Mathematical Programming Computation (2021) 13:339–413
https://doi.org/10.1007/s12532-020-00192-5

FULL LENGTH PAPER

Managing randomization in the multi-block alternating direction method of multipliers for quadratic optimization

Krešimir Mihić[1,2] · **Mingxi Zhu**[3] · **Yinyu Ye**[4]

Received: 17 April 2019 / Accepted: 24 August 2020 / Published online: 23 September 2020
© The Author(s) 2020

Abstract
The Alternating Direction Method of Multipliers (ADMM) has gained a lot of attention for solving large-scale and objective-separable constrained optimization. However, the two-block variable structure of the ADMM still limits the practical computational efficiency of the method, because one big matrix factorization is needed at least once even for linear and convex quadratic programming. This drawback may be overcome by enforcing a multi-block structure of the decision variables in the original optimization problem. Unfortunately, the multi-block ADMM, with more than two blocks, is not guaranteed to be convergent. On the other hand, two positive developments have been made: first, if in each cyclic loop one randomly permutes the updating order of the multiple blocks, then the method converges in expectation for solving any system of linear equations with any number of blocks. Secondly, such a randomly permuted ADMM also works for equality-constrained convex quadratic programming even when the objective function is not separable. The goal of this paper is twofold. First, we add more randomness into the ADMM by developing a randomly assembled cyclic ADMM (RAC-ADMM) where the decision variables in each block are randomly assembled. We discuss the theoretical properties of RAC-ADMM and show when random assembling helps and when it hurts, and develop a criterion to guarantee that it converges almost surely. Secondly, using the theoretical guidance on RAC-ADMM, we conduct multiple numerical tests on solving both randomly generated and large-scale benchmark quadratic optimization problems, which include continuous, and binary graph-partition and quadratic assignment, and selected machine learning problems. Our numerical tests show that the RAC-ADMM, with a variable-grouping strategy, could significantly improve the computation efficiency on solving most quadratic optimization problems.

Keywords Quadratic Optimization · ADMM · Decomposition · Randomization · Machine learning applications

✉ Krešimir Mihić
kresimir.mihic@oracle.com

Extended author information available on the last page of the article

Mathematics Subject Classification 90C20 · 65K05 · 90-04

1 Introduction

In this paper we consider the linearly constrained convex minimization model with an objective function that is the sum of multiple separable functions and a coupled quadratic function:

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \\ \text{s.t.} \ & \sum_{i=1}^{p} \mathbf{A}_i\,\mathbf{x}_i = \mathbf{b} \\ & \mathbf{x} \in \mathcal{X} \end{aligned} \tag{1}$$

where $\mathbf{H} \in \mathbb{R}^{n\times n}$ is a symmetric positive semidefinite matrix, vector $\mathbf{c} \in \mathbb{R}^n$ and the problem parameters are the matrix $\mathbf{A} = [\mathbf{A}_1, \ldots, \mathbf{A}_p]$, $\mathbf{A}_i \in \mathbb{R}^{m\times d_i}$, $i = 1, 2, \ldots, p$ with $\sum_{i=1}^{p} d_i = n$ and the vector $\mathbf{b} \in \mathbb{R}^m$. The constraint set $\mathcal{X}$ is the Cartesian product of possibly non-convex real, closed, nonempty sets, $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$, where $\mathbf{x}_i \in \mathcal{X}_i \subseteq \mathbb{R}^{d_i}$.

Problem (1) naturally arises from applications such as machine and statistical learning, image processing, portfolio management, tensor decomposition, matrix completion or decomposition, manifold optimization, data clustering and many other problems of practical importance. To solve problem (1), we consider in particular a randomly assembled multi-block and cyclic alternating direction method of multipliers (RAC-ADMM), a novel algorithm with which we hope to mitigate the problem of slow convergence and divergence issues of the classical alternating direction method of multipliers (ADMM) when applied to problems with cross-block coupled variables.

ADMM was originally proposed in 1970's [31,32] and after a long period without too much attention it has recently gained in popularity for a broad spectrum of applications [28,41,44,57,67]. Problems successfully solved by ADMM range from classical linear programming (LP), semidefinite programming (SDP) and quadratically constrained quadratic programming (QCQP) applied to partial differential equations, mechanics, image processing, statistical learning, computer vision and similar problems (for examples see [10,39,45,53,58,70]) to emerging areas such as deep learning [71], medical treatment [81] and social networking [2]. ADMM is shown to be a good choice for problems where high accuracy is not a requirement but a "good enough" solution is needed to be found fast.

Cyclic multi-block ADMM (C-ADMM) is an iterative algorithm that embeds a Gaussian-Seidel decomposition into each iteration of the augmented Lagrangian method (ALM) [36,59]. It consists of a cyclic update of the blocks of primal variables, $\mathbf{x}_i \in \mathcal{X}_i$, $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_p)$, and a dual ascent type update of the variable $\mathbf{y} \in \mathbb{R}^m$, i.e.,

$$\text{C-ADMM} := \begin{cases} \mathbf{x}_1^{k+1} = \arg\min\limits_{\mathbf{x}_1\in\mathcal{X}_1} \{L_\beta(\mathbf{x}_1, \mathbf{x}_2^k, \mathbf{x}_3^k, \ldots, \mathbf{x}_p^k; \mathbf{y}^k)\}, \\ \vdots \\ \mathbf{x}_p^{k+1} = \arg\min\limits_{\mathbf{x}_p\in\mathcal{X}_p} \{L_\beta(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \mathbf{x}_3^{k+1}, \ldots, \mathbf{x}_p; \mathbf{y}^k)\}, \\ \mathbf{y}^{k+1} = \mathbf{y}^k - \beta(\sum_{i=1}^{p} \mathbf{A}_i\,\mathbf{x}_i^{k+1} - \mathbf{b}) \end{cases} \tag{2}$$

where $\beta > 0$ is a penalty parameter of the Augmented Lagrangian function L_β,

$$L_\beta(\mathbf{x}_1, \ldots, \mathbf{x}_p; \mathbf{y}^k) := f(\mathbf{x}) - \mathbf{y}^T \left(\sum_{i=1}^{p} \mathbf{A}_i \, \mathbf{x}_i - \mathbf{b} \right) + \left\| \sum_{i=1}^{p} \mathbf{A}_i \, \mathbf{x}_i - \mathbf{b} \right\|^2. \tag{3}$$

Note that the classical ADMM [31,32] admits only optimization problems that are separable in blocks of variables and with $p = 2$.

Another variant of multi-block ADMM was suggested in [5], where the authors introduce the distributed multi-block ADMM (D-ADMM) for separable problems. The method creates a Dantzig-Wolfe-Benders decomposition structure and sequentially solves a "master" problem followed by solving distributed multi-block "slave" problems. It converts the multi-block problem into an equivalent two-block problem via variable splitting [6] and performs a separate augmented Lagrangian minimization over $\mathbf{x}_i$. The method assumes that the objective function is separable across blocks, $f(\mathbf{x}) = \sum_i f_i(\mathbf{x}_i) + \mathbf{c}^T \mathbf{x}$, and is not provably working for solving problems with non-separable objective functions.

$$\text{D-ADMM} := \begin{cases} \text{Update } \mathbf{x}_i, i = 1, \ldots, p \\ \quad \mathbf{x}_i^{k+1} = \arg\min\limits_{\mathbf{x}_i \in \mathcal{X}_i} f_i(\mathbf{x}_i) - (\mathbf{y}^k)^T (\mathbf{A}_i \, \mathbf{x}_i - \boldsymbol{\lambda}_i^k) + \frac{\beta}{2} \| A_i \, \mathbf{x}_i - \boldsymbol{\lambda}_i^k \|^2 \\ \text{Update } \boldsymbol{\lambda}_i, i = 1, \ldots, p \\ \quad \boldsymbol{\lambda}_i^{k+1} = \mathbf{A}_i \, \mathbf{x}_i^{k+1} - \frac{1}{p} \big(\sum_{j=1}^{p} \mathbf{A}_j \, \mathbf{x}_j^{k+1} - \mathbf{b} \big) \\ \mathbf{y}^{k+1} = \mathbf{y}^k - \frac{\beta}{p} (\sum_{i=1}^{p} \mathbf{A}_i \, \mathbf{x}_i^{k+1} - \mathbf{b}). \end{cases} \tag{4}$$

Because of the variable splitting, the distributed ADMM approach based on (4) increases the number of variables and constraints in the problem, which in turn makes the algorithm not very efficient for large p in practice.

