PRACTICAL ALGORITHMS FOR LARGE SCALE CONVEX COMPOSITE CONIC PROGRAMMING PROBLEM

LAM XIN YEE

(B.Sc. (Hons.), NUS)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF MATHEMATICS
NATIONAL UNIVERSITY OF SINGAPORE

2019

Supervisor:

Professor Toh Kim Chuan

Examiners:

Professor Zhao Gong Yun

Associate Professor Karthik Natarajan, Singapore University of Technology & Design

Professor Kim Sunyoung, Ewha W. University

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Lam Xin Yee

23 February 2019

Acknowledgements

First of all, I would like to express my utmost gratitude to my supervisor, Professor Toh Kim Chuan. Since I worked with him for my undergraduate thesis, Prof Toh has given me a lot of useful advice and guidance, especially in the fields of numerical optimization and software. His enthusiasm in mathematics has motivated me a lot; without his professional guidance, this work would not be possible. I also appreciate Prof Toh's financial support for various conference trips. On top of this, his kind understanding during my difficult situations also touched me. I sincerely thank him from the depth of my heart.

Secondly, I would like to thank Professor Sun Defeng for his wise advice and comments. Prof Sun has been my co-supervisor for four years and he has always tried to push our ability beyond our limits. I have benefited a lot from his depth of knowledge in mathematical programming, especially in the field of convex optimization.

My thanks also goes to the senior members in our optimization group, including but not limited to Dr Li Xudong, Dr Cui Ying, and Dr Yang Liuqin; as well as my fellow coursemates including Zhang Yangjing, Dr Yang Lei, Yuan Yancheng, Lin Meixia, and Liang Ling. All of them have inspired me a lot through our discussion on optimization topics. I also thank them for the friendship throughout my PhD journey.

Also, I am grateful to my internship supervisor Mr Max Wong Chan Yue, and

Mr Patrick Ge Pei. They have provided me insightful knowledge on financial markets and a handy working experience in the financial industry.

I am also grateful to the university and the Mathematics department for providing me research scholarship for four years as well as the financial support for conference trips.

I should not forget to mention my fellow friends that I have met in NUS. The friendship that we built has been memorable, and I would also like to thank them for the discussion we have on general mathematics problems and also for passing the PhD qualifying examination.

Last but not least, I would like to dedicate this thesis to my family including my parents, my sister, and my husband for their love and support. Their kind encouragement and understanding are most appreciated.

Contents

Summary

In this thesis, we design and implement specialized algorithms for solving various large scale optimization problems arising from literature. It is mainly divided into three parts.

For the first part of the thesis, we study a binary classification problem arising from the field of statistics and machine learning. High dimension low sample size statistical analysis is important in a wide range of applications. In such situations, the highly appealing discrimination method, support vector machine (SVM), can be improved to alleviate data piling at the margin. This leads naturally to the development of distance weighted discrimination (DWD), which can be modeled as a second-order cone programming problem and solved by interior-point methods when the scale (in sample size and feature dimension) of the data is moderate. Here we design a scalable and robust algorithm for solving large scale generalized DWD problems. Numerical experiments on real data sets from the UCI repository demonstrate that our algorithm is highly efficient in solving large scale problems, and sometimes even more efficient than the highly optimized LIBLINEAR and LIBSVM for solving the corresponding SVM problems.

In the second part of the thesis, we turn our focus on a problem of special structure. We propose a semi-proximal augmented Lagrangian based decomposition method for convex composite quadratic conic programming problems

with primal block angular structures. Using our algorithmic framework, we can naturally derive several well-known augmented Lagrangian based decomposition methods for stochastic programming such as the diagonal quadratic approximation method of Mulvey and Ruszczyński. Moreover, we derive novel enhancements and generalizations of these well-known methods. We also propose a semi-proximal symmetric Gauss-Seidel based alternating direction method of multipliers for solving the corresponding dual problem. Numerical results show that our algorithms can perform well even for very large instances of primal block angular convex QP problems. For example, one instance with more than $300,000$ linear constraints and $12,500,000$ nonnegative variables is solved in less than a minute whereas Gurobi took more than 3 hours, and another instance `qp-gridgen1` with more than $331,000$ linear constraints and $986,000$ nonnegative variables is solved in about 5 minutes whereas Gurobi took more than 35 minutes.

A natural extension of the second part of this study is to look at optimization problems having dual block angular structure. Thus in the last part of this thesis, we design an inexact proximal augmented Lagrangian based decomposition methods for convex composite conic programming problems with dual block angular structures. Similarly, the algorithmic framework is based on the recently developed symmetric Gauss-Seidel (sGS) decomposition theorem for solving a proximal convex composite quadratic programming problem. Besides the guaranteed convergence, the advantage of our algorithm is that the computation of subproblems are easy to be parallelized, thus it possesses great potential for solving optimization models with a huge number of constraints and/or variables. Our methods are particularly well suited for convex quadratic conic programming problems arising from stochastic programming models. Furthermore, we present an application of the proposed algorithms to the doubly nonnegative relaxations of uncapacitated facility location problems.

Chapter 1

Introduction

In this thesis, we focus on designing and implementing efficient algorithms for solving various large scale optimization models arising from the literature. In particular, we are interested in solving structured convex programming problem with huge numbers of constraints and/or variables.

1.1 Motivation and related methods

We examine several methods that are popular nowadays for solving large scale optimization problem. Firstly, we investigate several variants of the alternating direction method of multipliers (ADMM), which is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which is then easier to handle. It has been applied widely into a number of statistical problems, such as constrained sparse regression, image restoration, etc. The classical ADMM was originally proposed by Glowinski and Marroco (1975) and Gabay and Mercier (1976) for solving a 2-block convex optimization problem with a collection of coupling linear constraints. Over the years, there have been many variations of ADMM proposed and applied to a great variety of optimization problems. A natural modification is to extend the original ADMM from two-block to multi-block settings. However, in Chen et al. (2016), it was shown that the directly extended ADMM may not be convergent. Thus it is necessary to make some modifications to the directly extended ADMM in order to get a convergent algorithm. In Sun et al. (2015), the authors proposed a

semi-proximal ADMM for solving a convex conic programming problem with 3 blocks of variables and 4 types of constraints. The algorithm is a convergent modification of the ADMM with an additional inexpensive step in each iteration. In Li et al. (2016), the authors proposed a Schur complement based (SCB) convergent semi-proximal ADMM for solving a multi-block linearly constrained convex programming problem whose objective function is the sum of two proper closed convex functions plus an arbitrary number of convex quadratic or linear functions. One of the key contributions in Li et al. (2016) is the discovery of the Schur complement based decomposition method which allows the multi-block subproblems to be solved efficiently while ensuring the convergence of the algorithm. More recently, Li et al. (2015) generalized the SCB decomposition method in Li et al. (2016) to the *inexact* symmetric Gauss-Seidel decomposition method, which provides an elegant framework and simpler derivation of the SCB decomposition method. Based on this previous research, in Chen et al. (2017), the authors proposed an *inexact* symmetric Gauss-Seidel based multi-block semi-proximal ADMM for solving a class of high-dimensional convex composite conic optimization problems, which has been demonstrated to have much better performance than the possibly non-convergent directly extended ADMM in solving high dimensional convex quadratic semidefinite programming problems.

Inspired by the above research, we find that ADMM is a potentially powerful tool in application for solving high dimensional structured convex programming problems, henceforth our work will depend heavily on this method. In the subsections below, we elaborate in details the motivations and related methods for each class of problems that we consider.

1.1.1 Generalized distance weighted discrimination

In the first part of this thesis, we consider the problem of finding a (kernelized) linear binary classifier for a training data set. By far, the most popular and successful method for getting a good linear classifier from the training data is the support vector machine (SVM), originally proposed by Vapnik (1995). Indeed, it has been demonstrated in Fernández-Delgado et al. (2014) that the kernel

SVM is one of the best performers in the pool of 179 commonly used classifiers. Despite its success, it has been observed in Marron et al. (2007) that SVM may suffer from a "data-piling" problem in the high-dimension-low-sample size (HDLSS) setting (where the sample size is smaller than the feature dimension). The authors proposed a new linear classifier, called "Distance Weighted Discrimination" (DWD), as a superior alternative to the SVM. DWD has become a workhorse method for a number of statistical tasks, including data visualization (Marron and Alonso 2014), hypothesis testing linked with visualization in very high dimensions (Wei et al. 2016), and adjustment for data biases and heterogeneity (Benito et al. 2004; Liu et al. 2009).

It is well known that there is a strong need for efficient HDLSS methods for the settings where the feature dimension is large, say in the order of 10^4–10^5, especially in the area of genetic molecular measurements (usually having a small sample size, where many gene level features have been measured), chemometrics (typically a small sample of high dimensional spectra) and medical image analysis (a small sample of 3-d shapes represented by high-dimensional vectors). On the other hand, given the advent of a huge volume of data collectible through various sources, especially from the internet, it is also important for us to consider the case where the sample size is large, while the feature dimension may be moderate. Thus in this thesis, we are interested in the problem of finding a linear classifier through DWD for data instances where sample size and/or feature dimension are large.

In Marron et al. (2007), DWD is formulated as a second-order cone programming (SOCP) problem, and the resulting model is solved by using a primal-dual interior-point method for SOCP problems implemented in the software SDPT3 (Toh et al. 1999). However, the IPM based solver employed for DWD in Marron et al. (2007) is computationally very expensive for solving problems where sample size or feature dimension is large, thus making it impractical for large scale problems. A recent approach to overcome such a computational bottleneck has appeared in Wang and Zou (2015), where the authors proposed a novel reformulation of the primal DWD model which consists of minimizing a highly nonlinear convex objective function subject to a ball constraint. The resulting problem is

solved via its dual and an MM (minimization-majorization) algorithm is designed to compute the Lagrangian dual function for each given dual multiplier. The algorithm appears to be quite promising in theory for solving large scale DWD problems. However, the current numerical experiments and implementation of the proposed algorithm in Wang and Zou (2015) are preliminary and limited to small scale data instances, and it appears that substantial work must be done to make it efficient for large scale instances. As it is premature to compare our proposed algorithm with the one in Wang and Zou (2015), we will not consider the latter algorithm any further in this thesis.

1.1.2 Primal block angular convex composite quadratic conic programming problems

In the second part of the thesis, we focus on solving convex composite quadratic conic programming problems with a primal block angular structure, i.e. optimization problems with a separable convex objective function and conic constraints but the variables are coupled by linking linear constraints across different variables. Without specially designed strategies to exploit the underlying block angular structure, it is expected that the computational inefficiency of a generic algorithm for solving such a class of problems will be severe because the constraints cannot be decomposed completely.

In practical applications, quadratic and linear problems with primal block angular structure appear in many contexts, such as multicommodity flow problems (Assad, 1978) and statistical disclosure control (Hundepool et al., 2012). These problems are often very large scale in practice, and standard interior point methods such as those implemented in Gurobi or Mosek may not be efficient enough to solve such problems. In the literature, specialized algorithms designed to solve these problems have been studied extensively. Three of the most widely known algorithmic classes are (i) decomposition methods based on augmented Lagrangian and proximal-point algorithms, see for example Mulvey and Ruszczyński (1992); Rockafellar and Wets (1991); Ruszczyński (1986, 1989, 1995, 1999); (ii) interior-point log-barrier Lagrangian decomposition methods

such as those studied in Zhao (1999, 2001, 2005); Mehrotra and Özevin (2007, 2009a); and (iii) standard interior-point methods which incorporate novel numerical linear algebraic techniques to exploit the underlying block angular structures when solving the large linear systems arising in each iteration, for example in Birge and Qi (1988); Choi and Goldfarb (1993); Gondzio et al. (1997); Schultz and Meyer (1991); Todd (1988).

Besides quadratic and linear problems, semidefinite programming (SDP) problems with primal block angular structures are beginning to appear in the literature more frequently. It is gaining more attention as practitioners become more sophisticated in using SDP to model their application problems. For example, Hanasusanto and Kuhn (2017) reformulated a two-stage distributionally robust linear program as a completely positive cone program which bears a block angular structure and applied the reformulation to solve a multi-item newsvendor problem. Although linear programming problems with primal block angular structures have been studied extensively, the more complicated SDP version is still in its infancy stage. Apart from Mehrotra and Özevin (2007), Sivaramakrishnan (2010) and Zhu and Ariyawansa (2011), we are not aware of other works.

By focusing on designing efficient algorithms for solving general conic programming problems with primal block angular structures, we can in general also use the same algorithmic framework to solve the primal block angular linear and quadratic programming problems efficiently through designing novel numerical linear algebraic techniques to exploit the underlying structures. In this thesis, our main objective is to design efficient and robust (distributed) algorithms for solving large scale conic programming problems with block angular structures. Specifically, we will design an inexact semi-proximal augmented Lagrangian method (ALM) for the primal problem which attempts to exploit the block angular structure to solve the problem in parallel. Our algorithm is motivated by the recent theoretical advances in inexact semi-proximal ALM that is embedded in Chen et al. (2017). In contrast to most existing augmented Lagrangian based decomposition algorithms where the solution for each subproblem must be computed exactly or to very high accuracy, our algorithm has the key advantage of allowing the subproblems to be solved approximately with

progressive accuracy. We will also elucidate the connection of our algorithm to the well-known diagonal quadratic approximation (DQA) algorithm of Mulvey and Ruszczyński Mulvey and Ruszczyński (1992).

In the pioneering work in Kontogiorgis et al. (1996), an ADMM based framework was designed for the primal block angular problem (PBA) wherein the variables are duplicated and auxiliary variables are introduced to make the first ADMM subproblem in each iteration solvable in a distributed fashion and that the succeeding second ADMM subproblem is a sufficiently simple quadratic program which is assumed to be easy to solve. However, the problem might still be difficult to solve if the scale of the original problem gets very large. To overcome the potential computational inefficiency induced by the extra variables and constraints, and also the relatively expensive step of having to solve a QP subproblem in each iteration in the primal approach, in this thesis we will adopt the dual approach of solving PBA. Specifically, we will design and implement a semi-proximal symmetric Gauss-Seidel based alternating direction method of multipliers (sGS-ADMM) to directly solve the dual problem, which will also solve the primal problem as a by-product. The advantage of tackling the dual problem directly is that no extra variables are introduced to decouple the constraints and no coupled QP subproblems are needed to be solved in each iteration. We note that the sGS-ADMM is an algorithm designed based on the recent advances in Chen et al. (2017); more details will be presented later.

1.1.3 Dual block angular convex composite conic programming problems

In the final part of the thesis, we study a another structured problem that can be considered as the dual of those studied in the second part. In this case, the structure it bears is a dual block angular (DBA) instead of primal block angular. The DBA problem generally has a separable convex objective function and conic constraints but the linear constraints are coupled by a linking variable across diferent constraints. The dimension of these problems can grow large easily especially when the number of blocks increases, thus it is advisable to devise a

specialized algorithm to fully exploit the block angular structure; otherwise the computational inefficiency will be severe.

In practical applications, optimization problem with DBA structure arised in many contexts. One of the most common problem class is the two-stage stochastic conic programming problems. The deterministic equivalence of this kind of problem is a typical example class of DBA problems. In the literature, some commonly used methods are interior point log-barrier decomposition algorithms such as those implemented by Mehrotra and Özevin (2009a,b); Zhao (2001), which successfully decompose the two stage stochastic problems by solving the two stages subproblem separately instead of solving the entire deterministic equivalence model. While this may work well for small problem, when the scale of the problem gets larger, the Newton systems arised in solving the subproblem might become more time consuming to be solved.

Another problem class having DBA structure is the uncapacitated facility location (UFL) problem. This is a model used to determine a subset of facility locations to open and to allocate each customer to opened facilities so that the cost of facility opening as well as customer allocation are minimized. This binary integer program is usually solved by Benders Decomposition such as those implemented in Fischetti et al. (2017). However, it is well known that directly solving an integer program is hard and is often time consuming, not to mention when the problem size is big. Thus relaxation is often needed in the process to gain some information about the solution to the original integer problem. In the context of this thesis, we study the doubly nonnegative relaxations of UFL problems and solve the resulting dual block angular semidefinite problem using our specialized algorithmic framework.

Our main objective here is to design and implement an efficient and robust (distributed) algorithms for solving large scale conic programming problems with dual block angular structures. We propose an inexact symmetric Gauss-Seidel based proximal ADMM (sGS-ADMM) to solve the dual problem which attempts to exploit the block angular structure and hence enable us to solve the subproblems efficiently. The advantage of using sGS-ADMM is that when the dimension

of the linear system of equation arising from the subproblem get larger, we can simply add a symmetric positive semidefinite proximal term so that the linear system of equation can be decomposed and solved in parallel. This method is particularly well-suited for convex composite problem with multiple blocks of variables, and is again inspired by the recent theoretical advances in inexact sGS-ADMM that is proposed in Chen et al. (2017).

1.2 Contributions

In the first part of the thesis, the main contribution is to design a new method for solving large scale DWD problems, where we target to solve a problem with the sample size $n \approx 10^4$–10^6 and/or the dimension $d \approx 10^4$–10^5. Our method is a convergent 3-block semi-proximal alternating direction method of multipliers (ADMM), which is designed based on the recent advances in research on convergent multi-block ADMM-type methods (Sun et al. 2015; Li et al. 2016; Chen et al. 2017) for solving convex composite quadratic conic programming problems.

To be more specific, the first contribution we make is in reformulating the primal formulation of the *generalized* DWD model (using the terminology from Wang and Zou (2015)) and adapting the powerful inexact sGS-ADMM framework for solving the reformulated problem. This is in contrast to numerous SVM algorithms which are primarily designed for solving the dual formulation of the SVM model. The second contribution we make is in designing highly efficient numerical techniques to solve the subproblems in each of the inexact sGS-ADMM iterations. If n or d is moderate, then the complexity at each iteration is $O(nd) + O(n^2)$ or $O(nd) + O(d^2)$ respectively. If both n and d are large, then we employ the conjugate gradient iterative method for solving the large linear systems of equations involved. We also devise various strategies to speed up the practical performance of the sGS-ADMM algorithm in solving large scale instances (with the largest instance having $n = 256,000$ and $d \approx 3 \times 10^6$) of DWD problems with real data sets from the UCI machine learning repository (Lichman 2013). We should emphasize that the key in achieving high efficiency in our algorithm depends very much on the intricate numerical techniques and

sophisticated implementation we have developed.

In the second part of the thesis, our first contribution is in proposing several variants of augmented Lagrangian based algorithms for directly solving the primal form (PBA) of the convex composite quadratic conic programming problem with a primal block angular structure. We also show that they can be considered as generalizations of the well-known DQA method. Our second contribution is in the design and implementation of a specialized algorithm for solving the dual problem of (PBA). The algorithm is easy to implement and highly amenable to parallelization. Hence we expect it to be highly scalable for solving large scale problems with millions of variables and constraints.

In the third part of the thesis, our first contribution is to design an efficient and scalable algorithm for solving the dual block angular (DBA) form of the convex composite conic programming problem. Besides conducting numerical experiments on various practical dataset, we also generate some random DBA problems. Our numerical results show that our algorithm performs well on the random DBA problems. As a side products, we also derived concrete DBA convex composite conic programming problems arising from doubly nonnegative relaxations of common mixed integer programming problems such as uncapacitated facility location problems.

In every problem we mentioned, we implement the code in MATLAB and conduct comprehensive numerical experiments to evaluate the performance of our algorithms against other highly competitive state-of-the-art solvers such as Gurobi. Numerical results show that our algorithms can achieve good performance and sometimes much better than the state-of-the-art solvers.

1.3 Thesis organization

The rest of this thesis is organized as follows. In Chapter 2, we present some preliminaries that are related to our subsequent discussions, which includes the introduction of the symmetric Gauss-Seidel (sGS) decomposition theorem and the sGS based alternating direction method of multipliers. In Chapter 3, we design an efficient algorithm for solving the large scale generalized distance weighted

discrimination problems. We provide the convergence theorem of our proposed algorithm as well as the numerical experiment. Next, we solve primal block angular convex composite quadratic conic programming problems in chapter 4. Algorithms are presented in details for both primal and dual formulation. In chapter 5, we present an algorithmic framework for solving dual block angular convex composite conic programming problems. In particular, we focus on its dual formulation and the possible application in solving the SDP relaxation of uncapacitated facility location problem. Finally, we give conclusion of this thesis and discuss several possible future research directions in the last chapter.

Chapter 2

Preliminaries

2.1 Notations

We would like to emphasize that the meaning of a symbol might differ in each chapter. For example, y is a class label in chapter 3, but it is just an ordinary dual variable in chapter 4. On the other hand, the following mathematical notations are used throughout the entire thesis:

- We denote the 2-norm of a vector x by $\|x\|$, and the Frobenius norm of a matrix M by $\|M\|_F$. The inner product of two vectors x, y is denoted by $\langle x,\, y\rangle$.
- If S is a symmetric positive semidefinite matrix, then we denote the weighted norm of a vector x with the weight matrix S by $\|x\|_S := \sqrt{\langle x,\, Sx\rangle}$.
- We denote $[P; Q]$ or $(P; Q)$ as the matrix obtained by appending the matrix Q to the last row of the matrix P, whereas we denote $[P, Q]$ or (P, Q) as the matrix obtained by appending Q to the last column of matrix P, assuming that they have the same number of columns or rows respectively. We also use the same notation symbolically for P and Q which are linear maps with compatible domains and co-domains.
- For any linear map $\mathcal{T} : \mathcal{X} \to \mathcal{Y}$, we denote its adjoint as $\mathcal{T}^*$. If $\mathcal{X} = \mathcal{Y}$, and $\mathcal{T}$ is self-adjoint and positive semidefinite, then for any $x \in \mathcal{X}$ we have the notation $\|x\|_{\mathcal{T}} := \sqrt{\langle x,\, \mathcal{T}x\rangle}$.

- Let $f : \mathcal{X} \to (-\infty, +\infty]$ be an arbitrary closed proper convex function. We denote dom$f$ as its effective domain and ∂f as its subdifferential mapping. The Fenchel conjugate function of f is denoted as f^*.
- The Moreau-Yosida proximal mapping of f is defined by $\mathrm{Prox}_f(y) := \arg\min_x\{f(x) + \frac{1}{2}\|x - y\|^2\}$.

2.2 The symmetric Gauss-Seidel (sGS) decomposition theorem

In this section, we discuss the sGS decomposition theorem, which is an important theorem that our algorithms rely heavily on.

Let $\mathcal{Q} : \mathcal{X} \to \mathcal{X}$ be a given self-adjoint positive semidefinite linear operator, and $\eta \in \mathcal{X}$. Suppose that $\mathcal{Q}$ is partitioned according to $\mathcal{X}_1 \times \cdots \times \mathcal{X}_s$ as

$$\mathcal{Q} = \begin{bmatrix} Q_{1,1} & \cdots & Q_{1,s} \\ \vdots & \vdots & \vdots \\ Q_{1,s}^* & \cdots & Q_{s,s} \end{bmatrix} = \mathcal{U} + \mathcal{D} + \mathcal{U}^*, \tag{2.1}$$

where $\mathcal{D}$ and $\mathcal{U}$ denote the block diagonal part and the strictly upper block triangular part of $\mathcal{Q}$, respectively. That is,

$$\mathcal{U} = \begin{bmatrix} \mathbf{0} & Q_{1,2} & \cdots & Q_{1,s} \\ & \ddots & & \vdots \\ & & \ddots & Q_{s-1,s} \\ & & & \mathbf{0} \end{bmatrix}, \quad \mathcal{D} = \begin{bmatrix} Q_{1,1} & & & \\ & Q_{2,2} & & \\ & & \ddots & \\ & & & Q_{s,s} \end{bmatrix}. \tag{2.2}$$

Assuming that Q_{ii} is positive definite for all $i = 1, \ldots, s$. Then we can define the following symmetric Gauss-Seidel linear operator associated with $\mathcal{Q}$:

$$\mathbf{sGS}(\mathcal{Q}) = \mathcal{U}\mathcal{D}^{-1}\mathcal{U}^*.$$

For a given $x = (x_1; \ldots; x_s)$, we define

$$x_{\geq i} = (x_i; \ldots; x_s), \quad x_{\leq i} = (x_1; \ldots; x_i), \quad i = 1, \ldots, s.$$

We also define $x_{\geq s+1} = \emptyset$.

Theorem 2.1. (**sGS decomposition theorem**) Let $u \in \mathcal{X}$ be given. Suppose $\delta', \delta \in \mathcal{X}$ are two given error vectors with $\delta'_1 = \delta_1$. Define

$$\Delta(\delta', \delta) := \delta + \mathcal{U}\mathcal{D}^{-1}(\delta - \delta'). \tag{2.3}$$

Let $p : \mathcal{X}_1 \to (-\infty, \infty]$ be a proper closed convex function and $h(x) = \frac{1}{2}\langle x, \mathcal{Q}x\rangle - \langle \eta, x\rangle \ \forall\ x \in \mathcal{X}$. Consider the following convex composite quadratic programming problem:

$$x^+ := \operatorname{argmin} \Big\{ p(x_1) + h(x) + \frac{1}{2}\|x - u\|^2_{\mathcal{S}} - \langle x, \Delta(\delta', \delta)\rangle \ \mid \ x \in \mathcal{X} \Big\}, \tag{2.4}$$

where $\mathcal{S} = \mathbf{sGS}(\mathcal{Q})$ is the symmetric Gauss-Seidel linear operator associated with $\mathcal{Q}$. Then x^+ can be computed in the following symmetric Gauss-Seidel fashion. For $i = s, \ldots, 2$, compute $x'_i \in \mathcal{X}_i$ defined by

$$\begin{aligned} x'_i &:= \operatorname*{argmin}_{x_i \in \mathcal{X}_i} \ p(u_1) + h(u_{\leq i-1}; x_i; x'_{\geq i+1}) - \langle \delta'_i, x_i\rangle \\ &= Q^{-1}_{i,i}\big(\eta_i + \delta'_i - \textstyle\sum_{j=1}^{i-1} Q^*_{j,i} u_j - \sum_{j=i+1}^{s} Q_{i,j} x'_j\big). \end{aligned} \tag{2.5}$$

Then the optimal solution x^+ for (2.4) can be computed exactly via the following steps:

$$\begin{cases} x^+_1 = \operatorname*{argmin}_{x_1 \in \mathcal{X}_1} \ p(x_1) + h(x_1; x'_{\geq 2}) - \langle \delta_1, x_1\rangle, \\ x^+_i = \operatorname*{argmin}_{x_i \in \mathcal{X}_i} \ p(x^+_1) + h(x^+_{\leq i-1}; x_i; x'_{\geq i+1}) - \langle \delta_i, x_i\rangle \\ \qquad = Q^{-1}_{i,i}\big(\eta_i + \delta_i - \textstyle\sum_{j=1}^{i-1} Q^*_{j,i} x^+_j - \sum_{j=i+1}^{s} Q_{i,j} x'_j\big), \quad i = 2, \ldots, s. \end{cases} \tag{2.6}$$

Proof. See (Li et al., 2018a, Theorem1). □

Note that the role of the error vectors δ' and δ in the above block sGS decomposition theorem is to allow for inexact computation in (2.5) and (2.6). There is no need to know these error vectors in advanced. We should view the computed x_i' and x_i^+ from (2.5) and (2.6) as approximate solutions to the minimization subproblems without the terms involving δ_i' and δ_i. Once these approximate solutions have been computed, they would generate δ_i' and δ_i automatically. With these known error vectors, we know that the computed approximate solutions are the exact solutions to the slightly perturbed minimization problems in (2.5) and (2.6). In particular, for $i = s, \ldots, 2$, we know that δ_i' is the residual vector obtained when we solve the linear system of equations $Q_{ii}x_i' = \eta_i - \sum_{j=1}^{i-1} Q_{j,i}^* u_j - \sum_{j=i+1}^{s} Q_{i,j}x_j'$ in (2.5). Similar δ_i $(i = 2, \ldots, s)$ is the residual vector obtained when we solve the linear system of equations in (2.6).

2.3 The inexact sGS-ADMM

Now we shall describe briefly the inexact sGS based majorized semiproximal ADMM (sGS-imsPADMM) that was introduced in Chen et al. (2017).

Let $\mathcal{Z}$, $\mathcal{X}_1, \ldots, \mathcal{X}_s$ $(s \geq 2)$ and $\mathcal{Y}_1, \ldots, \mathcal{Y}_t$ $(t \geq 2)$ be given finite-dimensional inner product spaces. Also, define $\mathcal{X} := \mathcal{X}_1 \times \cdots \times \mathcal{X}_s$ and $\mathcal{Y} := \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_t$. Consider the following general convex composite programming model:

$$\begin{array}{lll} \text{(GCCP)} & \min_{x\in\mathcal{X}, y\in\mathcal{Y}} & p_1(x_1) + f(x_1, ..., x_s) + q_1(y_1) + g(y_1, ..., y_t) \\ & \text{s.t.} & \mathcal{A}^*x + \mathcal{B}^*y = c, \end{array}$$

where $p_1 : \mathcal{X}_1 \to (-\infty, \infty]$ and $q_1 : \mathcal{Y}_1 \to (-\infty, \infty]$ are two closed proper convex functions, $f : \mathcal{X} \to (-\infty, \infty)$ and $g : \mathcal{Y} \to (-\infty, \infty)$ are continuously differentiable convex functions whose gradients are Lipschitz continuous. The linear mappings $\mathcal{A} : \mathcal{Z} \to \mathcal{X}$ and $\mathcal{B} : \mathcal{Z} \to \mathcal{Y}$ are defined such that their adjoints are given by $\mathcal{A}^*x = \sum_{i=1}^{s} \mathcal{A}_i^* x_i$ for $x = (x_1, \cdots, x_s) \in \mathcal{X}$, and $\mathcal{B}^*y = \sum_{j=1}^{t} \mathcal{B}_j^* y_j$ for $y = (y_1, \cdots, y_t) \in \mathcal{Y}$, where $\mathcal{A}_i^* : \mathcal{X}_i \to \mathcal{Z}$, $i = 1, \cdots, s$ and $\mathcal{B}_j^* : \mathcal{Y}_j \to \mathcal{Z}$, $j = 1, ..., t$ are the adjoints of the linear maps $\mathcal{A}_i : \mathcal{Z} \to \mathcal{X}_i$ and $\mathcal{B}_j : \mathcal{Z} \to \mathcal{Y}_i$ respectively. For notational convenience, we define the functions $p : \mathcal{X} \to (-\infty, \infty]$ and $q : \mathcal{Y} \to (-\infty, \infty]$ by $p(x) := p_1(x_1)$ and $q(y) := q_1(y_1)$.

The augmented Lagrangian function of (GCCP) is given by

$$\mathcal{L}_\sigma(x, y; z) := p(x) + f(x) + q(y) + g(y) + \langle z,\, \mathcal{A}^* x + \mathcal{B}^* y - c\rangle + \frac{\sigma}{2}\|\mathcal{A}^* x + \mathcal{B}^* y - c\|^2.$$

(GCCP) could be solved by many methods in the literature; one of the most popular algorithm is the alternating direction method of multipliers (ADMM). As mentioned in section 1.1, the direct extension of the original ADMM from two-block to multi-block setting may not be convergent and hence various enhancements have been done to improve the convergence; see for example Sun et al. (2015); Li et al. (2016, 2015); Chen et al. (2017). In particular, based on the sGS decomposition theorem, by applying the corresponding sGS operator, the simplified version of the algorithmic framework for solving (GCCP) described by Chen et al. (2017) has the following template:

sGS-imsPADMM: An inexact sGS based majorized semi-proximal ADMM for solving (GCCP). Suppose $\mathcal{S}_{x_i},\ i=1,...,s$ and $\mathcal{S}_{y_j},\ j=1,...,t$ are self-adjoint positive semidefinite linear operators. Let $\tau \in (0, \frac{1+\sqrt{5}}{2})$ be the step-length and $\{\tilde{\epsilon}_k\}_{k\geq 0}$ be a summable sequence of nonnegative numbers. Choose $(x^0, y^0, z^0) \in \mathrm{dom}p \times \mathrm{dom}q \times \mathcal{Z}$. For $k = 0, 1, ...,$ perform the following steps:

Step 1a. (Backward GS sweep) Compute for $i = s, ..., 2$

$$\bar{x}_i^{k+1} \approx \underset{x_i\in\mathcal{X}_i}{\mathrm{argmin}}\{\mathcal{L}_\sigma(x_{\leq i-1}^k, x_i, \bar{x}_{\geq i+1}^{k+1}, y^k; z^k) + \frac{1}{2}\|x_i - x_i^k\|^2_{\mathcal{S}_{x_i}}\},$$

such that there exists

$$\bar{\delta}_i^k \in \partial_{x_i}\mathcal{L}_\sigma(x_{\leq i-1}^k, \bar{x}_i^{k+1}, \bar{x}_{\geq i+1}^{k+1}, y^k; z^k) + \mathcal{S}_{x_i}(\bar{x}_i^{k+1} - x_i^k)$$

satisfying $\|\bar{\delta}_i^k\| \leq \tilde{\epsilon}_k$.

Step 1b. (Forward GS sweep) Compute for $i = 1, ..., s$

$$x_i^{k+1} \approx \underset{x_i\in\mathcal{X}_i}{\mathrm{argmin}}\{\mathcal{L}_\sigma(x_{\leq i-1}^{k+1}, x_i, \bar{x}_{\geq i+1}^{k+1}, y^k; z^k) + \frac{1}{2}\|x_i - x_i^k\|^2_{\mathcal{S}_{x_i}}\},$$

such that there exists

$$\delta_i^k \in \partial_{x_i}\mathcal{L}_\sigma(x_{\leq i-1}^{k+1}, x_i^{k+1}, \bar{x}_{\geq i+1}^{k+1}, y^k; z^k) + \mathcal{S}_{x_i}(x_i^{k+1} - x_i^k)$$

with $\|\delta_i^k\| \leq \tilde{\epsilon}_k$.

Step 2a. (Backward GS sweep) Compute for $j = t, ..., 2$

$$\bar{y}_j^{k+1} \approx \underset{y_j\in\mathcal{Y}_j}{\mathrm{argmin}}\{\mathcal{L}_\sigma(x^{k+1}, y_{\leq j-1}^k, y_j, \bar{y}_{\geq j+1}^{k+1}; z^k) + \frac{1}{2}\|y_j - y_j^k\|^2_{\mathcal{S}_{y_j}}\},$$

such that there exists

$$\bar{\gamma}_j^k \in \partial_{y_j}\mathcal{L}_\sigma(x^{k+1}, y_{\leq j-1}^k, \bar{y}_j^{k+1}, \bar{y}_{\geq j+1}^{k+1}; z^k) + \mathcal{S}_{y_j}(\bar{y}_j^{k+1} - y_j^k)$$

with $\|\bar{\gamma}_j^k\| \leq \tilde{\epsilon}_k$.

Step 2b. (Forward GS sweep) Compute for $j = 1, ..., t$

$$y_j^{k+1} \approx \underset{y_j\in\mathcal{Y}_j}{\mathrm{argmin}}\{\mathcal{L}_\sigma(x^{k+1}, y_{\leq j-1}^{k+1}, y_j, \bar{y}_{\geq j+1}^{k+1}; z^k) + \frac{1}{2}\|y_j - y_j^k\|^2_{\mathcal{S}_{y_j}}\},$$

such that there exists

$$\gamma_j^k \in \partial_{y_j}\mathcal{L}_\sigma(x^{k+1}, y_{\leq j-1}^{k+1}, y_j^{k+1}, \bar{y}_{\geq j+1}^{k+1}; z^k) + \mathcal{S}_{y_j}(y_j^{k+1} - y_j^k)$$

with $\|\gamma_j^k\| \leq \tilde{\epsilon}_k$.

Step 3. Compute $z^{k+1} = z^k + \tau\sigma(\mathcal{A}^*x + \mathcal{B}^*y - c)$.

In short, by adding an inexpensive step in step 1a and step 2a respectively (note that these are the steps that do not involve the expensive computation of the possibly nonsmooth objective p and q), we can improve the convergence of the directly extended ADMM in multi-block setting. In the following parts of the thesis, most of our problems will have a multiblock setting and hence it would be useful to apply this sGS-imsPADMM framework. We defer the details, including the specific applications and the corresponding convergence theorems to later chapters.

Chapter 3

Fast algorithms for large scale generalized distance weighted discrimination

In this chapter, we focus on designing a new method for solving the generalized distance weighted discrimination model to find a linear classifier for a training data set $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and the class label $y_i \in \{-1, 1\}$ for all $i = 1, \ldots, n$ for data instances. Here we target to solve a problem with the sample size $n \approx 10^4$–10^6 and/or the dimension $d \approx 10^4$–10^5. Our method is a convergent 3-block semi-proximal alternating direction method of multipliers (ADMM), which is designed based on the recent advances in research on convergent multi-block ADMM-type methods (Sun et al. 2015; Li et al. 2016; Chen et al. 2017) for solving convex composite quadratic conic programming problems.

We will conduct extensive numerical experiments to evaluate the performance of our proposed algorithm against a few other alternatives. Relative to the primal-dual interior-point method used in Marron et al. (2007), our algorithm is vastly superior in terms of computational time and memory usage in solving large scale problems, where our algorithm can be a few thousands times faster. By exploiting all the highly efficient numerical techniques we have developed in the implementation of the sGS-ADMM algorithm for solving the generalized

DWD problem, we can also get an efficient implementation of the possibly non-convergent directly extended ADMM for solving the same problem. On the tested problems, our algorithm generally requires fewer iterations compared to the directly extended ADMM even when the latter is convergent. On quite a few instances, the directly extended ADMM actually requires many more iterations than our proposed algorithm to solve the problems. We also compare the efficiency of our algorithm in solving the generalized DWD problem against the highly optimized LIBLINEAR (Fan et al. 2008) and LIBSVM (Chang and Lin 2011) in solving the corresponding dual SVM problem. Surprisingly, our algorithm can even be more efficient than LIBSVM in solving large scale problems even though the DWD model is more complex, and on some instances, our algorithm is 50–100 times faster. Our DWD model is also able to produce the best test (or generalization) errors compared to LIBLINEAR and LIBSVM among the tested instances.

The remaining parts of this chapter are organized as follows. In section 3.1, we present the DWD formulation in full detail. In section 3.2, we propose our inexact sGS-based ADMM method for solving large scale DWD problems. We also discuss some essential computational techniques used in our implementation. We report our numerical results in section 3.3. We will also compare the performance of our algorithm to other solvers on the same data sets in this particular section.