The classical two-block ADMM (Eq. 2 with $p = 2$) and its convergence have been extensively studied in the literature (e.g. [20,22,31,35,54]. However, the two-block variable structure of the ADMM still limits the practical computational efficiency of the method, because one factorization of a large matrix is needed at least once even for linear and convex quadratic programming (e.g., [45,65]). This drawback may be overcome by enforcing a multi-block structure of the decision variables in the original optimization problem. Indeed, due to the simplicity and practical implications of a direct extension of ADMM to the multi-block variant (2), an active research recently has been going on in developing ADMM variants with provable convergence and competitive numerical efficiency and iteration simplicity (e.g. [17,35,37,58]), and on proving global convergence under some special conditions (e.g. [13,24,46,47]). Unfortunately, in general the Cyclic multi-block ADMM, with more than two blocks, is not guaranteed to be convergent even for solving a single system of linear equations, which settled a long-standing open question [15].

Moreover, in contrast to the work on separable convex problems, little work has been done on understanding properties of the multi-block ADMM for (1) with a non-separable convex quadratic or even non-convex objective function. One of the rare works that addresses coupled objectives is [17] where authors describe convergence

properties for non-separable convex minimization problems. A good description of the difficulties of obtaining a rigorous proof is given in [23]. For solving non-convex problems, a rigorous analysis of ADMM is by itself a very hard problem, with only a couple of works being done for generalized, but still limited (by an objective function), separable problems. For examples see [38,40,76,77,82].

Randomization is commonly used to reduce information and computation complexity for solving large-scale optimization problems. Typical examples include Q-Learning or Reinforced Learning, Stochastic Gradient Descent (SGD) for Deep Learning, Randomized Block-Coordinate-Descent (BCD) for convex programming, and so on. Randomization of ADMM has recently become a matter of interest as well. In [68] the authors devised randomly permuted multi-block ADMM (RP-ADMM) algorithm, in which on every cyclic loop the blocks are solved or updated in a randomly permuted order. Surprisingly the algorithm eliminated the divergence example constructed in [15], and RP-ADMM was shown to converge linearly in expectation for solving any square system of linear equations with any number of blocks. Subsequently, in [17] the authors focused on solving the linearly constrained convex optimization with coupled convex quadratic objective, and proved the convergence in expectation of RP-ADMM for the non separable multi-block convex quadratic programming, which is a much broader class of computational problems.

$$\text{RP-ADMM} := \begin{cases} \text{Randomly permute } (1, 2, ..., p) \text{ into } (\sigma_1, \sigma_2, ..., \sigma_p), \\ \text{then solve:} \\ \quad \mathbf{x}_{\sigma_1}^{k+1} = \arg\min\limits_{\mathbf{x}_{\sigma_1} \in \mathcal{X}_{\sigma_1}} \{L_\beta(\mathbf{x}_{\sigma_1}, \mathbf{x}_{\sigma_2}^k, x_{\sigma_3}^k, \ldots, \mathbf{x}_{\sigma_p}^k, \mathbf{y}^k)\}, \\ \quad \vdots \\ \quad \mathbf{x}_{\sigma_p}^{k+1} = \arg\min\limits_{\mathbf{x}_{\sigma_p} \in \mathcal{X}_{\sigma_p}} \{L_\beta(\mathbf{x}_{\sigma_1}^{k+1}, \mathbf{x}_{\sigma_2}^{k+1}, x_{\sigma_3}^{k+1} \ldots, \mathbf{x}_{\sigma_p}, \mathbf{y}^k)\}, \\ \quad \mathbf{y}^{k+1} = \mathbf{y}^k - \beta(\mathbf{A}\,\mathbf{x}^{k+1} - \mathbf{b}). \end{cases} \tag{5}$$

The main goal of the work proposed in this paper is twofold. First, we add more randomness into the ADMM by developing a randomly assembled cyclic ADMM (RAC-ADMM) where the decision variables in each block are randomly assembled. In contrast to RP-ADMM in which the variables in each block are fixed and unchanged, RAC-ADMM randomly assembles new blocks at each cyclic loop. It can be viewed as a decomposition-coordination procedure that decomposes the problem in a random fashion and combines the solutions to small local sub-problems to find the solution to the original large-scale problem. RAC-ADMM, in-line with RP-ADMM, admits multiple blocks with possibly cross-block coupled variables and updates the blocks in the cyclic order. The idea of re-constructing block variables at each cyclic loop was first mentioned in [51], where the authors present a framework for solving discrete optimization problems which decomposes a problem into sub-problems by randomly (without replacement) grouping variables into subsets. Each subset is then used to construct a sub-problem by considering variables outside the subset as fixed, and the sub-problems are then solved in a cyclic fashion. Subsets are constructed once per iteration. The algorithm presented in that paper is a variant of the block coordinate

descent (BCD) method with an addition of methodology to handle a small number of special constraints, which can be seen as a special case of RAC-ADMM. In the current paper we discuss the theoretical properties of RAC-ADMM and show when the additional random assembling helps and when it hurts.

Secondly, using the theoretical guidance on RAC-ADMM, we conduct multiple numerical tests on solving both randomly generated and benchmark quadratic optimization problems, which include continuous, and binary graph-partitioning and quadratic assignment problems, and selected machine learning problems such as linear regression, LASSO, elastic-net, and support vector machine. Our numerical tests show that RAC-ADMM, with a systematic variable-grouping strategy (designate a set of variables always belonging to a same block), could significantly improve the computation efficiency on solving most quadratic optimization problems.

The current paper is organized as follows. In the next section we present RAC-ADMM algorithm and present theoretical results with respect to convergence. Next we discuss the notion of special grouping, thus selecting variables in less-random fashion by analyzing a problem structure, and the use of partial Lagrangian, approaches, which improve convergence speed of the algorithm. In Sect. 3, we present a solver, RACQP, we built that uses RAC-ADMM to address linearly constrained quadratic problems. The solver is implemented in Matlab [50] and the source code available online [61]. The solver's performance is investigated in Sect. 4, where we compare RACQP with commercial solvers, Gurobi [34] and Mosek [55], and the academic OSQP which is a ADMM-based solver developed by [65]. We also consider machine learning problems and compare our general purpose solver with tailored heuristic solutions, Glmnet [30,64] and LIBSVM [14]. The summary of our contributions with concluding remarks is given in Sect. 5.

2 RAC-ADMM

In this section we describe our randomly assembled cyclic alternating direction method of multipliers (RAC-ADMM). We start by presenting the algorithm, then analyze its convergence for linearly constrained quadratic problems, and finalize the section by introducing accelerated procedures that improve the convergence speed of RAC-ADMM by means of a grouping strategy of highly coupled variables and a partial Lagrangian approach. Note that although our analysis of convergence is restricted to quadratic and/or special classes of problems, it serves as a good indicator of the convergence of the algorithm in more general case.

2.1 The algorithm

RAC-ADMM is an algorithm that is applied to solve convex problems (1). The algorithm addresses equality and inequality constraints separately, with the latter converted

into equalities using slack variables, $\mathbf{s}$:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{s}} \; & f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \\ \text{s.t.} \; & \mathbf{A}_{eq}\,\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} = \mathbf{b}_{ineq} \\ & \mathbf{x} \in \mathcal{X}, \mathbf{s} \geq \mathbf{0} \end{aligned} \tag{6}$$

where matrix $\mathbf{A}_{eq} \in \mathbb{R}^{m_e \times n}$ and vector $\mathbf{b}_{eq} \in \mathbb{R}^{m_e}$ describe equality constraints and matrix $\mathbf{A}_{ineq} \in \mathbb{R}^{m_i \times n}$ and the vector $\mathbf{b}_{ineq} \in \mathbb{R}^{m_i}$ describe inequality constraints. Primal variables $\mathbf{x} \in \mathcal{X}$ are in constraint set $\mathcal{X} \subseteq \mathbb{R}^n$ which is the Cartesian product of possibly non-convex real, closed, nonempty sets, and slack variables $\mathbf{s} \in \mathbb{R}_+^{m_i}$. The augmented Lagrangian function used by RAC-ADMM is then defined by

$$\begin{aligned} L_\beta(\mathbf{x}; \mathbf{s}; \mathbf{y}_{eq}; \mathbf{y}_{ineq}) := \; & f(\mathbf{x}) - \mathbf{y}_{eq}^T\big(\mathbf{A}_{eq}\,\mathbf{x} - \mathbf{b}_{eq}\big) - \mathbf{y}_{ineq}^T\big(\mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} - \mathbf{b}_{ineq}\big) \\ & + \tfrac{\beta}{2}\big(\|\,\mathbf{A}_{eq}\,\mathbf{x} - \mathbf{b}_{eq}\,\|^2 + \|\,\mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} - \mathbf{b}_{ineq}\,\|^2\big) \end{aligned} \tag{7}$$

with dual variables $\mathbf{y} \in \mathbb{R}^{m_e}$ and $\mathbf{z} \in \mathbb{R}^{m_i}$, and penalty parameter $\beta > 0$. In (6) we keep inequality and equality constraint matrices separate so to underline a separate slack variable update step of (8) which has a close form solution described in more details in Sect. 3.

RAC-ADMM is an iterative algorithm that embeds a Gaussian-Seidel decomposition into each iteration of the augmented Lagrangian method (ALM). It consists of a cyclic update of randomly constructed blocks† of primal variables, $\mathbf{x}_i \in \mathcal{X}_i$, followed by the update of slack variables $\mathbf{s}$ and a dual ascent type update for Lagrange multipliers $\mathbf{y}_{eq}$ and $\mathbf{y}_{ineq}$:

$$\text{RAC-ADMM} := \begin{cases} \text{Randomly (without replacement) assemble primal} \\ \text{variables } \mathbf{x}^\dagger \text{ into } p \text{ blocks, } \mathbf{x}_i, i = 1, \ldots, p, \text{ then solve :} \\ \quad \mathbf{x}_1^{k+1} = \arg\min\limits_{\mathbf{x}_1 \in X_1} \{L_\beta(\mathbf{x}_1, \mathbf{x}_2^k, \ldots, \mathbf{x}_p^k; \mathbf{s}^k; \mathbf{y}_{eq}^k; \mathbf{z}_{ineq}^k)\}, \\ \quad \vdots \\ \quad \mathbf{x}_p^{k+1} = \arg\min\limits_{\mathbf{x}_p \in X_p} \{L_\beta(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \ldots, \mathbf{x}_p; \mathbf{s}^k; \mathbf{y}_{eq}^k; \mathbf{z}_{ineq}^k)\}, \\ \quad \mathbf{s}^{k+1} = \arg\min\limits_{\mathbf{s} \geq \mathbf{0}} \{L_\beta(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \ldots, \mathbf{x}_p^{k+1}; \mathbf{s}; \mathbf{y}_{eq}^k; \mathbf{z}_{ineq}^k)\}, \\ \quad \mathbf{y}_{eq}^{k+1} = \mathbf{y}_{eq}^k - \beta(\mathbf{A}_{eq}\,\mathbf{x}^{k+1} - \mathbf{b}_{eq}), \\ \quad \mathbf{y}_{ineq}^{k+1} = \mathbf{y}_{ineq}^k - \beta(\mathbf{A}_{ineq}\,\mathbf{x}^{k+1} + \mathbf{s}^{k+1} - \mathbf{b}_{ineq}). \end{cases}$$

† Structure of a problem, if known, can be used to guide grouping as described in Sect. 2.3.1

(8)

Randomly assembled cyclic alternating direction method of multipliers (RAC-ADMM), can be seen as a generalization of cyclic ADMM, i.e. cyclic multi-block ADMM is a special case of RAC-ADMM in which the blocks are constructed at each iteration using a deterministic rule and optimized following a fixed block order. Using

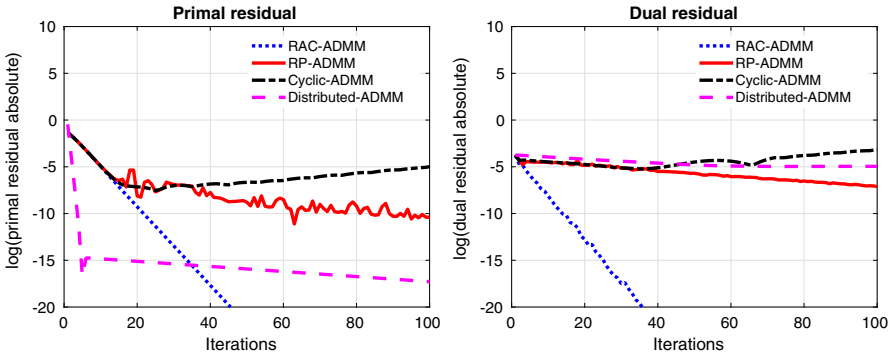


Fig. 1 Iteration evolution of primal and dual residuals of ADMM variants

the same analogy, RP-ADMM can be seen as a special case of RAC-ADMM, in which blocks are constructed using some predetermined rule and kept fixed at each iteration, but sub-problems (i.e. blocks minimizing primal variables) are solved in a random order.

The main advantage of RAC-ADMM over other multi-block ADMM variants is in its potential to significantly reduce primal and, especially, dual residuals, which is a common obstacle for applying multi-block ADMMs. Intuitively, switching variables between the blocks increases chances of finding descent directions which favor RAC-ADMM. The following example further explains such intuition.

Example 1 Consider the problem

$$\begin{aligned} \min_{x,y,u,v} \quad & x^2 - xy + u^2 - uv \\ \text{s.t.} \quad & (x, y, u, v) \geq 0 \end{aligned}$$

Starting from the origin with two blocks, if we group (x, u) and (y, v) then we cannot minimize further. But if we group (x, y) and (u, v) then we can find the problem is unbounded from below. Thus, re-grouping the variables gives RAC-ADMM higher chances of finding (better) descent direction(s), which consequently leads to a better performance for dual residuals.

To illustrate this feature we ran a simple experiment in which we fix the number of iterations and check the final residuals among the aforementioned multi-block ADMM variants. In Table 1 we show performance of the ADMMs when solving a simple quadratic problem with a single constraint, represented by a regularized Markowitz min–variance problem (defined in Sect. 4.1.3). Figure 1 gives the insight in evolution of the both residuals with iterations. From the figure, it is noticeable that both D-ADMM (Eq. 4) and RP-ADMM (Eq. 5) suffer from a very slow convergence speed, with the main difference that the latter gives a slightly lower error on dual residual. Multi-block Cyclic-ADMM (Eq. 2) does not converge to a KKT point for any k, but oscillates around a very weak solution. RAC-ADMM converges to the KKT solution very quickly with both residual errors below 10^{-8} in less than 40 iterations.

Table 1 Primal and dual residuals of the result returned by ADMM variants after k iterations for a randomly generated Markowitz min–variance problem

ADMM Variant	$k = 10$ iterations		$k = 50$ iterations		$k = 100$ iterations	
	Primal	Dual	Primal	Dual	Primal	Dual
RAC-ADMM (8)	$7.2 \cdot 10^{-3}$	$3.1 \cdot 10^{-4}$	$3.0 \cdot 10^{-10}$	$4.6 \cdot 10^{-12}$	$1.2 \cdot 10^{-14}$	$4.4 \cdot 10^{-16}$
RP-ADMM (5)	$7.4 \cdot 10^{-3}$	$1.0 \cdot 10^{-2}$	$2.0 \cdot 10^{-4}$	$3.3 \cdot 10^{-3}$	$4.3 \cdot 10^{-5}$	$6.8 \cdot 10^{-4}$
C-ADMM (2)	$7.4 \cdot 10^{-3}$	$1.2 \cdot 10^{-2}$	$6.8 \cdot 10^{-4}$	$4.9 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$	$2.5 \cdot 10^{-2}$
D-ADMM (4)	$3.7 \cdot 10^{-6}$	$1.8 \cdot 10^{-2}$	$1.2 \cdot 10^{-6}$	$8.0 \cdot 10^{-3}$	$3.1 \cdot 10^{-7}$	$6.2 \cdot 10^{-3}$

$n = 3000$, $p = 50$, $\beta = 1$

2.2 Convergence of RAC-ADMM

This section concerns with convergence properties of RAC-ADMM when applied to unbounded (i.e. $\mathbf{x} \in \mathbb{R}^n$) linearly-equality constrained quadratic optimization problems. To simplify the notation, we use $A = \mathbf{A}_{eq}$ and $\mathbf{b} = \mathbf{b}_{eq}$.

$$\begin{aligned} \min_{x} \ & \tfrac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \\ \text{s.t.} \ & \mathbf{A}\,\mathbf{x} = \mathbf{b} \end{aligned} \tag{9}$$

with $\mathbf{H} \in \mathbb{R}^{n\times n}$, $\mathbf{H} \succeq 0$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{x} \in \mathbb{R}^n$.