3.1 Generalized distance weighted discrimination

This section gives details on the optimization problems underlying the distance weighted discrimination. Let (x_i, y_i), $i = 1, \ldots, n$, be the training data where $x_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \{+1, -1\}$ is its corresponding class label. We let $X \in \mathbb{R}^{d\times n}$ be the matrix whose columns are the x_i's, and $y = [y_1, \ldots, y_n]^T$. In linear discrimination, we attempt to separate the vectors in the two classes by a hyperplane $H = \{x \in \mathbb{R}^d \mid w^T x + \beta = 0\}$, where $w \in \mathbb{R}^d$ is the unit normal and $|\beta|$ is its distance to the origin. Given a point $z \in \mathbb{R}^d$, the signed distance between z and the hyperplane H is given by $w^T z + \beta$. For binary classification

where the label $y_i \in \{-1, 1\}$, we want

$$y_i(\beta + x_i^T w) \geq 1 - \xi_i \quad \forall\, i = 1, ..., n,$$

where we have added a slack variable $\xi \geq 0$ to allow the possibility that the positive and negative data points may not be separated cleanly by the hyperplane. In matrix-vector notation, we need

$$r \;:=\; Z^T w + \beta y + \xi \;\geq\; \mathbf{1}, \tag{3.1}$$

where $Z = X\mathrm{diag}(y)$ and $\mathbf{1} \in \mathbb{R}^n$ is the vector of ones.

In SVM, w and β are chosen by maximizing the minimum residual, i.e.,

$$\max\Big\{\delta - C\langle \mathbf{1},\, \xi\rangle \mid Z^T w + \beta y + \xi \geq \delta \mathbf{1},\; \xi \geq 0,\; w^T w \leq 1\Big\}, \tag{3.2}$$

where $C > 0$ is a tuning parameter to control the level of penalization on ξ. For the DWD approach introduced in Marron et al. (2007), w and β are chosen instead by minimizing the sum of reciprocals of the r_i's, i.e.,

$$\min\Big\{\sum_{i=1}^{n} \frac{1}{r_i} + C\langle \mathbf{1},\, \xi\rangle \mid r = Z^T w + \beta y + \xi,\; r > 0,\; \xi \geq 0,\; w^T w \leq 1,\; w \in \mathbb{R}^d\Big\}. \tag{3.3}$$

Detailed discussions on the connections between the DWD model (3.3) and the SVM model (3.2) can be found in Marron et al. (2007). The DWD optimization problem (3.3) is shown to be equivalent to a second-order cone programming problem in Marron et al. (2007) and hence it can be solved by interior-point methods such as those implemented in the solver SDPT3 (Toh et al. 1999).

Here we design an algorithm which is capable of solving large scale generalized DWD problems of the following form:

$$\min\Big\{\Phi(r, \xi) := \sum_{i=1}^{n} \theta_q(r_i) + C\langle e,\, \xi\rangle \mid\; Z^T w + \beta y + \xi - r = 0,\;\; \|w\| \leq 1,\; \xi \geq 0\Big\}, \tag{3.4}$$

where $e \in \mathbb{R}^n$ is a given positive vector such that $\|e\|_\infty = 1$ (the last condition is for the purpose of normalization). The exponent q can be any given positive

number, though the values of most interest are likely to be $q = 0.5, 1, 2, 4$, and $\theta_q(r_i)$ is the function defined by

$$\theta_q(t) = \frac{1}{t^q} \quad \text{if } t > 0, \text{ and } \quad \theta_q(t) = \infty \quad \text{if } t \le 0.$$

Observe that in addition to allowing for a general exponent q in (3.4), we also allow for a nonuniform weight $e_i > 0$ in the penalty term for each ξ_i. By a simple change of variables and modification of the data vector y, (3.4) can also include the case where the terms in $\sum_{i=1}^n \frac{1}{r_i^q}$ are weighted non-uniformly. For brevity, we omit the details.

Proposition 1. Let $\kappa = \frac{q+1}{q} q^{\frac{1}{q+1}}$. The dual of problem (3.4) is given as follows:

$$-\min_{\alpha} \Big\{ \Psi(\alpha) := \|Z\alpha\| - \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} \mid 0 \le \alpha \le Ce, \langle y, \alpha \rangle = 0 \Big\}, \tag{3.5}$$

Proof. Consider the Lagrangian function associated with (3.4):

$$\begin{aligned} & L(r, w, \beta, \xi; \alpha, \eta, \lambda) \\ = \;& \textstyle\sum_{i=1}^n \theta_q(r_i) + C\langle e, \xi \rangle - \langle \alpha, Z^T w + \beta y + \xi - r \rangle + \frac{\lambda}{2}(\|w\|^2 - 1) - \langle \eta, \xi \rangle \\ = \;& \textstyle\sum_{i=1}^n \theta_q(r_i) + \langle r, \alpha \rangle + \langle \xi, Ce - \alpha - \eta \rangle - \beta\langle y, \alpha \rangle - \langle w, Z\alpha \rangle + \frac{\lambda}{2}(\langle w, w \rangle - 1), \end{aligned}$$

where $r \in \mathbb{R}^n$, $w \in \mathbb{R}^d$, $\beta \in \mathbb{R}$, $\xi \in \mathbb{R}^n$, $\alpha \in \mathbb{R}^n$, $\lambda, \eta \ge 0$. Now

$$\inf_{r_i} \Big\{ \theta_q(r_i) + \alpha_i r_i \Big\} = \begin{cases} \kappa\, \alpha_i^{\frac{q}{q+1}} & \text{if } \alpha_i \ge 0, \\ -\infty & \text{if } \alpha_i < 0; \end{cases}$$

$$\inf_{w} \Big\{ -\langle Z\alpha, w \rangle + \frac{\lambda}{2}\|w\|^2 \Big\} = \begin{cases} -\frac{1}{2\lambda}\|Z\alpha\|^2 & \text{if } \lambda > 0, \\ 0 & \text{if } \lambda = 0,\ Z\alpha = 0, \\ -\infty & \text{if } \lambda = 0,\ Z\alpha \ne 0; \end{cases}$$

$$\inf_{\xi} \Big\{ \langle \xi, Ce - \alpha - \eta \rangle \Big\} = \begin{cases} 0 & \text{if } Ce - \alpha - \eta = 0, \\ -\infty & \text{otherwise;} \end{cases}$$

$$\inf_{\beta} \Big\{ -\beta\langle y, \alpha \rangle \Big\} = \begin{cases} 0 & \text{if } \langle y, \alpha \rangle = 0, \\ -\infty & \text{otherwise.} \end{cases}$$

Let $F_D = \{\alpha \in \mathbb{R}^n \mid 0 \le \alpha \le Ce, \langle y,\, \alpha\rangle = 0\}$. Hence

$$\min_{r,w,\beta,\xi} L(r,w,\beta,\xi;\alpha,\eta,\lambda) = \begin{cases} \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} - \frac{1}{2\lambda}\|Z\alpha\|^2 - \frac{\lambda}{2}, & \text{if } \lambda > 0,\ \alpha \in F_D, \\ \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}}, & \text{if } \lambda = 0,\ Z\alpha = 0,\ \alpha \in F_D, \\ -\infty, & \text{if } \lambda = 0,\ Z\alpha \neq 0,\ \alpha \in F_D, \text{ or } \alpha \notin F_D. \end{cases}$$

Now for $\alpha \in F_D$, we have

$$\max_{\lambda\ge 0,\eta\ge 0} \left\{ \min_{r,w,\beta,\xi} L(r,w,\beta,\xi;\alpha,\eta,\lambda)\right\} = \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} - \|Z\alpha\|.$$

From here, we get the required dual problem. □

It is straightforward to show that the feasible regions of (3.4) and (3.5) both have nonempty interiors. Thus optimal solutions for both problems exist and they satisfy the following KKT (Karush-Kuhn-Tucker) optimality conditions:

$$\begin{aligned} &Z^T w + \beta y + \xi - r = 0, && \langle y,\, \alpha\rangle = 0, \\ &r > 0,\ \alpha > 0, \quad \alpha \le Ce, \quad \xi \ge 0, && \langle Ce - \alpha,\, \xi\rangle = 0, \\ &\alpha_i = \tfrac{q}{r_i^{q+1}},\ i = 1,\ldots,n, && \text{either } w = \tfrac{Z\alpha}{\|Z\alpha\|}, \text{ or } Z\alpha = 0,\ \|w\|^2 \le 1. \end{aligned} \tag{3.6}$$

Let $(r^*, \xi^*, w^*, \beta^*)$ and α^* be an optimal solution of (3.4) and (3.5), respectively. Next we analyse some properties of the optimal solution. In particular, we show that the optimal solution α^* is bounded away from 0.

Proposition 2. There exists a positive δ such that $\alpha_i^* \ge \delta \ \ \forall\, i = 1,\ldots,n$.

Proof. For convenience, let $F_P = \{(r,\xi,w,\beta) \mid Z^T w + \beta y + \xi - r = 0, \|w\| \le 1, \xi \ge 0\}$ be the feasible region of (3.4). Since $(\mathbf{1},\mathbf{1},0,0) \in F_P$, we have that

$$Ce_{\min}\xi_i^* \le C\langle e,\, \xi^*\rangle \le \Phi(r^*,\xi^*,w^*,\beta^*) \le \Phi(\mathbf{1},\mathbf{1},0,0) = n + C\textstyle\sum_{i=1}^n e_i \quad \forall\, i = 1,\ldots,n,$$

where $e_{\min} = \min_{1\le i\le n}\{e_i\}$. Hence we have $0 \le \xi^* \le \varrho\mathbf{1}$, where $\varrho := \frac{n + C\sum_{i=1}^n e_i}{Ce_{\min}}$.

Next, we establish a bound for $|\beta^*|$. Suppose $\beta^* > 0$. Consider an index i such that $y_i = -1$. Then $0 < \beta^* = Z_i^T w^* + \xi_i^* - r_i^* \le \|Z_i\|\|w^*\| + \xi_i^* \le K + \varrho$, where Z_i denotes the ith column of Z, $K = \max_{1\le j\le n}\{\|Z_j\|\}$. On the other

hand, if $\beta^* < 0$, then we consider an index k such that $y_k = 1$, and $0 < -\beta^* = Z_k^T w^* + \xi_k^* - r_k^* \leq K + \varrho$. To summarize, we have that $|\beta^*| \leq K + \varrho$.

Now we can establish an upper bound for r^*. For any $i = 1, \ldots, n$, we have that

$$r_i^* = Z_i^T w^* + \beta^* y_i + \xi_i^* \leq \|Z_i\|\|w^*\| + |\beta^*| + \xi_i^* \leq 2(K + \varrho).$$

From here, we get $\alpha_i^* = \frac{q}{(r_i^*)^{q+1}} \geq \delta := \frac{q}{(2K+2\varrho)^{q+1}} \; \forall \, i = 1, \ldots, n$. This completes the proof of the proposition. □

3.2 An inexact SGS-based ADMM for large scale DWD problems

We can rewrite the model (3.4) as:

$$\min \left\{ \sum_{i=1}^{n} \theta_q(r_i) + C\langle e, \xi \rangle + \delta_B(w) + \delta_{\mathbb{R}^n_+}(\xi) \Big| \begin{array}{l} Z^T w + \beta y + \xi - r = 0, \\ w \in \mathbb{R}^d, \; r, \xi \in \mathbb{R}^n \end{array} \right\},$$

where $B = \{w \in \mathbb{R}^d \mid \|w\| \leq 1\}$. Here, both $\delta_B(w)$ and $\delta_{\mathbb{R}^n_+}(\xi)$ are infinity indicator functions. In general, an infinity indicator function over a set $\mathcal{C}$ is defined by:

$$\delta_{\mathcal{C}}(x) := \begin{cases} 0, & \text{if } x \in \mathcal{C}; \\ +\infty, & \text{otherwise.} \end{cases}$$

The model above is a convex minimization problem with three nonlinear blocks. By introducing an auxiliary variable $u = w$, we can reformulate it as:

$$\begin{array}{ll} \min & \sum_{i=1}^n \theta_q(r_i) + C\langle e, \xi \rangle + \delta_B(u) + \delta_{\mathbb{R}^n_+}(\xi) \\ \text{s.t.} & Z^T w + \beta y + \xi - r = 0, \\ & D(w - u) = 0, \;\; w, u \in \mathbb{R}^d, \; \beta \in \mathbb{R}, \; r, \xi \in \mathbb{R}^n, \end{array} \tag{3.7}$$

where $D \in \mathbb{R}^{d \times d}$ is a given positive scalar multiple of the identity matrix which is introduced for the purpose of scaling the variables.

For a given parameter $\sigma > 0$, the augmented Lagrangian function associated with (3.7) is given by

$$\begin{aligned} & L_\sigma(r, w, \beta, \xi, u; \alpha, \rho) \\ = \; & \textstyle\sum_{i=1}^n \theta_q(r_i) + C\langle e,\, \xi\rangle + \delta_B(u) + \delta_{\mathbb{R}^n_+}(\xi) + \frac{\sigma}{2}\|Z^T w + \beta y + \xi - r - \sigma^{-1}\alpha\|^2 \\ & + \tfrac{\sigma}{2}\|D(w-u) - \sigma^{-1}\rho\|^2 - \tfrac{1}{2\sigma}\|\alpha\|^2 - \tfrac{1}{2\sigma}\|\rho\|^2. \end{aligned}$$

The algorithm which we will design later is based on recent progress in algorithms for solving multi-block convex conic programming. In particular, our algorithm is designed based on the inexact ADMM algorithm in Chen et al. (2017) and we made essential use of the inexact symmetric Gauss-Seidel decomposition theorem in Li et al. (2016) to solve the subproblems arising in each iteration of the algorithm.

We can view (3.7) as a linearly constrained nonsmooth convex programming problem with three blocks of variables grouped as (w, β), r, (u, ξ). The template for our inexact sGS based ADMM is described next. Note that the subproblems need not be solved exactly as long as they satisfy some prescribed accuracy.

Algorithm 1. An inexact sGS-ADMM for solving (3.7)**.**

Let $\{\varepsilon_k\}$ be a summable sequence of nonnegative nonincreasing numbers. Given an initial iterate $(r^0, w^0, \beta^0, \xi^0, u^0)$ in the feasible region of (3.7), and (α^0, ρ^0) in the dual feasible region of (3.7), choose a $d \times d$ symmetric positive semidefinite matrix $\mathcal{T}$, and perform the following steps in each iteration.

Step 1a. Compute

$$(\bar{w}^{k+1}, \bar{\beta}^{k+1}) \approx \mathrm{argmin}_{w,\beta} \left\{ L_\sigma(r^k, w, \beta, \xi^k, u^k; \alpha^k, \rho^k) + \frac{\sigma}{2}\|w - w^k\|^2_{\mathcal{T}} \right\}.$$

In particular, $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ is an approximate solution to the following

$(d+1)\times(d+1)$ linear system of equations:

$$\begin{aligned}&\underbrace{\begin{bmatrix} ZZ^T+D^2+\mathcal{T} & Zy \\ (Zy)^T & y^Ty \end{bmatrix}}_{A}\begin{bmatrix} w \\ \beta \end{bmatrix} \\ &= \bar{h}^k := \begin{bmatrix} -Z(\xi^k - r^k - \sigma^{-1}\alpha^k) + D^2u^k + D(\sigma^{-1}\rho^k) + \mathcal{T}w^k \\ -y^T(\xi^k - r^k - \sigma^{-1}\alpha^k) \end{bmatrix}.\end{aligned} \tag{3.8}$$

We require the residual of the approximate solution $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ to satisfy

$$\|\bar{h}^k - A[\bar{w}^{k+1}; \bar{\beta}^{k+1}]\| \le \varepsilon_k. \tag{3.9}$$

Step 1b. Compute $r^{k+1} \approx \mathrm{argmin}_{r\in\mathbb{R}^n}\ L_\sigma(r, \bar{w}^{k+1}, \bar{\beta}^{k+1}, \xi^k, u^k; \alpha^k, \rho^k)$. Specifically, by observing that the objective function in this subproblem is actually separable in r_i for $i = 1, \ldots, n$, we can compute r_i^{k+1} as follows:

$$\begin{aligned} r_i^{k+1} &\approx \arg\min_{r_i} \left\{\theta_q(r_i) + \tfrac{\sigma}{2}\|r_i - c_i^k\|^2\right\} \\ &= \arg\min_{r_i>0} \left\{\tfrac{1}{r_i^q} + \tfrac{\sigma}{2}\|r_i - c_i^k\|^2\right\} \quad \forall\ i = 1, \ldots, n, \end{aligned} \tag{3.10}$$

where $c^k = Z^T\bar{w}^{k+1} + y\bar{\beta}^{k+1} + \xi^k - \sigma^{-1}\alpha^k$. The details on how the above one-dimensional problems are solved will be given later. The solution r_i^{k+1} is deemed to be sufficiently accurate if

$$\Big| -\frac{q}{(r_i^{k+1})^{q+1}} + \sigma(r_i^{k+1} - c_i^k)\Big| \le \varepsilon_k/\sqrt{n} \quad \forall\ i = 1, \ldots, n.$$

Step 1c. Compute

$$(w^{k+1}, \beta^{k+1}) \approx \mathrm{argmin}_{w,\beta}\ \Big\{L_\sigma(r^{k+1}, w, \beta, \xi^k, u^k; \alpha^k, \rho^k) + \frac{\sigma}{2}\|w - w^k\|_{\mathcal{T}}^2\Big\},$$

which amounts to solving the linear system of equations (3.8) but with r^k in the right-hand side vector $\bar{h}^k$ replaced by r^{k+1}. Let h^k be the new right-hand side vector. We require the approximate solution to satisfy the accuracy condition that

$$\|h^k - A[w^{k+1}; \beta^{k+1}]\| \le 5\varepsilon_k.$$

Observe that the accuracy requirement here is more relaxed than that stated in (3.9) of Step 1a. The reason for doing so is that one may hope to use the solution $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ computed in Step 1a as an approximate solution for the current subproblem. If $(\bar{w}^{k+1}, \bar{\beta}^{k+1})$ indeed satisfies the above accuracy condition, then one can simply set $(w^{k+1}, \beta^{k+1}) = (\bar{w}^{k+1}, \bar{\beta}^{k+1})$ and the cost of solving this new subproblem can be saved.

Step 2. Compute $(u^{k+1}, \xi^{k+1}) = \operatorname{argmin}_{u,\xi} \; L_\sigma(r^{k+1}, w^{k+1}, \beta^{k+1}, \xi, u; \alpha^k, \rho^k)$. By observing that the objective function is actually separable in u and ξ, we can compute u^{k+1} and ξ^{k+1} separately as follows:

$$
\begin{aligned}
u^{k+1} &= \arg\min \left\{ \delta_B(u) + \frac{\sigma}{2}\|D(u - g^k)\|^2 \right\} = \begin{cases} g^k & \text{if } \|g^k\| \le 1, \\ g^k/\|g^k\| & \text{otherwise;} \end{cases} \\
\xi^{k+1} &= \Pi_{\mathbb{R}^n_+}\Big(r^{k+1} - Z^T w^{k+1} - y\beta^{k+1} + \sigma^{-1}\alpha^k - \sigma^{-1} Ce \Big),
\end{aligned}
$$

where $g^k = w^{k+1} - \sigma^{-1} D^{-1} \rho^k$, and $\Pi_{\mathbb{R}^n_+}(\cdot)$ denotes the projection onto $\mathbb{R}^n_+$.

Step 3. Compute

$$
\begin{aligned}
\alpha^{k+1} &= \alpha^k - \tau\sigma(Z^T w^{k+1} + y\beta^{k+1} + \xi^{k+1} - r^{k+1}), \\
\rho^{k+1} &= \rho^k - \tau\sigma D(w^{k+1} - u^{k+1}),
\end{aligned}
$$

where $\tau \in (0, (1+\sqrt{5})/2)$ is the steplength which is typically chosen to be 1.618.

In our implementation of Algorithm 1, we choose the summable sequence $\{\varepsilon_k\}_{k\ge 0}$ to be $\varepsilon_k = c/(k+1)^{1.5}$ where c is a constant that is inversely proportional to $\|Z\|_F$. Next we discuss the computational cost of Algorithm 1. As we shall see later, the most computationally intensive steps in each iteration of the above algorithm are in solving the linear systems of equations of the form (3.8) in Step 1a and 1c. The detailed analysis of their computational costs will be presented in subsection 3.2.3. All the other steps can be done in at most $O(n)$ or $O(d)$ arithmetic operations, together with the computation of $Z^T w^{k+1}$, which costs $2dn$ operations if we do not take advantage of any possible sparsity

in Z.

3.2.1 Convergence results

We have the following convergence theorem for the inexact sGS-ADMM, established by Chen, Sun and Toh in Chen et al. (2017, Theorem 1). This theorem guarantees the convergence of our algorithm to optimality, as a merit over the possibly non-convergent directly extended semi-proximal ADMM.

Theorem 3.1. Suppose that the system (3.6) has at least one solution. Let $\{(r^k, w^k, \beta^k, \xi^k, u^k; \alpha^k, \rho^k)\}$ be the sequence generated by the inexact sGS-ADMM in Algorithm 1. Then the sequence $\{(r^k, w^k, \beta^k, \xi^k, u^k)\}$ converges to an optimal solution of problem (3.7) and the sequence $\{(\alpha^k, \rho^k)\}$ converges to an optimal solution to the dual of problem (3.7).

Proof. In order to apply the convergence result in Chen et al. (2017), we need to express (3.7) as follows:

$$\min\Big\{p(r) + f(r, w, \beta) + q(\xi, u) + g(\xi, u) \mid \ A_1^* r + A_2^*[w; \beta] + B^*[\xi; u] = 0\Big\}, \quad (3.11)$$

where

$$p(r) = \textstyle\sum_{i=1}^n \theta_q(r_i), \ \ f(r, w, \beta) \equiv 0, \ \ q(\xi, u) = \delta_B(u) + C\langle e, \xi\rangle + \delta_{\mathbb{R}^n_+}(\xi), \ \ g(\xi, u) \equiv 0,$$

$$A_1^* = \begin{pmatrix} -I \\ 0 \end{pmatrix}, \ A_2^* = \begin{pmatrix} Z^T & y \\ D & 0 \end{pmatrix}, \ B^* = \begin{pmatrix} I & 0 \\ 0 & -D \end{pmatrix}.$$

Next we need to consider the following matrices:

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \left(A_1^*, \ A_2^*\right) + \begin{pmatrix} 0 & 0 & 0 \\ 0 & \mathcal{T} & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} I & [-Z^T, -y] \\ [-Z^T, -y]^T & M \end{pmatrix},$$

$$BB^* = \begin{pmatrix} I & 0 \\ 0 & D^2 \end{pmatrix},$$

where

$$M = \begin{pmatrix} ZZ^T + D^2 + \mathcal{T} & Zy \\ (Zy)^T & y^T y \end{pmatrix} \succ 0.$$

One can show that M is positive definite by using the Schur complement lemma. With the conditions that $M \succ 0$ and $BB^* \succ 0$, the conditions in Proposition 4.2 of Chen et al. (2017) are satisfied, and hence the convergence of Algorithm 1 follows by using Theorem 1 in Chen et al. (2017). □

We note here that the convergence analysis in Chen et al. (2017) is highly nontrivial. But it is motivated by the proof for the simpler case of an exact semi-proximal ADMM that is available in Appendix B of the paper by Fazel et al. (2013). In that paper, one can see that the convergence proof is based on the descent property of a certain function, while the augmented Lagrangian function itself does not have such a descent property.

3.2.2 Numerical computation of the subproblem (3.10) in Step 1b

In the presentation of Algorithm 1, we have described how the subproblem in each step can be solved except for the subproblem (3.10) in Step 1b. Now we discuss how it can be solved. Observe that for each i, we need to solve a one-dimensional problem of the form:

$$\min \Big\{ \varphi(s) := \frac{1}{s^q} + \frac{\sigma}{2}(s-a)^2 \mid s > 0 \Big\}, \tag{3.12}$$

where a is given. It is easy to see that $\varphi(\cdot)$ is a convex function and it has a unique minimizer in the domain $(0, \infty)$. The optimality condition for (3.12) is given by

$$s - a = \frac{q\sigma^{-1}}{s^{q+1}},$$

where the unique minimizer s^* is determined by the intersection of the line $s \mapsto s - a$ and the curve $s \mapsto \frac{q\sigma^{-1}}{s^{q+1}}$ for $s > 0$. We propose to use Newton's method to find the minimizer, and the template is given as follows. Given an

initial iterate s^0, perform the following iterations:

$$s_{k+1} = s_k - \varphi'(s_k)/\varphi''(s_k) = s_k \left(\frac{q(q+2)\sigma^{-1} + a s_k^{q+1}}{q(q+1)\sigma^{-1} + s_k^{q+2}} \right), \quad k = 0, 1, \ldots$$

Since $\varphi''(s^*) > 0$, Newton's method would have a local quadratic convergence rate, and we would expect it to converge in a small number of iterations, say less than 20, if a good initial point s^0 is given. In solving the subproblem (3.10) in Step 1b, we always use the previous solution r_i^k as the initial point to warm-start Newton's method. If a good initial point is not available, one can use the bisection technique to find one. In our tests, this technique was however never used.

Observe that the computational cost for solving the subproblem (3.10) in Step 1b is $O(n)$ if Newton's method converges within a fixed number of iterations (say 20) for all $i = 1, \ldots, n$. Indeed, in our experiments, the average number of Newton iterations required to solve (3.12) for each of the instances is less than 10.

3.2.3 Efficient techniques to solve the linear system (3.8)

Observe that in each iteration of Algorithm 1, we need to solve a $(d+1) \times (d+1)$ linear system of equations (3.8) with the same coefficient matrix A. For large scale problems where n and/or d are large, this step would constitute the most expensive part of the algorithm. In order to solve such a linear system efficiently, we design different techniques to solve it, depending on the dimensions n and d. We consider the following cases.

(1) The case where $d \ll n$ and d is moderate

This is the most straightforward case where we set $\mathcal{T} = 0$, and we solve (3.8) by computing the Cholesky factorization of the coefficient matrix A. The cost of computing A is $2nd^2$ arithmetic operations. Assuming that A is stored, then we can compute its Cholesky factorization at the cost of $O(d^3)$ operations, which needs only to be performed once at the very beginning of Algorithm 1. After

that, whenever we need to solve the linear system (3.8), we compute the right-hand-side vector at the cost of $2nd$ operations and solve two $(d+1)\times(d+1)$ triangular systems of linear equations at the cost of $2d^2$ operations.

(2) The case where $n \ll d$ and n is moderate

In this case, we also set $\mathcal{T}=0$. But solving the large $(d+1)\times(d+1)$ system of linear equations (3.8) requires more thought. In order to avoid inverting the high dimensional matrix A directly, we make use of the Sherman-Morrison-Woodbury formula to get A^{-1} by inverting a much smaller $(n+1)\times(n+1)$ matrix as shown in the following proposition.

Proposition 3. The coefficient matrix A can be rewritten as follows:

$$A = \widehat{D} + UEU^T, \quad U = \begin{bmatrix} Z & 0 \\ y^T & \|y\| \end{bmatrix}, \quad E = \mathrm{diag}(I_n, -1), \tag{3.13}$$

where $\widehat{D} = \mathrm{diag}(D, \|y\|^2)$. It holds that

$$A^{-1} = \widehat{D}^{-1} - \widehat{D}^{-1}UH^{-1}U^T\widehat{D}^{-1}, \tag{3.14}$$

where

$$H = E^{-1} + U^T\widehat{D}^{-1}U = \begin{bmatrix} I_n + Z^TD^{-1}Z + yy^T/\|y\|^2 & y/\|y\| \\ y^T/\|y\| & 0 \end{bmatrix}. \tag{3.15}$$

Proof. It is easy to verify that (3.13) holds and we omit the details. To get (3.14), we only need to apply the Sherman-Morrison-Woodbury formula in Golub and Loan (1996, p.50) to (3.13) and perform some simplifications. □

Note that in making use of (3.14) to compute $A^{-1}\bar{h}^k$, we need to find H^{-1}. A rather cost effective way to do so is to express H as follows and use the Sherman-Morrison-Woodbury formula to find its inverse:

$$H = J + \bar{y}\bar{y}^T, \quad J = diag(I_n + Z^TD^{-1}Z, -1), \quad \bar{y} = [y/\|y\|;\ 1].$$

With the above expression for H, we have that

$$H^{-1} = J^{-1} - \frac{1}{1 + \bar{y}^T J^{-1} \bar{y}} (J^{-1}\bar{y})(J^{-1}\bar{y})^T.$$

Thus to solve (3.8), we first compute the $n \times n$ matrix $I_n + Z^T D^{-1} Z$ in (3.15) at the cost of $2dn^2$ operations. Then we compute its Cholesky factorization at the cost of $O(n^3)$ operations. (Observe that even though we are solving a $(d+1) \times (d+1)$ linear system of equations for which $d \gg n$, we only need to compute the Cholesky factorization of a much smaller $n \times n$ matrix.) Also, we need to compute $J^{-1}\bar{y}$ at the cost of $O(n^2)$ operations by using the previously computed Cholesky factorization. These computations only need to be performed once at the beginning of Algorithm 1. After that, whenever we need to solve a linear system of the form (3.8), we can compute $\bar{h}^k$ at the cost of $2nd$ operations, and then make use of (3.14) to get $A^{-1}\bar{h}^k$ by solving two $n \times n$ triangular systems of linear equations at the cost of $2n^2$ operations, and performing two matrix-vector multiplications involving Z and Z^T at a total cost of $4nd$ operations. To summarize, given the Cholesky factorization of the first diagonal block of H, the cost of solving (3.8) via (3.14) is $6nd + 2n^2$ operations.

(3) The case where d and n are both large

The purpose of introducing the proximal term $\frac{1}{2}\|w - w^k\|_{\mathcal{T}}^2$ in Steps 1a and 1c is to make the computation of the solutions of the subproblems easier. However, one should note that adding the proximal term typically will make the algorithm converge more slowly, and the deterioration will become worse for larger $\|\mathcal{T}\|$. Thus in practice, one would need to strike a balance between choosing a symmetric positive semidefinite matrix $\mathcal{T}$ to make the computation easier while not slowing down the algorithm by too much.

In our implementation, we first attempt to solve the subproblem in Step 1a (similarly for 1c) without adding a proximal term by setting $\mathcal{T} = 0$. In particular, we solve the linear system (3.8) by using a preconditioned symmetric quasi-minimal residual (PSQMR) iterative solver (Freund 1997) when both n and d are large. Basically, it is a variant of the Krylov subspace method similar to the

idea in GMRES (Saad 2003). For more details on the PSQMR algorithm, the reader is referred to the appendix. In each step of the PSQMR solver, the main cost is in performing the matrix-vector multiplication with the coefficient matrix A, which costs $4nd$ arithmetic operations. As the number of steps taken by an iterative solver to solve (3.8) to the required accuracy (3.9) is dependent on the conditioning of A, in the event that the solver requires more than 50 steps to solve (3.8), we would switch to adding a suitable non-zero proximal term $\mathcal{T}$ to make the subproblem in Step 1a easier to solve.

The most common and natural choice of $\mathcal{T}$ to make the subproblem in Step 1a easy to solve is to set $\mathcal{T} = \lambda_{\max} I - ZZ^T$, where $\lambda_{\max}$ denotes the largest eigenvalue of ZZ^T. In this case the corresponding linear system (3.8) is very easy to solve. More precisely, for the linear system in (3.8), we can first compute $\bar{\beta}^{k+1}$ via the Schur complement equation in a single variable followed by computing $\bar{w}^k$ as follows:

$$\begin{aligned} &\big(y^T y - (Zy)^T(\lambda_{\max} I + D)^{-1}(Zy)\big)\beta = \bar{h}^k_{d+1} - (Zy)^T(\lambda_{\max} I + D)^{-1}\bar{h}^k_{1:d}, \\ &\bar{w}^{k+1} = (\lambda_{\max} I + D)^{-1}(\bar{h}^k_{1:d} - (Zy)\bar{\beta}^{k+1}), \end{aligned} \tag{3.16}$$

where $\bar{h}^k_{1:d}$ denotes the vector extracted from the first d components of $\bar{h}^k$. In our implementation, we pick a $\mathcal{T}$ which is less conservative than the above natural choice as follows. Suppose we have computed the first ℓ largest eigenvalues of ZZ^T such that $\lambda_1 \geq \ldots \geq \lambda_{\ell-1} > \lambda_\ell$, and their corresponding orthonormal set of eigenvectors, $v_1, \ldots, v_\ell$. We pick $\mathcal{T}$ to be

$$\mathcal{T} = \lambda_\ell I + \textstyle\sum_{i=1}^{\ell-1}(\lambda_i - \lambda_\ell) v_i v_i^T - ZZ^T, \tag{3.17}$$

which can be proved to be positive semidefinite by using the spectral decomposition of ZZ^T. In practice, one would typically pick ℓ to be a small integer, say 10, and compute the first ℓ largest eigenvalues and their corresponding eigenvectors via variants of the Lanczos method. The most expensive step in each iteration of the Lanczos method is a matrix-vector multiplication, which requires $O(d^2)$ operations. In general, the cost of computing the first few largest eigenvalues of ZZ^T is much cheaper than that of computing the full eigenvalue decomposition.

In MATLAB, such a computation can be done by using the routine `eigs`. To solve (3.8), we need the inverse of $ZZ^T + D + \mathcal{T}$. Fortunately, when $D = \mu I_d$, it can easily be inverted with

$$(ZZ^T + D + \mathcal{T})^{-1} = (\mu + \lambda_\ell)^{-1} I_d + \textstyle\sum_{i=1}^{\ell-1} \big((\mu + \lambda_i)^{-1} - (\mu + \lambda_\ell)^{-1}\big) v_i v_i^T.$$

One can then compute $\bar{\beta}^k$ and $\bar{w}^k$ as in (3.16) with $(\lambda_{\max} I + D)^{-1}$ replaced by the above inverse.

3.3 Numerical experiments

In this section, we test the performance of our inexact sGS-ADMM method on several publicly available data sets. The numerical results presented in the subsequent subsections are obtained from a computer with processor specifications: Intel(R) Xeon(R) CPU E5-2670 @ 2.5GHz (2 processors) and 64GB of RAM, running on a 64-bit Windows Operating System.

3.3.1 Tuning the penalty parameter

In the DWD model (3.7), we see that it is important to make a suitable choice of the penalty parameter C. In Marron et al. (2007), it has been noticed that a reasonable choice for the penalty parameter when the exponent $q = 1$ is a large constant divided by the square of a typical distance between the x_i's, where the typical distance, *dist*, is defined as the median of the pairwise Euclidean distances between classes. We found out that in a more general case, C should be inversely proportional to $dist^{q+1}$. On the other hand, we observed that a good choice of C also depends on the sample size n and the dimension of features d. In our numerical experiments, we empirically set the value of C to be $10^{q+1} \max\big\{1, \frac{10^{q-1}\log(n)\max\{1000,d\}^{\frac{1}{3}}}{dist^{q+1}}\big\}$, where $\log(\cdot)$ is the natural logarithm.

3.3.2 Scaling of data

A technique which is very important in implementing ADMM based methods in practice to achieve fast convergence is the data scaling technique. Empirically, we

have observed that it is good to scale the matrix Z in (3.7) so that the magnitude of all the blocks in the equality constraint would be roughly the same. Here we choose the scaling factor to be $Z_{\rm scale} = \sqrt{\|X\|_F}$, where $\|\cdot\|_F$ is the Frobenius norm. Hence the optimization model in (3.7) becomes:

$$\begin{array}{ll} \min & \sum_{i=1}^n \frac{1}{r_i^q} + C\langle e,\, \xi\rangle + \delta_{\widetilde{B}}(\tilde{u}) + \delta_{\mathbb{R}^n_+}(\xi) \\ \text{s.t.} & \widetilde{Z}^T\,\tilde{w} + \beta y + \xi - r = 0,\ r > 0, \\ & D(\tilde{w} - \tilde{u}) = 0,\ \ \tilde{w}, \tilde{u} \in \mathbb{R}^d,\ r, \xi \in \mathbb{R}^n, \end{array} \tag{3.18}$$

where $\widetilde{Z} = \frac{Z}{Z_{\rm scale}}$, $\tilde{w} = Z_{\rm scale} w$, $\tilde{u} = Z_{\rm scale} u$, and $\widetilde{B} = \{\tilde{w} \in \mathbb{R}^d \mid \|\tilde{w}\| \le Z_{\rm scale}\}$. Therefore, if we have computed an optimal solution $(r^*, \tilde{w}^*, \beta^*, \xi^*, \tilde{u}^*)$ of (3.18), then $(r^*, Z_{\rm scale}^{-1}\tilde{w}^*, \beta^*, \xi^*, Z_{\rm scale}^{-1}\tilde{u}^*)$ would be an optimal solution of (3.7).

3.3.3 Stopping condition for inexact sGS-ADMM

We measure the accuracy of an approximate optimal solution $(r, w, \beta, \xi, u, \alpha, \rho)$ for (3.18) based on the KKT optimality conditions (3.6) by defining the following relative residuals:

$$\eta_{C_1} = \tfrac{|y^T\alpha|}{1+C}, \qquad \eta_{C_2} = \tfrac{|\xi^T(Ce-\alpha)|}{1+C}, \qquad \eta_{C_3} = \tfrac{\|\alpha - s\|^2}{1+C} \text{ with } s_i = \tfrac{q}{r_i^{q+1}},$$

$$\eta_{P_1} = \tfrac{\|\widetilde{Z}^T\tilde{w} + \beta y + \xi - r\|}{1+C}, \ \eta_{P_2} = \tfrac{\|D(\tilde{w}-\tilde{u})\|}{1+C}, \qquad \eta_{P_3} = \tfrac{\max\{\|\tilde{w}\| - Z_{\rm scale}, 0\}}{1+C},$$

$$\eta_{D_1} = \tfrac{\|\min\{0,\alpha\}\|}{1+C}, \qquad \eta_{D_2} = \tfrac{\|\max\{0, \alpha - Ce\}\|}{1+C},$$

where $Z_{\rm scale}$ is a scaling factor which has been discussed in the last subsection. Additionally, we calculate the relative duality gap by:

$$\eta_{gap} := \frac{|\text{obj}_{\text{primal}} - \text{obj}_{\text{dual}}|}{1 + |\text{obj}_{\text{primal}}| + |\text{obj}_{\text{dual}}|},$$

where $\text{obj}_{\text{primal}} = \sum_{i=1}^n \frac{1}{r_i^q} + C\langle e,\, \xi\rangle$, $\text{obj}_{\text{dual}} = \kappa \sum_{i=1}^n \alpha_i^{\frac{q}{q+1}} - Z_{\rm scale}\|\widetilde{Z}\alpha\|$, with $\kappa = \frac{q+1}{q} q^{\frac{1}{q+1}}$. We should emphasize that although for machine learning problems, a high accuracy solution is usually not required, it is important however to use the KKT optimality conditions as the stopping criterion to find a moderately accurate solution in order to design a robust solver.

We terminate the solver when $\max\{\eta_P, \eta_D\} < 10^{-5}$, $\min\{\eta_C, \eta_{gap}\} < \sqrt{10^{-5}}$, and $\max\{\eta_C, \eta_{gap}\} < 0.05$. Here, $\eta_C = \max\{\eta_{C_1}, \eta_{C_2}, \eta_{C_3}\}$, $\eta_P = \max\{\eta_{P_1}, \eta_{P_2}, \eta_{P_3}\}$, and $\eta_D = \max\{\eta_{D_1}, \eta_{D_2}\}$. Furthermore, the maximum number of iterations is set to be 2000.