Convergence analysis of problems that include inequalities (bounds on variables and/or inequality constraints) is still an open question and will be addressed in our subsequent work.

2.2.1 Preliminaries

Double Randomness Interpretation Let $\Gamma_{RAC(n,p)}$ denote all possible updating combinations for RAC with n variables and p blocks, and let $\sigma_{RAC} \in \Gamma_{RAC(n,p)}$ denote one specific updating combination for RAC-ADMM. Then the total number of updating combinations for RAC-ADMM is given by

$$|\Gamma_{RAC(n,p)}| = \frac{n!}{(s!)^p}$$

where $s \in \mathbb{Z}_+$ denotes size of each block with $p \cdot s = n$.

RAC-ADMM could be viewed as a double-randomness procedure based on RP-ADMM with different block compositions. Let $\sigma_{RP} \in \Gamma_{RP(p)}$ denote an updating combinations of RP-ADMM with p blocks where the variable composition in each block is fixed. Clearly, the total number of updating combinations for RP-ADMM is given by

$$|\Gamma_{RP(p)}| = p!$$

the total number of possible updating orders of the p blocks. Then, one may consider RAC-ADMM first randomly chooses a block composition and then applies RP-ADMM. Let $\upsilon_i \in \Upsilon(n, p)$ denote one specific block composition or partition of n decision variables into p blocks, where $\Upsilon(n, p)$ is the set of all possible block compositions. Then, the total number of all possible block compositions is given by

$$|\Upsilon(n, p)| = \frac{|\Gamma_{RAC(n,p)}|}{|\Gamma_{RP(p)}|} = \frac{n!}{(s!)^p\, p!}$$

For convenience, in what follows let $\Gamma_{RP(p),\upsilon_i}$ denote all possible updating orders with a fixed block composition υ_i. To further illustrate the relations of RP-ADMM and RAC-ADMM, consider the following simple example.

Example 2 Let $n = 6$, $p = 3$, so $|\Gamma_{RP(6,3)}| = 3! = 6$, and the total number of block compositions or partitions is 15:

$$\begin{aligned}\upsilon_i \in \Upsilon(6,3) = \Big\{ & \{[x_1,x_2],[x_3,x_4],[x_5,x_6]\}, \{[x_1,x_2],[x_3,x_5],[x_4,x_6]\},\\ & \{[x_1,x_2],[x_3,x_6],[x_4,x_5]\}, \{[x_1,x_3],[x_2,x_4],[x_5,x_6]\},\\ & \{[x_1,x_3],[x_2,x_5],[x_4,x_6]\}, \{[x_1,x_3],[x_2,x_6],[x_4,x_5]\},\\ & \{[x_1,x_4],[x_2,x_3],[x_5,x_6]\}, \{[x_1,x_4],[x_2,x_5],[x_3,x_6]\},\\ & \{[x_1,x_4],[x_2,x_6],[x_3,x_5]\}, \{[x_1,x_5],[x_2,x_3],[x_4,x_6]\},\\ & \{[x_1,x_5],[x_2,x_4],[x_3,x_6]\}, \{[x_1,x_5],[x_2,x_6],[x_3,x_4]\},\\ & \{[x_1,x_6],[x_2,x_3],[x_4,x_5]\}, \{[x_1,x_6],[x_2,x_4],[x_3,x_5]\},\\ & \{[x_1,x_6],[x_2,x_5],[x_3,x_4]\}\Big\}\end{aligned}$$

RAC-ADMM could be viewed as if, at each cyclic loop, the algorithm first selects a block composition υ_i uniformly random from all possible 15 block compositions $\Upsilon(n, p)$, and then performs RP-ADMM with the chosen specific block composition υ_i. In other words, RAC-ADMM then randomly selects $\sigma \in \Gamma_{RP(p),\upsilon_i}$, which leads to a total of 90 possible updating combinations.

RAC-ADMM as a linear transformation Recall that the augmented Lagrangian function for (9) is given by

$$L_\beta(\mathbf{x},\mathbf{y}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} - \mathbf{y}^T(\mathbf{A}\mathbf{x}-\mathbf{b}) + \frac{1}{2}\beta||\mathbf{A}\mathbf{x}-\mathbf{b}||^2.$$

Consider one specific update order generated by RAC, $\sigma_{RAC} \in \Gamma_{RAC(n,p)}$. Note that we use σ instead σ_{RAC} when there is no confusion. One possible update combination generated by RAC, $\sigma = [\sigma_1, \ldots, \sigma_p]$, where σ_i is an index vector of size s, is as follows,

$$\text{RAC-ADMM}_{k+1} = \begin{cases} \mathbf{x}_{\sigma 1}^{k+1} = \arg\min\limits_{\mathbf{x}_{\sigma 1}}\{L_\beta(\mathbf{x}_{\sigma 1}, \mathbf{x}_{\sigma 2}^k, \ldots, \mathbf{x}_{\sigma p}^k; \mathbf{y}^k)\},\\ \vdots \\ \mathbf{x}_{\sigma p}^{k+1} = \arg\min\limits_{\mathbf{x}_{\sigma p}}\{L_\beta(\mathbf{x}_{\sigma 1}^{k+1}, \mathbf{x}_{\sigma 2}^{k+1}, \ldots, \mathbf{x}_{\sigma p}; \mathbf{y}^k)\},\\ \mathbf{y}^{k+1} = \mathbf{y}^k - \beta(\mathbf{A}\mathbf{x}^{k+1} - \mathbf{b}). \end{cases}$$

For convenience, we follow the notation in [17] and [68,69] to describe the iterative scheme of RAC-ADMM in a matrix form. Let $\mathbf{L}_\sigma \in \mathbb{R}^{n\times n}$ be $s \times s$ block matrix defined with respect to σ_i rows and σ_j columns as

$$(\mathbf{L}_\sigma)_{\sigma_i,\sigma_j} := \begin{cases} \mathbf{H}_{\sigma_i,\sigma_j} + \beta\,\mathbf{A}_{\sigma_i}^T\mathbf{A}_{\sigma_j}, & i \geq j\\ \mathbf{0}, & \text{otherwise} \end{cases} \tag{10}$$

and let $\mathbf{R}_\sigma$ be defined as

$$\mathbf{R}_\sigma := \mathbf{L}_\sigma - (\mathbf{H} + \beta\,\mathbf{A}^T\mathbf{A}).$$

By setting $\mathbf{z} := (\mathbf{x}; \mathbf{y})$, RAC-ADMM could be viewed as a linear system mapping iteration

$$\mathbf{z}^{k+1} := \mathbf{M}_\sigma \, \mathbf{z}^k + \bar{\mathbf{L}}_\sigma^{-1} \bar{\mathbf{b}}$$

where

$$\mathbf{M}_\sigma := \bar{\mathbf{L}}_\sigma^{-1} \bar{\mathbf{R}}_\sigma \tag{11}$$

and

$$\bar{\mathbf{L}}_\sigma := \begin{bmatrix} \mathbf{L}_\sigma & \mathbf{0} \\ \beta \, \mathbf{A} & \mathbf{I} \end{bmatrix} \bar{\mathbf{R}}_\sigma := \begin{bmatrix} \mathbf{R}_\sigma & \mathbf{A}^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \bar{\mathbf{b}} := \begin{bmatrix} -\mathbf{c} + \beta \, \mathbf{A}^T \, \mathbf{b} \\ \beta \, \mathbf{b} \end{bmatrix}$$

Define the matrix $\mathbf{Q}$ by

$$\begin{aligned} \mathbf{Q} := \mathbb{E}_\sigma[\mathbf{L}_\sigma^{-1}] &= \tfrac{1}{|\Gamma_{RAC(n,p)}|} \textstyle\sum_{\sigma \in \Gamma_{RAC(n,p)}} \mathbf{L}_\sigma^{-1} \\ &= \tfrac{1}{|\Upsilon(n,p)|} \textstyle\sum_{\upsilon_i \in \Upsilon(n,p)} \left\{ \tfrac{1}{p!} \textstyle\sum_{\sigma \in \Gamma_{RP(p),\upsilon_i}} \mathbf{L}_\sigma^{-1} \right\} \end{aligned}$$