3.3.4 Adjustment of Lagrangian parameter σ

Based upon some preliminary experiments, we set our initial Lagrangian parameter σ to be $\sigma_0 = \min\{10C, n\}^q$, where q is the exponent in (3.7), and adapt the following strategy to update σ to improve the convergence speed of the algorithm in practice:

Step 1. Set $\chi = \frac{\eta_P}{\eta_D}$, where η_P and η_D are defined in subsection 3.3.3;

Step 2. If $\chi > \theta$, set $\sigma_{k+1} = \zeta\sigma_k$; elseif $\frac{1}{\chi} > \theta$, set $\sigma_{k+1} = \frac{1}{\zeta}\sigma_k$.

Here we empirically set θ to be 5 and ζ to be 1.1. Nevertheless, if we have either $\eta_P \ll \eta_D$ or $\eta_D \ll \eta_P$, then we would increase ζ accordingly, say 2.2 if $\max\{\chi, \frac{1}{\chi}\} > 500$ or 1.65 if $\max\{\chi, \frac{1}{\chi}\} > 50$.

3.3.5 Performance of the sGS-ADMM on UCI data sets

In this subsection, we test our algorithm on instances from the UCI data repository (Lichman 2013). The datasets we have chosen here are all classification problems with two classes. However, the size for each class may not be balanced. To tackle the case of uneven class proportions, we use the weighted DWD model discussed in Qiao et al. (2010). Specifically, we consider the model (3.4) using $e = \mathbf{1}$ and the term $\sum_{i=1}^n 1/r_i^q$ is replaced by $\sum_{i=1}^n \tau_i^q/r_i^q$, with the weights τ_i given as follows:

$$\tau_i = \begin{cases} \frac{\tau_-}{\max\{\tau_+,\tau_-\}} & \text{if } y_i = +1, \\ \frac{\tau_+}{\max\{\tau_+,\tau_-\}} & \text{if } y_i = -1, \end{cases}$$

where $\tau_\pm = \left(|n_\pm|K^{-1}\right)^{\frac{1}{1+q}}$. Here $n_\pm$ is the number of data points with class label ± 1 respectively and $K := n/\log(n)$ is a normalizing factor.

Data	n	d	C	Iter	Time (s)	psqmr|double	Train-error (%)
a8a	22696	123	6.27e+02	201	2.28	0|201	15.10
a9a	32561	123	6.49e+02	201	2.31	0|201	14.93
covtype	581012	54	3.13e+03	643	104.34	0|191	23.74
gisette	6000	4972	1.15e+04	101	39.69	0| 49	0.17
gisette-scale	6000	4956	1.00e+02	201	59.73	0|201	0.00
ijcnn1	35000	22	4.23e+03	401	3.16	0|401	7.77
mushrooms	8124	112	3.75e+02	81	1.09	0| 81	0.00
real-sim	72309	20958	1.55e+04	210	47.69	875|210	1.45
w7a	24692	300	5.95e+02	701	4.84	0|701	1.17
w8a	49749	300	6.36e+02	906	9.43	0|906	1.20
rcv1	20242	44505	9.18e+03	81	9.18	234| 49	0.63
leu	38	7129	1.00e+02	489	2.84	0|489	0.00
prostate	102	6033	1.00e+02	81	3.19	0| 81	0.00
farm-ads	4143	54877	3.50e+03	81	6.92	792| 81	0.14
dorothea	800	88120	1.00e+02	51	4.44	0| 51	0.00
url-svm	256000	685896	1.18e+06	121	294.55	364|121	0.01

Table 3.1: The performance of our inexact sGS-ADMM method on the UCI data sets.

Table 3.1 presents the number of iterations and runtime required, as well as training error produced when we perform our inexact sGS-ADMM algorithm to solve 16 data sets. Here, the running time is the total time spent in reading the training data and in solving the DWD model. The timing for getting the best penalty parameter C is excluded. The results are generated using the exponent $q = 1$. In the table, "psqmr" is the iteration count for the preconditioned symmetric quasi-minimal residual method for solving the linear system (3.8). A '0' for "psqmr" means that we are using a direct solver as mentioned in subsection 3.2.3. Under the column "double" in Table 3.1, we also record the number of iterations for which the extra Step 1c is executed to ensure the convergence of Algorithm 1.

Denote the index set $S = \{i \mid y_i[\mathrm{sgn}(\beta + x_i^T w)] \leq 0, i = 1, \ldots, n\}$ for which the data instances are categorized wrongly, where $sgn(x)$ is the sign function. The training and testing errors are both defined by $\frac{|S|}{n} \times 100\%$, where $|S|$ is the cardinality of the set S.

Our algorithm is capable of solving all the data sets, even when the size of the data matrix is huge. In addition, for data with unbalanced class size, such as `w7a` and `w8a`, our algorithm is able to produce a classifier with small training error.

3.3.6 Comparison with other solvers

In this subsection, we compare our inexact sGS-ADMM method for solving (3.4) via (3.7) with the primal-dual interior-point method implemented in Toh et al. (1999) and used in Marron et al. (2007). We also compare our method with the directly extended (semi-proximal) ADMM (using the aggressive step-length 1.618) even though the latter's convergence is not guaranteed. Note that the directly extended ADMM we have implemented here follows exactly the same design used for sGS-ADMM, except that we switch off the additional Step 1c in the Algorithm 1. We should emphasize that our directly extended ADMM is not a simple adaption of the classical ADMM, but instead incorporates all the sophisticated techniques we have developed for sGS-ADMM.

We will report our computational results for two different values of the exponent, $q = 1$ and $q = 2$, in Tables 3.2 and 3.3, respectively.

exponent $q = 1$				sGS-ADMM			directADMM			IPM		
Data	n	d	C	Time(s)	Iter	Error(%)	Time(s)	Iter	Error(%)	Time(s)	Iter	Error(%)
a8a	22696	123	6.27e+02	2.28	201	15.10	2.01	219	15.10	1321.20	47	15.09
a9a	32561	123	6.49e+02	2.31	201	14.93	2.12	258	14.93	2992.60	43	14.93
covtype	581012	54	3.13e+03	104.34	643	23.74	100.26	700	23.74	-	-	-
gisette	6000	4972	1.15e+04	39.69	101	0.17	33.14	70	0.25	2403.40	62	20.50*
gisette-scale	6000	4956	1.00e+02	59.73	201	0.00	152.05	2000	0.00	49800.58	84	17.80*
ijcnn1	35000	22	4.23e+03	3.16	401	7.77	2.95	501	7.77	1679.34	34	7.77
mushrooms	8124	112	3.75e+02	1.09	81	0.00	1.42	320	0.00	136.67	50	0.00
real-sim	72309	20958	1.55e+04	47.69	210	1.45	89.55	702	1.42	-	-	-

exponent $q = 1$				sGS-ADMM			directADMM			IPM		
Data	n	d	C	Time(s)	Iter	Error(%)	Time(s)	Iter	Error(%)	Time(s)	Iter	Error(%)
w7a	24692	300	5.95e+02	4.84	701	1.17	8.52	2000	1.16	4474.13	53	1.17
w8a	49749	300	6.36e+02	9.43	906	1.20	13.58	2000	1.16	-	-	-
rcv1	20242	44505	9.18e+03	9.18	81	0.63	16.07	245	0.63	9366.41	43	0.17
leu	38	7129	1.00e+02	2.84	489	0.00	5.35	2000	0.00	1.33	11	0.00
prostate	102	6033	1.00e+02	3.19	81	0.00	4.13	2000	0.00	7.73	19	0.00
farm-ads	4143	54877	3.50e+03	6.92	81	0.14	4.20	61	0.14	642.79	54	0.24
dorothea	800	88120	1.00e+02	4.44	51	0.00	37.31	2000	0.00	15.53	23	0.00
url-svm	256000	685896	1.18e+06	294.55	121	0.01	264.52	195	0.00	-	-	-

Table 3.2: Comparison between the performance of our inexact sGS-ADMM, directly extended ADMM "directADMM", and the interior point method "IPM" on the UCI data sets. A '*' next to the error in the table means that the problem set cannot be solved properly by the respective solver; '-' means the algorithm cannot solve the dataset due to insufficient computer memory.

Table 3.2 reports the runtime, number of iterations required as well as the training error of 3 different solvers for solving the UCI data sets. We can observe that the interior point method is almost always the slowest to achieve optimality compared to the other two solvers, despite requiring the least number of iterations, especially when the sample size n is large. The inefficiency of the interior-point method is caused by its need to solve an $n \times n$ linear system of equations in each iteration, which could be very expensive if n is large. In addition, it cannot solve the DWD problem where n is huge due to the excessive computer memory needed to store the large $n \times n$ matrix.

On the other hand, our inexact sGS-ADMM method outperforms the directly extended (semi-proximal) ADMM for 9 out of 16 cases in terms of runtime. For the other cases, we are only slower by a relatively small margin. Furthermore, when our algorithm outperforms the directly extended ADMM, it often shortens the runtime by a large margin. In terms of number of iterations, for 14 out of 16 cases, the directly extended ADMM requires at least the same number of iterations as our inexact sGS-ADMM method. We can say that our algorithm is remarkably efficient and it further possesses a convergence guarantee. In

contrast, the directly extended ADMM is not guaranteed to converge although it is also very efficient when it does converge. We can observe that the directly extended ADMM sometimes would take many more iterations to solve a problem compared to our inexact sGS-ADMM, especially for the instances in Table 3.3, possibly because the lack of a convergence guarantee makes it difficult for the method to find a sufficiently accurate approximate optimal solution.

To summarize, our inexact sGS-ADMM method is an efficient yet convergent algorithm for solving the primal form of the DWD model. It is also able to solve large scale problems which cannot be handled by the interior point method.

exponent $q = 2$				sGS-ADMM			directADMM			IPM		
Data	n	d	C	Time(s)	Iter	Error(%)	Time(s)	Iter	Error(%)	Time(s)	Iter	Error(%)
a8a	22696	123	1.57e+04	2.77	248	15.10	2.97	533	15.11	7364.34	45	15.17
a9a	32561	123	1.62e+04	2.92	279	14.94	5.78	1197	14.94	16402.05	48	14.95
covtype	581012	54	1.52e+05	81.63	368	23.74	275.29	2000	23.74	-	-	-
gisette	6000	4972	1.01e+06	39.72	101	0.03	28.42	81	0.03	2193.74	55	0.00
gisette-scale	6000	4956	1.00e+03	88.89	439	0.00	147.80	2000	1.20	31827.33	53	0.00
ijcnn1	35000	22	2.69e+05	2.76	233	7.94	4.80	1056	7.87	1959.68	38	7.98
mushrooms	8124	112	7.66e+03	1.59	301	0.00	3.63	2000	0.17	594.33	52	0.00
real-sim	72309	20958	1.10e+06	43.20	180	1.51	23.56	181	1.51	-	-	-
w7a	24692	300	1.44e+04	3.96	473	1.15	7.83	2000	2.69	5448.78	48	1.18
w8a	49749	300	1.54e+04	7.07	543	1.13	13.57	2000	2.68	-	-	-
rcv1	20242	44505	4.69e+05	9.47	81	1.16	7.54	101	1.16	8562.76	47	0.24*
leu	38	7129	1.00e+03	1.72	204	0.00	4.52	2000	0.00	2.82	23	0.00
prostate	102	6033	1.00e+03	3.32	111	0.00	3.87	2000	0.00	8.33	21	0.00
farm-ads	4143	54877	5.21e+04	37.79	401	0.19	8.31	146	0.19	343.13	43	0.27
dorothea	800	88120	1.00e+03	4.39	51	0.00	31.49	2000	0.00	16.84	25	0.00
url-svm	256000	685896	5.49e+07	452.81	205	0.02	587.46	490	0.02	-	-	-

Table 3.3: Same as Table 3.2 but for $q = 2$.

Table 3.3 reports the runtime, number of iterations required as well as the training error of 3 different solvers for solving the UCI data sets for the case when $q = 2$. Again, we can see that the interior point method is almost always

the slowest to converge to optimality.

Our sGS-ADMM algorithm outperforms the directly extended ADMM algorithm in 12 out of 16 data sets in terms of runtime. In terms of the number of iterations, it has the best performance among almost all the data sets. On the other hand, for 8 data sets, the number of iterations required by the directly extended ADMM hits the maximum iterations allowed, probably implying non-convergence of the method. For the interior point method, it takes an even longer time to solve the problems compared to the case when $q = 1$. This is due to an increase in the number of constraints generated in the second-order cone programming formulation of the DWD model with $q = 2$.

The numerical result we obtained in this case is consistent with the one we obtained for the case $q = 1$. This further shows the merit of our algorithm in a more general setting. We could also expect the similar result when the exponent is 4 or 8.

3.3.7 Comparison with LIBSVM and LIBLINEAR

In this subsection, we will compare the performance of our DWD model to the state-of-the-art model support vector machine (SVM). We apply our sGS-ADMM algorithm as the DWD model and use LIBSVM in Chang and Lin (2011) as well as LIBLINEAR in Fan et al. (2008) to implement the SVM model. LIBSVM is a general solver for solving SVM models with different kernels; while LIBLINEAR is a solver highly specialized in solving SVM with linear kernels. LIBLINEAR is a fast linear classifier; in particular, we would apply it to the dual of L2-regularized L1-loss support vector classification problem. We would like to emphasize that the solution given by LIBSVM using linear kernel and that given by LIBLINEAR is not exactly the same. This may be due to the reason that LIBLINEAR has internally preprocessed the data and assumes that there is no bias term in the model.

The parameters used in LIBSVM are chosen to be the same as in Ito et al. (2015), whereas for LIBLINEAR, we make use of the default parameter $C = 1$ for all the datasets.

Table 3.4 shows the runtime and number of iterations needed for solving the binary classification problem via the DWD and SVM models respectively on the UCI datasets. It also gives the training and testing classification error produced by the three algorithms. Note that the training time excludes the time for computing the best penalty parameter C. The stopping tolerance for all algorithms is set to be 10^{-5}. For LIBLINEAR, we observed that the default maximum number of iteration is breached for many datasets. Thus we increase the maximum number of iteration from the default 1000 to 20000.

In terms of runtime, LIBLINEAR is almost always the fastest to solve the problem, except for the two largest datasets (rcv1,url-svm) for which our algorithm is about 2-3 times faster. Note that the maximum iteration is reached for these two datasets. On the other hand, LIBSVM is almost always the slowest solver. It may only be faster than sGS-ADMM for small datasests (3 cases). Our algorithm can be 50-100 times faster than LIBSVM when solving large data instances. Furthermore, LIBSVM may have the risk of not being able to handle extremely large-scaled datasets. For example, it cannot solve the biggest dataset (url-svm) within 24 hours.

In terms of training and testing error, we may observe from the table that the DWD and SVM models produced comparable training classification errors, although there are some discrepancies due to the differences in the models and penalty parameters used. On the other hand, the testing errors vary across different solvers. For most datasets, the DWD model (solved by sGS-ADMM) produced smaller (sometimes much smaller) testing errors than the other algorithms (8 cases); whereas the SVM model (solved by LIBLINEAR) produced the worst testing errors among all algorithms (5 cases). The discrepancy between the testing errors given by LIBSVM and LIBLINEAR may be due to the different treatment of the bias term in-built into the algorithms.

It is reasonable to claim that our algorithm is more efficient than the extremely fast solver LIBLINEAR in solving large data instances even though our algorithm is designed for the more complex DWD model compared to the simpler SVM model. Moreover, our algorithm for solving the DWD model is able to produce

testing errors which are generally better than those produced by LIBLINEAR for solving the SVM model.

				DWD via sGS-ADMM				SVM via LIBLINEAR				SVM via LIBSVM			
Data	n	d	n_{test}	Time	Iter	Err_{tr}	Err_{test}	Time	Iter	Err_{tr}	Err_{test}	Time	Iter	Err_{tr}	Err_{test}
a8a	22696	123	9865	2.28	201	15.10	14.67	0.69	20000	15.32	14.87	50.45	34811	15.44	14.80
a9a	32561	123	16281	2.31	201	14.93	15.19	0.79	6475	15.01	15.02	93.91	25721	15.24	15.03
covtype	581012	54	/	104.34	643	23.74	/	23.53	20000	23.69	/	19641.19	224517	23.70	/
gisette	6000	4972	1000	39.69	101	0.17	3.00	4.34	132	0.00	10.70	85.93	14818	0.40	14.30
gisette-scale	6000	4956	1000	59.73	201	0.00	2.50	33.07	484	0.00	2.30	152.27	15738	0.40	2.20
ijcnn1	35000	22	91701	3.16	401	7.77	7.82	0.55	11891	8.70	8.35	27.04	10137	9.21	8.76
mushrooms	8124	112	/	1.09	81	0.00	/	0.19	326	0.00	/	0.64	1003	0.00	/
real-sim	72309	20958	/	47.69	210	1.45	/	7.56	1190	1.07	/	3846.07	52591	1.37	/
w7a	24692	300	25057	4.84	701	1.17	1.30	0.43	1689	1.28	10.12	14.79	62830	1.37	1.38
w8a	49749	300	14951	9.43	906	1.20	1.31	2.45	13095	1.18	9.64	63.27	124373	1.36	1.43
rcv1	20242	44505	677399	9.18	81	0.63	5.12	33.98	20000	0.15	5.17	819.32	33517	0.62	13.82
leu	38	7129	34	2.84	489	0.00	5.88	0.30	27	0.00	20.59	0.59	184	0.00	17.65
prostate	102	6033	/	3.19	81	0.00	/	2.73	353	0.00	/	2.54	1063	0.00	/
farm-ads	4143	54877	/	6.92	81	0.14	/	6.11	5364	0.14	/	10.34	15273	0.14	/
dorothea	800	88120	350	4.44	51	0.00	5.14	0.83	11	0.00	8.86	7.48	4553	0.00	7.71
url-svm	256000	685896	/	294.55	121	0.01	/	660.39	20000	0.01	/	-	-	-	-

Table 3.4: Comparison between the performance of our inexact sGS-ADMM on DWD model with LIBLINEAR and LIBSVM on SVM model. n_{test} is the size of testing sample, Err_{tr} is the percentage of training error; while Err_{test} is that of the testing error. '-' means the result cannot be obtained within 24 hours, and '/' means test sets are not available.

Chapter 4

A semi-proximal augmented Lagrangian based decomposition method for primal block angular convex composite quadratic conic programming problems

In this chapter, we will focus on designing efficient and robust (distributed) algorithms for solving large scale convex composite quadratic conic programming problems with a primal block angular structure, i.e. optimization problems with a separable convex objective function and conic constraints but the variables are coupled by linking linear constraints across different variables.

We consider the following primal block-angular optimization problem:

$$\text{(PBA-P)} \quad \min \ \sum_{i=0}^{N} f_i(x_i) := \theta_i(x_i) + \tfrac{1}{2}\langle x_i,\, \mathcal{Q}_i x_i\rangle + \langle c_i,\, x_i\rangle$$

$$\text{s.t.} \ \underbrace{\begin{bmatrix} \mathcal{A}_0 & \mathcal{A}_1 & \dots & \mathcal{A}_N \\ & \mathcal{D}_1 & & \vdots \\ & & \ddots & \vdots \\ & & & \mathcal{D}_N \end{bmatrix}}_{\mathcal{B}} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{bmatrix},$$

$$x_i \in \mathcal{K}_i,\ i = 0, 1, \dots, N,$$

where for each $i = 0, 1, \dots, N$, $\theta_i : \mathcal{X}_i \to (-\infty, \infty]$ is a proper closed convex function, $\mathcal{Q}_i : \mathcal{X}_i \to \mathcal{X}_i$ is a positive semidefinite linear operator, $\mathcal{A}_i : \mathcal{X}_i \to \mathcal{Y}_0$ and $\mathcal{D}_i : \mathcal{X}_i \to \mathcal{Y}_i$ are given linear maps, $c_i \in \mathcal{X}_i$ and $b_i \in \mathcal{Y}_i$ are given data, $\mathcal{K}_i \subset \mathcal{X}_i$ is a closed convex set that is typically a cone but not necessarily so, and $\mathcal{X}_i, \mathcal{Y}_i$ are real finite dimensional Euclidean spaces each equipped with an inner product $\langle \cdot, \cdot \rangle$ and its induced norm $\| \cdot \|$. Note that the addition of the proper closed convex functions in the objective gives us the flexibility to add nonsmooth terms such as ℓ_1 regularization terms. We should also mention that a constraint of the form $b_i - \mathcal{D}_i x_i \in \mathcal{C}_i$, where $\mathcal{C}_i$ is a closed convex set can be put in the form in (PBA-P) by introducing a slack variable s_i so that $[\mathcal{D}_i, I](x_i; s_i) = b_i$ and $(x_i; s_i) \in \mathcal{K}_i \times \mathcal{C}_i$.

Without loss of generality, we assume that the constraint matrix $\mathcal{B}$ in (PBA-P) has full row-rank. Let $n_i = \dim(\mathcal{X}_i)$ and $m_i = \dim(\mathcal{Y}_i)$. Observe that the problem (PBA-P) has $\sum_{i=0}^{N} m_i$ linear constraints and the dimension of the decision variable is $\sum_{i=0}^{N} n_i$. Thus even if m_i and/or n_i are moderate numbers, the overall dimension of the problem can easily get very large when N is large.

In the important special case of a block angular linear programming problem for which $\mathcal{Q}_i = 0$ and $\theta_i = 0$ for all $i = 0, \dots, N$, the Dantzig-Wolfe decomposition method (which may be viewed as a dual method based on the Lagrangian function $\sum_{i=0}^{N} \langle c_i,\, x_i\rangle - \langle u,\, b_0 - \sum_{i=0}^{N} \mathcal{A}_i x_i\rangle$) is a well known classical approach for solving such a problem. The Dantzig-Wolfe decomposition method has the attractive property that in each iteration, x_i can be computed individually from

a smaller linear program (LP) for $i = 1, \ldots, N$. However, it is generally acknowledged that an augmented Lagrangian approach has a number of important advantages over the usual Lagrangian dual method. For example, Ruszczyński stated in Ruszczyński (1995) that the dual approach based on the ordinary Lagrangian can suffer from the nonuniqueness of the solutions of subproblems. In addition, solving the subproblem of the augmented Lagrangian approach would be more stable. In that paper, the well-known diagonal quadratic approximation (DQA) method is introduced. The DQA method is a very successful decomposition method and it has been a popular tool in stochastic programming. Thus it would be a worthwhile effort to analyse it again to see whether further enhancements are possible.

This chapter is organized as follows. We will derive the dual of (PBA-P) in section 4.1. In section 4.2, we will present our inexact semi-proximal augmented Lagrangian methods for the primal problem (PBA-P). In section 4.3, we will propose a semi-proximal symmetric Gauss-Seidel based ADMM for the dual problem of (PBA-P). For all algorithms we introduce, we conduct numerical experiments to evaluate their performance and the numerical results are reported in section 4.4.

4.1 Derivation of the dual of (PBA-P)

For notational convenience, we define

$$\boldsymbol{\mathcal{X}} = \mathcal{X}_0 \times \mathcal{X}_1 \times \cdots \times \mathcal{X}_N, \quad \boldsymbol{\mathcal{Y}} = \mathcal{Y}_0 \times \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_N, \quad \boldsymbol{\mathcal{K}} = \mathcal{K}_0 \times \mathcal{K}_1 \times \cdots \times \mathcal{K}_N. \tag{4.1}$$

For each $x \in \boldsymbol{\mathcal{X}}$, $y \in \boldsymbol{\mathcal{Y}}$, and $c \in \boldsymbol{\mathcal{X}}$, $b \in \boldsymbol{\mathcal{Y}}$, we can express them as

$$\begin{aligned} &x = (x_0; x_1; \cdots; x_N), \quad y = (y_0; y_1; \cdots; y_N), \\ &c = (c_0; c_1; \cdots; c_N), \quad b = (b_0; b_1; \cdots; b_N). \end{aligned} \tag{4.2}$$

We also define $\mathcal{A}$, $\mathcal{Q}$ and $\boldsymbol{\theta}$ as follows:

$$\mathcal{A} = [\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_N], \ \mathcal{Q}(x) = \big(\mathcal{Q}_0(x_0); \mathcal{Q}_1(x_1); \ldots; \mathcal{Q}_N(x_N)\big), \ \boldsymbol{\theta}(x) = \textstyle\sum_{i=0}^{N} \theta_i(x_i). \tag{4.3}$$

Using the notation in (4.1)–(4.3), we can write (PBA-P) compactly in the form of a general convex composite quadratic conic programming problem:

$$\min\Big\{\boldsymbol{\theta}(x) + \tfrac{1}{2}\langle x,\, \mathcal{Q}x\rangle + \langle c,\, x\rangle \mid \mathcal{B}x - b = 0,\; x \in \boldsymbol{\mathcal{K}}\Big\}. \tag{4.4}$$

By introducing auxiliary variables $u, v \in \boldsymbol{\mathcal{X}}$, problem (4.4) can equivalently be written as the following model:

$$\begin{array}{ll} \min & \boldsymbol{\theta}(u) + \frac{1}{2}\langle x,\, \mathcal{Q}x\rangle + \langle c,\, x\rangle + \delta_{\boldsymbol{\mathcal{K}}}(v) \\ \text{s.t.} & \mathcal{B}x - b = 0,\; u - x = 0,\; v - x = 0. \end{array} \tag{4.5}$$

To derive the dual of (4.4), consider the following Lagrangian function for (4.5):

$$\begin{aligned} & \mathcal{L}(x, u, v; y, s, z) \\ = \;& \boldsymbol{\theta}(u) + \frac{1}{2}\langle x,\, \mathcal{Q}x\rangle + \langle c,\, x\rangle + \delta_{\boldsymbol{\mathcal{K}}}(v) - \langle y,\, \mathcal{B}x - b\rangle - \langle s,\, x - u\rangle - \langle z,\, x - v\rangle \\ = \;& \frac{1}{2}\langle x,\, \mathcal{Q}x\rangle + \langle c - \mathcal{B}^* y - s - z,\, x\rangle + \boldsymbol{\theta}(u) + \langle s,\, u\rangle + \delta_{\boldsymbol{\mathcal{K}}}(v) + \langle z,\, v\rangle + \langle y,\, b\rangle, \end{aligned}$$

where $x, u, v, s, z \in \boldsymbol{\mathcal{X}}, y \in \boldsymbol{\mathcal{Y}}$. Now for a given subspace $\mathcal{W} \subset \boldsymbol{\mathcal{X}}$ containing Range($\mathcal{Q}$), the range space of $\mathcal{Q}$, we have

$$\begin{aligned} \inf_x \mathcal{L}(x, u, v; y, s, z) &= \inf_x \Big\{\frac{1}{2}\langle x,\, \mathcal{Q}x\rangle + \langle c - \mathcal{B}^* y - s - z,\, x\rangle\Big\} \\ &= \begin{cases} -\frac{1}{2}\langle w,\, \mathcal{Q}w\rangle, & \text{if } c - \mathcal{B}^* y - s - z = -\mathcal{Q}w \text{ for some } w \in \mathcal{W}, \\ -\infty, & \text{otherwise.} \end{cases} \end{aligned}$$

Also,

$$\begin{aligned} \inf_u \mathcal{L}(x, u, v; y, s, z) &= \inf_u \big[\boldsymbol{\theta}(u) + \langle s,\, u\rangle\big] = -\boldsymbol{\theta}^*(-s); \\ \inf_v \mathcal{L}(x, u, v; y, s, z) &= \inf_v \big[\delta_{\boldsymbol{\mathcal{K}}}(v) + \langle z,\, v\rangle\big] = -\delta^*_{\boldsymbol{\mathcal{K}}}(-z). \end{aligned}$$

Hence the dual of (4.5) is given by

$$\max_{y,s,z} \inf_{x,u,v} \mathcal{L}(x, u, v; y, s, z)$$

$$= \max_{y,s,z,w} \left\{ -\boldsymbol{\theta^*}(-s) - \frac{1}{2}\langle w,\, \mathcal{Q}w\rangle + \langle y,\, b\rangle - \delta^*_{\boldsymbol{\mathcal{K}}}(-z) \,\middle|\, \begin{array}{l} -\mathcal{Q}w + \mathcal{B}^* y + s + z = c, \\ w \in \mathcal{W} \end{array} \right\},$$

or equivalently,

$$\begin{array}{rl} -\min & \boldsymbol{\theta}^*(-s) + \frac{1}{2}\langle w,\, \mathcal{Q}w\rangle - \langle b,\, y\rangle + \delta^*_{\boldsymbol{\mathcal{K}}}(-z) \\ \text{s.t.} & -\mathcal{Q}w + \mathcal{B}^* y + s + z = c, \\ & s \in \boldsymbol{\mathcal{X}},\ y \in \boldsymbol{\mathcal{Y}},\ w \in \mathcal{W}. \end{array} \tag{4.6}$$

It is not difficult to check that for all $z = (z_0; z_1; \ldots; z_N)$, $s = (s_0; s_1; \ldots; s_N) \in \boldsymbol{\mathcal{X}}$, we have

$$\delta^*_{\boldsymbol{\mathcal{K}}}(-z) = \textstyle\sum_{i=0}^N \delta^*_{\mathcal{K}_i}(-z_i), \quad \boldsymbol{\theta}^*(-s) = \sum_{i=0}^N \theta^*_i(-s_i). \tag{4.7}$$

Assume that both the primal and dual problems satisfy the (generalized) Slater's condition. Then the optimal solutions for both problems exist and they satisfy the following Karush-Kuhn-Tucker (KKT) optimality conditions:

$$\left\{ \begin{array}{l} \mathcal{B}x - b = 0, \\ -\mathcal{Q}w + \mathcal{B}^* y + s + z - c = 0, \quad \mathcal{Q}w - \mathcal{Q}x = 0, \quad w \in \mathcal{W}, \\ -s \in \partial\boldsymbol{\theta}(x) \ \Leftrightarrow\ x - \mathrm{Prox}_{\boldsymbol{\theta}}(x - s) = 0, \\ x - \Pi_{\boldsymbol{\mathcal{K}}}(x - z) = 0. \end{array} \right. \tag{4.8}$$

By applying the structures in (4.1)–(4.3) and (4.7) to (4.6), we get explicitly the dual of (PBA-P):

$$\begin{array}{rll} \text{(PBA-D)} & -\min & \sum_{i=0}^N \theta^*_i(-s_i) + \delta^*_{\mathcal{K}_i}(-z_i) + \frac{1}{2}\langle w_i,\, \mathcal{Q}_i(w_i)\rangle - \langle b_i,\, y_i\rangle \\ \text{s.t.} & \left[\begin{array}{c} \mathcal{A}_0^* \\ \mathcal{A}_1^* \\ \vdots \\ \mathcal{A}_N^* \end{array}\right] y_0 + \left[\begin{array}{c} -\mathcal{Q}_0 w_0 + s_0 + z_0 \\ \mathcal{D}_1^* y_1 - \mathcal{Q}_1 w_1 + s_1 + z_1 \\ \vdots \\ \mathcal{D}_N^* y_N - \mathcal{Q}_N w_N + s_N + z_N \end{array}\right] = c, \\ & w_i \in \mathcal{W}_i,\ i = 0, 1, \ldots, N, \end{array} \tag{4.9}$$

where for each $i = 0, 1, \ldots, N$, $\mathcal{W}_i \subset \mathcal{X}_i$ is a given subspace containing Range($\mathcal{Q}_i$).

4.2 Inexact semi-proximal augmented Lagrangian methods for the primal problem (PBA-P)

First we rewrite (PBA-P) in the following form:

$$\min \Big\{ \sum_{i=0}^{N} f_i(x_i) + \delta_{F_i}(x_i) \;\mid\; \mathcal{A}x = b_0,\; x = (x_0; x_1; \ldots; x_N) \in \boldsymbol{\mathcal{X}} \Big\}, \qquad (4.10)$$

where $F_0 = \mathcal{K}_0$, and $F_i = \{x_i \in \mathcal{X}_i \mid \mathcal{D}_i x_i = b_i, x_i \in \mathcal{K}_i\}$, $i = 1, \ldots, N$. For a given parameter $\sigma > 0$, we consider the following augmented Lagrangian function associated with (4.10):

$$L_\sigma(x; y_0) = \textstyle\sum_{i=0}^{N} f_i(x_i) + \delta_{F_i}(x_i) + \frac{\sigma}{2}\|\mathcal{A}x - b_0 - \sigma^{-1} y_0\|^2 - \frac{1}{2\sigma}\|y_0\|^2. \quad (4.11)$$

The augmented Lagrangian method for solving (4.10) has the following template.

ALM. Given $\sigma > 0$ and $y_0^0 \in \mathcal{Y}_0$. Perform the following steps in each iteration.

Step 1. $x^{k+1} \approx \operatorname{argmin}_x L_\sigma(x; y_0^k)$.

Step 2. $y_0^{k+1} = y_0^k + \tau\sigma(b_0 - \mathcal{A}x^{k+1})$, where $\tau \in (0, 2)$ is the step-length.

As one may observe from Step 1 of the ALM, an undesirable feature in the method is that it destroys the separable structure in the Dantzig-Wolfe decomposition method. Although the feasible sets for the x_i's are separable, the objective function has a quadratic term which couples all the x_i's.

Here we propose to add a semi-proximal term to the augmented Lagrangian function to overcome the difficulty of non-separability. In this case, the function $L_\sigma(x; y_0^k)$ in Step 1 of the ALM is majorized by an additional semi-proximal term at the point x^k, i.e.,

$$L_\sigma(x; y_0^k) + \frac{\sigma}{2}\|x - x^k\|_{\mathcal{T}}^2,$$

where $\mathcal{T}$ is a given positive semidefinite self-adjoint linear operator which should

be chosen appropriately to decompose the computation of the x_i's in Step 1 of the ALM while at the same time the added proximal term should be as small as possible. In this section, we choose $\mathcal{T}$ to be the following positive semidefinite linear operator:

$$\mathcal{T} = \mathrm{diag}(\mathcal{J}_0, \ldots, \mathcal{J}_N) - \mathcal{A}^*\mathcal{A}, \tag{4.12}$$

where $\mathcal{J}_i \succeq \beta_i I + \mathcal{A}_i^*\mathcal{A}_i$, with $\beta_i = \sum_{j=0, j\neq i}^N \|\mathcal{A}_i^*\mathcal{A}_j\|_2$ for each $i = 0, 1, \ldots, N$. Such a choice is generally less conservative than the usual choice of $\widehat{\mathcal{T}}$ that will be given later in (4.20). It is especially a good choice when $\mathcal{A}_i$ and $\mathcal{A}_j$ are nearly orthogonal to one another for most of the index pairs (i, j).

With the choice in (4.12), we get

$$\begin{aligned}
& L_\sigma(x; y_0^k) + \frac{\sigma}{2}\|x - x^k\|_{\mathcal{T}}^2 \\
= & \sum_{i=0}^N \big(f_i(x_i) + \delta_{F_i}(x_i)\big) + \frac{\sigma}{2}\|\mathcal{A}x - b_0 - \sigma^{-1}y_0\|^2 - \frac{1}{2\sigma}\|y_0\|^2 + \frac{\sigma}{2}\|x - x^k\|_{\mathcal{T}}^2 \\
= & \sum_{i=0}^N \Big(f_i(x_i) + \delta_{F_i}(x_i) + \frac{\sigma}{2}\Big[\langle x_i,\, \mathcal{J}_i x_i\rangle - 2\langle x_i,\, \mathcal{A}_i^*(b_0 + \sigma^{-1}y_0 - \mathcal{A}x^k) + \mathcal{J}_i x_i^k\rangle\Big]\Big) \\
& + \frac{\sigma}{2}\Big[\|b_0 + \sigma^{-1}y_0\|^2 + \|x^k\|_{\mathcal{T}}\Big] - \frac{1}{2\sigma}\|y_0\|^2.
\end{aligned}$$

The inexact semi-proximal ALM (sPALM) we consider for solving the primal block angular problem (PBA-P) through (4.10) is given as follows.

sPALM. Given $\sigma > 0$ and $y_0^0 \in \mathcal{Y}_0$. Let $\{\varepsilon_k\}$ be a given summable sequence of nonnegative numbers. Perform the following steps in each iteration.

Step 1. Compute

$$x^{k+1} \approx \widehat{x}^{k+1} := \mathrm{argmin}\{L_\sigma(x; y_0^k) + \frac{\sigma}{2}\|x - x^k\|_{\mathcal{T}}^2\}, \tag{4.13}$$

with residual

$$d^{k+1} \in \partial_x L_\sigma(x^{k+1}; y_0^k) + \sigma\mathcal{T}(x^{k+1} - x^k), \tag{4.14}$$

satisfying $\|d^{k+1}\| \le \varepsilon_k$. Let $\mathcal{G}_i = \mathcal{Q}_i + \sigma\mathcal{J}_i$, $g_i^k = \mathcal{Q}_i x_i^k + c_i + \sigma\mathcal{A}_i^*(\mathcal{A}x^k - b_0 - \sigma^{-1}y_0^k) - \mathcal{G}_i x_i^k$. Due to the separability of the variables in (4.13) because of the specially chosen $\mathcal{T}$, one can compute **in parallel** for $i = 0, 1, \ldots, N$,

$$x_i^{k+1} \approx \widehat{x}_i^{k+1} := \mathrm{argmin}\Big\{\theta_i(x_i) + \frac{1}{2}\langle x_i, \mathcal{G}_i x_i\rangle + \langle g_i^k, x_i\rangle \mid x_i \in F_i\Big\}, \tag{4.15}$$

with the residual $d_i^{k+1} := v_i^{k+1} + \mathcal{G}_i x_i^{k+1} + g_i^k$ for some $v_i^{k+1} \in \partial(\theta_i + \delta_{F_i})(x_i^{k+1})$ and satisfying

$$\|d_i^{k+1}\| \le \frac{1}{\sqrt{N+1}}\varepsilon_k. \tag{4.16}$$

Step 2. $y_0^{k+1} = y_0^k + \tau\sigma(b_0 - \mathcal{A}x^{k+1})$, where $\tau \in (0, 2)$ is the steplength.

Observe that with the introduction of the semi-proximal term $\frac{\sigma}{2}\|x - x^k\|_{\mathcal{T}}^2$ to the augmented Lagrangian function in Step 1 of the sPALM, we have decomposed the large coupled problem involving x in ALM into $N + 1$ smaller independent problems that can be solved in parallel. For the case of a quadratic or linear program, we can employ a powerful solver such as Gurobi or Mosek to efficiently solve these smaller problems.