Notice that for any block structure υ_i any update order within this fixed block structure $\sigma \in \Gamma_{RP(p),\upsilon_i}$, we have $\mathbf{L}_\sigma^T = \mathbf{L}_{\bar{\sigma}}$, where $\bar{\sigma}$ is a reverse permutation of $\sigma \in \Gamma_{RP(p),\upsilon_i}$. Specifically, let $\sigma = [\sigma_1, \ldots, \sigma_p]$, we have $\bar{\sigma} = [\bar{\sigma}_1, \ldots, \bar{\sigma}_p]$, and $\bar{\sigma}_i = \sigma_{p+1-i}$. For a specific fixed block structure υ_i, define matrix $\mathbf{Q}_{\upsilon_i}$ as

$$\mathbf{Q}_{\upsilon_i} := \mathbb{E}[\mathbf{L}_{\sigma_i}^{-1} \, | \upsilon_i] = \frac{1}{p!} \sum\nolimits_{\sigma_i \in \Gamma_{RP(n,\upsilon_i)}} \mathbf{L}_{\sigma_i}^{-1},$$

and because $\mathbf{L}_\sigma^T = \mathbf{L}_{\bar{\sigma}}$, matrix $\mathbf{Q}_{\upsilon_i}$ is symmetric for all i, and

$$\mathbf{Q} := \frac{1}{|\Upsilon(n, p)|} \sum\nolimits_{\upsilon_i \in \Upsilon(n,p)} \mathbf{Q}_{\upsilon_i} \tag{12}$$

Finally, the expected mapping matrix $\mathbf{M}$ is given by

$$\mathbf{M} := \mathbb{E}_\sigma[\mathbf{M}_\sigma] = \frac{1}{|\Gamma_{RAC(n,p)}|} \sum\nolimits_{\sigma \in \Gamma_{RAC(n,p)}} \mathbf{M}_\sigma$$

or, by direct computation,

$$\mathbf{M} := \begin{bmatrix} \mathbf{I} - \mathbf{Q}\,\mathbf{S} & \mathbf{Q}\,\mathbf{A}^T \\ \beta(\mathbf{A}\,\mathbf{Q}\,\mathbf{S} - \mathbf{A}) & \mathbf{I} - \beta\,\mathbf{A}\,\mathbf{Q}\,\mathbf{A}^T \end{bmatrix} \tag{13}$$

where $\mathbf{S} = \mathbf{H} + \beta \, \mathbf{A}^T \, \mathbf{A}$.

2.2.2 Expected convergence of RAC-ADMM

With the preliminaries defined, we are now ready to show that RAC-ADMM converges in expectation under the following assumption:

Assumption 1 *Assume that for any block of indices σ_i that generated by RAC-ADMM*

$$\mathbf{H}_{\sigma_i,\sigma_i} + \beta\, \mathbf{A}_{\sigma_i}^T\, \mathbf{A}_{\sigma_i} \succ 0$$

where σ_i is the index vector describing indices of primal variables of the block i.

Theorem 2 *Suppose that Assumption (1) holds, and that RAC-ADMM (8) is employed to solve problem (9). Then the expected output converges to some KKT point of (9).*

Theorem 2 suggests that the expected output converges to some KKT point of (9). Such convergence in expectation criteria has been widely used in many randomized algorithms, including convergence analysis for RP-BCD and RP-ADMM (e.g. [16, 68]), and stochastic quasi-newton methods (e.g. [12]). It is worth mentioning that if the optimization problem is strictly convex (H $>$ 0), we are able to prove that the expected mapping matrix has spectrum that is strictly less than 1 (Corollary 1).

Although convergence in expectation is widely used in many literature, it is still a relatively weak convergence criteria. This is why in Sect. 2.2.4 we propose a sufficient condition for almost surely convergence of RAC-ADMM. The section also provides an example showing a problem with $\rho(M) < 1$ which does not converge. Rather it oscillates almost surely (Example 3). To the best of our knowledge, this is the first example showing that even if a randomized optimization algorithm has expected spectrum radius *strictly less than 1*, the algorithm may still oscillate—to construct an example with expected spectrum radius equals to 1 that does not converge is an easy task. Consider for example a sequence $\{x_t, t \geq 0\}$ with $x_t = -1$ and $x_t = 1$, chosen with equal probabilities (prob=1/2). Then, the sequence does not converge with probability 1. However, under the such example, the expected spectrum of this mapping procedure $\rho(M)$ actually equals to 1, which implies that the sequence may not converge. Despite the fact that such example exists for RAC-ADMM, in all the numerical tests provided in Sect. 4, RAC-ADMM converges to the KKT point of the optimization problem under few iterations. Such strong numerical evidences imply that in practice, our algorithm does not require taking expectation over many iterations to converge.

The proof of Theorem 2 follows the proof structure of [17,68,69] to show that under Assumption 1:

(1) $\text{eig}(\mathbf{Q}\,\mathbf{S}) \in [0, \frac{4}{3})$;
(2) $\forall \lambda \in \text{eig}(\mathbf{M}), \text{eig}(\mathbf{Q}\,\mathbf{S}) \in [0, \frac{4}{3}) \implies \|\lambda\| < 1$ or $\lambda = 1$;
(3) if $1 \in \text{eig}(\mathbf{M})$, then the eigenvalue 1 has a complete set of eigenvectors;
(4) Steps (2) and (3) imply the convergence in expectation of the RAC-ADMM.

The proof builds on Theorem 2 from [17], which describes RP-ADMM convergence in expectation under specific conditions put on matrices H and A, and Weyl's inequality, which gives the upper bound on maximum eigenvalue and the lower bound on minimum eigenvalue of a sum of Hermitian matrices. Proofs for items (2) and (3) are identical to proofs given in [17, Section 3.2], so here the focus in on proving item (1). The following lemma completes the proof of expected convergence of RAC.

Lemma 1 *Under Assumption 1, the matrix Q is positive definite, and*

$$eig(\mathbf{Q}\,\mathbf{S}) \subset [0, \frac{4}{3})$$

To prove Lemma 1, we first show that for any block structure υ_i, the following proposition holds:

Proposition 1 $\mathbf{Q}_{\upsilon_i}\,\mathbf{S}$ *is positive semi-definite and symmetric, and*

$$eig(\mathbf{Q}_{\upsilon_i}\,\mathbf{S}) \subseteq [0, \frac{4}{3})$$

Intuitively, a different block structure of RAC-ADMM iteration could be viewed as relabeling variables and performing RP-ADMM procedure as described in [17].

Proof Define block structure $\{[x_1, \ldots, x_s], [x_{s+1}, \ldots, x_{2s}], [x_{(p-1)s+1}, \ldots, x_{ps}]\}$ as υ_1. For any block structure υ_i, there exists $\tilde{\mathbf{S}}$ and $\tilde{\mathbf{Q}}_{\upsilon_1}$ s.t.

$$\text{eig}(\mathbf{Q}_{\upsilon_i}\,\mathbf{S}) = \text{eig}(\tilde{\mathbf{Q}}_{\upsilon_1}\tilde{\mathbf{S}})$$

where $\tilde{\mathbf{Q}}_{\upsilon_1}$ represents formulation of $\mathbb{E}_\sigma[\mathbf{L}_\sigma^{-1}]$ matrix with respect to block structure υ_1 and matrix $\tilde{\mathbf{S}}$. To prove this, we introduce permutation matrix $\mathbf{P}_{\upsilon_1 \to \upsilon_i}$ as follows. Given

$$\begin{aligned}\upsilon_1 &= \{[1, \ldots, s], [s+1, \ldots, 2s], [(p-1)s+1, \ldots, ps]\}\\ \upsilon_i &= \{[\pi(1), \ldots, \pi(s)], [\pi(s+1), \ldots, \pi(2s)], [\pi((p-1)s+1), \ldots, \pi(ps)]\}\end{aligned}$$

define

$$\mathbf{P}_{\upsilon_i} = \begin{bmatrix} \mathbf{e}_{\pi(1)} \\ \mathbf{e}_{\pi(2)} \\ \vdots \\ \mathbf{e}_{\pi(ps)} \end{bmatrix}$$

Where $\mathbf{e}_i$ is the row vector with ith element equal to 1. Notice $\mathbf{P}_{\upsilon_i}$ is orthogonal matrix for any υ_i, i.e. $\mathbf{P}_{\upsilon_i}\,\mathbf{P}_{\upsilon_i}^T = I$. For any fixed block structure υ_i, with an update order within $\sigma_{RP} \in \Gamma_{RP}(p)$, the following equality holds

$$\mathbf{L}_{\sigma_{RP},\mathbf{S},\upsilon_i} = \mathbf{P}_{\upsilon_i}^T\, L_{\sigma_{RP},\tilde{\mathbf{S}},\upsilon_1}\,\mathbf{P}_{\upsilon_i}$$

where $\mathbf{L}_{\sigma_{RP},\mathbf{S},\upsilon_i}$ is the construction of $\mathbf{L}$ following update order $\sigma_{RP} \in \Gamma_{RP}(p)$ and block structure υ_i with respect to $\mathbf{S}$, and $\mathbf{L}_{\sigma_{RP},\tilde{\mathbf{S}},\upsilon_1}$ is the construction of $\mathbf{L}$ following update order $\sigma_{RP} \in \Gamma_{RP}(p)$ and block structure υ_1, with coefficient matrix $\tilde{\mathbf{S}}$, and

$$\tilde{\mathbf{S}} = \mathbf{P}_{\upsilon_i}\,\mathbf{S}\,\mathbf{P}_{\upsilon_i}^T$$

and

$$\mathbf{L}^{-1}_{\sigma,\mathbf{S},\upsilon_i} = (\mathbf{P}^T_{\upsilon_i}\,\mathbf{L}_{\sigma,\tilde{\mathbf{S}},\upsilon_1}\,\mathbf{P}_{\upsilon_i})^{-1} = \mathbf{P}^T_{\upsilon_i}\,\mathbf{L}^{-1}_{\sigma,\tilde{\mathbf{S}},\upsilon_1}\,\mathbf{P}_{\upsilon_i}\,.$$

Then by the definition of $\mathbf{Q}$ matrix (Eq. 12), we get

$$\mathbf{Q}_{\upsilon_i,\mathbf{S}} = \mathbf{P}^T_{\upsilon_i}\,\tilde{\mathbf{Q}}_{\upsilon_1,\tilde{\mathbf{S}}}\,\mathbf{P}_{\upsilon_i}$$

so that

$$\mathbf{Q}_{\upsilon_i,\mathbf{S}}\,\mathbf{S} = \mathbf{P}^T_{\upsilon_i}\,\tilde{\mathbf{Q}}_{\upsilon_1,\tilde{\mathbf{S}}}\,\mathbf{P}_{\upsilon_i}\,\mathbf{P}^{-1}_{\upsilon_i}\,\tilde{\mathbf{S}}\,\mathbf{P}_{\upsilon_i} = \mathbf{P}^T_{\upsilon_i}\,\tilde{\mathbf{Q}}_{\upsilon_1,\tilde{\mathbf{S}}}\tilde{\mathbf{S}}\,\mathbf{P}_{\upsilon_i}\,.$$

Considering the eigenvalues of $\mathbf{Q}_{\upsilon_i,\mathbf{S}}\,\mathbf{S}$,

$$\mathrm{eig}(\mathbf{Q}_{\upsilon_i,\mathbf{S}}\,\mathbf{S}) = \mathrm{eig}(\mathbf{P}^T_{\upsilon_i}\,\tilde{\mathbf{Q}}_{\upsilon_1,\tilde{\mathbf{S}}}\tilde{\mathbf{S}}\,\mathbf{P}_{\upsilon_i}) = \mathrm{eig}(\tilde{\mathbf{Q}}_{\upsilon_1,\tilde{\mathbf{S}}}\tilde{\mathbf{S}})$$

and from [17], under Assumption (1), $\tilde{\mathbf{Q}}_{\upsilon_1,\tilde{\mathbf{S}}}$ is positive definite, and

$$\mathrm{eig}(\tilde{\mathbf{Q}}_{\upsilon_1,\tilde{\mathbf{S}}}\tilde{\mathbf{S}}) \subset [0, \frac{4}{3})$$

which implies $\mathbf{Q}_{\upsilon_i}$ is positive definite, and

$$\mathrm{eig}(\mathbf{Q}_{\upsilon_i}\,\mathbf{S}) \subset [0, \frac{4}{3}).$$

Notice that by definition of $\mathbf{Q}$, we have

$$\mathbf{Q}\,\mathbf{S} = \frac{1}{\Upsilon(d,n)}\sum_{\upsilon_i}\mathbf{Q}_{\upsilon_i}\,\mathbf{S}\,.$$

and since $\mathbf{S}$ is positive definite and symmetric, we could write $\mathbf{S} = \mathbf{B}^T\,\mathbf{B}$, so

$$\mathbf{Q}\,\mathbf{S} = \frac{1}{\Upsilon(d,n)}\sum_{\upsilon_i}\mathbf{Q}_{\upsilon_i}\,\mathbf{B}^T\,\mathbf{B}\,.$$

Because $\frac{1}{\Upsilon(d,n)}\sum_{\upsilon_i}\mathbf{Q}_{\upsilon_i}$ is real symmetric, we have

$$\begin{aligned}\mathrm{eig}(\mathbf{Q}\,\mathbf{S}) &= \mathrm{eig}\left(\tfrac{1}{\Upsilon(d,n)}\textstyle\sum_{\upsilon_i}\mathbf{Q}_{\upsilon_i}\,\mathbf{B}^T\,\mathbf{B}\right)\\ &= \mathrm{eig}(\mathbf{B}(\tfrac{1}{\Upsilon(d,n)}\textstyle\sum_{\upsilon_i}\mathbf{Q}_{\upsilon_i})\,\mathbf{B}^T)\\ &= \mathrm{eig}\left(\tfrac{1}{\Upsilon(d,n)}\textstyle\sum_{\upsilon_i}\mathbf{B}\,\mathbf{Q}_{\upsilon_i}\,\mathbf{B}^T\right)\end{aligned}$$

Let $\lambda_1(\mathbf{A})$ denote the maximum eigenvalue of matrix $\mathbf{A}$, then as all $\mathbf{B}\,\mathbf{Q}_{\upsilon_i}\,\mathbf{B}^T$ are Hermitian matrices, by Weyl's theorem, we have

$$\begin{aligned}\lambda_1(\mathbf{Q}\,\mathbf{S}) &= \lambda_1\left(\tfrac{1}{\Upsilon(d,n)}\textstyle\sum_{\upsilon_i}\mathbf{B}\,\mathbf{Q}_{\upsilon_i}\,\mathbf{B}^T\right)\\ &\le \tfrac{1}{\Upsilon(d,n)}\textstyle\sum_{\upsilon_i}\lambda_1(\mathbf{B}\,\mathbf{Q}_{\upsilon_i}\,\mathbf{B}^T)\\ &= \tfrac{1}{\Upsilon(d,n)}\textstyle\sum_{\upsilon_i}\lambda_1(\mathbf{Q}_{\upsilon_i}\,\mathbf{S})\end{aligned}$$

and as $\lambda_1(\mathbf{Q}_{\upsilon_i}\,\mathbf{S}) < \frac{4}{3}$ for each i,

$$\mathrm{eig}(\mathbf{Q}\,\mathbf{S}) \subseteq \left[0, \frac{4}{3}\right)$$

which completes the proof of Lemma 1, and thus establishes that RAC-ADMM is guaranteed to converge in expectation. □

When the problem is strongly convex ($\mathbf{H} \succ 0$), we introduce the following corollary.

Corollary 1 *Under Assumption 1, and* $\mathbf{H} \succ 0$,

$$\rho(\mathbf{M}) < 1$$

Proof When $\mathbf{H} \succ 0$, by definition $\mathbf{S} = \mathbf{H} + \beta\,\mathbf{A}^T\,\mathbf{A} \succ 0$, and by Lemma 1, $\mathbf{Q} \succ 0$, hence $\mathrm{eig}(\mathbf{Q}\,\mathbf{S}) \subseteq (0, \frac{4}{3})$, and this implies $\rho(\mathbf{M}) < 1$. □

Note that there are random sequences converging in expectation where their spectrum-radius equal to one. Therefore, for solving strongly non-separable convex quadratic optimization, the expected convergence rate of RAC-ADMM is proved to be linear, which result is stronger than just "convergence in expectation".

2.2.3 Convergence speed of RAC-ADMM versus RP-ADMM

Following is a corollary to show that on average or in expectation, RAC-ADMM outperforms RP-ADMM with a fixed block composition in sense of spectral radius of mapping matrix.