In order to judge how accurately the decomposed subproblems in Step 1 must be solved, we need to analyse the stopping condition for (4.15) in detail. In particular, we need to find $v_i^{k+1} \in \partial(\theta_i + \delta_{F_i})(x_i^{k+1})$ for $i = 0, 1, \ldots, N$. This can be done by considering the dual of the subproblem (4.15), which could be

written as:

$$\begin{aligned} -\min \quad & \theta_i^*(-s_i) + \tfrac{1}{2}\langle w_i,\, \mathcal{G}_i w_i\rangle - \langle b_i,\, y_i\rangle + \delta^*_{\mathcal{K}_i}(-z_i) \\ \text{s.t.} \quad & -\mathcal{G}_i w_i + \mathcal{D}_i^* y_i + s_i + z_i = g_i^k, \\ & s_i \in \mathcal{X}_i,\ \ y_i \in \mathcal{Y}_i,\ w_i \in \mathcal{W}_i, \quad i = 1, \ldots, N. \end{aligned} \tag{4.17}$$

Note that for $i = 0$, we have a similar problem as the above but the terms involving y_i are absent. For the discussion below, we will just focus on the case where $i = 1, \ldots, N$, the case for $i = 0$ can be derived similarly. One can estimate v_i^{k+1} to be $-\mathcal{D}_i^* y_i^{k+1} - s_i^{k+1} - z_i^{k+1}$ for a computed dual solution $(y_i^{k+1}, s_i^{k+1}, z_i^{k+1})$ and the residual d_i^{k+1} is simply the residual in the dual feasibility constraint in the above problem.

Remark 1. In the sPALM, some of the dual variables for (PBA-D) are not explicitly constructed. Here we describe how they can be estimated. Recall that for (PBA-D), we want to get

$$-\mathcal{Q}_i x_i + \mathcal{A}_i^* y_0 + \mathcal{D}_i^* y_i + s_i + z_i - c_i = 0 \quad \forall\, i = 0, 1, \ldots, N.$$

Note that for convenience, we introduced $\mathcal{D}_0^* = 0$. From the KKT conditions for (4.15) and (4.17), we have that

$$\begin{aligned} -\mathcal{G}_i w_i^{k+1} + \mathcal{D}_i^* y_i^{k+1} + s_i^{k+1} + z_i^{k+1} - g_i^k =: R_i^d \ &\approx\ 0, \\ \mathcal{G}_i w_i^{k+1} - \mathcal{G}_i x_i^{k+1} \ &\approx\ 0. \end{aligned}$$

By using the expression for $\mathcal{G}_i$, g_i^k and y_0^{k+1}, we get

$$\begin{aligned} & -\mathcal{Q}_i x_i^{k+1} + \mathcal{A}_i^* y_0^{k+1} + \mathcal{D}_i^* y_i^{k+1} + s_i^{k+1} + z_i^{k+1} - c_i \\ = \ & R_i^d + (\mathcal{G}_i w_i^{k+1} - \mathcal{G}_i x_i^{k+1}) + \sigma \mathcal{J}_i (x_i^{k+1} - x_i^k) + \sigma \mathcal{A}_i^* \mathcal{A}(x^k - x^{k+1}) \\ & + (\tau - 1)\sigma \mathcal{A}_i^* (b_0 - \mathcal{A} x^{k+1}). \end{aligned}$$

Note that the right-hand-side quantity in the above equation will converge to 0 based on the convergence of sPALM and the KKT conditions for (4.15) and

(4.17). Thus by using the dual variables computed from solving (4.17), we can generate the dual variables for (PBA-D).

4.2.1 Convergence of the inexact sPALM

The convergence of the inexact sPALM for solving (4.10) can be established readily by using known results in Chen et al. (2018). To do that, we need to first reformulate (4.10) into the form required in Chen et al. (2018) as follows:

$$\min\Big\{h(x)+\psi(x)\ \mid\ \mathcal{A}x=b_0,\ x=(x_0;x_1;\ldots;x_N)\in\boldsymbol{\mathcal{X}}\Big\},\tag{4.18}$$

where $h(x)=\sum_{i=0}^{N}\frac{1}{2}\langle x_i,\,Q_ix_i\rangle+\langle c_i,\,x_i\rangle$ and $\psi(x)=\sum_{i=0}^{N}\theta_i(x_i)+\delta_{F_i}(x_i)$. Its corresponding KKT residual mapping is given by

$$\mathcal{R}(x,y_0)=\begin{pmatrix} b_0-\mathcal{A}x \\ \\ x-\mathrm{Prox}_{\psi}(x-\mathcal{Q}x-c-\mathcal{A}^*y_0)\end{pmatrix}\quad\forall\ x\in\boldsymbol{\mathcal{X}},\ y_0\in\mathcal{Y}_0.\tag{4.19}$$

Note that (x,y_0) is a solution of the KKT system of (4.18) if and only $\mathcal{R}(x,y_0)=0$.

Now we state the global convergence theorem here for the convenience of the readers. Define the self-adjoint positive definite linear operator $\mathcal{V}:\boldsymbol{\mathcal{X}}\to\boldsymbol{\mathcal{X}}$ by

$$\mathcal{V}:=\tau\sigma\Big(\mathcal{Q}+\sigma\mathcal{T}+\frac{2-\tau}{6}\sigma\mathcal{A}^*\mathcal{A}\Big).$$

We have the following convergence result for the inexact sPALM.

Theorem 4.1. Assume that the solution set to the KKT system of (4.10) is nonempty and $(\overline{x},\overline{y}_0)$ is a solution. Then, the sequence $\{(x^k,y_0^k)\}$ generated by sPALM is well-defined such that for any $k\geq 1$,

$$\|x^{k+1}-\widehat{x}^{k+1}\|^2_{\mathcal{Q}+\sigma\mathcal{T}+\sigma\mathcal{A}^*\mathcal{A}}\leq\langle d^{k+1},x^{k+1}-\widehat{x}^{k+1}\rangle,$$

and for all $k = 0, 1, \ldots$,

$$\begin{aligned}&\left(\|x^{k+1} - \overline{x}\|^2_{\widehat{\mathcal{V}}^{1/2}} + \|y_0^{k+1} - \overline{y}_0\|^2\right) - \left(\|x^k - \overline{x}\|^2_{\widehat{\mathcal{V}}^{1/2}} + \|y_0^k - \overline{y}_0\|^2\right)\\ &\leq -\left(\tfrac{2-\tau}{3\tau}\|y_0^k - y_0^{k+1}\|^2 + \|x^{k+1} - x^k\|^2_{\mathcal{V}} - 2\tau\sigma\langle d^k, x^{k+1} - \overline{x}\rangle\right),\end{aligned}$$

where $\widehat{\mathcal{V}} = \mathcal{V} + \frac{2-\tau}{6}\tau\sigma^2\mathcal{A}^*\mathcal{A}$. Moreover, the sequence $\{(x^k, y_0^k)\}$ converges to a solution to the KKT system of (4.10).

Proof. The result can be proved directly from the convergence result in (Chen et al., 2018, Theorem 1). □

The local linear convergence of sPALM can also be established if the KKT residual mapping $\mathcal{R}$ satisfies the following error bound condition: there exist positive constants κ and r such that $\text{dist}((x, y_0), \Omega) \leq \kappa\|\mathcal{R}(x, y_0)\|$ for all (x, y_0) satisfying $\|(x, y_0) - (x^*, y_0^*)\| \leq r$, where Ω is the solution set of (4.18) and (x^*, y_0^*) is a particular solution of (4.18). In order to save some space, we will not state the theorem here but refer the reader to (Chen et al., 2018, Theorem 2).

4.2.2 Comparison of sPALM with the diagonal quadratic approximation method and its recent variants

Let $\rho := (N+1)^{-1}$. Consider the following linear operator

$$\widehat{\mathcal{T}} = \text{diag}(\mathcal{E}_0, \ldots, \mathcal{E}_N) - \mathcal{A}^*\mathcal{A}, \tag{4.20}$$

where $\mathcal{E}_i \succeq \rho^{-1}\mathcal{A}_i^*\mathcal{A}_i$ for all $i = 0, 1, \ldots, N$. It is not difficult to show that $\widehat{\mathcal{T}} \succeq 0$.

If instead of (4.12), we choose $\mathcal{T}$ to be the linear operator given in (4.20), then instead of **sPALM**, we get the following variant of the inexact sPALM.

sPALM-b. Given $\sigma > 0$ and $y_0^0 \in \mathcal{Y}_0$. Let $\{\varepsilon_k\}$ be a given summable sequence of nonnegative numbers. Perform the following steps in each iteration.

Step 1. Let $g_i^k = \mathcal{Q}_i x_i^k + c_i + \sigma \mathcal{A}_i^*(\mathcal{A}x^k - b_0 - \sigma^{-1} y_0^k) - (\mathcal{Q}_i + \sigma \mathcal{E}_i) x_i^k$. Compute (in parallel) for $i = 0, 1, \ldots, N$,

$$x_i^{k+1} \approx \operatorname{argmin}\Big\{\theta_i(x_i) + \frac{1}{2}\langle x_i,\, (\mathcal{Q}_i + \sigma\mathcal{E}_i)x_i\rangle + \langle g_i^k,\, x_i\rangle \mid x_i \in F_i\Big\}, \quad (4.21)$$

with the residual $d_i^{k+1} := v_i^{k+1} + (\mathcal{Q}_i + \sigma\mathcal{E}_i)x_i^{k+1} + g_i^k$ for some $v_i^{k+1} \in \partial(\theta_i + \delta_{F_i})(x_i^{k+1})$ and satisfying $\|d_i^{k+1}\| \le \frac{1}{\sqrt{N+1}}\varepsilon_k$.

Step 2. $y_0^{k+1} = y_0^k + \tau\sigma(b_0 - \mathcal{A}x^{k+1})$, where $\tau \in (0, 2)$ is the steplength.

In Ruszczyński (1989), Ruszczyński proposed the diagonal quadratic approximation (DQA) augmented Lagrangian method that aims to solve a problem of the form (PBA-P). As already mentioned, the DQA method is a very successful decomposition method that is frequently used in stochastic programming. Although it was not derived in our way in Ruszczyński (1989), we shall see later that the DQA method can roughly be derived as the augmented Lagrangian method described in **ALM** where the minimization problem in Step 1 is solved approximately by a proximal gradient method, with the proximal term chosen specially using the linear operator $\widehat{\mathcal{T}}$ in (4.20) to make the resulting subproblem separable.

ALM-DQA-mod. Given $\sigma > 0$, $y_0^0 \in \mathcal{Y}_0$ and $x^0 \in \boldsymbol{\mathcal{X}}$. Let $\{\varepsilon_k\}$ be a given summable sequence of nonnegative numbers. Perform the following steps in each iteration.

Step 1. Starting with $\hat{x}^0 = x^k$, iterate the following step for $s = 0, 1, \ldots$ until convergence:

- Compute $\hat{x}^{s+1} \approx \operatorname{argmin}\{L_\sigma(x; y_0^k) + \frac{\sigma}{2}\|x - \hat{x}^s\|_{\widehat{\mathcal{T}}}^2 \mid x \in \boldsymbol{\mathcal{X}}\}$. As the problem is separable, one can compute in parallel for $i = 0, 1, \ldots, N$,

$$\begin{aligned} \hat{x}_i^{s+1} &\approx \operatorname{argmin}\Big\{ \begin{array}{c} f_i(x_i) + \frac{\sigma}{2}\langle x_i - \hat{x}_i^s,\, \mathcal{E}_i(x_i - \hat{x}_i^s)\rangle \\ +\langle x_i - \hat{x}_i^s,\, \hat{g}_i^s\rangle \end{array} \mid x_i \in F_i \Big\} \\ &= \operatorname{argmin}\Big\{ \begin{array}{c} \theta_i(x_i) + \frac{1}{2}\langle x_i,\, (\mathcal{Q}_i + \sigma\mathcal{E}_i)x_i\rangle \\ +\langle x_i,\, \bar{g}_i^s\rangle \end{array} \mid x_i \in F_i \Big\}, \end{aligned} \tag{4.22}$$

  where $\hat{g}_i^s = \sigma\mathcal{A}_i^*(\mathcal{A}\hat{x}^s - b_0 - \sigma^{-1}y_0^k)$, $\bar{g}_i^s = \mathcal{Q}_i\hat{x}_i^s + c_i + \sigma\mathcal{A}_i^*(\mathcal{A}\hat{x}^s - b_0 - \sigma^{-1}y_0^k) - (\mathcal{Q}_i + \sigma\mathcal{E}_i)\hat{x}_i^s$.

  At termination, set $x^{k+1} = \hat{x}^{s+1}$.

Step 2. $y_0^{k+1} = y_0^k + \tau\sigma(b_0 - \mathcal{A}x^{k+1})$, where $\tau \in (0, 2)$ is the steplength.

Observe that the subproblem (4.21) in Step 1 of sPALM-b is exactly one step of the proximal gradient method (4.22) in Step 1 of the ALM-DQA-mod. As solving the problem of the form in (4.22) multiple times for each iteration of the ALM-DQA-mod may be expensive, it is highly conceivable that the overall efficiency of sPALM-b could be better than that of the ALM-DQA-mod.

Next, we elucidate the connection between **ALM-DQA-mod** and the DQA method described in Ruszczyński (1989). Given $\hat{x}_i^s \in F_i$, we can parameterize a given x_i as

$$x_i = \hat{x}_i^s + \rho d_i = (1 - \rho)\hat{x}_i^s + \rho(\hat{x}_i^s + d_i),\ i = 0, 1, \ldots, N,$$

with $\rho = (N+1)^{-1} \in (0, 1]$. Then by convexity, $f_i(x_i) \le (1-\rho)f_i(\hat{x}_i^s) + \rho f_i(\hat{x}_i^s + d_i)$. Also, if $\hat{x}_i^s + d_i \in F_i$, then $x_i \in F_i$ since $\hat{x}_i^s \in F_i$. From here, we have that

for all $x \in F_0 \times F_1 \times \cdots \times F_N$,

$$\begin{aligned} & L_\sigma(x; y_0^k) + \frac{\sigma}{2}\|x - \hat{x}^s\|_{\widehat{\mathcal{T}}}^2 + \frac{1}{2\sigma}\|y_0^k\|^2 \\ \leq\ & (1-\rho)\sum_{i=0}^N f_i(\hat{x}_i^s) + \rho\sum_{i=0}^N f_i(\hat{x}_i^s + d_i) + \frac{\sigma}{2}\|\mathcal{A}(\hat{x}^s + \rho d) - b_0 - \sigma^{-1}y_0^k\|^2 + \frac{\sigma\rho^2}{2}\|d\|_{\widehat{\mathcal{T}}}^2 \\ =\ & \sum_{i=0}^N \rho f_i(\hat{x}_i^s + d_i) + \rho\langle d_i,\, \hat{g}_i^s\rangle + \frac{\sigma\rho^2}{2}\langle d_i,\, \mathcal{E}_i d_i\rangle + (1-\rho)\sum_{i=0}^N f_i(\hat{x}_i^s) \\ & + \frac{\sigma}{2}\|\mathcal{A}\hat{x}^s - b_0 - \sigma^{-1}y_0^k\|^2. \end{aligned} \tag{4.23}$$

Hence instead of (4.22), we may consider to minimize the majorization of $L_\sigma(x; y_0^k) + \frac{\sigma}{2}\|x - \hat{x}^s\|_{\widehat{\mathcal{T}}}^2$ in (4.23), and compute for $i = 0, 1, \ldots, N$,

$$d_i^{s+1} = \operatorname{argmin} \rho\big\{ f_i(\hat{x}_i^s + d_i) + \frac{\sigma\rho}{2}\langle d_i,\, \rho\mathcal{E}_i d_i\rangle + \langle d_i,\, \hat{g}_i^s\rangle \mid \hat{x}_i^s + d_i \in F_i,\ d_i \in \mathcal{X}_i \big\}. \tag{4.24}$$

We get the DQA method of Ruszczyński (1989) if we take $\mathcal{E}_i = \rho^{-1}\mathcal{A}_i^*\mathcal{A}_i$, compute d^{s+1} exactly in the above subproblem (4.24), and set

$$\hat{x}_i^{s+1} = \hat{x}_i^s + \rho d_i^{s+1}, \quad i = 0, 1, ..., N,$$

instead of the solution in (4.22). Thus we may view the DQA method as an augmented Lagrangian method for which the subproblem in Step 1 is solved by a majorized proximal gradient method with the proximal term chosen to be $\frac{\sigma}{2}\|x - \hat{x}^s\|_{\widehat{\mathcal{T}}}^2$ in each step.

Remark 2. When the $\mathcal{A}_i$'s are matrices, the majorization $\mathcal{A}^*\mathcal{A} \preceq \operatorname{diag}(\mathcal{E}_0, \ldots, \mathcal{E}_N)$ can be improved as follows, as has been done in Chatzipanagiotis et al. (2015). Let

$$I_j = \big\{ i \in \{0, 1, \ldots, N\} \mid e_j^T \mathcal{A}_i \neq 0 \big\}, \quad \chi := \max\{|I_j| \mid j = 1, \ldots, m\} \ \leq\ N{+}1 = \rho^{-1}.$$

Then

$$\|\mathcal{A}x\|^2 = \|\sum_{i=0}^N \mathcal{A}_i x_i\|^2 = \sum_{j=1}^m |\sum_{i=0}^N e_j^T \mathcal{A}_i x_i|^2 = \sum_{j=1}^m |\sum_{i\in I_j} e_j^T \mathcal{A}_i x_i|^2$$
$$\leq \sum_{j=1}^m \left(|I_j| \sum_{i\in I_j} |e_j^T \mathcal{A}_i x_i|^2\right) \leq \chi \sum_{j=1}^m \sum_{i\in I_j} |e_j^T \mathcal{A}_i x_i|^2$$
$$= \chi \sum_{j=1}^m \sum_{i=0}^N |e_j^T \mathcal{A}_i x_i|^2 = \chi \sum_{i=0}^N \|\mathcal{A}_i x_i\|^2.$$

That is, $\mathcal{A}^*\mathcal{A} \preceq \mathrm{diag}(\chi\mathcal{A}_0^*\mathcal{A}_0, \cdots, \chi\mathcal{A}_N^*\mathcal{A}_N)$. Such an improvement has been considered in Chatzipanagiotis et al. (2015). It is straightforward to incorporate the improvement into ALM-DQA-mod by simply replacing $\mathcal{E}_i = \rho^{-1}\mathcal{A}_i^*\mathcal{A}_i$ in (4.20) by $\chi\mathcal{A}_i^*\mathcal{A}_i$ for each $i = 0, 1, \ldots, N$.

With the derivation of the **ALM-DQA-mod** as an augmented Lagrangian method with its subproblems solved by a specially chosen proximal gradient method, we can leverage on this viewpoint to design an accelerated variant of this method. Specifically, we can improve the efficiency in solving the subproblems by using an inexact accelerated proximal gradient (iAPG) method, and we will also use a proximal term based on the linear operator (4.12), which is typically less conservative than the term $\frac{\sigma}{2}\|x - x^k\|_{\widehat{\mathcal{T}}}$ used in the DQA method.

ALM-iAPG. Given $\sigma > 0$, $y_0^0 \in \mathcal{Y}_0$ and $x^0 \in \boldsymbol{\mathcal{X}}$. Let $\{\varepsilon_k\}$ be a given summable sequence of nonnegative numbers. Perform the following steps in each iteration.

Step 1. Starting with $\hat{x}^0 = \bar{x}^0 = x^k$, $t_0 = 1$, iterate the following step for $s = 0, 1, \ldots$ until convergence:

- Compute $\hat{x}^{s+1} \approx \mathrm{argmin}\{L_\sigma(x; y_0^k) + \frac{\sigma}{2}\|x - \bar{x}^s\|_{\mathcal{T}}^2 \mid x_i \in F_i, i = 0, 1, \ldots, N\}$. As the problem is separable, one can compute in parallel for $i = 0, 1, \ldots, N$,

$$\hat{x}_i^{s+1} \approx \mathrm{argmin}\Big\{\theta_i(x_i) + \tfrac{1}{2}\langle x_i, \mathcal{G}_i x_i\rangle + \langle x_i, \bar{g}_i^s\rangle \mid x_i \in F_i\Big\}, \quad (4.25)$$

  where $\mathcal{G}_i = \mathcal{Q}_i + \sigma\mathcal{J}_i$, $\bar{g}_i^s = \mathcal{Q}_i\bar{x}_i^s + c_i + \sigma\mathcal{A}_i^*(\mathcal{A}\bar{x}_i^s - b_0 - \sigma^{-1}y_0^k) - \mathcal{G}_i\bar{x}_i^s$.
- Compute $t_{s+1} = (1 + \sqrt{1 + 4t_s^2})/2$, $\beta_{s+1} = (t_s - 1)/t_{s+1}$.
- Compute $\bar{x}^{s+1} = (1 + \beta_{s+1})\hat{x}^{s+1} - \beta_{s+1}\hat{x}^s$.

At termination, set $x^{k+1} = \hat{x}^{s+1}$.

Step 2. $y_0^{k+1} = y_0^k + \tau\sigma(b_0 - \mathcal{A}x^{k+1})$, where $\tau \in (0, 2)$ is the steplength.

4.2.3 Numerical performance of sPALM and ALM-DQA-mod

In this subsection, we compare the performance of the sPALM and ALM-DQA-mod algorithms for solving several linear and quadratic test instances. The detailed description of the datasets is given in Section 4.4. We also report the number of constraints and variables of the instances in the table. For all the instances, we have $m_1 = m_2 = ... = m_N$ and $n_1 = n_2 = ... = n_N$. Hence we denote them as m_i and n_i respectively.

Table 4.1 compares the performance of the two solvers sPALM and ALM-DQA-mod for the primal problem (PBA-P) through (4.10) against that of the solver sGS-ADMM for the dual problem (PBA-D). The details of the dual approach will be presented in the next section. Here, we could observe that sPALM and ALM-DQA-mod always require much longer runtime to achieve the same accuracy level in the relative KKT residual when compare to sGS-ADMM, although the former algorithms generally take a smaller number of outer iterations. In

addition, the ALM-DQA-mod algorithm is slightly slower than sPALM on the whole though the difference is not too significant. Note that our preliminary implementation of the algorithms is in MATLAB which does not have a good support for parallel computing. In a full scale implementation, one may try to implement these algorithms on an appropriate parallel computing platform with a good parallelization support. Nevertheless, the inferior performance of the two primal approaches has motivated us to instead consider the dual approach of designing an efficient algorithm for the dual problem (PBA-D).

				sGS-ADMM		sPALM		ALM-DQA-mod	
Data	$m_0 \mid m_i$	$n_0 \mid n_i$	N	Iter	Time(s)	Iter	Time (s)	Iter	Time (s)
qp-rand-m1-n20-N10-t1	1 | 1	20 | 20	10	321	0.50	153	8.43	12	15.62
qp-rand-m50-n80-N10-t1	50 | 50	80 | 80	10	421	0.64	268	59.88	44	192.17
qp-rand-m10-n20-N10-t2	10 | 10	20 | 20	10	1501	1.20	2971	208.56	54	137.83
qp-rand-m50-n80-N10-t2	50 | 50	80 | 80	10	141	0.20	92	19.20	32	150.32
tripart1	2096 | 192	2096 | 2096	16	1981	3.01	3880	1212.86	1422	1113.26
tripart2	8432 | 768	8432 | 8432	16	6771	51.65	5000	6369.20	1610	5723.31
qp-tripart1	2096 | 192	2096 | 2096	16	653	1.44	308	94.60	114	202.45
qp-tripart2	8432 | 768	8432 | 8432	16	971	9.79	347	419.16	124	1043.54
qp-pds1	87 | 126	372 | 372	11	971	0.99	538	49.18	535	79.24
qp-SDC-r100-c50-l100-p1000-t1	5000 | 150	0 | 5000	100	32	1.52	10	37.49	7	61.51
qp-SDC-r100-c50-l100-p1000-t2	5000 | 150	0 | 5000	100	31	1.34	9	34.22	2	33.40
qp-SDC-r100-c50-l100-p5000-t1	5000 | 150	0 | 5000	100	32	1.40	10	37.75	8	75.85
qp-SDC-r100-c50-l100-p5000-t2	5000 | 150	0 | 5000	100	31	1.37	9	34.50	3	36.63
qp-SDC-r100-c50-l100-p10000-t1	5000 | 150	0 | 5000	100	32	1.37	10	37.93	9	86.82
qp-SDC-r100-c50-l100-p10000-t2	5000 | 150	0 | 5000	100	31	1.35	9	34.78	3	37.30
qp-SDC-r100-c100-l100-p1000-t1	10000 | 200	0 | 10000	100	31	2.67	10	73.50	7	116.72
qp-SDC-r100-c100-l100-p1000-t2	10000 | 200	0 | 10000	100	31	2.67	9	68.08	2	63.91
qp-SDC-r100-c100-l100-p5000-t1	10000 | 200	0 | 10000	100	31	2.70	10	74.02	8	137.19
qp-SDC-r100-c100-l100-p5000-t2	10000 | 200	0 | 10000	100	31	2.63	9	68.04	2	64.26
qp-SDC-r100-c100-l100-p10000-t1	10000 | 200	0 | 10000	100	32	2.65	10	74.65	8	147.16
qp-SDC-r100-c100-l100-p10000-t2	10000 | 200	0 | 10000	100	31	2.63	9	67.66	3	71.18
qp-SDC-r100-c100-l200-p20000-t1	10000 | 200	0 | 10000	200	41	6.68	10	183.21	8	302.29
qp-SDC-r200-c100-l200-p20000-t1	20000 | 300	0 | 20000	200	34	11.96	10	360.48	7	513.61
qp-SDC-r200-c200-l200-p20000-t1	40000 | 400	0 | 40000	200	31	22.33	10	783.49	7	1068.04
M64-64	405 | 64	511 | 511	64	1991	3.16	5000	2874.88	625	1241.12

				sGS-ADMM		sPALM		ALM-DQA-mod	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Iter	Time(s)	Iter	Time (s)	Iter	Time (s)

Table 4.1: Comparison of computational results between sGS-ADMM and two variants of ALM for primal block angular problem. All the run result are obtained using **single thread**. Here, "Iter" is the number of outer iterations performed, and "Time" is the total runtime in seconds.

4.3 A semi-proximal symmetric Gauss-Seidel based ADMM for the dual problem (PBA-D)

In the last section, we have designed the sPALM algorithm to solve the primal problem (PBA-P) directly. One can also attempt to solve (PBA-P) via its dual problem (PBA-D). Based on the structure in (PBA-D), we find that it is highly conducive for us to employ a symmetric Gauss-Seidel based ADMM (sGS-ADMM) to solve the problem, as we shall see later when the details are presented.

To derive the sGS-ADMM algorithm for solving (PBA-D), it is more convenient for us to express (PBA-D) in a more compact form as follows:

$$\min \{p(s) + f(y_{1:N}, w, s) + q(z) + g(y_0, z) \mid \mathcal{F}^*[y_{1:N}; w; s] + \mathcal{G}^*[y_0; z] = c\}, \quad (4.26)$$

where $y_{1:N} = [y_1; \ldots; y_N]$, and

$$\begin{aligned}
\mathcal{F}^* &:= \left[\ \mathcal{D}^*,\ \ -\mathcal{Q},\ \ I\ \right], \quad \mathcal{G}^* := \left[\ \mathcal{A}^*,\ \ I\ \right], \\
p(s) &:= \boldsymbol{\theta}^*(-s),\ f(y_{1:N}, w, s) := -\langle b_{1:N},\, y_{1:N}\rangle + \frac{1}{2}\langle w,\, \mathcal{Q}w\rangle + \delta_{\mathcal{W}}(w), \\
q(z) &:= \delta^*_{\boldsymbol{\mathcal{K}}}(-z),\ g(y_0, z) := -\langle b_0,\, y_0\rangle.
\end{aligned}$$

Here we take $\mathcal{W} = \mathrm{Range}(\mathcal{Q})$. This is a multi-block linearly constrained convex programming problem for which the direct application of the classical ADMM is not guaranteed to converge. Thus we adapt the recently developed sGS-ADMM (Chen et al. 2017; Li et al. 2016) whose convergence is guaranteed to solve the

dual problem (PBA-D).

Given a positive parameter σ, the augmented Lagrangian function for (PBA-D) is given by

$$\begin{aligned}\mathcal{L}_\sigma(y,w,s,z;x) &= p(s)+f(y_{1:N},w,s)+q(z)+g(y_0,z)+\\ &\quad \tfrac{\sigma}{2}\|\mathcal{F}^*[y_{1:N};w;s]+\mathcal{G}^*[y_0;z]-c+\tfrac{1}{\sigma}x\|^2-\tfrac{1}{2\sigma}\|x\|^2\\ &= \textstyle\sum_{i=0}^N\theta_i^*(-s_i)+\delta^*_{\mathcal{K}_i}(-z_i)+\frac{1}{2}\langle w_i,\,\mathcal{Q}_i w_i\rangle-\langle b_i,\,y_i\rangle\\ &\quad +\tfrac{\sigma}{2}\|-\mathcal{Q}_0w_0+\mathcal{A}_0^*y_0+s_0+z_0-c_0+\sigma^{-1}x_0\|^2-\tfrac{1}{2\sigma}\|x_0\|^2.\\ &\quad +\textstyle\sum_{i=1}^N\frac{\sigma}{2}\|-\mathcal{Q}_iw_i+\mathcal{A}_i^*y_0+\mathcal{D}_i^*y_i+s_i+z_i-c_i+\sigma^{-1}x_i\|^2-\frac{1}{2\sigma}\|x_i\|^2.\end{aligned}$$

Now to develop the sGS-ADMM, we need to analyze the block structure of the quadratic terms in $\mathcal{L}_\sigma(y,w,s,z;x)$ corresponding the blocks $[y_{1:N};w;s]$ and $[y_0;z]$, which are respectively given as follows:

$$\begin{aligned}\mathcal{F}\mathcal{F}^* &= \begin{bmatrix}\mathcal{D}\mathcal{D}^* & -\mathcal{D}\mathcal{Q} & \mathcal{D}\\ -\mathcal{Q}\mathcal{D}^* & \mathcal{Q}^2 & -\mathcal{Q}\\ \mathcal{D}^* & -\mathcal{Q} & I\end{bmatrix}\\ &= \underbrace{\begin{bmatrix}0 & -\mathcal{D}\mathcal{Q} & \mathcal{D}\\ 0 & 0 & -\mathcal{Q}\\ 0 & 0 & 0\end{bmatrix}}_{\mathcal{U}_\mathcal{F}}+\underbrace{\begin{bmatrix}\mathcal{D}\mathcal{D}^* & 0 & 0\\ 0 & \mathcal{Q}^2 & 0\\ 0 & 0 & I\end{bmatrix}}_{\mathcal{D}_\mathcal{F}}+\mathcal{U}_\mathcal{F}^*\\ \mathcal{G}\mathcal{G}^* &= \begin{bmatrix}\mathcal{A}\mathcal{A}^* & \mathcal{A}\\ \mathcal{A}^* & I\end{bmatrix}=\underbrace{\begin{bmatrix}0 & \mathcal{A}\\ 0 & 0\end{bmatrix}}_{\mathcal{U}_\mathcal{G}}+\underbrace{\begin{bmatrix}\mathcal{A}\mathcal{A}^* & 0\\ 0 & I\end{bmatrix}}_{\mathcal{D}_\mathcal{G}}+\mathcal{U}_\mathcal{G}^*.\end{aligned}$$

Based on the above (symmetric Gauss-Seidel) decompositions, we define the following positive semidefinite linear operators associated with the decompositions:

$$\mathrm{sGS}(\mathcal{F}\mathcal{F}^*)\;=\;\mathcal{U}_\mathcal{F}^*\mathcal{D}_\mathcal{F}^{-1}\mathcal{U}_\mathcal{F},\quad \mathrm{sGS}(\mathcal{G}\mathcal{G}^*)\;=\;\mathcal{U}_\mathcal{G}^*\mathcal{D}_\mathcal{G}^{-1}\mathcal{U}_\mathcal{G}. \tag{4.27}$$

Note that here we view $\mathcal{Q}$ as a linear operator defined on $\mathcal{W}$ and because we

take $\mathcal{W} = \mathrm{Range}(\mathcal{Q})$, $\mathcal{Q}^2$ is positive definite on $\mathcal{W}$ and hence $\mathcal{D}_{\mathcal{F}}$ is invertible. Since $\mathcal{A}$ is assumed to have full row-rank, $\mathcal{D}_{\mathcal{G}}$ is also invertible.

Given the current iterate $(y^k, s^k, w^k, z^k, x^k)$, the basic template of the sGS-ADMM for (4.26) at the k-th iteration is given as follows.

Step 1. Compute

$$(y_{1:N}^{k+1}, w^{k+1}, s^{k+1}) \;=\; \mathrm{argmin}_{y_{1:N},w,s} \left\{ \begin{array}{l} p(s) + f(y_{1:N}, w, s) \\ +\frac{\sigma}{2}\|\mathcal{F}^*[y_{1:N}; w; s] + \mathcal{G}^*[y_0^k; z^k] - c + \frac{1}{\sigma}x^k\|^2 \\ +\frac{\sigma}{2}\|[y_{1:N}; w; s] - [y_{1:N}^k; w^k; s^k]\|^2_{\mathrm{sGS}(\mathcal{F}\mathcal{F}^*)} \end{array} \right\}.$$

Step 2. Compute

$$(y_0^{k+1}, z^{k+1}) = \mathrm{argmin}_{y_0,z} \left\{ \begin{array}{l} q(z) + g(y_0, z) \\ +\frac{\sigma}{2}\|\mathcal{F}^*[y_{1:N}^{k+1}; w^{k+1}; s^{k+1}] + \mathcal{G}^*[y_0; z] - c + \frac{1}{\sigma}x^k\|^2 \\ +\frac{\sigma}{2}\|[y_0; z] - [y_0^k; z^k]\|^2_{\mathrm{sGS}(\mathcal{G}\mathcal{G}^*)} \end{array} \right\}.$$

Step 3. Compute $x^{k+1} = x^k + \tau\sigma(\mathcal{F}^*[y_{1:N}^{k+1}; w^{k+1}; s^{k+1}] + \mathcal{G}^*[y_0^{k+1}; z^{k+1}] - c)$, where $\tau \in (0, \frac{1+\sqrt{5}}{2})$ is the steplength.

By using the sGS-decomposition theorem in Li et al. (2018a), we can show that the computation in Step 1 can be done by updating the blocks $(y_{1:N}, w, s)$ in a symmetric Gauss-Seidel fashion. Similarly, the computation in Step 2 can be done by updating the blocks (y_0, z) in a symmetric Gauss-Seidel fashion. With the above preparations, we can now give the detailed description of the sGS-ADMM algorithm for solving (4.9).

sGS-ADMM on (4.9)**.** Given $(y^0, w^0, s^0, z^0, x^0) \in \boldsymbol{\mathcal{Y}} \times \mathcal{W} \times \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}}$, perform the following steps in each iteration. Note that for notational convenience, we define $\mathcal{D}_0 = 0$ in the algorithm.

Step 1a. Let $g^k = \mathcal{A}^* y_0^k + z^k - c + \sigma^{-1}x^k$. Compute

$$(\bar{y}_1^k, \ldots, \bar{y}_N^k) \;=\; \mathrm{argmin}_{y_1,\ldots,y_N} \Big\{ \mathcal{L}_\sigma\big((y_0^k, y_1, \ldots, y_N), w^k, s^k, z^k; x^k\big) \Big\},$$

which can be done in parallel by computing for $i = 1, \ldots, N$,

$$\bar{y}_i^k \;=\; \mathrm{argmin}_{y_i}\Big\{ -\langle b_i,\, y_i\rangle + \frac{\sigma}{2}\| - \mathcal{Q}_i w_i^k + \mathcal{D}_i^* y_i + s_i^k + g_i^k\|^2\Big\}.$$

Specifically, for $i = 1, \ldots, N$, $\bar{y}_i^k$ is the solution of the following linear system:

$$\mathcal{D}_i\mathcal{D}_i^*\, y_i \;=\; \sigma^{-1} b_i - \mathcal{D}_i(-\mathcal{Q}_i w_i^k + s_i^k + g_i^k). \tag{4.28}$$

Step 1b Compute $\bar{w}^k = \mathrm{argmin}\big\{\mathcal{L}_\sigma\big((y_0^k, \bar{y}_1^k, \ldots, \bar{y}_N^k), w, s^k, z^k; x^k\big)\big\}$ by computing in parallel for $i = 0, 1, \ldots, N$,

$$\bar{w}_i^k \;=\; \mathrm{argmin}_{w_i}\Big\{\frac{1}{2}\langle w_i,\, \mathcal{Q}_i w_i\rangle + \frac{\sigma}{2}\| - \mathcal{Q}_i w_i + \mathcal{D}_i^*\bar{y}_i^k + s_i^k + g_i^k\|^2 \mid w_i \in \mathrm{Range}(\mathcal{Q}_i)\Big\}.$$

It is important to note that $\bar{w}_i^k$ is only needed theoretically but not needed explicitly in practice. This is because in practical computation, only $\mathcal{Q}_i\bar{w}_i^k$ is needed. To compute $\mathcal{Q}_i\bar{w}_i^k$, we first compute the solution $\tilde{w}_i^k$ of the linear system below:

$$(I + \sigma\mathcal{Q}_i)\tilde{w}_i \;=\; \sigma(\mathcal{D}_i^*\bar{y}_i^k + s_i^k + g_i^k). \tag{4.29}$$

Then we can compute $\mathcal{Q}_i\bar{w}_i^k = \mathcal{Q}_i\tilde{w}_i^k$. The precise mechanism as to why the latter equality is valid will be given in the remark after the presentation of this algorithm.

Step 1c. Compute

$$(s_0^{k+1}, \ldots, s_N^{k+1}) \;=\; \mathrm{argmin}_{s_0,\ldots,s_N}\Big\{\mathcal{L}_\sigma((y_0^k, \bar{y}_1^k, \ldots, \bar{y}_N^k), \bar{w}^k, (s_0, s_1, \ldots, s_N), z^k; x^k)\Big\},$$

which can be done in parallel by computing for $i = 0, 1, \ldots, N$,

$$\begin{aligned} s_i^{k+1} \;&=\; \mathrm{argmin}_{y_i}\Big\{\theta_i^*(-s_i) + \frac{\sigma}{2}\| - \mathcal{Q}_i\bar{w}_i^k + \mathcal{D}_i^*\bar{y}_i^k + s_i + g_i^k\|^2\Big\} \\ &=\; -\mathrm{Prox}_{\theta_i^*/\sigma}(-\mathcal{Q}_i\bar{w}_i^k + \mathcal{D}_i^*\bar{y}_i^k + g_i^k) \\ &=\; \frac{1}{\sigma}\mathrm{Prox}_{\sigma\theta_i}\big(\sigma(-\mathcal{Q}_i\bar{w}_i^k + \mathcal{D}_i^*\bar{y}_i^k + g_i^k)\big) - (-\mathcal{Q}_i\bar{w}_i^k + \mathcal{D}_i^*\bar{y}_i^k + g_i^k). \end{aligned}$$

Step 1d Compute $w^{k+1} = \text{argmin}\big\{\mathcal{L}_\sigma\big((y_0^k, \bar{y}_1^k, \ldots, \bar{y}_N^k), w, s^{k+1}, z^k; x^k\big)\big\}$ by computing in parallel for $i = 0, 1, \ldots, N$,

$$w_i^{k+1} \;=\; \text{argmin}_{w_i} \left\{ \begin{array}{l} \frac{1}{2}\langle w_i,\, \mathcal{Q}_i w_i\rangle \\ +\frac{\sigma}{2}\| - \mathcal{Q}_i w_i + \mathcal{D}_i^* \bar{y}_i^k + s_i^{k+1} + g_i^k\|^2 \end{array} \Big| w_i \in \text{Range}(\mathcal{Q}_i) \right\}.$$

Note that the same remark in Step 1b is applicable here.

Step 1e Compute

$$(y_1^{k+1}, \ldots, y_N^{k+1}) \;=\; \text{argmin}_{y_1,\ldots,y_N}\Big\{\mathcal{L}_\sigma\big((y_0^k, y_1, \ldots, y_N), w^{k+1}, s^{k+1}, z^k; x^k\big)\Big\},$$

which can be done in parallel by computing for $i = 1, \ldots, N$,

$$y_i^{k+1} \;=\; \text{argmin}_{y_i}\Big\{ -\langle b_i,\, y_i\rangle + \frac{\sigma}{2}\| - \mathcal{Q}_i w_i^{k+1} + \mathcal{D}_i^* y_i + s_i^{k+1} + g_i^k\|^2\Big\}.$$

Step 2a. Let $\boldsymbol{h}^k = -\mathcal{Q}w^{k+1} + \mathcal{D}^* y^{k+1} + s^{k+1} - c + \sigma^{-1}x^k$. Compute

$$\begin{aligned} \bar{y}_0^k \;&=\; \text{argmin}_{y_0}\Big\{\mathcal{L}_\sigma\big((y_0, y_1^{k+1}, \ldots, y_N^{k+1}), w^{k+1}, s^{k+1}, z^k; x^k\big)\Big\} \\ &=\; \text{argmin}_{y_0}\Big\{ -\langle b_0,\, y_0\rangle + \frac{\sigma}{2}\|\mathcal{A}_0^* y_0 + z_0^k + h_0^k\|^2 + \sum_{i=1}^N \frac{\sigma}{2}\|\mathcal{A}_i^* y_0 + z_i^k + h_i^k\|^2\Big\}. \end{aligned}$$

Specifically, $\bar{y}_0^k$ is the solution to the following linear system of equations:

$$\Big(\sum_{i=0}^N \mathcal{A}_i\mathcal{A}_i^*\Big) y_0 \;=\; \sigma^{-1}b_0 - \sum_{i=0}^N \mathcal{A}_i(z_i^k + h_i^k). \tag{4.30}$$

Step 2b Compute $z^{k+1} = \text{argmin}\big\{\mathcal{L}_\sigma\big((\bar{y}_0^k, y_1^{k+1}, \ldots, y_N^{k+1}), w^{k+1}, s^{k+1}, z; x^k\big)\big\}$ by computing in parallel for $i = 0, 1, \ldots, N$,

$$\begin{aligned} z_i^{k+1} \;&=\; \text{argmin}_{z_i}\Big\{\delta_{\mathcal{K}_i}^*(-z_i) + \frac{\sigma}{2}\|\mathcal{A}_i^*\bar{y}_0^k + z_i + h_i^k\|^2\Big\} \\ &=\; -\text{Prox}_{\sigma^{-1}\delta_{\mathcal{K}_i}^*}(\mathcal{A}_i^*\bar{y}_0^k + h_i^k) \\ &=\; \frac{1}{\sigma}\Pi_{\mathcal{K}_i}\big(\sigma(\mathcal{A}_i^*\bar{y}_0^k + h_i^k)\big) - (\mathcal{A}_i^*\bar{y}_0^k + h_i^k). \end{aligned}$$

Step 2c Compute

$$y_0^{k+1} = \mathrm{argmin}_{y_0}\Big\{\mathcal{L}_\sigma\big((y_0, y_1^{k+1}, \ldots, y_N^{k+1}), w^{k+1}, s^{k+1}, z^{k+1}; x^k\big)\Big\}$$
$$= \mathrm{argmin}_{y_0}\Big\{ -\langle b_0,\, y_0\rangle + \frac{\sigma}{2}\|\mathcal{A}_0^* y_0 + z_0^{k+1} + h_0^k\|^2 + \sum_{i=1}^{N}\frac{\sigma}{2}\|\mathcal{A}_i^* y_0 + z_i^{k+1} + h_i^k\|^2\Big\}.$$

Note that the computation in Step 2a is applicable here.

Step 3 Compute

$$x^{k+1} = x^k + \tau\sigma(-\mathcal{Q}w^{k+1} + \mathcal{B}^* y^{k+1} + s^{k+1} + z^{k+1} - c),$$

where $\tau \in (0, \frac{1+\sqrt{5}}{2})$ is the steplength.

Now we make some important remarks concerning the computations in sGS-ADMM.

1. If the term $\boldsymbol{\theta} \equiv 0$ in Step 1c, then this step is vacuous, and Step 1b and Step 1d are identical. Hence the computation needs only to be done for Step 1d. Hence Step 1 only consists of Step 1a, 1d, and 1e.

2. If $\mathcal{Q} \equiv 0$, then Step 1b and 1d are vacuous. Therefore Step 1 only consists of Step 1a, 1c, and 1e.

3. The computation in Step 1d can be omitted if the quantity $\bar{w}_i^k$ computed in Step 1b is already a sufficiently good approximate solution to the current subproblem. More precisely, if the approximation $\bar{w}_i^k$ for w_i^{k+1} satisfies the admissible accuracy condition required in the inexact sGS-ADMM designed in Chen et al. (2017), then we can just set $w_i^{k+1} = \bar{w}_i^k$ instead of using the exact solution to the current subproblem. Similar remark is also applicable to the computation in Step 1e and Step 2c.

4. The sGS-ADMM in fact has the flexibility of allowing for inexact computations as already shown in Chen et al. (2017). While the computation in Step 1a and 1e (similarly for Step 1b and 1d, Step 2a and 2c) are assumed to be done exactly (up to machine precision), the computation can in fact

be done inexactly subject to a certain predefined accuracy requirement on the computed approximate solution. Thus iterative methods such as the preconditioned conjugate gradient (PCG) method can be used to solve the linear systems when their dimensions are too large. We omit the details here for the sake of brevity.

5. In solving the linear system (4.30), the $m_0 \times m_0$ symmetric positive definite matrix $\sum_{i=0}^{N} \mathcal{A}_i \mathcal{A}_i^*$ is fixed, and one can pre-compute the matrix if it can be stored in the memory and its Cholesky factorization can be computed at a reasonable cost. Then in each sGS-ADMM iteration, $\bar{y}_0^k$ and y_0^{k+1} can be computed cheaply by solving triangular linear systems. In the event when computing the coefficient matrix or its Cholesky factorization is out of reach, one can use a PCG method to solve the linear system. In that case, one can implement the computation of the matrix-vector product in parallel by computing $\mathcal{A}_i \mathcal{A}_i^* y_0$ in parallel for $i = 0, 1, \ldots, N$, given any y_0. Note that when the PCG method is employed, the use of the inexact sGS-ADMM framework just mentioned above will become necessary.

   The same remark above also applies to the linear system (4.28) for each $i = 1, \ldots, N$.

   For the multi-commodity flow problem which we will consider later in the numerical experiments, we note that the linear system in (4.30) has a very simple coefficient matrix given by $\sum_{i=0}^{N} \mathcal{A}_i \mathcal{A}_i^* = (N+1) I_m$, and the coefficient matrix $\mathcal{D}_i \mathcal{D}_i^*$ in (4.28) is equal to the Laplacian matrix of the network graph for all $i = 1, \ldots, N$. Thus both (4.30) and (4.28) can be solved efficiently by a direct solver.

6. In Step 1b, we claimed that $\mathcal{Q}_i \bar{w}_i^k = \mathcal{Q}_i \tilde{w}_i^k$. Here we show why the result holds. For simplicity, we assume that $\mathcal{Q}_i$ is a symmetric positive semidefinite matrix rather than a linear operator. Consider the spectral decomposition $\mathcal{Q}_i = UDU^T$, where $D \in \Re^{r \times r}$ is a diagonal matrix whose diagonal elements are the positive eigenvalues of $\mathcal{Q}_i$ and the columns of $U \in \Re^{m_i \times r}$ are their corresponding orthonormal set of eigenvectors. We let $V \in \Re^{m_i \times (m_i - r)}$ be the matrix whose columns form an orthonormal set

of eigvectors of $\mathcal{Q}_i$ correspond to the zero eigenvalues. With this decomposition and the parameterization $w_i = U\xi$ (because $w_i \in \mathrm{Range}(\mathcal{Q}_i)$), the minimization for $\bar{w}_i^k$ is equivalent to the following:

$$\mathrm{argmin}\Big\{\frac{1}{2}\langle \xi,\, D\xi\rangle + \frac{\sigma}{2}\|D\xi - U^T g\|^2 + \frac{\sigma}{2}\|V^T g\|^2 \mid \xi \in \Re^r\Big\}, \qquad (4.31)$$

where we have set $g = z_i^k + h_i^k$ for convenience. Now from solving (4.29), we get that

$$(I + \sigma D)U^T\tilde{w}_i^k = \sigma U^T g, \quad V^T\tilde{w}_i^k = \sigma V^T g.$$

This show that $U^T\tilde{w}_i^k$ is the unique solution to the problem (4.31). Hence $\bar{w}_i^k = U(U^T\tilde{w}_i^k)$ is the unique solution to (4.29). From here, we have that $\mathcal{Q}_i\bar{w}_i^k = UDU^T(UU^T\tilde{w}_i^k) = UDU^T\tilde{w}_i^k = \mathcal{Q}_i\tilde{w}_i^k$.

4.3.1 Convergence theorems of sGS-ADMM

The convergence theorem of sGS-ADMM can be established directly by using known results from Chen et al. (2017) and Zhang et al. (2018). Here we present the global convergence result and the linear rate of convergence for the convenience of reader.

In order to state the convergence theorems, we need some definitions.

Definition 4.3.1. Let $\mathbb{F} : \mathcal{X} \rightrightarrows \mathcal{Y}$ be a multivalued mapping and denote its inverse by $\mathbb{F}^{-1}$. The graph of multivalued function $\mathbb{F}$ is defined by $\mathrm{gph}\mathbb{F} := \{(x, y) \in \mathcal{X} \times \mathcal{Y} \mid y \in \mathbb{F}(x)\}$.

Denote $u := (y, w, s, z, x) \in \mathcal{U} := \boldsymbol{\mathcal{Y}} \times \mathcal{W} \times \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}}$. The KKT mapping $\mathcal{R} : \mathcal{U} \to \mathcal{U}$ of (4.4) is defined by

$$\mathcal{R}(u) := \begin{pmatrix} \mathcal{B}x - b \\ -\mathcal{Q}w + \mathcal{B}^* y + s + z - c \\ \mathcal{Q}w - \mathcal{Q}x \\ x - \mathrm{Prox}_{\boldsymbol{\theta}}(x - s) \\ x - \Pi_{\boldsymbol{\mathcal{K}}}(x - z) \end{pmatrix}. \qquad (4.32)$$

Denote the set of KKT points by $\bar{\Omega}$. The KKT mapping $\mathcal{R}$ is said to be metrically subregular at $(\bar{u}, 0) \in \text{gph}\mathcal{R}$ with modulus $\eta > 0$ if there exists a scalar $\rho > 0$ such that

$$\text{dist}(u, \bar{\Omega}) \leq \eta \|\mathcal{R}(u)\| \quad \forall u \in \{u \in \mathcal{U} : \|u - \bar{u}\| \leq \rho\}.$$

Now we are ready to present the convergence theorem of sGS-ADMM.

Theorem 4.2. Let $\{u^k := (y^k, w^k, s^k, z^k; x^k)\}$ be the sequence generated by sGS-ADMM. Then, we have the following results.

(a) The sequence $\{(y^k, w^k, s^k, z^k)\}$ converges to an optimal solution of the compact form (4.6) of the dual problem (PBA-D), and the sequence $\{x^k\}$ converges to an optimal solution of the compact form (4.4) of the primal problem (PBA).

(b) Suppose that the sequence $\{u^k\}$ converges to a KKT point $\bar{u} := (\bar{y}^k, \bar{w}^k, \bar{s}^k, \bar{z}^k, \bar{x}^k)$ and the KKT mapping $\mathcal{R}$ is metrically subregular at $(\bar{u}, 0) \in \text{gph}\mathcal{R}$. Then the sequence $\{u^k\}$ is linearly convergent to $\bar{u}$.

Proof. (a) The global convergence result follows from that in Chen et al. (2017). (b) The result follows directly by applying the convergence result in (Zhang et al., 2018, Proposition 4.1) (which slightly improves an earlier result in Han et al. (2017)) to the compact formulation (4.6) of (PBA-D). □

Remark 3. By Theorem 1 and Remark 1 in Li et al. (2018b), we know that when (PBA-P) is a convex programming problem where for each $i = 0, \ldots, N$, θ_i is piecewise linear-quadratic or strongly convex, and $\mathcal{K}_i$ is polyhedral, then $\mathcal{R}$ is metrically subregular at $(\bar{u}, 0) \in \text{gph}\mathcal{R}$ for any KKT point $\bar{u}$. Thus sGS-ADMM converges locally at a linear rate to an optimal solution of (PBA-P) and (PBA-D) under the previous conditions on θ_i and $\mathcal{K}_i$. In particular, for the special case of a primal block angular quadratic programming problem where $\theta_i \equiv 0$ and $\mathcal{K}_i = \mathbb{R}^{n_i}_+$ for all i, we know that sGS-ADMM is locally linearly convergent, which can even be proven to converge globally linearly.

4.3.2 Computational cost

Now we would discuss the main computational cost of sGS-ADMM. We could observe that the most time-consuming computations are in solving large linear system of equations in Step 1a, 1b, 1d, 1e, 2a, and 2c.

In general, suppose for every iteration we need to solve a $d \times d$ linear system of equations:

$$Mx = r. \tag{4.33}$$

Assuming that M is stored, then we can compute its Cholesky factorization at the cost of $O(d^3)$ operations, which needs only to be done once at the very beginning of the algorithm. After that, whenever we need to solve the equation, we just need to compute the right-hand-side vector r and solve two $d \times d$ triangular systems of linear equations at the cost of $O(d^2)$ operations.

We can roughly summarize the costs incurred in solving $Mx = r$ as follows:

(C_1) Cost for computing the coefficient matrix M (only once at the beginning of algorithm);

(C_2) Cost for computing Cholesky factorization of M (only once at the beginning of algorithm);

(C_3) Cost for computing right-hand-side vector r;

(C_4) Cost for solving two triangular systems of linear equations.

The computational cost C_1, C_2, C_3, C_4 above for each of the equations in Step 1a, 1b, 1d, 1e, 2a, and 2c are tabulated in Table 4.2.

Step	C_1 (once)	C_2 (once)	C_3 (each iteration)	C_4 (each iteration)
1a and 1e $(i = 1, \ldots, N)$	$O(m_i^2 n_i)$	$O(m_i^3)$	$O(n_i^2 + m_i n_i)$	$O(m_i^2)$
1b and 1d $(i = 1, \ldots, N)$	$O(n_i^2)$	$O(n_i^3)$	$O(m_i n_i)$	$O(n_i^2)$

2a and 2c	$O(m_0^2 n_0)$	$O(m_0^3)$	$O(m_0 n_0)$	$O(m_0^2)$

Table 4.2: Computational cost for solving the linear systems of equations in each of the steps.

4.4 Numerical experiments

In this section, we evaluate the performance of the algorithm we have designed for solving the problem (PBA). We conduct numerical experiments on three major types of primal block angular model, including linear, quadratic, and nonlinear problems. Apart from randomly generated datasets, we would demonstrate that our algorithms can be quite efficient in solving realistic problems encountered in the literature.

4.4.1 Stopping condition

Based on the optimality conditions in (4.8), we measure the accuracy of a computed solution by the following relative residuals:

$$\eta = \max\{\eta_P, \eta_D, \eta_Q, \eta_K, \eta_S\},$$

where

$$\begin{aligned} \eta_P &= \frac{\|\mathcal{B}x - b\|}{1 + \|b\|}, \quad \eta_D = \frac{\| - \mathcal{Q}w + \mathcal{B}^* y + s + z - c\|}{1 + \|c\|}, \quad \eta_Q = \frac{\|\mathcal{Q}w - \mathcal{Q}x\|}{1 + \|\mathcal{Q}\|}, \\ \eta_K &= \frac{\|x - \Pi_{\boldsymbol{\mathcal{K}}}(x - z)\|}{1 + \|x\| + \|z\|}, \quad \eta_S = \frac{\|x - \mathrm{Prox}_\theta(x - s)\|}{1 + \|x\| + \|s\|}. \end{aligned}$$

We terminate our algorithm when $\eta \leq 10^{-5}$.

4.4.2 Block angular problems with linear objective functions

In this subsection, we perform numerical experiments on minimization problems having linear objective functions and primal block angular constraints. Multicommodity flow (MCF) problems are one of the main representative in this

class of problems. It is a model to solve the routing problem of multiple commodities throughout a network from a set of supply nodes to a set of demand nodes. These problems usually exhibit primal block angular structures due to the network nature in the constraints.

Consider a connected network graph $(\mathcal{N}, \mathcal{E})$ with m nodes and $n = |\mathcal{E}|$ arcs for which N commodities must be transported through the network. We assume that each commodity has a single source-sink pair (s_k, t_k) and we are given the flow r_k that must be transported from s_k to t_k, for $k = 1, \ldots, N$. Let $M \in \mathbb{R}^{m \times |\mathcal{E}|}$ be node-arc incidence matrix of the graph. Then the MCF problem can be expressed in the form given in (PBA-D) with the following data:

$$\mathcal{K}_0 = \{x_0 \in \mathbb{R}^n \mid 0 \leq x_0 \leq u\}, \quad \mathcal{K}_i = \mathbb{R}^n_+, \;\; i = 1, \ldots, N,$$

$$\mathcal{Q}_i = 0, \;\; \theta_i(\cdot) = 0, \;\; \forall i = 0, 1, ..., N,$$

$$\mathcal{A}_0 = I_n, \;\; \mathcal{A}_i = -I_n, \;\; \forall i = 1, ..., N,$$

$$\mathcal{D}_1 = \mathcal{D}_2 = \cdots = \mathcal{D}_N = M \text{ is the node-arc incidence matrix.}$$

For this problem, x_i denotes the flow of the i-th commodity $(i = 1, \ldots, N)$ through the network, x_0 is the total flow, and u is a given upper bound vector on the total flow.

Description of datasets

Following Castro and Cuesta (2011), the datasets we used are as follows.

tripart and gridgen: These are five multicommodity instances obtained with the Tripart and Gridgen generators. They could be downloaded from `http://www-eio.upc.es/~jcastro/mmcnf_data.html`.

pds: The PDS problems come from a model of transporting patients away from a place of military conflict. It could be downloaded from `http://www.di.unipi.it/optimize/Data/MMCF.html#Pds`.

M{n}-{k}: These are the problems generated by the Mnetgen generator, which is one of the most famous random generator of Multicommodity Min Cost

Flow instances. Here n is the number of nodes in the network and k is the number of commodity. It could be downloaded from `http://www.di.unipi.it/optimize/Data/MMCF.html#MNetGen`.

Numerical results

In Table 4.3, we compare our sGS-ADMM algorithm against the solvers Gurobi and BlockIP. We should emphasize that Gurobi is a state-of-the-art solver for solving general linear and quadratic programming problems. Although it is not a specialized algorithm for primal block angular problems, it has been so powerful in solving sparse general linear and convex quadratic programming problems that it should be used as the benchmark for any newly developed algorithm. On the other hand, BlockIP (Castro, 2016) is an efficient interior-point algorithm specially designed for solving primal block angular problems, especially those arising from MCF problems. As reported in Castro (2016), it has been successful in solving many large scale instances of primal block angular LP and QP problems.

In the following numerical experiments, we employ Gurobi directly on the compact formulation (4.4). To be more specific, we input $\mathcal{B}$ as a general sparse matrix. The feasibility and objective gap tolerance is set to be 1e-5, and the number of threads is set to be 1. All the other parameters remain as default setting. Similarly for BlockIP, all the three tolerances (primal and dual feasibility, and relative objective gap) are set to be 1e-5 for consistency. Its maximum number of iteration is set to be 500.

				sGS-ADMM		Gurobi		BlockIP	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Iter	Time(s)	Iter	Time (s)	Iter	Time (s)
tripart1	2096 | 192	2096 | 2096	16	1981	3.01	5155	0.78	48	1.23
tripart2	8432 | 768	8432 | 8432	16	6771	51.65	42070	42.81	67	10.32
tripart3	16380 | 1200	16380 | 16380	20	5561	104.96	85390	189.37	81	48.70
tripart4	24815 | 1050	24815 | 24815	35	8581	343.32	246340	1685.50	115	139.36
gridgen1	3072 | 1025	3072 | 3072	320	7541	409.75	497709	8039.40	203	1589.04
pds15	1812 | 2125	7756 | 7756	11	2893	22.60	8545	1.01	81	12.19
pds30	3491 | 4223	16148 | 16148	11	4471	111.49	27645	4.79	110	51.66

				sGS-ADMM		Gurobi		BlockIP	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Iter	Time(s)	Iter	Time (s)	Iter	Time (s)
pds60	6778 | 8423	33388 | 33388	11	7719	465.06	70168	17.57	145	403.23
pds90	8777 | 12186	46161 | 46161	11	5315	479.59	100858	25.18	162	822.45
M64-64	405 | 64	511 | 511	64	1991	3.16	7601	0.77	51	0.85
M128-64	936 | 128	1171 | 1171	64	2601	7.32	18108	3.93	52	3.15
M128-128	979 | 128	1204 | 1204	128	3801	28.89	32736	7.24	127	11.75
M256-256	1802 | 256	2204 | 2204	256	6821	225.31	103561	18.53	97	89.92
M512-64	3853 | 512	4768 | 4768	64	2631	45.77	48235	8.76	72	48.44
M512-128	3882 | 512	4786 | 4786	128	3581	137.23	87659	17.96	97	144.77
M512-512	707 | 512	1797 | 1797	512	7021	373.58	199260	16.79	146	308.95

Table 4.3: Comparison of computational results between sGS-ADMM, Gurobi, and BlockIP for **linear** primal block angular problems. All the results are obtained using a **single thread**. 'Iter' under the column for Gurobi means the total number of simplex iterations.

From Table 4.3, we observe that Gurobi is the fastest to solve 11 out of 16 instances. Gurobi is extremely fast in solving the `pdsxx` and `Mxxx-xx` problems but have difficulty in solving `tripart4` and `gridgen1` efficiently. On the other hand, sGS-ADMM and BlockIP are highly efficient in solving the latter instances. On the other hand, BlockIP is the fastest when solving the `tripart2,3,4` instances while sGS-ADMM is the fastest in solving the `gridgen1` and `M512-128` instances.

Our sGS-ADMM solver outperforms Gurobi when the instance is both hard and huge, for example, `tripart4` and `gridgen1`. For the latter instance, it is in fact the fastest solver. Intuitively, we can expect sGS-ADMM to perform stablely as the scale of the data increases. We also noticed that BlockIP is quite sensitive to the practical setting of the upper bound on the unbounded variables. For example, setting "9e6" and "9e8" as the upper bounds for the unbounded variables can lead to a significant difference in the number of iterations.

4.4.3 Block angular problems with convex quadratic objective functions

In this subsection, we perform numerical experiments on optimization problems having convex quadratic objective functions and primal block angular constraints.

One of the main class of this type of problem is again from the multicommodity flow problem. Following Castro (2016), we add in the quadratic objective term, $\mathcal{Q}_i = 0.1I,\ \forall i = 0, ..., N$. The corresponding datasets start with a prefix `"qp-"`, including `tripart`, `gridgen` and `pds`.

Another main class of quadratic primal block angular problems arises in the field of statistical disclosure control. Castro (2005) studied the controlled tabular adjustment (CTA) to find a closest, perturbed, yet safe table given a three-dimensional table for which the content need to be protected. In particular, we have

$$\mathcal{Q}_i = I,\ \theta_i(\cdot) = 0,\ i = 0, \ldots, N,$$
$$\mathcal{A}_0 = I,\ \mathcal{A}_i = -I,\ \forall i = 1, ..., N,$$
$$\mathcal{D}_1 = \mathcal{D}_2 = \cdots = \mathcal{D}_N \text{ is a node-arc incidence matrix}$$

and $\mathcal{K}_i$ $(i = 0, 1, \ldots, N)$ is the same as in section 4.4.2.

Description of datasets

The datasets we used are as follows.

rand: These instances are randomly generated sparse problems. Here we generated two types of problems.

- Type 1 problem (with suffix `-t1`) has diagonal quadratic objective cost, i.e. $\mathcal{Q}_i$ is a random diagonal matrix given by `spdiags(rand(n_i,1),0,n_i,n_i)`.
- Type 2 problem (with suffix `-t2`) does not necessarily have diagonal

quadratic objective cost. In this case $\mathcal{Q}_i$ is still very sparse but remained to be positive semidefinite. We use the following routine to generate $\mathcal{Q}_i$ for every $i = 0, 1, ..., N$:

`tmp=sprandn(n_i,n_i,0.1);` Q_i `= tmp*tmp'.`

For both types of problems, we generate $\mathcal{A}_i$ and $\mathcal{D}_i$ similarly for $i = 0, ..., N$ using MATLAB command `sprandn` with density 0.5 and 0.3 respectively. Note that by convention we have $\mathcal{D}_0 = 0$.

L2CTA3D: This is an extra large instance (with a total of 10M variables and 210K constraints) provided in `http://www-eio.upc.es/~jcastro/huge_sdc_3D.html`.

SDC: These are some of the CTA instances we generated using the generator provided by J. Castro at `http://www-eio.upc.es/~jcastro/CTA_3Dtables.html`.

Numerical results

As in the last subsection, we compare our sGS-ADMM algorithm against Gurobi and BlockIP solver in Table 4.4.

				sGS-ADMM		Gurobi		BlockIP	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Iter	Time(s)	Iter	Time (s)	Iter	Time (s)
qp-rand-m50-n80-N10-t1	50 | 50	80 | 80	10	421	0.64	14	0.28	29	0.13
qp-rand-m1000-n1500-N10-t1	1000 | 1000	1500 | 1500	10	748	57.67	15	1641.91	39	360.12
qp-rand-m100-n200-N100-t1	100 | 100	200 | 200	100	331	3.81	18	14.09	54	8.51
qp-rand-m1000-n1500-N100-t1	1000 | 1000	1500 | 1500	100	361	312.61	18	17175.81	/	/
qp-rand-m100-n200-N150-t1	100 | 100	200 | 200	150	341	6.56	19	20.94	58	60.17
qp-rand-m1000-n1500-N150-t1	1000 | 1000	1500 | 1500	150	448	559.20	17	36591.57	/	/
qp-rand-m10-n20-N10-t2	10 | 10	20 | 20	10	1501	1.20	14	0.25	*	*
qp-rand-m50-n80-N10-t2	50 | 50	80 | 80	10	141	0.20	14	0.44	*	*
qp-rand-m1000-n1500-N10-t2	1000 | 1000	1500 | 1500	10	131	50.81	12	6916.43	*	*
qp-rand-m100-n200-N100-t2	100 | 100	200 | 200	100	81	3.61	14	28.40	*	*
qp-rand-m1000-n1500-N100-t2	1000 | 1000	1500 | 1500	100	220	576.62	13	8823.43	*	*
qp-rand-m100-n200-N150-t2	100 | 100	200 | 200	150	74	5.36	15	45.91	*	*
qp-rand-m1000-n1500-N150-t2	1000 | 1000	1500 | 1500	150	252	930.81	13	15299.33	*	*

				sGS-ADMM		Gurobi		BlockIP	
Data	$m_0 \mid m_i$	$n_0 \mid n_i$	N	Iter	Time(s)	Iter	Time (s)	Iter	Time (s)
qp-tripart1	2096 | 192	2096 | 2096	16	653	1.44	15	1.17	24	0.22
qp-tripart2	8432 | 768	8432 | 8432	16	971	9.79	19	6.66	38	1.36
qp-tripart3	16380 | 1200	16380 | 16380	20	1034	27.09	22	30.08	55	6.78
qp-tripart4	24815 | 1050	24815 | 24815	35	5871	413.35	22	238.92	67	17.46
qp-gridgen1	3072 | 1025	3072 | 3072	320	4081	308.05	40	2143.19	208	1197.24
qp-pds15	1812 | 2125	7756 | 7756	11	1110	10.40	48	14.49	90	11.56
qp-pds30	3491 | 4223	16148 | 16148	11	1941	57.58	53	59.95	113	44.32
qp-pds60	6778 | 8423	33388 | 33388	11	4685	337.13	58	226.56	134	192.85
qp-pds90	8777 | 12186	46161 | 46161	11	3021	318.43	58	402.51	165	547.32
qp-L2CTA3D_100x100x1000_5000	110000 | 1000	0 | 100000	100	21	31.24	8	6696.47	7	22.72
qp-SDC-r100-c50-l100-p1000-t1	5000 | 150	0 | 5000	100	32	1.52	8	101.91	7	0.84
qp-SDC-r100-c50-l100-p1000-t2	5000 | 150	0 | 5000	100	31	1.34	6	96.47	6	0.80
qp-SDC-r100-c50-l100-p5000-t1	5000 | 150	0 | 5000	100	32	1.40	8	100.84	8	0.93
qp-SDC-r100-c50-l100-p5000-t2	5000 | 150	0 | 5000	100	31	1.37	6	106.82	6	0.79
qp-SDC-r100-c50-l100-p10000-t1	5000 | 150	0 | 5000	100	32	1.37	8	97.74	8	0.94
qp-SDC-r100-c50-l100-p10000-t2	5000 | 150	0 | 5000	100	31	1.35	6	102.74	6	0.77
qp-SDC-r100-c100-l100-p1000-t1	10000 | 200	0 | 10000	100	31	2.67	8	810.58	7	2.16
qp-SDC-r100-c100-l100-p1000-t2	10000 | 200	0 | 10000	100	31	2.67	6	1107.95	6	2.10
qp-SDC-r100-c100-l100-p5000-t1	10000 | 200	0 | 10000	100	31	2.70	8	1266.75	7	2.12
qp-SDC-r100-c100-l100-p5000-t2	10000 | 200	0 | 10000	100	31	2.63	6	751.24	6	2.02
qp-SDC-r100-c100-l100-p10000-t1	10000 | 200	0 | 10000	100	32	2.65	8	779.24	8	2.42
qp-SDC-r100-c100-l100-p10000-t2	10000 | 200	0 | 10000	100	31	2.63	6	810.03	6	1.98
qp-SDC-r100-c100-l200-p20000-t1	10000 | 200	0 | 10000	200	41	6.68	8	1418.31	8	4.87
qp-SDC-r200-c100-l200-p20000-t1	20000 | 300	0 | 20000	200	34	11.96	8	5194.45	8	9.47
qp-SDC-r200-c200-l200-p20000-t1	40000 | 400	0 | 40000	200	31	22.33	8	53964.31	7	23.34
qp-SDC-r500-c50-l500-p50000-t1	25000 | 550	0 | 25000	500	41	43.36	8	11025.98	8	24.56
qp-SDC-r500-c500-l50-p5000-t1	250000 | 1000	0 | 250000	50	20	27.04	8	11360.16	/	/

Table 4.4: Comparison of computational results between sGS-ADMM, Gurobi, and BlockIP for **quadratic** primal block angular problems. All the results are obtained using **single thread**. 'Iter' under the column for Gurobi means the total number of barrier iterations. A '/' under the column for BlockIP means that the solver runs out of memory, and a '*' means the solver is not compatible to solve the problem.

Table 4.4 shows that Gurobi is almost always slowest to solve the test instances in this case, whereas our sGS-ADMM performs almost as efficiently as BlockIP in solving these quadratic primal block angular problems. It is worth noting that our sGS-ADMM method works very well on the large scale randomly generated problems compared to BlockIP, because for these instances the matrices $\mathcal{A}_i$ and $\mathcal{Q}_i$ are no longer simple identity matrices for which the BlockIP solver can take special advantage of. Also, BlockIP runs out of memory for three of the huge instances `qp-rand-m1000-n1500-N100-t1`, `qp-rand-m1000-n1500-N150-t1` and `qp-SDC-r500-c500-l50-p5000-t1`.

It is also observed that BlockIP solver could not solve for the `qp-rand-xxx-t2` problem because it is not designed to cater for solving problems with nondiagonal quadratic objective cost. For these types of problem, our sGS-ADMM algorithm can substantially outperform Gurobi, sometimes by a factor of more than 10.

4.4.4 Block angular problems with nonlinear convex objective functions

In this subsection, we perform numerical experiments on optimization problems having nonlinear convex objective functions and primal block angular constraints. Nonlinear multicommodity flow problems usually arise in transportation and telecommunication. The two most commonly used nonlinear objective functions are:

$$h(t) = \begin{cases} \sum_{i=1}^{m} f_{\mathrm{Kr}}(t_i; \mathrm{cap}_i), & \text{known as Kleinrock function;} \\ \sum_{i=1}^{m} f_{\mathrm{BPR}}(t_i; \mathrm{cap}_i, r_i), & \text{known as BPR (Bureau of Public Roads) function,} \end{cases}$$

where

$$f_{\mathrm{Kr}}(\alpha; c) = \begin{cases} \frac{\alpha}{c-\alpha} & \text{if } 0 \le \alpha < c, \\ +\infty & \text{otherwise,} \end{cases} \qquad f_{\mathrm{BPR}}(\alpha; c, r) = \begin{cases} r\alpha[1 + B(\frac{\alpha}{c})^{\beta}] & \text{if } \alpha \ge 0, \\ +\infty & \text{otherwise.} \end{cases}$$

The Kleinrock function is normally used to model delay in a telecommunication problem; whereas the BPR function is mainly used to model congestion in a

transportation problem. Here cap_i is the capacity of arc i, r_i is the free flow time of arc i, and β, B are two positive parameters.

Thus in our problem setting, we have

$$\theta_0(x_0) = h(x_0),\ \theta_i(x_i) = 0,\ \forall\ i = 1, ..., N,$$

$$\mathcal{Q}_i = 0,\ c_i = 0,\ \forall\ i = 0, ..., N,$$

$$\mathcal{A}_0 = I,\ \mathcal{A}_i = -I,\ \forall\ i = 1, ..., N,$$

$\mathcal{D}_1 = \mathcal{D}_2 = \cdots = \mathcal{D}_N$ is a node-arc incidence matrix,

$b_0 = 0,\ b_i = d_i\ \forall i = 1, ..., N$ for some demand d_i for each commodity i,

$$\mathcal{K}_i = \begin{cases} [0, \text{cap}_i], & \text{for Kleinrock function;} \\ \mathbb{R}^{n_i}_+, & \text{for BPR function.} \end{cases}$$

Following Babonneau and Vial (2009), the datasets we used are the **planar** and **grid** problems, which could be downloaded from `http://www.di.unipi.it/optimize/Data/MMCF.html#Plnr`.

Remark 4. In Step 1c of the sGS-ADMM algorithm, we need to update s_i^{k+1} by

$$s_i^{k+1} \ = \ \frac{1}{\sigma}\text{Prox}_{\sigma\theta_i}\big(\sigma(-\mathcal{Q}_i\bar{w}_i^k + \mathcal{D}_i^*\bar{y}_i^k + g_i^k)\big) - \big(-\mathcal{Q}_i\bar{w}_i^k + \mathcal{D}_i^*\bar{y}_i^k + g_i^k\big) \quad i = 0, 1, \ldots, N.$$

To compute the proximal mapping for a given s:

$$\text{Prox}_{\sigma\theta_i}(s) = \arg\min\big\{g(t) := \sigma\theta_i(t) + \frac{1}{2}\|t - s\|^2\big\},$$

we can use Newton's method to solve the equation $\nabla g(t) = 0$. In each sGS-ADMM iteration, we warm-start Newton's method by using the quantity already computed in the previous iteration to generate s_i^k.

Another point to note is that although s_i^{k+1} is not computed exactly, the convergence of the sGS-ADMM algorithm is not affected as long as s_i^{k+1} is computed to satisfy the admissible accuracy condition required in each iteration of the inexact sGS-ADMM method developed in Chen et al. (2017).

Numerical results

In this subsection, we compare our sGS-ADMM algorithm against BlockIP and IPOPT. IPOPT is one of the state-of-the-art solvers for solving general nonlinear programs. We use the Kleinrock function as our objective function here.

				sGS-ADMM		BlockIP		IPOPT	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Iter	Time(s)	Iter	Time(s)	Iter	Time (s)
grid1	80 | 24	80 | 80	50	591	0.66	28	0.13	76	3.10
grid3	360 | 99	360 | 360	50	381	0.53	41	1.60	86	21.30
grid5	840 | 224	840 | 840	100	581	1.95	-	-	90	127.50
grid8	2400 | 624	2400 | 2400	500	4171	261.73	215	4568.28	51	5027.50
grid10	2400 | 624	2400 | 2400	2000	3432	893.98	221	36035.85	14	5340.39
planar30	150 | 29	150 | 150	92	431	0.44	93	1.59	90	7.55
planar80	440 | 79	440 | 440	543	1875	20.07	-	-	430	1400.91
planar100	532 | 99	532 | 532	1085	2614	70.99	-	-	117	1184.46

Table 4.5: Comparison of computational results between sGS-ADMM, BlockIP and IPOPT for nonlinear primal block angular problem. A '-' under the column for BlockIP means that the solver encounters memory issue.

Table 4.5 shows that IPOPT is almost always the slowest to solve the test instances but it is very robust in the sense that it is able to solve all the test instances to the required accuracy. It is not surprising for it to perform less efficiently since it is a general solver for nonlinear programs.

On the other hand, we observed that BlockIP runs into memory issue when solving almost half of the instances. This may be due to the fact that BlockIP uses a preconditioned conjugate gradient (PCG) method and Cholesky factorization to solve the linear systems arising in each iteration of the interior-point method. At some point of the iteration, the PCG method did not converge and the algorithm switches to use a Cholesky factorization to solve the linear system, which causes the out-of-memory error. Even when the PCG method works well, it might still converge in almost 10 times slower than our algorithm.

Acknowledgements

We would like to thank Professor Jordi Castro for sharing with us his solver BlockIP so that we are able to evaluate the performance of our algorithm more comprehensively. We are also grateful to him for providing us with the test instances he has taken great effort to generate over the years when developing BlockIP. Thanks also go to the Optimization Group at the Department of Computer Science of the University of Pisa for collecting/generating several suites of test data and making them publicly available.