Corollary 2 *Under Assumption 1, with* $\mathbf{H} = \mathbf{0}$ *so that* $\mathbf{S} = \beta\,\mathbf{A}^T\,\mathbf{A}$, *where* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *is a non-singular matrix, there exists some RP-ADMM (with specific block compositions), such that expected spectral radius of RAC-ADMM mapping matrix is (weakly) smaller than expected spectral radius of that of RP-ADMM.*

Proof We prove the corollary in solving linear system with A non singular, with null objective function. In this setup, the expected output converges to the unique primal dual optimal solution to (9).

Notice in this setup, we have

$$\lambda \in \mathrm{eig}(\mathbf{M}) \Leftrightarrow \tau = \frac{(1-\lambda)^2}{1-2\lambda} \in \mathrm{eig}(\mathbf{Q}\,\mathbf{A}^T\,\mathbf{A})$$
$$\lambda_{\upsilon_i} \in \mathrm{eig}(\mathbf{M}_{RP,\upsilon_i}) \Leftrightarrow \tau_{\upsilon_i} = \frac{(1-\lambda_{\upsilon_i})^2}{1-2\lambda_{\upsilon_i}} \in \mathrm{eig}(\mathbf{Q}_{\upsilon_i}\,\mathbf{A}^T\,\mathbf{A})$$

By calculation, we could characterize λ as roots of quadratic polynomial [69],

$$\lambda_1 = 1-\tau+\sqrt{\tau(\tau-1)}, \quad \lambda_2 = 1-\tau-\sqrt{\tau(\tau-1)}.$$

Suppose corollary doesn't hold, $\rho(\mathbb{E}[\mathbf{M}_{RAC}]) \geq \rho(\mathbb{E}[\mathbf{M}_{RP,\upsilon_i}])$ for all possible block structure. Define $\underline{\tau}_{\upsilon_i}$ as the the smallest eigenvalue with respect to $\mathbf{Q}_{\upsilon_i}\,\mathbf{S}$, and $\bar{\tau}_{\upsilon_i}$ as the largest eigenvalue with respect to $\mathbf{Q}_{\upsilon_i}\,\mathbf{S}$. Similarly, $\underline{\tau}$ as the smallest eigenvalue with respect to $\mathbf{Q}\,\mathbf{S}$, and $\bar{\tau}$ the largest eigenvalue of $\mathbf{Q}\,\mathbf{S}$. Consider the following two cases:

Case 1. $\lambda^* = \max_i |\lambda_i| \in \mathbb{C}$ and $\lambda^* \notin \mathbb{R} \Leftrightarrow \tau_{\lambda^*} < 1$, where $\tau_{\lambda^*} \in \mathrm{eig}(\mathbf{Q}\,\mathbf{S})$ satisfies $\dfrac{(1-\lambda^*)^2}{1-2\lambda^*} = \tau_{\lambda^*}$.

We have, $\rho(\mathbb{E}[\mathbf{M}_{RAC}]) \geq \rho(\mathbb{E}[\mathbf{M}_{RP,\upsilon_i}])\ \forall\, i$, which implies that

$$\sqrt{1-\tau_{\lambda^*}} > \max\left\{\sqrt{1-\tau_{\upsilon_i}}, \tau_{\upsilon_i} - 1 + \sqrt{\tau_{\upsilon_i}(\tau_{\upsilon_i}-1)}\right\} \quad \forall i.$$

Specifically

$$\sqrt{1-\tau_{\lambda^*}} > \sqrt{1-\underline{\tau}_{\upsilon_i}} \quad \forall \upsilon_i,$$

As $f(x) = \sqrt{1-x}$ is monotone decreasing with respect to x, the above implies that

$$\tau_{\lambda^*} < \underline{\tau}_{\upsilon_i} \quad \forall \upsilon_i,$$

and as $\tau_{\lambda^*} \geq \underline{\tau}$, the above equation implies

$$\underline{\tau} < \underline{\tau}_{\upsilon_i} \quad \forall \upsilon_i,$$

which is impossible, as by Weyl's theorem,

$$\underline{\tau} \geq \frac{1}{|\Upsilon(d,b)|}\sum_i \underline{\tau}_{\upsilon_i} \geq \min_i\ \underline{\tau}_{\upsilon_i}.$$

Case 2. $\lambda^* = \max_i |\lambda_i| \in \mathbb{R} \Leftrightarrow \tau_{\lambda^*} > 1$.

We have $\rho(\mathbb{E}[\mathbf{M}_{RAC}]) \geq \rho(\mathbb{E}[\mathbf{M}_{RP,\upsilon_i}])\ \forall i$, what implies that

$$\tau_{\lambda^*} - 1 + \sqrt{\tau_{\lambda^*}(\tau_{\lambda^*}-1)} > \max\{\sqrt{1-\tau_{\upsilon_i}}, \tau_{\upsilon_i} - 1 + \sqrt{\tau_{\upsilon_i}(\tau_{\upsilon_i}-1)}\} \quad \forall i.$$

Specifically,

$$\tau_{\lambda^*} - 1 + \sqrt{\tau_{\lambda^*}(\tau_{\lambda^*} - 1)} > \tau_{\upsilon_i} - 1 + \sqrt{\tau_{\upsilon_i}(\tau_{\upsilon_i} - 1)} \quad \forall i, \quad \forall \upsilon_i.$$

As $g(x) = x - 1 + \sqrt{x(x-1)}$ is a monotone increasing function for $x \in [1, \infty)$, the above implies

$$\overline{\tau} \geq \tau_{\lambda^*} > \overline{\tau}_{\upsilon_i} \quad \forall \upsilon_i,$$

which is impossible, as by Weyl's theorem,

$$\bar{\tau} \leq \frac{1}{|\Upsilon(d,b)|} \sum_i \bar{\tau}_{\upsilon_i} \leq \max_i \ \underline{\tau}_{\upsilon_i}.$$

□

2.2.4 Variance of RAC-ADMM

Convergence in expectation may not be a good indicator of convergence for solving all problems, as there may exist a problem for which RAC-ADMM is not stable or possesses greater variance. In order to give another probabilistic measure on performance of RAC-ADMM, this section introduces convergence almost surely (a.s.) as an indicator of the algorithm convergence. Convergence almost surely as a measure for stability has been used in linear control systems for quite some time, and is based on the mean-square stability criterion for stochastically varying systems [19]. The criterion establishes conditions for asymptotic convergence of covariance of the system states (e.g. variables).

This section builds on those results and establishes sufficient condition for RAC-ADMM to converge almost surely when applied to solve (9). The condition utilizes the Kronecker product of the mapping matrix, which captures the dynamics of the second moments of the random sequences generated by RAC-ADMM algorithm, and the expectation over the products of mapping matrices that provides the bounds on the variance of the distance between the KKT point and the random sequence generated by our algorithm.

Theorem 3 *Suppose that Assumption 1 holds, and that RAC-ADMM (8) is employed to solve problem (9). Then the output of RAC-ADMM converges almost surely to some KKT point of (9) if*

$$\rho(\mathbb{E}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]) < 1$$

where $\mathbf{M} \otimes \mathbf{M}$ *is the Kronecker product of* $\mathbf{M}$ *with itself.*

Proof Let $\overline{\mathbf{z}} = [\overline{\mathbf{x}}; \overline{\mathbf{y}}] \in \mathbb{R}^N$ denote the KKT point of (9), then, at $k+1th$ iteration we have

$$(\mathbf{z}_{k+1} - \overline{\mathbf{z}}) = \mathbf{M}_{\sigma_k}(\mathbf{z}_k - \overline{\mathbf{z}}).$$

Define $\mathbf{d}_k = \mathbf{z}_k - \overline{\mathbf{z}}$, and

$$\mathbf{P}_k = \mathbb{E}[\mathbf{d}_k \, \mathbf{d}_k^T].$$

There exists a linear operator $\mathcal{T}$ s.t.