Chapter 5

On proximal augmented Lagrangian based decomposition methods for dual block angular convex composite conic programming problems

Let $\mathcal{X}$ and $\mathcal{Y}$ be an n-dimensional and m-dimensional inner product spaces respectively. Similarly $\mathcal{X}_i$ and $\mathcal{Y}_i$ are n_i-dimensional and m_i-dimensional ($i = 1, \cdots, N$) inner product spaces respectively. We consider the following convex composite conic programming problem with the so called dual block angular structure (DBA):

$$\begin{aligned} \min \quad & \theta(x) + \langle c,\, x\rangle + \textstyle\sum_{i=1}^{N} \bar{\theta}_i(\bar{x}_i) + \langle \bar{c}_i,\, \bar{x}_i\rangle \\ \text{s.t.} \quad & \begin{bmatrix} A & & & & \\ B_1 & \bar{B}_1 & & & \\ B_2 & & \bar{B}_2 & & \\ \vdots & & & \ddots & \\ B_N & & & & \bar{B}_N \end{bmatrix} \begin{bmatrix} x \\ \bar{x}_1 \\ \vdots \\ \bar{x}_N \end{bmatrix} = \begin{bmatrix} b \\ \bar{b}_1 \\ \vdots \\ \bar{b}_N \end{bmatrix}, \\ & x \in \mathcal{K}, \quad \bar{x}_i \in \mathcal{K}_i \subset \mathcal{X}_i,\ \forall\, i = 1, \ldots, N, \end{aligned} \tag{5.1}$$

where $b \in \mathcal{Y}$, $c \in \mathcal{X}$, $\bar{b}_i \in \mathcal{Y}_i$ and $c_i \in \mathcal{X}_i$ $(i = 1, \ldots, N)$ are given data; $A : \mathcal{X} \to \mathcal{Y}$, $B_i : \mathcal{X} \to \mathcal{Y}_i$ and $\bar{B}_i : \mathcal{X}_i \to \mathcal{Y}_i$ are given linear maps; $\theta : \mathcal{X} \to (-\infty, \infty]$ and $\bar{\theta}_i : \mathcal{X}_i \to (-\infty, \infty]$ $(i = 1, \ldots, N)$ are given proper closed convex functions; $\mathcal{K} \subset \mathcal{X}$ and $\mathcal{K}_i \subset \mathcal{X}_i$ $(i = 1, \ldots, N)$ are given closed convex sets. In the above problem, we allow the case where A is absent.

Let $\bar{m} = \sum_{i=1}^N m_i$ and $\bar{n} = \sum_{i=1}^N n_i$ with n_i being the dimension of $\mathcal{X}_i$. Define

$$\begin{aligned}
&\bar{\mathcal{X}} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_N, \quad \bar{\mathcal{Y}} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_N, \quad \bar{\mathcal{K}} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_N, \\
&\bar{x} = [\bar{x}_1; \ldots; \bar{x}_N] \in \bar{\mathcal{X}}, \ \ \bar{c} = [\bar{c}_1; \ldots; \bar{c}_N] \in \bar{\mathcal{X}}, \ \ \bar{b} = [\bar{b}_1; \ldots; \bar{b}_N] \in \bar{\mathcal{Y}}, \\
&B = [B_1; \ldots; B_N], \ \ \bar{B} = \mathrm{diag}(\bar{B}_1, \ldots, \bar{B}_N), \quad \bar{\theta}(\bar{x}) = \textstyle\sum_{i=1}^N \bar{\theta}_i(\bar{x}_i). \qquad (5.2)
\end{aligned}$$

Note that the dimension of $\bar{\mathcal{X}}$ is $\bar{n}$ and that of $\bar{\mathcal{Y}}$ is $\bar{m}$. For convenience, we will sometimes refer to $\bar{B}_i$ as an $m_i \times n_i$ matrix to mean its matrix representation with respect to the standard basis in $\mathcal{X}_i$ and $\mathcal{Y}_i$. A similar convention may also be used for A and B_i.

We can rewrite (5.1) as an instance of the following general conic programming problem with a block angular structure:

$$\begin{array}{lllllll}
\text{(DBA-P)} & \min & \theta(x) + \langle c, x\rangle & + \bar{\theta}(\bar{x}) + \langle \bar{c}, \bar{x}\rangle & & \\
& \text{s.t.} & Ax & & = b, & x \in \mathcal{K}, \\
& & Bx & + \bar{B}\bar{x} & = \bar{b}, & \bar{x} \in \bar{\mathcal{K}},
\end{array}$$

where $A : \mathcal{X} \to \mathcal{Y}$, $B : \mathcal{X} \to \bar{\mathcal{Y}}$, $\bar{B} : \bar{\mathcal{X}} \to \bar{\mathcal{Y}}$. The above structure is often referred to in the literature as a dual block angular structure. We should mention that the problem (DBA-P) is very general. For example, it includes block angular convex quadratic conic programming problems as a special case when $\theta(\cdot)$ and $\bar{\theta}_i(\cdot)$ $(i = 1, \ldots, N)$ are convex quadratic functions.

We shall show later that the dual of (DBA-P) is given by

$$\begin{aligned}(\text{DBA-D}) \quad -\min \quad & \theta^*(-v) + \delta^*_{\mathcal{K}}(-z) - \langle b,\, y\rangle \; + \bar{\theta}^*(-\bar{v}) + \delta^*_{\bar{\mathcal{K}}}(-\bar{z}) - \langle \bar{b},\, \bar{y}\rangle \\ \text{s.t.} \quad & A^*y + B^*\bar{y} + z + v = c, \;\; z \in \mathcal{X},\; v \in \mathcal{X}, \\ & \bar{B}^*\bar{y} + \bar{z} + \bar{v} = \bar{c}, \;\; \bar{z} \in \bar{\mathcal{X}},\; \bar{v} \in \bar{\mathcal{X}},\end{aligned}$$

where $\bar{z} = [\bar{z}_1; \ldots; \bar{z}_N] \in \bar{\mathcal{X}}$, $\bar{v} = [\bar{v}_1; \ldots; \bar{v}_N] \in \bar{\mathcal{X}}$, $\bar{y} = [\bar{y}_1; \ldots; \bar{y}_N] \in \bar{\mathcal{Y}}$, $\mathcal{K}^*$ the dual cone of $\mathcal{K}$ and $\bar{\mathcal{K}}^* = \mathcal{K}_1^* \times \cdots \times \mathcal{K}_N^*$. Observe that the problem (DBA-D) has six blocks of variables $(y, \bar{y}, z, \bar{z}, v, \bar{v})$ coupled together by the linear constraints $A^*y + B^*\bar{y} + z + v = c$ and $\bar{B}^*\bar{y} + \bar{z} + \bar{v} = \bar{c}$.

We made the following assumption on the problem data throughout this chapter.

Assumption 1. $A \in \mathbb{R}^{m\times n}$ has full row rank (if it is present), and $[B, \bar{B}]$ has full row rank.

Assumption 2. The projection $\Pi_{\mathcal{K}}(x)$ and $\Pi_{\bar{\mathcal{K}}}(\bar{z})$ can be computed efficiently such as in the case when $\mathcal{K}$ is a nonnegative orthant, a second-order cone or $\mathbb{S}^n_+$ (the cone of symmetric positive semidefinite matrices). We also assume that the proximal mappings for θ and $\bar{\theta}$ can be computed analytically or very efficiently.

This chapter is organized as follows. We will elaborate some example classes of the conic programming problem with dual block angular structure in section 5.2. In section 5.3, we will derive the dual (DBA-D) of the primal problem (DBA-P). We present some preliminary in section 5.4 before we provide the algorithmic details of our inexact symmetric Gauss-Seidel proximal ADMM and ALM for solving (DBA-D) in section 5.5. In section 5.6, we discuss an extension where we incorporate a semismooth Newton-CG method in our proposed algorithm. Finally, we conduct numerical experiment in section 5.7 and conclude the chapter in the last section.

5.1 Example classes of (DBA)

In this section, we shall give three example classes of the dual block angular problems.

5.1.1 Examples from 2-stage stochastic conic programming problems

Here we show that the dual block angular structure shown in (5.1) naturally arises from a 2-stage stochastic conic programming problem. Consider the following 2-stage stochastic optimization problem:

$$\min_{x\in\mathcal{X}} \Big\{ \theta(x) + \langle c,\, x\rangle + E_\xi\, Q(x;\xi) \mid Ax \;=\; b,\; x \in \mathcal{K} \Big\}. \tag{5.3}$$

Here ξ is a random vector, $E_\xi(\cdot)$ denotes the expectation with respect to ξ, and

$$Q(x;\xi) \;=\; \min_{\tilde{x}\in\mathcal{X}_2} \Big\{ \tilde{\theta}_\xi(\tilde{x}) + \langle \tilde{c}_\xi,\, \tilde{x}\rangle \;:\; \tilde{B}_\xi\tilde{x} = \tilde{b}_\xi - B_\xi x,\; \tilde{x} \in \mathcal{K}_1 \Big\},$$

where for each given scenario ξ, $\tilde{c}_\xi \in \mathcal{X}_1$, $\tilde{b}_\xi \in \mathcal{Y}_1$ are given data, $B_\xi : \mathcal{X} \to \mathcal{Y}_1$, $\tilde{B}_\xi : \mathcal{X}_1 \to \mathcal{Y}_1$ are given linear maps, and $\mathcal{K}_1 \subset \mathcal{X}_1$ is a given closed convex set, $\tilde{\theta}_\xi : \mathcal{X}_1 \to (-\infty, \infty]$ is a given proper closed convex function. Note that there is no loss of generality in considering only equality constraints in (5.3) since constraints of the form $b - Ax \in \mathcal{Q}$, $x \in \mathcal{K}$ can always be reformulated as $[A, I](x; s) = b$, $(x; s) \in \mathcal{K} \times \mathcal{Q}$.

By sampling N scenarios for ξ, one may approximate $E_\xi Q(x;\xi)$ by the sample mean $\sum_{i=1}^N p_i Q(x;\xi_i)$, with p_i being the probability of occurrence of the ith scenario, and hence approximately solve (5.3) via the following deterministic optimization problem which has exactly the same form as (5.1):

$$\min \left\{ \begin{array}{l|l} \theta(x) + \langle c,\, x\rangle & Ax = b,\; x \in \mathcal{K} \\ + \sum_{i=1}^N \bar{\theta}_i(\bar{x}_i) + \langle \bar{c}_i,\, \bar{x}_i\rangle & B_i x + \bar{B}_i \bar{y}_i = \bar{b}_i,\; \bar{y}_i \in \mathcal{K}_1,\; \forall\, i = 1, \ldots, N \end{array} \right\}, \tag{5.4}$$

where $\bar{c}_i = p_i \tilde{c}_{\xi_i}$ and $\bar{\theta}_i = p_i \tilde{\theta}_{\xi_i}$, $B_i = B_{\xi_i}, \bar{B}_i = \tilde{B}_{\xi_i}, \bar{b}_i = \tilde{b}_{\xi_i}$ are the data, and $\bar{x}_i = \tilde{x}_{\xi_i} \in \mathcal{X}_1$ is the second stage decision variable associated with the ith

scenario.

5.1.2 Examples from doubly nonnegative relaxations of uncapacitated facility location problems

The uncapacitated facility location (UFL) problem is one of the most studied problems in operations research. Here we consider the general UFL with linear and/or separable convex quadratic allocation costs that was introduced in Günlük et al. (2007):

$$\begin{aligned} \text{(UFL)} \quad \min \quad & \textstyle\sum_{i=1}^{p} c_i u_i + \sum_{i=1}^{p} \sum_{j=1}^{q} c_{ij} s_{ij} + \frac{1}{2} q_{ij} s_{ij}^2 \\ \text{s.t} \quad & \textstyle\sum_{i=1}^{n} s_{ij} = 1, \ \forall\, j = 1, \ldots, q; \\ & s_{ij} \le u_i, \ \forall\, i = 1, \ldots, p, \ j = 1, \ldots, q; \\ & s_{ij} \ge 0, \ \forall\, i = 1, \ldots, p, \ j = 1, \ldots, q; \\ & u_i \in \{0, 1\} \ \forall\, i = 1, \ldots, p, \end{aligned} \tag{5.5}$$

where $c_i \ge 0$ for all i, and $c_{ij}, q_{ij} \ge 0$ for all i, j are given data. Observe that the allocation cost for customer j is a convex quadratic function of his demand s_{ij} served by the opened facility i.

Let $U = uu^T$. One can see that the constraint $u_i \in \{0, 1\}$ is equivalent to the constraint that $u_i^2 = U_{ii} = u_i$. Also, by introducing the nonnegative slack variable z_{ij}, we can convert the inequality constraint $s_{ij} \le u_i$ to the equality constraint $s_{ij} + z_{ij} = u_i$.

Let $S = (s_{ij}) \in \mathbb{R}^{p \times q}$ and $Z = (z_{ij}) \in \mathbb{R}^{p \times q}$. Also let $e \in \mathbb{R}^p$ be the vector of all ones and S_j be the jth column of S. Then one can express the problem (5.5) equivalently as follows:

$$\begin{aligned} \min \quad & \textstyle\sum_{i=1}^{p} c_i u_i + \sum_{i=1}^{p} \sum_{j=1}^{q} c_{ij} s_{ij} + \frac{1}{2} q_{ij} s_{ij}^2 \\ \text{s.t} \quad & e^T S_j = 1, \ \ \forall\, j = 1, \ldots, q; \\ & S_j + Z_j = u, \ \ S_j, Z_j \ge 0, \ \ \forall\, j = 1, \ldots, q; \\ & u - \operatorname{diag}(U) = 0, \quad U = uu^T, \ u \ge 0, \ U \in \mathbb{S}^p. \end{aligned}$$

Let $C = (c_{ij}) \in \mathbb{R}^{p\times q}$ and $Q = (q_{ij}) \in \mathbb{R}^{p\times q}$. By relaxing the rank-1 constraint $U = uu^T$ to $U \succeq uu^T$ and using the equivalence that $U \succeq uu^T$ if and only if $[1,\ u^T; u,\ U] \succeq 0$, we get the following doubly nonnegative (DNN) relaxation of (5.5):

$$\begin{array}{ll} \min & \langle [0; c],\, [\alpha; u]\rangle + \sum_{j=1}^{q} \langle C_j,\, S_j\rangle + \frac{1}{2}\langle S_j,\, \mathrm{diag}(Q_j)S_j\rangle \\ \text{s.t} & \left[\begin{array}{c} 0 \\ u \end{array}\right] + \left[\begin{array}{cc} e^T & 0^T \\ -I_p & -I_p \end{array}\right] \left[\begin{array}{c} S_j \\ Z_j \end{array}\right] = \left[\begin{array}{c} 1 \\ 0 \end{array}\right], \quad j = 1, \dots q; \\ & u - \mathrm{diag}(U) = 0, \ \ \alpha = 1, \\ & \boldsymbol{U} := \left[\begin{array}{cc} \alpha & u^T \\ u & U \end{array}\right] \in \mathbb{S}^{1+p}_{+}, \ \ \boldsymbol{U} \geq 0, \ \ S_j, Z_j \geq 0,\ j = 1, \dots, q. \end{array} \tag{5.6}$$

Now we can express the above DNN programming problem in the form (5.1) by defining the following inner product spaces, functions, cones and linear maps:

$$\mathcal{X} = \mathbb{S}^{1+p}, \ \ \mathcal{Y} = \mathbb{R}^{1+p}, \ \ \mathcal{X}_j = \mathbb{R}^{2q}, \ \ \mathcal{Y}_j = \mathbb{R}^{1+p},\ j = 1, \dots, q,$$

$$\theta(\boldsymbol{U}) = \delta_{\mathcal{N}}(\boldsymbol{U}),\ \mathcal{K} = \mathbb{S}^{1+p}_{+},$$

$$\bar{\theta}_j([S_j; Z_j]) = \frac{1}{2}\langle S_j,\, \mathrm{diag}(Q_j)S_j\rangle,\ \mathcal{K}_j = \mathbb{R}^{2q}_{+},\ j = 1, \dots, q,$$

$$b = \left[\begin{array}{c} 1 \\ 0 \end{array}\right] \in \mathbb{R}^{1+p}, \ \ A(\boldsymbol{U}) = \left[\begin{array}{c} \alpha \\ u - \mathrm{diag}(U) \end{array}\right] = \left[\begin{array}{c} \langle A_0,\, \boldsymbol{U}\rangle \\ \langle A_1,\, \boldsymbol{U}\rangle \\ \vdots \\ \langle A_p,\, \boldsymbol{U}\rangle \end{array}\right],$$

$$B_j(\boldsymbol{U}) = \left[\begin{array}{c} 0 \\ u \end{array}\right] = \left[\begin{array}{c} \langle O,\, \boldsymbol{U}\rangle \\ \langle E_1,\, \boldsymbol{U}\rangle \\ \vdots \\ \langle E_p,\, \boldsymbol{U}\rangle \end{array}\right], \ \ \bar{B}_j = \left[\begin{array}{cc} e^T & 0^T \\ -I_p & -I_p \end{array}\right] \in \mathbb{R}^{(1+p)\times 2q},$$

$$\bar{b}_j = \left[\begin{array}{c} 1 \\ 0 \end{array}\right] \in \mathbb{R}^{1+p}, \ \ j = 1, \dots, q,$$

where $\mathcal{N} = \{X \in \mathbb{S}^{1+p} \mid X \geq 0\}$ and

$$A_0 = \begin{bmatrix} 1 & 0^T \\ 0 & 0_{p\times p} \end{bmatrix}, \quad A_i = \begin{bmatrix} 0 & \frac{1}{2}e_i^T \\ \frac{1}{2}e_i & -e_ie_i^T \end{bmatrix}, \quad E_i = \begin{bmatrix} 0 & \frac{1}{2}e_i^T \\ \frac{1}{2}e_i & 0_{p\times p} \end{bmatrix}, \quad i = 1, \ldots, p.$$

In the above, e_i is the ith unit vector in $\mathbb{R}^p$. Observe that for the problem (5.6), we have

$$B_1 = \cdots = B_q, \quad \bar{B}_1 = \cdots = \bar{B}_q.$$

Thus given any $\boldsymbol{U} \in \mathcal{X}$ and $\bar{y} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_q$, the evaluation of $B(\boldsymbol{U})$ and $B^*\bar{y}$ can be done very efficiently since $B(\boldsymbol{U}) = [B_1(\boldsymbol{U})]_{j=1}^q$ and $B^*\bar{y} = B_1^*(\bar{y}_1 + \cdots + \bar{y}_q)$. For later purpose, we note that $\bar{B}_j\bar{B}_j^*$ has a very simple inverse given by

$$(\bar{B}_j\bar{B}_j^*)^{-1} = \begin{bmatrix} 0 & 0^T \\ 0 & \frac{1}{2}I_p \end{bmatrix} + \frac{1}{2p}\begin{bmatrix} 2 \\ e \end{bmatrix}[2,\ e^T], \quad j = 1, \ldots, q. \tag{5.7}$$

5.1.3 Computation of a Wasserstein barycenter for a family of discrete distributions

Consider a family of discrete distributions $\{(s_j^{(i)}, p_j^{(i)}) \mid j = 1, \ldots, m_i\},\ i = 1, \ldots, N$, such that the probability of the ith distribution taking the value $s_j^{(i)} \in \mathbb{R}^d$ is given by $p_j^{(i)}$ for $j = 1, \ldots, m_i$. The Wasserstein barycenter for the family is the distribution $\{(s_j, p_j) \mid j = 1, \ldots, m\}$ that solves the following minimization problem:

$$\min \left\{ \sum_{i=1}^N \langle M^{(i)},\ X^{(i)} \rangle \;\middle|\; \begin{array}{c} X^{(i)}e^{(i)} = p,\ (X^{(i)})^T e = p^{(i)},\ p \in \Delta, \\ X^{(i)} \in \mathbb{R}^{m\times m_i},\ X^{(i)} \geq 0,\ i = 1, \ldots, N \end{array} \right\}, \tag{5.8}$$

where $\Delta = \{p \in \mathbb{R}^m \mid e^T p = 1,\ p \geq 0\}$, e is the vector of all ones in $\mathbb{R}^m$, and $e^{(i)}$ is the vector of all ones in $\mathbb{R}^{m_i}$, $i = 1, \ldots, N$. In the above problem, $M^{(i)} \in \mathbb{R}^{m\times m_i}$ is defined by $M_{kj}^{(i)} = \|s_k - s_j^{(i)}\|^2$. In (5.8), we consider the version of the problem where the support vectors $x_1, \ldots, x_m$ of the centroid are given. (Note that in practice, one may cluster the support vectors $\cup_{i=1}^N \{s_j^{(i)} \mid j = 1, \ldots, m_i\}$ into m clusters and the vectors s_j may be taken to be the center of mass of the $j$the cluster.)

Next, we show that the problem (5.8) has a dual block angular structure. Define

$$\mathcal{X} = \mathbb{R}^m,\ \mathcal{Y} = \mathbb{R}^m,\ \mathcal{K} = \Delta,\ \ B_i = I_m,\ i = 1, \ldots, N,$$

$$\mathcal{X}_i = \mathbb{R}^{m \times m_i},\ \mathcal{Y}_i = \mathbb{R}^m,\ \mathcal{K}_i = \{X^{(i)} \in \mathcal{X}_i \mid (X^{(i)})^T e = p^{(i)},\ X^{(i)} \geq 0\},$$

$$\bar{B}_i : \mathcal{X}_i \to \mathcal{Y}_i \quad \text{such that} \quad \bar{B}_i(X^{(i)}) = -X^{(i)} e^{(i)},\ \ i = 1, \ldots, N.$$

Then we can express (5.8) as the following block angular problem:

$$\begin{array}{ll} \min & \delta_{\mathcal{K}}(p) + \sum_{i=1}^N \langle M^{(i)},\, X^{(i)} \rangle + \delta_{\mathcal{K}_i}(X^{(i)}) \\ \text{s.t.} & \begin{bmatrix} B_1 \\ \vdots \\ B_N \end{bmatrix} p + \begin{bmatrix} \bar{B}_1(X^{(1)}) \\ \vdots \\ \bar{B}_N(X^{(N)}) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}. \end{array} \tag{5.9}$$

Observe that for the above problem, we have that $B_1 = \ldots = B_N = I_m$, $\bar{B}_i^*(\bar{y}_i) = -\bar{y}_i (e^{(i)})^T$, and

$$(\bar{B}_i \bar{B}_i^*)^{-1} = \frac{1}{m_i} I_m, \quad i = 1, \ldots, N.$$

Note that for this problem, the projection onto $\mathcal{K}$ can be computed analytically. Similarly, the projection onto $\mathcal{K}_i$ can also be computed analytically as follows. Given $G \in \mathbb{R}^{m \times m_i}$,

$$\begin{aligned} \Pi_{\mathcal{K}_i}(G) &= \operatorname{argmin}\Big\{ \frac{1}{2}\|X - G\|^2 \mid X^T e = p^{(i)}, X \geq 0 \Big\} \\ &= \operatorname{argmin}\Big\{ \sum_{j=1}^{m_i} \frac{1}{2}\|X_j - G_j\|^2 \mid e^T X_j = p_j^{(i)},\ X_j \in \mathbb{R}^m_+,\ j = 1, \ldots, m_i \Big\}, \end{aligned}$$

where X_j and G_j denote the jth column of X and G, respectively. Thus the jth column of $\Pi_{\mathcal{K}_i}(G)$ can be computed analytically for $j = 1, \ldots, m_i$.

5.2 The derivation of (DBA-D) and KKT conditions

In this section, we will provide the derivation of dual problem (DBA-D) of (DBA-P) in details. We introduce the auxiliary variables, $u = x$, $\bar{u} = \bar{x}$, $s = x$, $\bar{s} = \bar{x}$ and replace $\theta(x)$, $\bar{\theta}(\bar{x})$, $\delta_{\mathcal{K}}(x)$, $\delta_{\bar{\mathcal{K}}}(\bar{x})$ by $\theta(u)$, $\bar{\theta}(\bar{u})$, $\delta_{\mathcal{K}}(s)$, $\delta_{\bar{\mathcal{K}}}(\bar{s})$ respectively. Consider the Lagrangian function for (DBA-P):

$$\begin{aligned}
& \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z}) \\
:= \; & \theta(u) + \langle c,\, x\rangle + \bar{\theta}(\bar{u}) + \langle \bar{c},\, \bar{x}\rangle + \delta_{\mathcal{K}}(s) + \delta_{\bar{\mathcal{K}}}(\bar{s}) - \langle y,\, Ax - b\rangle - \langle \bar{y},\, Bx + \bar{B}\bar{x} - \bar{b}\rangle \\
& -\langle v,\, x - u\rangle - \langle \bar{v},\, \bar{x} - \bar{u}\rangle - \langle z,\, x - s\rangle - \langle \bar{z},\, \bar{x} - \bar{s}\rangle \\
= \; & \langle c - A^*y - B^*\bar{y} - v - z,\, x\rangle + \langle -\bar{B}^*\bar{y} - \bar{v} - \bar{z},\, \bar{x}\rangle + \theta(u) + \langle v,\, u\rangle + \bar{\theta}(\bar{u}) + \langle \bar{v},\, \bar{u}\rangle \\
& +\delta_{\mathcal{K}}(s) + \langle z,\, s\rangle + \delta_{\bar{\mathcal{K}}}(\bar{s}) + \langle \bar{z},\, \bar{s}\rangle + \langle b,\, y\rangle + \langle \bar{b},\, \bar{y}\rangle,
\end{aligned}$$

where $x, u, s, v, z \in \mathcal{X}$, $\bar{x}, \bar{u}, \bar{s}, \bar{v}, \bar{z} \in \bar{\mathcal{X}}$, $y \in \mathcal{Y}$ and $\bar{y} \in \bar{\mathcal{Y}}$.

We have the following:

$$\begin{aligned}
\inf_{x} \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z}) &= \inf_{x}\{\langle c - A^*y - B^*\bar{y} - v - z,\, x\rangle\} \\
&= \begin{cases} 0, & \text{if } c - A^*y - B^*\bar{y} - v - z = 0, \\ -\infty, & \text{otherwise;} \end{cases} \\
\inf_{\bar{x}} \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z}) &= \inf_{\bar{x}}\{\langle -\bar{B}^*\bar{y} - \bar{v} - \bar{z},\, \bar{x}\rangle\} \\
&= \begin{cases} 0, & \text{if } \bar{B}^*\bar{y} + \bar{v} + \bar{z} = 0, \\ -\infty, & \text{otherwise;} \end{cases} \\
\inf_{u} \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z}) &= \inf_{u}\{\theta(u) + \langle v,\, u\rangle\} = -\theta^*(-v); \\
\inf_{\bar{u}} \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z}) &= \inf_{\bar{u}}\{\bar{\theta}(\bar{u}) + \langle \bar{v},\, \bar{u}\rangle\} = -\bar{\theta}^*(-\bar{v}); \\
\inf_{s} \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z}) &= \inf_{u}\{\delta_{\mathcal{K}}(s) + \langle z,\, s\rangle\} = -\delta^*_{\mathcal{K}}(-z); \\
\inf_{\bar{s}} \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z}) &= \inf_{\bar{s}}\{\delta_{\bar{\mathcal{K}}}(\bar{s}) + \langle \bar{z},\, \bar{s}\rangle\} = -\delta^*_{\bar{\mathcal{K}}}(-\bar{z}).
\end{aligned}$$

Hence the dual (DBA-D) of (DBA-P) is given by

$$\max_{y, \bar{y}, v, \bar{v}, z, \bar{z}} \; \inf_{x, \bar{x}, u, \bar{u}, s, \bar{s}} \mathcal{L}(x, \bar{x}, u, \bar{u}, s, \bar{s}; y, \bar{y}, v, \bar{v}, z, \bar{z})$$

$$= \max_{y,\bar{y},v,\bar{v},z,\bar{z}} \left\{ \begin{array}{c|c} -\theta^*(-v) - \bar{\theta}^*(-\bar{v}) - \delta^*_{\mathcal{K}}(-z) & c - A^*y - B^*\bar{y} - v - z = 0, \\ -\delta^*_{\bar{\mathcal{K}}}(-\bar{z}) + \langle b,\, y\rangle + \langle \bar{b},\, \bar{y}\rangle & \bar{B}^*\bar{y} + \bar{v} + \bar{z} = 0 \end{array} \right\}.$$

Assume that the feasible regions of both the primal and dual problem has nonempty interiors. Then the optimal solutions for both problems exist. The Karush-Kuhn-Tucker (KKT) conditions for the problems (DBA-P) and (DBA-D) are given as follows:

$$\begin{aligned} Ax = b, \quad Bx + \bar{B}\bar{x} = \bar{b}, \\ A^*y + B^*\bar{y} + z + v = c, \quad \bar{B}^*\bar{y} + \bar{z} + \bar{v} = \bar{c}, \\ x = \Pi_{\mathcal{K}}(x - z), \quad \bar{x} = \Pi_{\bar{\mathcal{K}}}(\bar{x} - \bar{z}), \\ x = \mathrm{Prox}_{\theta}(x - v), \quad \bar{x} = \mathrm{Prox}_{\bar{\theta}}(\bar{x} - \bar{v}). \end{aligned} \tag{5.10}$$

5.3 Inexact symmetric Gauss-Seidel proximal ADMM and ALM for solving (DBA-D)

Now we are ready to discuss our proposed inexact symmetric Gauss-Seidel (sGS) proximal ADMM algorithm for solving (DBA-D). We shall also show that the algorithm reduced to sGS proximal augmented Lagrangian method (ALM) for a special case where the nonsmooth objective terms disappear.

Given a penalty parameter $\sigma > 0$, the augmented Lagrangian function associated with (DBA-D) is given as follows: for $(y, \bar{y}, z, \bar{z}, v, \bar{v}) \in \mathcal{Y}\times\bar{\mathcal{Y}}\times\mathcal{X}\times\bar{\mathcal{X}}\times\mathcal{X}\times\bar{\mathcal{X}}$, $(x, \bar{x}) \in \mathcal{X} \times \bar{\mathcal{X}}$,

$$\begin{aligned} & \mathcal{L}_\sigma(y, \bar{y}, z, \bar{z}, v, \bar{v}; x, \bar{x}) \\ = \;& \theta^*(-v) + \delta^*_{\mathcal{K}}(-z) - \langle b,\, y\rangle + \bar{\theta}^*(-\bar{v}) + \delta^*_{\bar{\mathcal{K}}}(-\bar{z}) - \langle \bar{b},\, \bar{y}\rangle - \frac{1}{2\sigma}\|x\|^2 - \frac{1}{2\sigma}\|\bar{x}\|^2 \\ & + \frac{\sigma}{2}\|A^*y + B^*\bar{y} + z + v - c + \sigma^{-1}x\|^2 + \frac{\sigma}{2}\|\bar{B}^*\bar{y} + \bar{z} + \bar{v} - \bar{c} + \sigma^{-1}\bar{x}\|^2. \end{aligned}$$

5.3.1 An inexact symmetric Gauss-Seidel proximal ADMM for solving (DBA-D)

The inexact symmetric Gauss-Seidel based proximal ADMM method for solving the dual problem (DBA-D) of the block angular problem (DBA-P) has the following template:

Algorithm DBA-sGS-ADMM. Given the initial points $(y^0, \bar{y}^0, z^0, \bar{z}^0, v, \bar{v}^0) \in \mathcal{Y} \times \bar{\mathcal{Y}} \times \mathcal{X} \times \bar{\mathcal{X}} \times \mathcal{X} \times \bar{\mathcal{X}}$, $(x^0, \bar{x}^0) \in \mathcal{X} \times \bar{\mathcal{X}}$, perform the following steps in each iteration.

Step 1. Let $J : \mathcal{Y} \to \mathcal{Y}$ be a given self-adjoint positive semidefinite linear operator, and $\mathcal{S}$ be a positive semidefinite linear operator that is to be chosen later. Compute

$$(\bar{z}^{k+1}, z^{k+1}, y^{k+1}) \approx \mathrm{argmin} \left\{ \begin{array}{l|l} \mathcal{L}_\sigma(y, \bar{y}^k, z, \bar{z}, v^k, \bar{v}^k; x^k, \bar{x}^k) + & y \in \mathcal{Y}, \\ \frac{\sigma}{2}\|y - y^k\|_J^2 + \frac{\sigma}{2}\|(z; y) - (z^k; y^k)\|_{\mathcal{S}}^2 & z \in \mathcal{X}, \bar{z} \in \bar{\mathcal{X}} \end{array} \right\}.$$

Let $R^{1,k} = A^* y^k + B^* \bar{y}^k + z^k + v^k - c^k$, $R^{2,k} = \bar{B}^* \bar{y}^k + \bar{z}^k + \bar{v}^k - \bar{c}^k$, where $c^k = c - \sigma^{-1} x^k$ and $\bar{c}^k = \bar{c} - \sigma^{-1} \bar{x}^k$. Observe that $\bar{z}$ and (y, z) are separable. Hence we can compute $\bar{z}^{k+1}$ and (z^{k+1}, y^{k+1}) separately by solving

$$\bar{z}^{k+1} = \mathrm{argmin}\Big\{ \frac{\sigma}{2}\|\bar{z} - \bar{z}^k + R^{2,k}\|^2 + \delta^*_{\bar{\mathcal{K}}}(-\bar{z}) \mid \bar{z} \in \bar{\mathcal{X}} \Big\}, \tag{5.11}$$

$$(z^{k+1}, y^{k+1}) \approx \mathrm{argmin} \left\{ \begin{array}{l|l} -\langle b,\, y\rangle + \delta^*_{\mathcal{K}}(-z) + & \\ \frac{\sigma}{2}\|A^* y + z + B^* \bar{y}^k + v^k - c^k\|^2 & y \in \mathcal{Y}, \\ \frac{\sigma}{2}\|y - y^k\|_J^2 + \frac{\sigma}{2}\|(z; y) - (z^k; y^k)\|_{\mathcal{S}}^2 & z \in \mathcal{X} \end{array} \right\}. \tag{5.12}$$

Notice that $\bar{z}^{k+1}$ can be computed by solving the following N subproblems in parallel, i.e.,

$$\begin{aligned} \bar{z}_i^{k+1} &= \mathrm{argmin}\Big\{ \frac{\sigma}{2}\|\bar{z}_i - \bar{z}_i^k + R_i^{2,k}\|^2 + \delta^*_{\mathcal{K}_i}(-\bar{z}_i) \mid \bar{z}_i \in \mathcal{X}_i \Big\} \quad (5.13) \\ &= -\mathrm{Prox}_{\sigma^{-1}\delta^*_{\mathcal{K}_i}}\big(R_i^{2,k} - \bar{z}_i^k\big), \quad i = 1, \ldots, N. \end{aligned}$$

To compute (y^{k+1}, z^{k+1}) efficiently, we can choose $\mathcal{S} = \mathbf{sGS}(\mathcal{Q}_{z,y})$, the

sGS linear operator associated with the following operator to decompose the computation of y^{k+1} and z^{k+1}:

$$\mathcal{Q}_{z,y} := \begin{bmatrix} I_n & A^* \\ A & AA^* + J \end{bmatrix}.$$

Then we can compute (y^{k+1}, z^{k+1}) in a symmetric Gauss-Seidel fashion as follows:

$$\begin{aligned}
y^{k+1}_{\rm tmp} &= \mathrm{argmin}\big\{ -\langle b,\, y\rangle + \tfrac{\sigma}{2}\|A^*(y-y^k)+R^{1,k}\|^2 + \tfrac{\sigma}{2}\|y-y^k\|_J^2 - \langle \delta^{k+1}_{\rm tmp},\, y\rangle \big\} \\
&= y^k + (AA^*+J)^{-1}\big(\sigma^{-1}(b+\delta^{k+1}_{\rm tmp}) - AR^{1,k}\big), \\
z^{k+1} &= \mathrm{argmin}\big\{ \tfrac{\sigma}{2}\|z - z^k + A^*(y^{k+1}_{\rm tmp}-y^k) + R^{1,k}\|^2 + \delta^*_{\mathcal{K}}(-z) \mid z\in\mathcal{X}\big\} \\
&= -\mathrm{Prox}_{\sigma^{-1}\delta^*_{\mathcal{K}}}\big(A^*(y^{k+1}_{\rm tmp}-y^k) + R^k_1 - z^k\big), \\
y^{k+1} &= \mathrm{argmin}\left\{ \begin{array}{l} -\langle b,\, y\rangle + \tfrac{\sigma}{2}\|A^*(y-y^k)+R^{1,k}+z^{k+1}-z^k\|^2 \\ +\tfrac{\sigma}{2}\|y-y^k\|_J^2 - \langle \delta^{k+1},\, y\rangle \end{array} \right\} \\
&= y^k + (AA^*+J)^{-1}(\sigma^{-1}(b+\delta^{k+1}) - A(R^{1,k}+z^{k+1}-z^k)).
\end{aligned}$$

Step 2. Let $\bar{J}$ be a given block diagonal linear operator with $\bar{J} = \mathrm{diag}(\bar{J}_1, \ldots, \bar{J}_N)$ such that each $\bar{J}_i : \mathcal{Y}_i \to \mathcal{Y}_i$ is self-adjoint positive semidefinite, and $\mathcal{T}$ be a positive semidefinite linear operator that is to be chosen later. Compute

$$(v^{k+1}, \bar{v}^{k+1}, \bar{y}^{k+1}) \approx \mathrm{argmin}\left\{ \begin{array}{l|l} \mathcal{L}_\sigma(y^{k+1}, \bar{y}, z^{k+1}, \bar{z}^{k+1}, v, \bar{v}; x^k, \bar{x}^k) + & \bar{y}\in\mathbb{R}^{\bar{m}}, \\ \tfrac{\sigma}{2}\|\bar{y}-\bar{y}^k\|^2_{\bar{J}} & v\in\mathcal{X}, \\ +\tfrac{\sigma}{2}\|(v-v^k;\bar{v}-\bar{v}^k;\bar{y}-\bar{y}^k)\|^2_{\mathcal{T}} & \bar{v}\in\bar{\mathcal{X}} \end{array} \right\}.$$

Let

$$M = BB^* + \bar{B}\bar{B}^* + \bar{J}, \tag{5.14}$$

and $R^{3,k} = A^*y^{k+1} + B^*\bar{y}^k + z^{k+1} + v^k - c^k = R^{1,k} + A^*(y^{k+1}-y^k) + (z^{k+1}-z^k)$, $R^{4,k} = \bar{B}^*\bar{y}^k + \bar{z}^{k+1} + \bar{v}^k - \bar{c}^k = R^{2,k} + (\bar{z}^{k+1}-\bar{z}^k)$. To compute $(v^{k+1}, \bar{v}^{k+1}, \bar{y}^{k+1})$ efficiently, we choose $\mathcal{T} = \mathbf{sGS}(\mathcal{Q}_{v\bar{v},\bar{y}})$, the sGS

linear operator associated with the following linear operator

$$\mathcal{Q}_{v\bar{v},\bar{y}} = \left[\begin{array}{cc|c} I_n & 0 & B^* \\ 0 & I_{\bar{n}} & \bar{B}^* \\ \hline B & \bar{B} & M \end{array}\right].$$

Then we can decompose the computation of $(v^{k+1}, \bar{v}^{k+1}, \bar{y}^{k+1})$ in a symmetric Gauss-Seidel fashion as follows:

$$\begin{aligned} \bar{y}^{k+1}_{\rm tmp} &= \mathrm{argmin}\left\{ \begin{array}{l} -\langle \bar{b},\, \bar{y}\rangle + \frac{\sigma}{2}\|B^*(\bar{y}-\bar{y}^k) + R^{3,k}\|^2 + \frac{\sigma}{2}\|\bar{B}^*(\bar{y}-\bar{y}^k) + R^{4,k}\|^2 \\ +\frac{\sigma}{2}\|\bar{y}-\bar{y}^k\|^2_{\bar{J}} - \langle \bar{\delta}^{k+1}_{\rm tmp},\, \bar{y}\rangle \end{array} \right\} \\ &= \bar{y}^k + M^{-1}(\sigma^{-1}(\bar{b}+\bar{\delta}^{k+1}_{\rm tmp}) - BR^{3,k} - \bar{B}R^{4,k}); \end{aligned}$$

$$\left\{ \begin{array}{rcl} v^{k+1} &=& \mathrm{argmin}\big\{\theta^*(-v) + \frac{\sigma}{2}\|v - v^k + B^*(\bar{y}^{k+1}_{\rm tmp} - \bar{y}^k) + R^{3,k}\|^2 \mid v \in \mathcal{X}\big\}; \\ \bar{v}^{k+1} &=& \mathrm{argmin}\big\{\bar{\theta}^*(-\bar{v}) + \frac{\sigma}{2}\|\bar{v} - \bar{v}^k + \bar{B}^*(\bar{y}^{k+1}_{\rm tmp} - \bar{y}^k) + R^{4,k}\|^2 \mid \bar{v} \in \bar{\mathcal{X}}\big\}; \end{array} \right. \tag{5.15}$$

$$\begin{aligned} \bar{y}^{k+1} &= \mathrm{argmin}\left\{ \begin{array}{l} -\langle \bar{b},\, \bar{y}\rangle + \frac{\sigma}{2}\|B^*(\bar{y}-\bar{y}^k) + v^{k+1} - v^k + R^{3,k}\|^2 \\ +\frac{\sigma}{2}\|\bar{B}^*(\bar{y}-\bar{y}^k) + \bar{v}^{k+1} - \bar{v}^k + R^{4,k}\|^2 + \frac{\sigma}{2}\|\bar{y}-\bar{y}^k\|^2_{\bar{J}} - \langle \bar{\delta}^{k+1},\, \bar{y}\rangle \end{array} \right\} \\ &= \bar{y}^k + M^{-1}\big(\sigma^{-1}(\bar{b}+\bar{\delta}^{k+1}) - B(R^{3,k} + v^{k+1} - v^k) - \bar{B}(R^{4,k} + \bar{v}^{k+1} - \bar{v}^k)\big). \end{aligned}$$

We should note that the computation of $\bar{v}^{k+1}$ in (5.15) can be done in parallel by solving N smaller subproblems where

$$\begin{aligned} \bar{v}^{k+1}_i &= \mathrm{argmin}\Big\{\bar{\theta}^*_i(-\bar{v}_i) + \frac{\sigma}{2}\|\bar{v}_i - \bar{v}^k_i + \bar{B}^*_i\big((\bar{y}^{k+1}_{\rm tmp})_i - \bar{y}^k_i\big) + (R^k_4)_i\|^2 \mid \bar{v}_i \in \mathcal{X}_i\Big\} \\ &= -\mathrm{Prox}_{\sigma^{-1}\bar{\theta}^*_i}\Big(\bar{B}^*_i\big((\bar{y}^{k+1}_{\rm tmp})_i - \bar{y}^k_i\big) + R^{4,k}_i - \bar{v}^k_i\Big), \quad i = 1, \ldots, N. \end{aligned} \tag{5.16}$$

Step 3. Compute

$$\begin{aligned} x^{k+1} &= x^k + \tau\sigma(A^*y^{k+1} + B^*\bar{y}^{k+1} + z^{k+1} + v^{k+1} - c), \\ \bar{x}^{k+1} &= \bar{x}^k + \tau\sigma(\bar{B}^*\bar{y}^{k+1} + \bar{z}^{k+1} + \bar{v}^{k+1} - \bar{c}), \end{aligned}$$

where $\tau \in (0, (1+\sqrt{5})/2)$ is the step-length, which is usually set to be 1.618.

5.3.2 An inexact symmetric Gauss-Seidel proximal ALM for solving (DBA-D)

When the functions $\theta(\cdot)$ and $\bar{\theta}_i(\cdot)$ $(i = 1, \ldots, N)$ are absent in (5.1), we can design a symmetric Gauss-Seidel proximal augmented Lagrangian method (sGS-ALM) to solve the problem. In this case, the variables v and $\bar{v}$ are absent in (DBA-D), and the associated augmented Lagrangian function is given as follows: for $(y, \bar{y}, z, \bar{z}) \in \mathcal{Y} \times \bar{\mathcal{Y}} \times \mathcal{X} \times \bar{\mathcal{X}}$, $(x, \bar{x}) \in \mathcal{X} \times \bar{\mathcal{X}}$,

$$\begin{aligned}\mathcal{L}_\sigma(y, \bar{y}, z, \bar{z}; x, \bar{x}) \;=\; & -\langle b,\, y\rangle - \langle \bar{b},\, \bar{y}\rangle + \delta^*_{\mathcal{K}}(-z) + \delta^*_{\bar{\mathcal{K}}}(-\bar{z}) \\ & + \tfrac{\sigma}{2}\|A^*y + B^*\bar{y} + z - c + \sigma^{-1}x\|^2 \\ & + \tfrac{\sigma}{2}\|\bar{B}^*\bar{y} + \bar{z} - \bar{c} + \sigma^{-1}\bar{x}\|^2 - \tfrac{1}{2\sigma}\|x\|^2 - \tfrac{1}{2\sigma}\|\bar{x}\|^2.\end{aligned}$$

Algorithm DBA-sGS-ALM. Given the initial points $(\bar{z}^0, z^0, y^0, \bar{y}^0) \in \bar{\mathcal{X}} \times \mathcal{X} \times \mathcal{Y} \times \bar{\mathcal{Y}}$, $(x^0, \bar{x}^0) \in \mathcal{X} \times \bar{\mathcal{X}}$, perform the following steps in each iteration.

Step 1. Let $J : \mathcal{Y} \to \mathcal{Y}$ and $\bar{J} : \bar{\mathcal{Y}} \to \bar{\mathcal{Y}}$ be given self-adjoint positive semidefinite linear operators, and $\mathcal{S}$ be a positive semidefinite linear operator that is to be chosen later. Let $\{\epsilon_k\}$ be a given summable sequence of nonnegative numbers. Compute

$$\begin{aligned}&\alpha^k := (\bar{z}^{k+1}, z^{k+1}, y^{k+1}, \bar{y}^{k+1}) \\ &\approx \widehat{\alpha}^k = \operatorname{argmin}\left\{ \begin{array}{l|l} \mathcal{L}_\sigma(y, \bar{y}, z, \bar{z}; x^k, \bar{x}^k) & \\ +\frac{\sigma}{2}\|y - y^k\|_J^2 + \frac{\sigma}{2}\|\bar{y} - \bar{y}^k\|_{\bar{J}}^2 & \begin{array}{l} y \in \mathcal{Y}, \bar{y} \in \bar{\mathcal{Y}}, \\ z \in \mathcal{X}, \bar{z} \in \bar{\mathcal{X}} \end{array} \\ +\frac{\sigma}{2}\|(\bar{z}; z; y; \bar{y}) - (\bar{z}^k; z^k; y^k; \bar{y}^k)\|_{\mathcal{S}}^2 & \end{array} \right\},\end{aligned}$$

with residual $d^{k+1} \in \partial_\alpha L_\sigma(\alpha^{k+1}, \beta^k) + \sigma\mathcal{T}(\alpha^{k+1} - \alpha^k)$ satisfying $\|d^{k+1}\| \le \epsilon^k$, where $\mathcal{T} := \mathcal{S} + \operatorname{diag}(0, 0, J, \bar{J})$. If we choose $\mathcal{S} = \mathbf{sGS}(\mathcal{Q}_{\bar{z}z,y,\bar{y}})$, the sGS linear operator associated with the following operator

$$\mathcal{Q}_{\bar{z}z,y,\bar{y}} = \left[\begin{array}{cc|c|c} I_{\bar{n}} & 0 & 0 & \bar{B}^* \\ 0 & I_n & A^* & B^* \\ \hline 0 & A & AA^* + J & AB^* \\ \hline \bar{B} & B & BA^* & BB^* + \bar{B}\bar{B}^* + \bar{J} \end{array}\right],$$

then we can decompose the computation of $(\bar{z}^{k+1}, z^{k+1}, y^{k+1}, \bar{y}^{k+1})$ in a symmetric Gauss-Seidel fashion as follows. Let $c^k = c - \sigma^{-1}x^k$ and $\bar{c}^k = \bar{c} - \sigma^{-1}\bar{x}^k$. Compute

$$\bar{y}^{k+1}_{\rm tmp} = \mathrm{argmin}\left\{ \begin{array}{l} -\langle \bar{b},\, \bar{y}\rangle + \frac{\sigma}{2}\|B^*\bar{y} + A^*y^k + z^k - c^k\|^2 \\ +\frac{\sigma}{2}\|\bar{B}^*\bar{y} + \bar{z}^k - \bar{c}^k\|^2 + \frac{\sigma}{2}\|\bar{y} - \bar{y}^k\|^2_{\bar{J}} - \langle \bar{\delta}^{k+1}_{\rm tmp},\, \bar{y}\rangle \end{array} \right\};$$

$$y^{k+1}_{\rm tmp} = \mathrm{argmin}\left\{ \begin{array}{l} -\langle b,\, y\rangle + \frac{\sigma}{2}\|A^*y + B^*\bar{y}^k_{\rm tmp} + z^k - c^k\|^2 \\ +\frac{\sigma}{2}\|y - y^k\|^2_J - \langle \delta^{k+1}_{\rm tmp},\, y\rangle \end{array} \right\};$$

$$\left\{ \begin{array}{rcl} z^{k+1} &=& \mathrm{argmin}\{ \frac{\sigma}{2}\|z + A^*y^{k+1}_{\rm tmp} + B^*\bar{y}^{k+1}_{\rm tmp} - c^k\|^2 + \delta^*_{\mathcal{K}}(-z) \mid z \in \mathcal{X}\} \\ &=& -\mathrm{Prox}_{\sigma^{-1}\delta^*_{\mathcal{K}}}\big(A^*y^{k+1}_{\rm tmp} + B^*\bar{y}^{k+1}_{\rm tmp} - c^k\big); \\ \bar{z}^{k+1}_i &=& \mathrm{argmin}\{ \frac{\sigma}{2}\|\bar{z}_i + \bar{B}^*_i(\bar{y}^{k+1}_{\rm tmp})_i - \bar{c}^k_i\|^2 + \delta^*_{\mathcal{K}_i}(-\bar{z}_i) \mid \bar{z}_i \in \mathcal{X}_i\} \\ &=& -\mathrm{Prox}_{\sigma^{-1}\delta^*_{\mathcal{K}_i}}\big(\bar{B}^*_i(\bar{y}^{k+1}_{\rm tmp})_i - \bar{c}^k_i\big) \quad i = 1, \ldots, N; \end{array} \right.$$

$$y^{k+1} = \mathrm{argmin}\left\{ \begin{array}{l} -\langle b,\, y\rangle + \frac{\sigma}{2}\|A^*y + B^*\bar{y}^{k+1}_{\rm tmp} + z^{k+1} - c^k\|^2 \\ +\frac{\sigma}{2}\|y - y^k\|^2_J - \langle \delta^{k+1},\, y\rangle \end{array} \right\};$$

$$\bar{y}^{k+1} = \mathrm{argmin}\left\{ \begin{array}{l} -\langle \bar{b},\, \bar{y}\rangle + \frac{\sigma}{2}\|B^*\bar{y} + A^*y^{k+1} + z^{k+1} - c^k\|^2 \\ +\frac{\sigma}{2}\|\bar{B}^*\bar{y} + \bar{z}^{k+1} - \bar{c}^k\|^2 + \frac{\sigma}{2}\|\bar{y} - \bar{y}^k\|^2_{\bar{J}} - \langle \bar{\delta}^{k+1},\, \bar{y}\rangle \end{array} \right\}.$$

Step 2. Compute

$$\begin{aligned} x^{k+1} &= x^k + \tau\sigma(A^*y^{k+1} + B^*\bar{y}^{k+1} + z^{k+1} - c), \\ \bar{x}^{k+1} &= \bar{x}^k + \tau\sigma(\bar{B}^*\bar{y}^{k+1} + \bar{z}^{k+1} - \bar{c}), \end{aligned}$$

where $\tau \in (0, 2)$ is the step-length which is usually set to be 1.9.

5.3.3 Convergence of the inexact sGS-ADMM and sGS-ALM algorithms

The convergence theorem of the inexact sGS-ADMM and sGS-ALM algorithms could be established directly using known results from Chen et al. (2017), Zhang

et al. (2018), and Chen et al. (2018). Here we state the theorems again for the convenience of readers.

Denote $u := (y, \bar{y}, v, \bar{v}, z, \bar{z}, x, \bar{x}) \in \mathcal{U} := \mathcal{Y} \times \bar{\mathcal{Y}} \times \mathcal{X} \times \bar{\mathcal{X}} \times \mathcal{X} \times \bar{\mathcal{X}} \times \mathcal{X} \times \bar{\mathcal{X}}$. The KKT mapping $\mathcal{R} : \mathcal{U} \to \mathcal{U}$ of (5.1) is defined by

$$\mathcal{R}(u) := \begin{pmatrix} Ax - b \\ Bx + \bar{B}\bar{x} - \bar{b} \\ A^*y + B^*\bar{y} + z + v - c \\ \bar{B}^*\bar{y} + \bar{z} + \bar{v} - \bar{c} \\ x - \Pi_{\mathcal{K}}(x - z) \\ \bar{x} - \Pi_{\bar{\mathcal{K}}}(\bar{x} - \bar{z}) \\ x - \mathrm{Prox}_{\theta}(x - v) \\ \bar{x} - \mathrm{Prox}_{\bar{\theta}}(\bar{x} - \bar{v}) \end{pmatrix}. \tag{5.17}$$

Theorem 5.1. Let $\{u^k := (y^k, \bar{y}^k, v^k, \bar{v}^k, z^k, \bar{z}^k, x^k, \bar{x}^k)\}$ be the sequence generated by sGS-ADMM. Then, we have the following results.

(a) The sequence $\{(y^k, \bar{y}^k, v^k, \bar{v}^k, z^k, \bar{z}^k)\}$ converges to an optimal solution of the dual problem (DBA-D) and the sequence $\{(x^k, \bar{x}^k)\}$ converges to an optimal solution of the primal problem (DBA-P).

(b) Suppose that the sequence $\{u^k\}$ converges to a KKT point $\hat{u} := (\hat{y}^k, \hat{\bar{y}}^k, \hat{v}^k, \hat{\bar{v}}^k, \hat{z}^k, \hat{\bar{z}}^k, \hat{x}^k, \hat{\bar{x}}^k)$ and the KKT mapping $\mathcal{R}$ is metrically subregular at $(\hat{u}, 0) \in \mathrm{gph}\mathcal{R}$. Then the sequence $\{u^k\}$ is linearly convergent to $\hat{u}$.

Proof. We refer the reader to section 4.3.1 for the definition of metrically subregularity. The result of the theorem follows directly by applying the convergence result in (Zhang et al., 2018, Proposition 4.1) to (DBA-D). □

Remark 5. By Theorem 1 and Remark 1 in Li et al. (2018b), we know that when (DBA-P) is a convex programming problem where for each $i = 1, \ldots, N$, θ and $\bar{\theta}_i$ are piecewise linear-quadratic or strongly convex, and $\mathcal{K}, \mathcal{K}_i$ are polyhedral, then $\mathcal{R}$ is metrically subregular at $(\bar{u}, 0) \in \mathrm{gph}\mathcal{R}$ for any KKT point $\bar{u}$.

Similarly, the convergence of the sGS-ALM can be established readily by using known results in Chen et al. (2018). Define the self-adjoint positive definite linear operator $\mathcal{H}, \mathcal{V} : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{X}}$ by

$$\mathcal{H} := \begin{bmatrix} 0 \\ I \\ A \\ B \end{bmatrix} \begin{bmatrix} 0 & I & A^* & B^* \end{bmatrix} + \begin{bmatrix} I \\ 0 \\ 0 \\ \bar{B} \end{bmatrix} \begin{bmatrix} I & 0 & 0 & \bar{B}^* \end{bmatrix},$$

$$\mathcal{V} := \tau\sigma^2\Big(\mathcal{T} + \frac{2-\tau}{6}\mathcal{H}\Big).$$

We have the following convergence result for the inexact sGS-ALM.

Theorem 5.2. Define $\alpha := (\bar{z}, z, y, \bar{y})$ and $\beta := (x, \bar{x})$. Assume that the solution set to the KKT system of (DBA-D) is nonempty and $(\bar{\alpha}, \bar{\beta})$ is such a solution. Then, the sequence $\{(\alpha^k, \beta^k)\}$ generated by DBA-sGS-ALM is well-defined such that for any $k \geq 1$,

$$\|\alpha^{k+1} - \widehat{\alpha}^{k+1}\|^2_{\sigma(\mathcal{T}+\mathcal{H})} \leq \langle d^{k+1}, \alpha^{k+1} - \widehat{\alpha}^{k+1}\rangle,$$

and for all $k = 0, 1, \ldots$,

$$\begin{aligned} &\Big(\|\alpha^{k+1} - \overline{\alpha}\|^2_{\hat{\mathcal{V}}^{1/2}} + \|\beta^{k+1} - \overline{\beta}\|^2\Big) - \Big(\|\alpha^{k} - \overline{\alpha}\|^2_{\hat{\mathcal{V}}^{1/2}} + \|\beta^{k} - \overline{\beta}\|^2\Big) \\ &\leq -\left(\tfrac{2-\tau}{3\tau}\|\beta^k - \beta^{k+1}\|^2 + \|\alpha^{k+1} - \alpha^k\|^2_{\mathcal{V}} - 2\tau\sigma\langle d^k, \alpha^{k+1} - \overline{\alpha}\rangle\right), \end{aligned}$$

where $\hat{\mathcal{V}} = \mathcal{V} + \frac{2-\tau}{6}\tau\sigma^2\mathcal{H}$. Moreover, the sequence $\{(\alpha^k, \beta^k)\}$ converges to a solution to the KKT system of (DBA-D).

Proof. The result can be proved directly from the convergence result in (Chen et al., 2018, Theorem 1). □

5.3.4 The efficient computation of $\bar{y}^{k+1}_{\text{tmp}}$ and $\bar{y}^{k+1}$

To implement the DBA-sGS-ALM or DBA-sGS-ADMM efficiently, the efficient computation of $\bar{y}^{k+1}_{\text{tmp}}$ and $\bar{y}^{k+1}$ must be carefully addressed. To compute the solution $\bar{y}^{k+1}_{\text{tmp}}$ in (5.15), we would need to solve a generally very large $\bar{m} \times \bar{m}$

system of linear equations of the form:

$$M\bar{y} = h, \tag{5.18}$$

where the $\bar{m} \times \bar{m}$ matrix M given in (5.14) has the following structure:

$$M = \begin{bmatrix} B_1 \\ \vdots \\ B_N \end{bmatrix} \begin{bmatrix} B_1 \\ \vdots \\ B_N \end{bmatrix}^* + \begin{bmatrix} \bar{B}_1\bar{B}_1^* & & \\ & \ddots & \\ & & \bar{B}_N\bar{B}_N^* \end{bmatrix} + \bar{J}. \tag{5.19}$$

Here we design several strategies which are crucial for cutting down the computational cost of solving (5.18).

(a) Suppose we choose $\bar{J} = \mathrm{diag}(\bar{J}_1, \ldots, \bar{J}_N)$ where each $\bar{J}_i : \mathcal{X}_i \to \mathcal{X}_i$ is a self-adjoint positive semidefinite linear operator. To compute the inverse of M in (5.19), we may make use of the Sherman-Morrison-Woodbury (SMW) formula which we shall describe next. Let $\bar{D} = \mathrm{diag}(\bar{D}_1, \ldots, \bar{D}_N)$, where $\bar{D}_i = \bar{B}_i\bar{B}_i^* + \bar{J}_i$, $i = 1, \ldots, N$. Assume that $\bar{D}_i$ is invertible for all $i = 1, \ldots, N$, then we have that

$$M^{-1}h = (\bar{D} + BB^*)^{-1}h = \bar{D}^{-1}h - \bar{D}^{-1}BG^{-1}B^*\bar{D}^{-1}h,$$

where $G : \mathcal{X} \to \mathcal{X}$ is given by

$$G := I_n + \sum_{i=1}^{N} B_i^*\bar{D}_i^{-1}B_i. \tag{5.20}$$

Thus to solve (5.18), we only need to solve a linear system of the form $Gx = g$.

Observe that to compute G, the component matrices $B_i^*\bar{D}_i^{-1}B_i$, $i = 1, \ldots, N$, can be computed in parallel, and operations such as $\bar{D}_i^{-1}h_i$, $i = 1, \ldots, N$, can also be done in parallel. Similarly operations such as evaluating Bx and $B^*\bar{y}$ for given $x \in \mathcal{X}$ and $\bar{y} \in \bar{\mathcal{Y}}$ can also be done in parallel.

(i) Generally, one would expect the $n \times n$ matrix $B_i^*\bar{D}_i^{-1}B_i$ to be dense even if B_i is sparse, unless $\bar{D}_i$ has a nice structure such as being a diagonal

matrix. If the memory required to store the matrix G and its Cholesky factor is prohibitive, then one may choose $\bar{J}_i$ such that $\bar{D}_i$ is a diagonal matrix. For example, we may choose

$$\bar{J}_i = \lambda_{\max}(\bar{B}_i\bar{B}_i^*)I_{m_i} - \bar{B}_i\bar{B}_i^*, \quad i = 1, \ldots, N.$$

Hence $\bar{D}_i = \lambda_{\max}(\bar{B}_i\bar{B}_i^*)I_{m_i}$. In which case, $G = I_n + \sum_{i=1}^N \lambda_{\max}(\bar{B}_i\bar{B}_i^*)^{-1}B_i^*B_i$, and it is more likely to be sparse.

(ii) In general, we can expect the matrix symmetric positive definite matrix G in (5.20) to be well conditioned, especially if for each i, we choose $\bar{J}_i = \alpha_i I_{m_i}$ for a sufficiently large nonnegative scalar α_i. In this case, a preconditioned conjugate gradient (PCG) method applied to solve $Gx = g$ is expected to converge in a small number of steps. In addition, for a given x, the matrix-vector product Gx can also be computed very efficiently since the component vectors, $\bar{B}_i^*\bar{D}_i^{-1}\bar{B}_i x$, $i = 1, \ldots, N$, can be computed in parallel.

(b) It is possible to solve (5.18) in a parallel fashion if we choose $\bar{J}$ appropriately. For example, if we choose

$$\bar{J} = \mathrm{diag}(B_iB_i^* + \textstyle\sum_{j=1, j\neq i}^N \|B_iB_j^*\|_2\, I_{m_i} \mid i = 1, \ldots, N) - BB^* \succeq 0, \quad (5.21)$$

then

$$M = \mathrm{diag}(\bar{E}, \ldots, \bar{E}_N), \quad (5.22)$$

where $\bar{E}_i = \bar{B}_i\bar{B}_i^* + B_iB_i^* + \sum_{j=1, j\neq i}^N \|B_iB_j^*\|_2 I_{m_i}$, $i = 1, \ldots, N$. Then the solution $\bar{y}$ in (5.18) can be computed in parallel by solving

$$\bar{E}_i\bar{y}_i = h_i, \quad i = 1, \ldots, N.$$

We should mention that in the literature, a standard way to decompose

M into a block diagonal form is to choose $\bar{J}$ to be:

$$\bar{J}^{\rm std} = N{\rm diag}(B_1B_1^*, \ldots, B_NB_N^*) - BB^* \succeq 0,$$

where the positive semidefiniteness of $\bar{J}^{\rm std}$ can be shown by using the inequality that $\|\sum_{i=1}^N \bar{x}_i\|^2 \le N\sum_{i=1}^N \|\bar{x}_i\|^2$ for any $\bar{x} \in \bar{\mathcal{X}}$. However, such a choice is usually too conservative compared to the one in (5.21).

(c) We should also take special consideration when the optimization problem (5.1) has the property that $B_1 = B_2 = \cdots = B_N$. In this case

$$G = I_n + B_1^*\big(\textstyle\sum_{i=1}^N \bar{D}_i^{-1}\big)B_1.$$

Thus if m_1 is not too large, one can precompute the $m_1 \times m_1$ matrix $\sum_{i=1}^N \bar{D}_i^{-1}$ once at the beginning of the algorithm so that the matrix-vector product Gx can be evaluated very efficiently for any given $x \in \mathcal{X}$. If in addition, we also have that $\bar{B}_1 = \cdots = \bar{B}_N$ such as in the case of the DNN relaxation (5.6) of the UFL problem, then we get

$$G = I_n + NB_1^*\bar{D}_1^{-1}B_1.$$

In this case, obviously there is tremendous saving in the computation of the matrix-vector product Gx for any given x.

5.4 Incorporating a semismooth Newton-CG method in DBA-sGS-ADMM and DBA-sGS-ALM for solving (DBA-D)

Recall that in computing (z^{k+1}, y^{k+1}) in (5.12) for Algorithm DBA-sGS-ADMM, we have added a proximal term based on the sGS linear operator $\mathbf{sGS}(\mathcal{Q}_{z,y})$ to decouple the computation of z^{k+1} and y^{k+1}. We should note that while the computation of the individual z^{k+1} and y^{k+1} has been made easier, the negative effect of adding a possibly large proximal term is that the overall algorithm

may take more iterations to converge to a required level of accuracy. Thus in practice, a judicious choice must always be made to balance the two competing factors. In particular, if an efficient solver is available to compute (z^{k+1}, y^{k+1}) simultaneously in (5.12) without the need to add a proximal term, i.e., $J = 0$ and $\mathcal{S} = 0$, then one should always adopted this option. In this case, we have that

$$(z^{k+1}, y^{k+1}) = \operatorname{argmin}\left\{ \begin{array}{l|l} -\langle b,\, y\rangle + \delta^*_{\mathcal{K}}(-z) - \langle \delta^{k+1},\, y\rangle & y \in \mathcal{Y}, \\ +\frac{\sigma}{2}\|A^*y + z + B^*\bar{y}^k + v^k - c^k\|^2 & z \in \mathcal{X} \end{array} \right\}. \quad (5.23)$$

In Step 1 of Algorithm DBA-sGS-ALM, if we choose $J = 0$ and $\mathcal{S} = \mathbf{sGS}(\mathcal{Q}_{\bar{z}zy,\bar{y}})$, the sGS linear operator associated with the following operator

$$\mathcal{Q}_{\bar{z}zy,\bar{y}} = \left[\begin{array}{ccc|c} I_{\bar{n}} & 0 & 0 & \bar{B}^* \\ 0 & I_n & A^* & B^* \\ 0 & A & AA^* & AB^* \\ \hline \bar{B} & B & BA^* & BB^* + \bar{B}\bar{B}^* + \bar{J} \end{array}\right],$$

then we can compute $(\bar{z}^{k+1}, z^{k+1}, y^{k+1}, \bar{y}^{k+1})$ in a symmetric Gauss-Seidel fashion as follows. Let $c^k = c - \sigma^{-1}x^k$ and $\bar{c}^k = \bar{c} - \sigma^{-1}\bar{x}^k$. Compute

$$\begin{aligned} \bar{y}^{k+1}_{\rm tmp} &= \operatorname{argmin}\left\{ \begin{array}{l} -\langle \bar{b},\, \bar{y}\rangle + \frac{\sigma}{2}\|B^*\bar{y} + A^*y^k + z^k - c^k\|^2 + \frac{\sigma}{2}\|\bar{B}^*\bar{y} + \bar{z}^k - \bar{c}^k\|^2 \\ +\frac{\sigma}{2}\|\bar{y} - \bar{y}^k\|^2_{\bar{J}} - \langle \bar{\delta}^{k+1}_{\rm tmp},\, \bar{y}\rangle \end{array} \right\}, \\ \left\{ \begin{array}{l} (z^{k+1}, y^{k+1}) = \operatorname{argmin}\left\{ \begin{array}{l|l} -\langle b,\, y\rangle + \delta^*_{\mathcal{K}}(-z) + & y \in \mathcal{Y}, \\ \frac{\sigma}{2}\|z + A^*y + B^*\bar{y}^{k+1}_{\rm tmp} - c^k\|^2 - \langle \delta^{k+1},\, y\rangle & z \in \mathcal{X} \end{array} \right\}, \\ \bar{z}^{k+1} = \operatorname{argmin}\big\{ \frac{\sigma}{2}\|\bar{z} + \bar{B}^*\bar{y}^{k+1}_{\rm tmp} - \bar{c}^k\|^2 + \delta^*_{\bar{\mathcal{K}}}(-\bar{z}) \mid \bar{z} \in \bar{\mathcal{X}} \big\}, \end{array} \right. \quad (5.24) \\ \bar{y}^{k+1} &= \operatorname{argmin}\left\{ \begin{array}{l} -\langle \bar{b},\, \bar{y}\rangle + \frac{\sigma}{2}\|B^*\bar{y} + A^*y^{k+1} + z^{k+1} - c^k\|^2 + \frac{\sigma}{2}\|\bar{B}^*\bar{y} + \bar{z}^{k+1} - \bar{c}^k\|^2 \\ +\frac{\sigma}{2}\|\bar{y} - \bar{y}^k\|^2_{\bar{J}} - \langle \bar{\delta}^{k+1},\, \bar{y}\rangle \end{array} \right\}. \end{aligned}$$

In both (5.23) and (5.24), we need to solve a problem of the form:

$$(z^{k+1}, y^{k+1}) \;=\; \operatorname{argmin} \left\{ \begin{array}{l} -\langle b,\, y\rangle + \delta^*_{\mathcal{K}}(-z) \\ +\frac{\sigma}{2}\|z + A^*y - \widehat{c}^k\|^2 - \langle \delta^{k+1},\, y\rangle \end{array} \;\Big|\; y \in \mathcal{Y},\; z \in \mathcal{X} \right\}, \quad (5.25)$$

for some given $\widehat{c}^k$. By making use of the projection onto $\mathcal{K}$, we can first compute y^{k+1} from the following minimization problem, and obtain z^{k+1} once y^{k+1} is computed as follows:

$$\left\{ \begin{array}{rcl} y^{k+1} & = & \operatorname{argmin} \left\{ -\langle b,\, y\rangle + \frac{\sigma}{2}\|\Pi_{\mathcal{K}}(A^*y - \widehat{c}^k)\|^2 - \langle \delta^{k+1},\, y\rangle \;\big|\; y \in \mathcal{Y} \right\}, \\ z^{k+1} & = & \Pi_{\mathcal{K}^*}\big(\widehat{c}^k - A^*y^{k+1}\big). \end{array} \right.$$

In this case, we could employ a Newton based conjugate gradient method (Newton CG) to update the variable y and hence are able to compute (z^{k+1}, y^{k+1}) simultaneously. This method is however not implemented in the context of this thesis and might be a possible future enhancement for our algorithm in solving dual block angular problems.

5.5 Numerical experiments

Here we describe the numerical experiments we have conducted on several data classes of (DBA).

5.5.1 Stopping conditions

Based on the optimality conditions in (5.10), we measure the accuracy of a computed solution by the following relative residuals:

$$\eta = \max\{\eta_P, \eta_D, 0.1\eta_{\mathcal{K}}, \eta_{\bar{P}}, \eta_{\bar{D}}, 0.1\eta_{\bar{\mathcal{K}}}\},$$

where

$$\eta_P = \tfrac{\|Ax-b\|}{1+\|b\|}, \quad \eta_D = \tfrac{\|A^*y+\bar{B}^*\bar{y}+z+v-c\|}{1+\|c\|}, \quad \eta_{\mathcal{K}} = \tfrac{\|x-\Pi_{\mathcal{K}}(x-z)\|}{1+\|x\|+\|z\|},$$
$$\eta_{\bar{P}} = \tfrac{\|Bx+\bar{B}\bar{x}-\bar{b}\|}{1+\|\bar{b}\|}, \quad \eta_{\bar{D}} = \tfrac{\|\bar{B}^*\bar{y}+\bar{z}+\bar{v}-\bar{c}\|}{1+\|\bar{c}\|}, \quad \eta_{\bar{\mathcal{K}}} = \tfrac{\|\bar{x}-\Pi_{\bar{\mathcal{K}}}(\bar{x}-\bar{z})\|}{1+\|\bar{x}\|+\|\bar{z}\|}.$$

In addition, we also compute the duality gap by:

$$\eta_{\text{gap}} = \frac{|\text{Obj}_{\text{Primal}} - \text{Obj}_{\text{Dual}}|}{1 + |\text{Obj}_{\text{Primal}}| + |\text{Obj}_{\text{Dual}}|},$$

where

$$\begin{aligned} \text{Obj}_{\text{Primal}} &:= \theta(x) + \langle c, x\rangle + \bar{\theta}(\bar{x}) + \langle \bar{c}, \bar{x}\rangle, \quad \text{and} \\ \text{Obj}_{\text{Dual}} &:= \theta^*(-v) + \delta^*_{\mathcal{K}}(-z) - \langle b, y\rangle + \bar{\theta}^*(-\bar{v}) + \delta^*_{\bar{\mathcal{K}}}(-\bar{z}) - \langle \bar{b}, \bar{y}\rangle, \end{aligned}$$

are the primal and dual objective functions respectively. We terminate our algorithm when $\eta \leq 10^{-5}$ and $\eta_{\text{gap}} \leq 10^{-4}$.

5.5.2 2-stage Stochastic Conic Programming Problems

In this subsection, we run numerical experiment on the two-stage stochastic conic programming problems. Our main objective here is to test the efficiency of our algorithm for quadratic stochastic programming (QSP). Since there are not much QSP test data available, we will follow the idea mentioned in Lau and Womersley (2001) to extend stochastic linear programming to QSP by artificially adding a small quadratic term in the objective. That is, we choose to set $\theta(x) = \frac{0.1}{2}\|x\|^2$, $\bar{\theta}(\bar{x}) = \frac{0.1}{2}\|\bar{x}\|^2$.

Other than those data sets that have been studied in Linderoth et al. (2006), we also examine several other data sets found in the literature. The datasets `assets`, `env`, `phone`, and `4node` are from the website `http://www4.uwsp.edu/math/afelt/slptestset/download.html`; `pltexp` and `storm` are from `http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html`; and at last `gbd` could be downloaded from `http://pages.cs.wisc.edu/~swright/stochastic/sampling/`. Below we describe each of the dataset in more details.

assets. This network model represents the management of assets. Its nodes are asset categories and its arcs are transactions. The problem is to maximize the return of an investment from every stage with the balance of material at each node.

env. This model assists the Canton of Geneva in planning its energy supply infrastructure and policies. The main objective is to minimize the installation cost of various types of energy, while meeting all the supply-demand at each node and satisfying several realistic constraints such as the environmental constraints.

phone. This is a problem which models the service of providing private lines to telecommunication customers, often used by large corporations between business locations for high speed, private data transmission. The goal is to minimize the unserved requests, while staying within budget.

4node. This is a two stage network problem for scheduling cargo transportation. While the flight schedule is completely determined in stage one, the amounts of cargo to be shipped are uncertain and shall be determined in stage two.

pltexp. This is a stochastic capacity expansion model that tries to allocate new production capacity across a set of plants so as to maximize profit subject to uncertain demand.

storm. This is a two period freight scheduling problem described in Mulvey and Ruszczynski. In this model, routes are scheduled to satisfy a set of demands at stage 1, demands occur, and unmet demands are delivered at higher costs in stage 2 to account for shortcomings.

gbd. This is the aircraft allocation problem where aircraft of different types are to be allocated to routes in a way that maximizes profit under uncertain demand, and minimizes the cost of operating the aircraft as well as costs associated with bumping passengers when the demand for seats outstrips the capacity.

In Table 5.1, we compare the numerical performance of our proposed sGS-ADMM method against the state-of-the-art solver Gurobi. As mentioned in previous chapter, although Gurobi is not a specialized algorithm for solving DBA problem, it is so efficient that it should be a benchmark for any newly-developed algorithm. When running Gurobi, we set the stopping tolerance to be 1e-5 for

consistency and input the DBA problem as the usual quadratic programming problem without any special structure.

					sGS-ADMM			Gurobi	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Obj	Iter	Time(s)	$d_y \vert d_{\bar{y}}$	Iter	Time(s)
assets_small	5 | 5	13 | 13	100	1.86e+05	1351	1.23	86 | 697	20	0.16
assets_large	5 | 5	13 | 13	37500	5.09e+06	20000	761.06	1752 | 5190	33	17.83
env_aggr	32 | 48	69 | 85	5	6.38e+04	501	0.43	501 | 18	23	0.16
env_imp	32 | 48	69 | 85	15	1.22e+05	1022	0.50	1022 | 46	25	0.17
env_1200	32 | 48	69 | 85	1200	6.74e+06	5251	24.37	5251 | 630	36	4.66
env_1875	32 | 48	69 | 85	1875	1.05e+07	4800	37.66	4800 | 635	38	7.45
env_3780	32 | 48	69 | 85	3780	2.11e+07	5099	105.69	5099 | 667	39	15.50
env_5292	32 | 48	69 | 85	5292	2.95e+07	5361	156.86	5361 | 707	40	23.94
env_lrge	32 | 48	69 | 85	8232	4.58e+07	4611	225.64	4611 | 838	44	37.07
env_xlrge	32 | 48	69 | 85	32928	1.83e+08	5633	1389.10	5633 | 1667	39	157.56
phone	1 | 23	9 | 93	32768	2.74e+05	4201	489.65	4199 | 1579	195	573.03
phone_1	1 | 23	9 | 93	1	4.77e+01	157	0.09	64 | 28	12	0.12
4node_256	14 | 74	60 | 198	256	1.25e+04	881	1.48	199 | 45	26	1.92
4node_512	14 | 74	60 | 198	512	1.39e+04	764	1.98	160 | 31	29	4.08
4node_1024	14 | 74	60 | 198	1024	1.64e+04	691	5.17	182 | 14	29	8.39
4node_2048	14 | 74	60 | 198	2048	2.15e+04	763	11.69	138 | 14	28	18.49
4node_4096	14 | 74	60 | 198	4096	3.14e+04	873	26.02	99 | 14	51	65.65
4node_8192	14 | 74	60 | 198	8192	5.05e+04	912	54.08	80 | 10	59	166.00
4node_16384	14 | 74	60 | 198	16384	8.65e+04	839	103.17	71 | 14	54	305.55
4node_32768	14 | 74	60 | 198	32768	1.55e+05	646	178.71	93 | 40	135	1237.64
4node_base_1024	16 | 74	60 | 198	1024	1.60e+04	501	4.10	111 | 54	68	17.14
4node_base_2048	16 | 74	60 | 198	2048	2.02e+04	501	8.93	163 | 112	49	31.17
4node_base_4096	16 | 74	60 | 198	4096	2.85e+04	501	18.15	170 | 160	55	67.81
4node_base_8192	16 | 74	60 | 198	8192	4.51e+04	811	65.50	265 | 500	229	535.85
4node_base_16384	16 | 74	60 | 198	16384	7.83e+04	931	148.14	234 | 500	156	1390.91
4node_base_32768	16 | 74	60 | 198	32768	1.44e+05	1271	419.45	296 | 500	62	695.07
pltexpA2_6	62 | 104	188 | 272	6	6.50e+04	20000	10.35	19968 | 8	25	0.18
pltexpA2_16	62 | 104	188 | 272	16	1.54e+05	20000	14.93	19988 | 104	24	0.27

					sGS-ADMM			Gurobi	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Obj	Iter	Time(s)	$d_y \vert d_{\bar{y}}$	Iter	Time(s)
stormG2_8	58 | 528	179 | 1377	8	1.83e+07	14401	30.49	14398 | 0	65	0.99
stormG2_27	58 | 528	179 | 1377	27	2.11e+07	8101	26.98	8096 | 0	76	5.44
stormG2_125	58 | 528	179 | 1377	125	3.04e+07	15501	180.01	15494 | 0	98	64.89
gbd	4 | 5	21 | 10	646425	8.21e+08	1211	436.94	1002 | 372	21	167.61

Table 5.1: Comparison of computational result between sGS-ADMM and Gurobi for two-stage quadratic stochastic programming problem. All the run result are obtained using **single thread**. Under the column $d_y|d_{\bar{y}}$, we also record the number of times y and $\bar{y}$ are updated twice.

From table 5.1, we could observe that Gurobi is indeed very efficient for solving the quadratic stochastic programming problems. Overall, we can still say that our algorithm is on comparable term with Gurobi, and it has better performance for 15 out of 32 datasets, particularly for the `phone` and `4node` datasets.

5.5.3 Uncapacitated Facility Location (UFL) Problem

(1) Linear UFL problem

In this subsection, we aim to solve some actual data of UFL problems arising from the literature. We have two benchmark datasets of different scales, namely the small scale `Bilde-Krarup` dataset and the large scale `Koerkel-Ghosh` dataset. Both of these data instances could be downloaded from the UflLib at `http://resources.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/index.html`. These are all linear UFL problem, i.e. $Q_j = 0 \; \forall j = 1, ..., q$ in equation (5.6).

We first solve the linear relaxation of the UFL problem by relaxing the integrality constraint in (5.5), i.e. for all $i = 1, ..., p$, we replace the constraint $u_i \in \{0, 1\}$ by $u_i \in [0, 1]$. Since the nonsmooth objective does not exist in this case, we would employ the sGS-ALM algorithm to solve the linear relaxation instead. The numerical results obtained by sGS-ALM will be compared against Gurobi and reported in table 5.2.

Next, we solve the SDP relaxation (5.6) of the UFL problem arised from relaxing the uncapacitated facility location problem (5.5) using sGS-ADMM. We would compare our numerical results against MOSEK. MOSEK is one of the state-of-the-art SDP solver for solving general SDP problems. When MOSEK is employed to solve the dual block angular problem, we input the left hand side matrix in the constraint without assuming any special structure.

In both tables, we would also report the optimal objective function value of the original integer problem and the resulting relaxation gap.

				Original IP	LP relaxation via sGS-ALM					LP relaxation via Gurobi			
Data	$m_0 \mid m_i$	$n_0 \mid n_i$	N	Obj	Obj	Gap	Iter	Time(s)	$d_y \mid d_{\bar{y}}$	Obj	Gap	Iter	Time(s)
B1.1	0 | 51	50 | 100	100	23468.00	23372.97	0.40	3639	2.61	0 | 3639	23368.40	0.42	1429	0.12
B1.10	0 | 51	50 | 100	100	21864.00	21193.70	3.07	2358	1.62	0 | 2358	21192.18	3.07	1406	0.12
C1.1	0 | 51	50 | 100	100	16781.00	16038.66	4.42	2021	1.38	0 | 2021	16036.80	4.43	1003	0.13
C1.10	0 | 51	50 | 100	100	17994.00	17150.08	4.69	1427	0.96	0 | 1427	17149.54	4.69	1191	0.16
E1.1	0 | 51	50 | 100	100	15042.00	13877.78	7.74	1461	1.01	0 | 1461	13876.52	7.75	1015	0.16
E1.10	0 | 51	50 | 100	100	14630.00	13324.01	8.93	1511	1.02	0 | 1511	13324.52	8.92	985	0.17
E5.1	0 | 51	50 | 100	100	32377.00	29288.08	9.54	1011	0.70	0 | 1011	29288.97	9.54	1965	0.19
E5.10	0 | 51	50 | 100	100	34086.00	30123.69	11.62	1731	1.17	0 | 1731	30122.65	11.63	2001	0.18
E10.1	0 | 51	50 | 100	100	46832.00	42496.39	9.26	1411	0.95	0 | 1411	42495.70	9.26	2709	0.20
E10.10	0 | 51	50 | 100	100	47449.00	43179.36	9.00	1791	1.18	0 | 1791	43179.42	9.00	2682	0.21
ga250a-1	0 | 251	250 | 500	250	257957.00	257602.28	0.14	2061	10.12	0 | 2061	257618.98	0.13	970	3.03
ga250b-2	0 | 251	250 | 500	250	275141.00	272715.83	0.88	1251	6.14	0 | 1251	272721.53	0.88	3897	3.56
ga250c-3	0 | 251	250 | 500	250	333662.00	322880.36	3.23	1581	7.71	0 | 1581	322884.50	3.23	15500	3.23
ga500a-4	0 | 501	500 | 1000	500	511047.00	510526.04	0.10	2201	90.41	0 | 2201	510382.61	0.13	3073	17.05
ga500b-5	0 | 501	500 | 1000	500	537482.00	533027.15	0.83	1301	53.05	0 | 1301	532968.81	0.84	12685	17.43
ga500c-1	0 | 501	500 | 1000	500	621360.00	602873.29	2.98	1791	74.01	0 | 1791	602860.99	2.98	44632	19.58
ga750a-2	0 | 751	750 | 1500	750	763674.00	762602.53	0.14	3501	320.01	0 | 3501	762520.15	0.15	5504	67.87
ga750b-3	0 | 751	750 | 1500	750	796130.00	789639.63	0.82	1951	182.52	0 | 1951	789618.46	0.82	22224	56.82
ga750c-4	0 | 751	750 | 1500	750	900044.00	875458.52	2.73	1941	183.79	0 | 1941	875565.93	2.72	82495	59.52
gs250a-1	0 | 251	250 | 500	250	257964.00	257647.60	0.12	2601	14.29	0 | 2601	257640.74	0.13	881	2.55
gs250b-2	0 | 251	250 | 500	250	275675.00	273062.38	0.95	1251	6.61	0 | 1251	273034.93	0.96	4271	2.66
gs250c-3	0 | 251	250 | 500	250	333000.00	321994.92	3.30	1621	8.10	0 | 1621	321988.94	3.31	15368	3.42
gs500a-4	0 | 501	500 | 1000	500	511137.00	510439.57	0.14	2001	81.60	0 | 2001	510369.57	0.15	2770	17.76
gs500b-5	0 | 501	500 | 1000	500	538270.00	533107.87	0.96	1361	54.44	0 | 1361	533098.15	0.96	11974	17.90
gs500c-1	0 | 501	500 | 1000	500	620041.00	601950.27	2.92	1731	71.17	0 | 1731	601986.20	2.91	43477	17.68
gs750a-2	0 | 751	750 | 1500	750	763548.00	762632.87	0.12	3851	330.23	0 | 3851	762529.44	0.13	5631	62.84
gs750b-3	0 | 751	750 | 1500	750	796589.00	790041.76	0.82	1751	155.15	0 | 1751	789935.57	0.84	22439	51.68
gs750c-4	0 | 751	750 | 1500	750	901339.00	875326.27	2.89	1951	168.32	0 | 1951	875410.57	2.88	82798	49.20

Table 5.2: Comparison of computational result for the LP relaxation of UFL problem using sGS-ALM and Gurobi. Under the column $d_y|d_{\bar{y}}$, we also record the number of times y and $\bar{y}$ are updated twice. Under the column 'Gap', the relaxation gap is also reported.

				Original IP	SDP relaxation via sGS-ADMM					SDP relaxation via MOSEK			
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Obj	Obj	Gap	Iter	Time(s)	$d_y \vert d_{\bar{y}}$	Obj	Gap	Iter	Time(s)
B1.1	51 | 51	51 | 100	100	23468.00	23367.68	0.43	1883	3.33	768 | 1883	23296.71	0.73	21	49.71
B1.10	51 | 51	51 | 100	100	21864.00	21192.78	3.07	1911	3.21	379 | 1911	21147.47	3.28	21	48.60
C1.1	51 | 51	51 | 100	100	16781.00	16034.28	4.45	2151	3.82	847 | 2151	16000.42	4.65	30	63.74
C1.10	51 | 51	51 | 100	100	17994.00	17150.73	4.69	1973	3.47	538 | 1973	17113.25	4.89	26	57.58
E1.1	51 | 51	51 | 100	100	15042.00	13876.46	7.75	1969	3.80	586 | 1969	13833.35	8.04	24	53.73
E1.10	51 | 51	51 | 100	100	14630.00	13323.19	8.93	1872	3.64	561 | 1872	13292.16	9.14	31	66.52
E5.1	51 | 51	51 | 100	100	32377.00	29289.55	9.54	1603	3.05	247 | 1603	29223.74	9.74	22	49.99
E5.10	51 | 51	51 | 100	100	34086.00	30122.05	11.63	1854	3.53	365 | 1854	30057.47	11.82	18	43.66
E10.1	51 | 51	51 | 100	100	46832.00	42495.90	9.26	1321	2.52	310 | 1321	42367.00	9.53	20	46.25
E10.10	51 | 51	51 | 100	100	47449.00	43179.16	9.00	1869	3.56	133 | 1869	43082.90	9.20	18	42.99
ga250a-1	251 | 251	251 | 500	250	257957.00	257640.39	0.12	1751	39.20	1570 | 1751	-	-	-	-
ga250b-2	251 | 251	251 | 500	250	275141.00	272733.10	0.88	1646	34.87	1249 | 1646	-	-	-	-
ga250c-3	251 | 251	251 | 500	250	333662.00	322880.76	3.23	1504	31.52	526 | 1504	-	-	-	-
ga500a-4	501 | 501	501 | 1000	500	511047.00	510427.92	0.12	2601	259.07	2150 | 2601	-	-	-	-
ga500b-5	501 | 501	501 | 1000	500	537482.00	532995.96	0.83	2901	282.99	2777 | 2901	-	-	-	-
ga500c-1	501 | 501	501 | 1000	500	621360.00	602860.72	2.98	1796	173.45	809 | 1796	-	-	-	-
ga750a-2	751 | 751	751 | 1500	750	763674.00	762536.78	0.15	3001	765.85	2428 | 3001	-	-	-	-
ga750b-3	751 | 751	751 | 1500	750	796130.00	789652.09	0.81	5601	1402.37	5475 | 5601	-	-	-	-
ga750c-4	751 | 751	751 | 1500	750	900044.00	875561.41	2.72	2197	544.13	1328 | 2197	-	-	-	-
gs250a-1	251 | 251	251 | 500	250	257964.00	257690.01	0.11	2091	45.41	1699 | 2091	-	-	-	-
gs250b-2	251 | 251	251 | 500	250	275675.00	273048.49	0.95	1578	32.31	1263 | 1578	-	-	-	-
gs250c-3	251 | 251	251 | 500	250	333000.00	321990.63	3.31	1707	34.93	736 | 1707	-	-	-	-
gs500a-4	501 | 501	501 | 1000	500	511137.00	510417.22	0.14	3101	313.15	2983 | 3101	-	-	-	-
gs500b-5	501 | 501	501 | 1000	500	538270.00	533119.62	0.96	3401	332.51	3297 | 3401	-	-	-	-
gs500c-1	501 | 501	501 | 1000	500	620041.00	601982.83	2.91	1871	180.30	1002 | 1871	-	-	-	-
gs750a-2	751 | 751	751 | 1500	750	763548.00	762567.38	0.13	3251	829.55	2790 | 3251	-	-	-	-
gs750b-3	751 | 751	751 | 1500	750	796589.00	789952.48	0.83	4601	1154.16	4410 | 4601	-	-	-	-
gs750c-4	751 | 751	751 | 1500	750	901339.00	875413.80	2.88	2608	648.66	1972 | 2608	-	-	-	-

Table 5.3: Comparison of computational result for the SDP relaxation of UFL problem using sGS-ADMM and MOSEK. Under the column $d_y|d_{\bar{y}}$, we also record the number of times y and $\bar{y}$ are updated twice. Under the column 'Gap', the relaxation gap is also reported.

From Table 5.2, we could observe that Gurobi is still very efficient in solving this kind of problem even if the scale is very huge. However, we should emphasize that our main focus is not on its LP relaxation, but on the SDP relaxation. From Table 5.3, we could observe that sGS-ADMM is always the fastest in solving the SDP relaxation of the UFL problem when compared against MOSEK, as expected. The runtime is almost 10-20 times faster than that is achieved by

MOSEK. For the large scale UFL problems, we also tried to use MOSEK to solve them but they often run out of memory in our machine. In addition, due to the inefficiency of MOSEK as observed when small scale UFL dataset is used, we would not report the numerical result of MOSEK for large scale UFL problem.

(2) Quadratic UFL Problem

In this subsection, we would generate some random quadratic UFL problem following a procedure mentioned in Günlük et al. (2007). We first solve the QP relaxation of these qUFL problems with our sGS-ADMM algorithm and Gurobi in Table 5.4, and then solve its SDP relaxation with sGS-ADMM and MOSEK in Table 5.5.

We should take note that MOSEK could not accept the model with both quadratic objective function value and semidefinite constraint, thus we reformulate (5.6) to be a conic programming model with rotated quadratic cone constraint. Assuming we can find the Cholesky factorization for each $Q_j := H_j^T H_j$, we have:

$$
\begin{aligned}
\min \quad & \langle [0; c],\, [\alpha; u] \rangle + \textstyle\sum_{j=1}^{q} \langle C_j,\, S_j \rangle + \sum_{j=1}^{q} t_j \\
\text{s.t} \quad & \begin{bmatrix} 0 \\ u \end{bmatrix} + \begin{bmatrix} e^T & 0^T \\ -I_p & -I_p \end{bmatrix} \begin{bmatrix} S_j \\ Z_j \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad j = 1, \dots q; \\
& u - \mathrm{diag}(U) = 0, \;\; \alpha = 1; \\
& \boldsymbol{U} := \begin{bmatrix} \alpha & u^T \\ u & U \end{bmatrix} \in \mathbb{S}_+^{1+p}, \;\; \boldsymbol{U} \geq 0, \;\; S_j, Z_j \geq 0, \; j = 1, \dots, q; \\
& y_j = H_j S_j, \quad j = 1, \dots q; \\
& \beta_j = 1, \quad \begin{bmatrix} \beta_j \\ t_j \\ y_j \end{bmatrix} \in \mathcal{Q}_j := \{ x \in \mathbb{R}^{p+2} : 2x_1 x_2 \geq \textstyle\sum_{i=3}^{p+2} x_i^2, \; x_1 \geq 0, \; x_2 \geq 0 \}.
\end{aligned}
\tag{5.26}
$$

				QP relaxation via sGS-ADMM				QP relaxation via Gurobi		
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Obj	Iter	Time(s)	$d_y \vert d_{\bar{y}}$	Obj	Iter	Time(s)
randQUFL-m50-n10	0 | 11	10 | 20	50	83.73	151	0.22	2 | 0	83.73	15	0.26
randQUFL-m50-n20	0 | 21	20 | 40	50	68.11	196	0.10	8 | 0	68.11	17	0.24
randQUFL-m50-n30	0 | 31	30 | 60	50	45.78	411	0.14	11 | 0	45.78	16	0.27
randQUFL-m200-n50	0 | 51	50 | 100	200	73.52	754	0.79	9 | 0	73.52	22	0.38
randQUFL-m200-n100	0 | 101	100 | 200	200	46.97	813	1.06	14 | 0	46.97	21	0.56
randQUFL-m200-n150	0 | 151	150 | 300	200	42.00	773	1.40	21 | 0	42.00	30	1.01
randQUFL-m200-n200	0 | 201	200 | 400	200	38.35	1016	2.29	27 | 0	38.35	27	1.36
randQUFL-m1000-n200	0 | 201	200 | 400	1000	79.85	1665	26.34	14 | 0	79.85	30	5.26
randQUFL-m1000-n500	0 | 501	500 | 1000	1000	47.94	3037	119.68	36 | 0	47.94	41	17.59
randQUFL-m1000-n1000	0 | 1001	1000 | 2000	1000	35.29	2559	233.04	51 | 0	35.29	44	37.79
randQUFL-m1000-n2000	0 | 2001	2000 | 4000	1000	24.95	2056	436.06	58 | 0	24.95	40	73.93

Table 5.4: Comparison of computational result for the QP relaxation of random large scale QUFL problem using sGS-ADMM and Gurobi. All the run result are obtained using **single thread**. Under the column $d_y|d_{\bar{y}}$, we also record the number of times y and $\bar{y}$ are updated twice.

				SDP relaxation via sGS-ADMM				SDP relaxation via MOSEK		
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Obj	Iter	Time(s)	$d_y \vert d_{\bar{y}}$	Obj	Iter	Time(s)
randQUFL-m50-n10	11 | 11	11 | 20	50	83.73	101	0.22	5 | 49	83.73	8	0.81
randQUFL-m50-n20	21 | 21	21 | 40	50	68.11	107	0.14	8 | 50	68.11	8	1.33
randQUFL-m50-n30	31 | 31	31 | 60	50	45.78	173	0.20	37 | 92	45.77	11	3.03
randQUFL-m200-n50	51 | 51	51 | 100	200	73.52	402	0.81	11 | 62	73.52	10	108.46
randQUFL-m200-n100	101 | 101	101 | 200	200	46.97	385	1.60	30 | 50	46.97	15	980.51
randQUFL-m200-n150	151 | 151	151 | 300	200	42.00	469	3.35	34 | 40	42.00	16	3536.43
randQUFL-m200-n200	201 | 201	201 | 400	200	38.35	501	6.13	36 | 34	-	-	-
randQUFL-m1000-n200	201 | 201	201 | 400	1000	79.85	611	16.59	28 | 24	-	-	-
randQUFL-m1000-n500	501 | 501	501 | 1000	1000	47.94	1021	109.07	68 | 11	-	-	-
randQUFL-m1000-n1000	1001 | 1001	1001 | 2000	1000	35.29	791	312.37	80 | 10	-	-	-
randQUFL-m1000-n2000	2001 | 2001	2001 | 4000	1000	24.95	651	999.62	111 | 0	-	-	-

Table 5.5: Comparison of computational result for the SDP relaxation of random large scale QUFL problem using sGS-ADMM and MOSEK. All the run result are obtained using **single thread**. Under the column $d_y|d_{\bar{y}}$, we also record the number of times y and $\bar{y}$ are updated twice. '-' means that the numerical experiment is not performed due to the inefficiency.

Table 5.4 shows that Gurobi is still quite efficient in solving the QUFL problems, probably because the quadratic objective is diagonal that makes the decomposition easy. In Table 5.5, our sGS-ADMM algorithm clearly overtakes MOSEK to be the most efficient algorithm in solving the QUFL problems. Although the number of iterations required by MOSEK is small, the runtime needed in each iteration is huge and thus slows down the overall process. For the last five test instances, we do not run MOSEK due to the inefficiency observed in the smaller test instances.

5.5.4 Randomly generated problems

To demonstrate the generality and superiority of our algorithm, we conduct numerical experiment on some randomly generated dual block angular problem in this section. We have two classes of random data sets, namely the DBA-QP and DBA-SDP problems.

(1) DBA-QP

Here we generate some random quadratic problems with dual block angular structure. In particular, we have the following:

$$\theta(x) := \frac{1}{2}\langle x,\, Qx\rangle \;\text{ and }\; \bar{\theta}_i(\bar{x}_i) := \frac{1}{2}\langle \bar{x}_i,\, \bar{Q}_i\bar{x}_i\rangle \;\forall i = 1, ..., N,$$
$$\text{with the bounds } \mathcal{K} := \mathbb{R}^n_+,\; \bar{\mathcal{K}}_i := \mathbb{R}^{n_i}_+ \;\forall i = 1, ..., N.$$

We generate A, B by `MATLAB` command `sprand` with density $10/n_0$. Similarly $\bar{B}$ is generated with the same command with density $10/n_i$. On the other hand, we generate the symmetric positive definite matrices Q and $\bar{Q}_i$ by the `MATLAB` routine `sprandsym(n0,2/n0,1,1)` and `sprandsym(ni,2/ni,1,1)` respectively.

					sGS-ADMM			Gurobi	
Data	$m_0 \mid m_i$	$n_0 \mid n_i$	N	Obj	Iter	$d_y \mid d_{\bar{y}}$	Time(s)	Iter	Time(s)
randQP-m10-n20-N10	10 | 10	20 | 20	10	3.523e+02	104	23| 0	0.22	13	0.25
randQP-m50-n80-N10	50 | 50	80 | 80	10	1.806e+03	151	70| 0	0.16	11	0.29

					sGS-ADMM			Gurobi	
Data	$m_0\|m_i$	$n_0\|n_i$	N	Obj	Iter	$d_y\|d_{\bar{y}}$	Time(s)	Iter	Time(s)
randQP-m100-n200-N10	100 | 100	200 | 200	10	3.707e+03	283	269| 0	0.59	13	0.99
randQP-m200-n300-N10	200 | 200	300 | 300	10	7.048e+03	374	340| 0	4.10	11	4.48
randQP-m500-n800-N10	500 | 500	800 | 800	10	1.770e+04	335	334| 0	6.56	12	53.51
randQP-m100-n200-N50	100 | 100	200 | 200	50	1.713e+04	810	803| 0	4.09	14	1.53
randQP-m200-n300-N50	200 | 200	300 | 300	50	3.362e+04	959	906| 0	10.88	13	6.60
randQP-m500-n800-N50	500 | 500	800 | 800	50	8.494e+04	791	787| 0	60.33	14	53.87
randQP-m100-n200-N100	100 | 100	200 | 200	100	3.460e+04	973	963| 0	8.70	16	3.18
randQP-m200-n300-N100	200 | 200	300 | 300	100	6.687e+04	1023	1007| 0	30.45	14	15.91
randQP-m500-n800-N100	500 | 500	800 | 800	100	1.688e+05	601	580| 0	13.14	15	130.15
randQP-m100-n200-N200	100 | 100	200 | 200	200	6.897e+04	1169	1156| 0	23.76	18	7.71
randQP-m200-n300-N200	200 | 200	300 | 300	200	1.324e+05	1187	1138| 0	73.10	15	35.29
randQP-m500-n800-N200	500 | 500	800 | 800	200	3.369e+05	859	829| 0	34.37	17	312.68
randQP-m1000-n2000-N200	1000 | 1000	2000 | 2000	200	7.003e+05	1326	1320| 0	156.92	17	2314.25
randQP-m2000-n3000-N200	2000 | 2000	3000 | 3000	200	1.351e+06	916	883| 0	217.22	16	8898.54

Table 5.6: Comparison of computational result between sGS-ADMM and Gurobi for randomly generated **QP** problem. All the run result are obtained using **single thread**. Under the column $d_y|d_{\bar{y}}$, we also record the number of times y and $\bar{y}$ are updated twice.

From Table 5.6, we can observe that Gurobi is no longer the most efficient algorithm in solving the general DBA-QP problems. This may be due to the reason that the problem is harder in the sense that quadratic terms Q or $\bar{Q}_i$ is no longer a simple diagonal matrix. In particular, the largest dataset having 402,000 constraints and 603,000 variables is solved within 4 minutes by sGS-ADMM while Gurobi takes more than 2 hours to achieve the required optimality.

(2) DBA-SDP

We also generated a random data set of semidefinite programming problem with dual block angular structure. In particular, we have

$\theta(x) := 0,\ \bar{\theta}_i(\bar{x}_i) := 0\ \forall i = 1, ..., N;$

with bounds $\mathcal{K} := S^n_+,\ \bar{\mathcal{K}}_i := S^{n_i}_+\ \forall i = 1, ..., N;$

$$\text{and mappings } A(x) := \begin{bmatrix} \langle A_1,\, x\rangle \\ \langle A_2,\, x\rangle \\ \vdots \\ \langle A_m,\, x\rangle \end{bmatrix},\ B_i(x) := \begin{bmatrix} \langle B_{i,1},\, x\rangle \\ \langle B_{i,2},\, x\rangle \\ \vdots \\ \langle B_{i,\bar{m}},\, x\rangle \end{bmatrix},\ \bar{B}_i(\bar{x}_i) := \begin{bmatrix} \langle \bar{B}_{i,1},\, \bar{x}_i\rangle \\ \langle \bar{B}_{i,2},\, \bar{x}_i\rangle \\ \vdots \\ \langle \bar{B}_{i,\bar{m}},\, \bar{x}_i\rangle \end{bmatrix}.$$

We generate the matrix representation of the linear mapping A_i for $i = 1, ..., m$ using `MATLAB` routine:

`Ai = sprand(n0,n0,0.2); Ai = Ai*Ai';`.

Similarly, the matrix representation of $B_{i,k}$ and $\bar{B}_{i,k}$ for $i = 1, ..., N$, $k = 1, ..., \bar{m}$ is generated using the same routine except with density $5/n_i$.

					sGS-ADMM			MOSEK	
Data	$m_0 \vert m_i$	$n_0 \vert n_i$	N	Obj	Iter	$d_y \vert d_{\bar{y}}$	Time(s)	Iter	Time(s)
randSDP-m10-n20-N10	10 | 10	20 | 20	10	5.222e+03	2375	2369|2375	6.13	9	1.61
randSDP-m50-n80-N10	50 | 50	80 | 80	10	2.327e+05	236	154| 236	8.15	10	6.68
randSDP-m100-n200-N10	100 | 100	200 | 200	10	3.286e+06	212	110| 212	55.81	11	116.40
randSDP-m200-n300-N10	200 | 200	300 | 300	10	1.646e+07	309	165| 309	243.02	12	1109.49

Table 5.7: Comparison of computational result between sGS-ADMM and MOSEK for randomly generated **SDP** problem. All the run result are obtained using **single thread**. Under the column $d_y|d_{\bar{y}}$, we also record the number of times y and $\bar{y}$ are updated twice.

From Table 5.7, we observe that MOSEK is not efficient in solving the general DBA-SDP problems. Even when there is only 2,200 constraints and 3,300 variables, MOSEK needs about 18 minutes to reach optimality, which is almost 4.5 times slower than our proposed algorithm.

Chapter 6

Conclusions

In this thesis, we have proposed several efficient algorithms to solve various classes of structured optimization problems.

In the first part of the thesis, by making use of the recent advances in ADMM from the work in Sun et al. (2015), Li et al. (2016) and Chen et al. (2017), we proposed a convergent 3-block inexact symmetric Gauss-Seidel-based semi-proximal ADMM algorithm for solving large scale DWD problems. We applied the algorithm successfully to the primal formulation of the DWD model and designed highly efficient routines to solve the subproblems arising in each of the inexact sGS-ADMM iterations. Numerical experiments for the cases when the exponent equals to 1 and 2 demonstrated that our algorithm is capable of solving large scale problems, even when the sample size and/or the feature dimension is huge. In addition, it is also highly efficient while guaranteeing the convergence to optimality.

In the second part of the thesis, we have designed efficient sGS decomposition based ADMM methods for solving convex composite quadratic conic programming problems with a primal block angular structure. Numerical experiments show that our algorithm is especially efficient for large instances with convex quadratic objective functions. As a future project, we plan to implement our algorithm for solving semidefinite programming problems with primal block angular structures. Also, it would be ideal to utilize a good parallel computing and programming platform to implement the algorithm to realize its full potential.

In the last part of the thesis, we have proposed a distributed sGS decomposition based ADMM method for solving general convex composite quadratic conic programming problems with a dual block angular structure. An improvement that could be made to improve the numerical performance might be incorporating the semismooth Newton-CG method for solving the subproblem. In addition, instead of the simple doubly nonnegative relaxations, we could first derive the equivalence copositive conic reformulation of the UFL problems followed by double nonnegative relaxations to tighten the relaxation gap.

There are some open questions that we would like to bring out for discussion. For example, can we accelerate the sGS-ADMM? If so, what's the rate of convergence? Overall, we could see that sGS-ADMM as a newly emerging tool has many potential in application to various optimization model in the literature. In the future, we would be interested to design decomposed based algorithms for other optimization problem with special structure, such as staircase and block triangular structure.

Bibliography

Assad, A. (1978). Multicommodity network flows – A survey. *Networks*, 8:37–91.

Babonneau, F. and Vial, J.-P. (2009). ACCPM with a nonlinear constraint and an active set strategy to solve nonlinear multicommodity flow problems. *Mathematical Programming*, 120:179–210.

Benito, M., Parker, J., Du, Q., Wu, J., Xiang, D., Perou, C. M., and Marron, J. S. (2004). Adjustment of systematic microarray data biases. *Bioinformatics*, 20:105–114.

Birge, J.-R. and Qi, L. (1988). Computing block-angular Karmarkar projections with applications to stochastic programming. *Management Science*, 34:1472–1479.

Castro, J. (2005). Quadratic interior-point methods in statistical disclosure control. *Computational Management Science*, 2:107–121.

Castro, J. (2016). Interior-point solver for convex separable block-angular problems. *Optimization Methods and Software*, 31:1:88–109.

Castro, J. and Cuesta, J. (2011). Quadratic regularizations in an interior-point method for primal block-angular problems. *Mathematical Programming*, 130:415–445.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM : a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(27).

Chatzipanagiotis, N., Dentcheva, D., and Zavlanos, M.-M. (2015). An augmented Lagrangian method for distributed optimization. *Mathematical Programming*, 152:405–434.

Chen, C., He, B., Ye, Y., and Yuan, X. (2016). The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155:57–79.

Chen, L., Li, X.-D., Sun, D.-F., and Toh, K.-C. (2018). On the equivalence of inexact proximal ALM and ADMM for a class of convex composite programming. *arXiv:1803.10803*.

Chen, L., Sun, D. F., and Toh, K. C. (2017). An efficient inexact symmetric gauss-seidel based majorized ADMM for high-dimensional convex composite conic programming. *Mathematical Programming*, 161:237–270.

Choi, I.-C. and Goldfarb, D. (1993). Exploiting special structure in a primal-dual path following algorithm. *Mathematical Programming*, 58:33–52.

Fan, P.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Fazel, M., Pong, T. K., Sun, D. F., and Tseng, P. (2013). Hankel matrix rank minimization with applications to system identification and realization. *SIAM J. Matrix Analysis and Applications*, 34:946–977.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181.

Fischetti, M., Ljubi, I., and Sinnl, M. (2017). Redesigning benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162.

Freund, R. W. (1997). Preconditioning of symmetric, but highly indefinite linear systems. In *Proceedings of 15th IMACS World Congress on Scientific Computation Modelling and Applied Mathematics*, pages 551 – 556. Berlin, Germany.

Gabay, D. and Mercier, B. (1976). A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2:17–40.

Glowinski, R. and Marroco, A. (1975). Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problémes de dirichlet non linéaires. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathámatique et Analyse Numérique*, 9.R2:41–76.

Golub, G. H. and Loan, C. F. V. (1996). *Matix Computation*. John Hopkins University Press, third edition.

Gondzio, J., Sarkissian, R., and Vial, J.-P. (1997). Using an interior point method for the master problem in a decomposition approach. *European Journal of Operational Research*, 101:577 – 587.

Günlük, O., Lee, J., and Weismantel, R. (2007). MINLP strengthening for separable convex quadratic transportation-cost UFL. Technical Report Technical Report RC24213 (W0703-042), Research Division, IBM, Yorktown Heights, NY.

Han, D.-R., Sun, D.-F., and Zhang, L.-W. (2017). Linear rate convergence of the alternating direction method of multipliers for convex composite programming. *Mathematics of Operations Research*, 42:622–637.

Hanasusanto, G.-A. and Kuhn, D. (2017). Conic programming reformulation of two-stage distributionally robust linear programs over Wasserstein balls. *arXiv:1609.07505*.

Hundepool, A., Domingo-Ferrer, J., Franconi, L., Giessing, S., Nordholt, E.-S., Spicer, K., and de Wolf, P.-P. (2012). *Statistical Disclosure Control*. Wiley.

Ito, N., Takeda, A., and Toh, K. C. (2015). A fast united classification algorithm

based on accelerated proximal gradient method. *Journal of Machine Learning Research*.

Kontogiorgis, S., De Leone, R., and Meyer, R.-R. (1996). Alternating direction splitting for block angular parallel optimization. *Journal of Optimization Theory and Applications*, 99:1–29.

Lau, K. K. and Womersley, R. S. (2001). Multistage quadratic stochastic programming. *Journal of Computational and Applied Mathematics*, 129(1):105 – 138. Nonlinear Programming and Variational Inequalities.

Li, X. D., Sun, D. F., and Toh, K. C. (2015). QSDPNAL: A two-phase Newton-CG proximal augmented Lagrangian method for convex quadratic semidefinite programming problems. *arXiv:1512.08872*.

Li, X. D., Sun, D. F., and Toh, K. C. (2016). A Schur complement based semi-proximal ADMM for convex quadratic conic programming and extensions. *Mathematical Programming*, 155:333–373.

Li, X.-D., Sun, D.-F., and Toh, K.-C. (2018a). A block symmetric Gauss-Seidel decomposition theorem for convex composite quadratic programming and its applications. *Mathematical Programming*. in print.

Li, X.-D., Sun, D.-F., and Toh, K.-C. (2018b). A highly efficient semismooth Newton augmented Lagrangian method for solving Lasso problems. *SIAM journal on optimization*, 28:433–458.

Lichman, M. (2013). UCI machine learning repository.

Linderoth, J., Shapiro, A., and Wright, S. (2006). The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, 142(1):215–241.

Liu, X., Parker, J., Fan, C., Perou, C. M., and Marron, J. S. (2009). Visualization of cross-platform microarray normalization. In *Batch Effects and Noise in Microarray Experiments: Sources and Solutions (ed A. Scherer)*, chapter 14, pages 167–181. John Wiley & Sons Ltd, Chichester, UK.

Marron, J. S. and Alonso, A. M. (2014). Overview of object oriented data analysis. *Biometrical Journal*, 56:732–753.

Marron, J. S., Todd, M. J., and Ahn, J. (2007). Distance weighted discrimination. *Journal of the American Statistical Association*, 102:1267–1271.

Mehrotra, S. and Özevin, M.-G. (2007). Decomposition-based interior point methods for two-stage stochastic semidefinite programming. *SIAM journal on optimization*, 18:206–222.

Mehrotra, S. and Özevin, M.-G. (2009a). Decomposition based interior point methods for two-stage stochastic convex quadratic programs with recourse. *Operations Research*, 57:964–974.

Mehrotra, S. and Özevin, M. G. (2009b). On the implementation of interior point decomposition algorithms for two-stage stochastic conic programs. *SIAM J. on Optimization*, 19(4):1846–1880.

Mulvey, J.-M. and Ruszczyński, A. (1992). A diagonal quadratic approximation method for large scale linear programs. *Operations Research Letters*, 12:205–215.

Qiao, X., Zhang, H. H., Liu, Y., Todd, M. J., and Marron, J. S. (2010). Weighted distance weighted discrimination and its asymptotic properties. *Journal of the American Statistical Association*, 105:401–414.

Rockafellar, R.-T. and Wets, R.-J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16:119–147.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333.

Ruszczyński, A. (1989). An augmented Lagrangian decomposition method for block diagonal linear programming problems. *Operations Research Letters*, 8:287–294.

Ruszczyński, A. (1995). On convergence of an augmented Lagrangian decomposition method for sparse convex optimization. *Mathematics of Operations Research*, 20:634–656.

Ruszczyński, A. (1999). Some advances in decomposition methods for stochastic linear programming. *Annals of Operations Research*, 85:153–172.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition.

Schultz, G. and Meyer, R.-R. (1991). An interior point method for block angular optimization. *SIAM Journal on Optimization*, 1:583–602.

Sivaramakrishnan, K. (2010). A parallel interior point decomposition algorithm for block angular semidefinite programs. *Computational Optimization and Applications*, 46:1–29.

Sun, D. F., Toh, K. C., and Yang, L. Q. (2015). A convergent 3-block semiproximal alternating direction method of multipliers for conic programming with 4-type constraints. *SIAM Journal on Optimization*, 25:882–915.

Todd, M.-J. (1988). Exploiting special structure in Karmarkar's linear programming algorithm. *Mathematical Programming*, 41:81–103.

Toh, K. C., Todd, M. J., and Tutuncu, R. H. (1999). SDPT3 – a Matlab software package for semidefinite programming. *Optimization Methods and Software*, 11:545–581.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

Wang, B. and Zou, H. (2015). Another look at DWD: Thrifty Algorithm and Bayes Risk Consistency in RKHS. *arXiv:1508.05913*.

Wei, S., Lee, C., Wichers, L., and Marron, J. S. (2016). Direction-projection-permutation for high dimensional hypothesis tests. *Journal of Computational and Graphical Statistics*, 25(2):549–569.

Zhang, N., Wu, J., and Zhang, L. (2018). A linearly convergent majorized ADMM with indefinite proximal terms for convex composite programming and its applications. *arXiv:1706.01698*.

Zhao, G. (1999). Interior-point methods with decomposition for solving large-scale linear programs. *Optimization Theory and Applications*, 102:169–192.

Zhao, G. (2001). A log-barrier method with Benders decomposition for solving two-stage stochastic programs. *Mathematical Programming*, 90:507–536.

Zhao, G. (2005). A Lagrangian dual method with self-concordant barrier for multi-stage stochastic convex programming. *Mathematical Programming*, 102:1–24.

Zhu, Y. and Ariyawansa, K.-A. (2011). A preliminary set of applications leading to stochastic semidefinite programs and chance-constrained semidefinite programs. *Applied Mathematical Modelling*, 35:2425–2442.