$$\mathrm{vec}(\mathbf{P}_{k+1}) = \mathcal{T}\,\mathrm{vec}(\mathbf{P}_k) \tag{14}$$

where $\mathrm{vec}(\cdot)$ is vectorization of a matrix, and $\mathcal{T} = \mathbb{E}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]$, as

$$\begin{aligned}\mathrm{vec}(\mathbf{P}_{k+1}) &= \mathrm{vec}(\mathbb{E}[\mathbf{d}_{k+1}\,\mathbf{d}_{k+1}^T]) \\ &= \frac{1}{|\Upsilon(n,\,p)|}\sum_{i=1}^{|\Upsilon(n,p)|}\mathrm{vec}(\mathbf{M}_i\,\mathbb{E}[\mathbf{d}_k\,\mathbf{d}_k^T]\,\mathbf{M}_i^T) \\ &= \mathbb{E}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]\,\mathrm{vec}(\mathbf{P}_k)\end{aligned}$$

and $\rho(\mathbb{E}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]) < 1$ implies $d_k \overset{a.s.}{\to} \mathbf{0}$. To prove this, let $||\cdot||$ be the Frobenius norm of a matrix, $||\,\mathbf{A}\,|| = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$, thus we can write

$$\mathbb{E}[||\,\mathbf{d}_k\,||^2] = \mathrm{Trace}(\mathbf{P}_k) \le ||\,\mathrm{vec}(\mathbf{P}_k)||^2,$$

and by (14),

$$||\,\mathrm{vec}(\mathbf{P}_k)||^2 = ||\mathcal{T}\,\mathrm{vec}(\mathbf{P}_{k-1})||^2 = ||\mathcal{T}^k\,\mathrm{vec}(\mathbf{P}_0)||^2 \le ||\mathcal{T}^k||^2 \cdot ||\,\mathrm{vec}(\mathbf{P}_0)||^2$$

If $\rho(\mathcal{T}) < 1$, we know that $\mathcal{T}$ is convergent, and there exists $\mu > 0$, $0 < \gamma < 1$, s.t.

$$||\mathcal{T}^k||^2 \le \mu\gamma^k,$$

thus there exists $\mathbf{M}$ such that,

$$\sum_{k=0}^{\infty}\mathbb{E}[||\,\mathbf{d}_k\,||^2] \le \mathbf{M}\sum_{k=0}^{\infty}\gamma^k \le C < \infty$$

For any $\epsilon > 0$, by Markov inequality we have

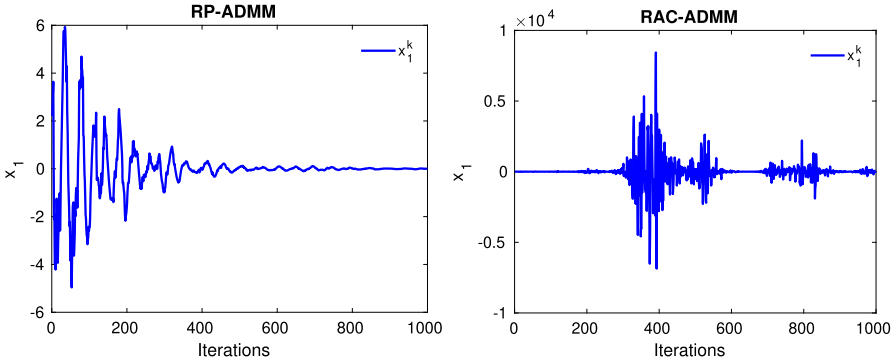
$$\sum_{k=0}^{\infty}\mathbb{E}[||\,\mathbf{d}_k\,||^2] \le C \Rightarrow \sum_{k=0}^{\infty}\mathrm{Prob}(||\,\mathbf{d}_k\,||^2 > \epsilon) < \infty,$$

and as $\sum_{k=0}^{\infty}\mathrm{Prob}(||\,\mathbf{d}_k\,||^2 < \epsilon) < \infty$, by Borel–Cantelli, and $||\,\mathbf{d}_k\,||^2 \in m\mathcal{F}_+$,

$$\mathbf{d}_k \overset{a.s.}{\to} \mathbf{0} \quad \text{as } k \to \infty$$

which then implies that randomized ADMM converges almost surely. □

To illustrate the stability issues with RAC-ADMM, consider the following example.



(a) $\rho(\mathbb{E}_{RP}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]) = 0.99$ **(b)** $\rho(\mathbb{E}_{RAC}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]) = 1.09$

Fig. 2 Effect of variance on convergence for problem (15). Evaluation of x_1^k (optimal $x_1^* = 0$)

Example 3 Consider the following problem

$$\begin{array}{ll} \max & \mathbf{0} \cdot \mathbf{x} \\ \text{s.t.} & \mathbf{A}\,\mathbf{x} = \mathbf{0} \end{array}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1+\gamma \\ 1 & 1 & 1 & 1 & 1+\gamma & 1+\gamma \\ 1 & 1 & 1 & 1+\gamma & 1+\gamma & 1+\gamma \\ 1 & 1 & 1+\gamma & 1+\gamma & 1+\gamma & 1+\gamma \\ 1 & 1+\gamma & 1+\gamma & 1+\gamma & 1+\gamma & 1+\gamma \end{bmatrix} \tag{15}$$

Let $[x_0, y_0] \sim N(0, 5I)$, $\beta = 1$, $\gamma = 1$, and number of blocks $p = 3$. Consider RP-ADMM with the fixed block composition $[x_1,\ x_2], [x_3,\ x_4], [x_5,\ x_6]$.

Convergence in expectation for this particular block structure finds $\rho(\mathbb{E}[\mathbf{M}_{RP,\upsilon_1}]) = 0.9887 > \rho(\mathbb{E}[\mathbf{M}_{RAC}]) = 0.8215$. In fact, for all block compositions for this example we have, $\rho(\mathbb{E}[\mathbf{M}_{RAC}]) > \rho(\mathbb{E}[\mathbf{M}_{RP,\upsilon_i}]$. However, RAC-ADMM does not converge, as shown in Fig. 2, showing that convergence in expectation may not be a sufficient indicator from this particular example.

Indeed, if we apply Theorem 3, we find out that RAC-ADMM does not converge almost surely, while RP-ADMM does for solving this example: $\rho(\mathbb{E}_{RAC}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]) = 1.0948 > 1$ while $\rho(\mathbb{E}_{RP}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]) = 0.9852 < 1$, what explains the results shown in Fig. 2. In fact, RP-ADMM converges almost surely for all 15 block compositions of this example.

2.3 Variance reduction in RAC-ADMM

The previous section described sufficient condition for the almost sure convergence of RAC-ADMM algorithm. This section address controlability of the algorithm. More

precisely we ask, given a linearly constrained quadratic problem (LCQP) (Eq. 9), what means do we have at our disposal to control convergence of a LCQP– how to bound the covariance and how to improve the convergence rate.

2.3.1 Detecting and utilizing a structure in LCQP

Although some problem types inherit a known structure (e.g. network-flow problems), in general, the structure is not known. There are many sophisticated techniques used to detect a structure of a matrix one can use and apply towards improving performance of RAC-ADMM. Although such elaborate methods have a potential of detecting hidden structure of Hessian and Jacobian matrices almost perfectly, using them or developing our own is beyond the scope of this paper. Instead, we adopt a simple matrix partitioning approach outlined in [27].

In general, for RAC-ADMM we are interested in a structure of a constraint matrix, which can be detected using the following simple approach. Given a constraint matrix $\mathbf{A}$ (describing equalities, inequalities or both), a desirable structure such as one shown in (16) can be derived by applying a graph partitioning method.

$$\underbrace{\begin{bmatrix} \mathbf{V}_1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \mathbf{V}_v & 0 \\ \mathbf{W}_1 & \cdots & \mathbf{W}_v & \mathbf{W}_{v+1} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_v \\ \mathbf{x}_{v+1} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_v \\ \mathbf{b}_{v+1} \end{bmatrix}}_{\mathbf{b}}. \tag{16}$$

The outline of the process is as follows:

1. Build a graph representation of matrix $\mathbf{A}$: Each row i and column j is a vertex; vertices are connected with edges if $a_{i,j} \neq 0$.
2. Partition the graph using a graph partitioning algorithm or solver, for example [43].
3. Recreate $\mathbf{A}$ as a block matrix from the graph partitions.

Using graph partitioning as a procedure for decomposing a problem in a way that maximizes the modularity has been studied for a while. Although the problem itself is an NP-hard integer program, many efficient algorithms have been proposed that achieve near-optimal performance with good computational scalability [1,7,56]. In addition, successful implementations such as those used by GCG (generic solver for mixed integer programs, part of SCIP Optimization Suite [63]) exist. Currently our solver RACQP, described in Sect. 3, does not implement an automatic routine for modularity detection, but we plan to add it.

2.3.2 Smart grouping

Smart-grouping is a pre-processing method in which we use block structure of constraint matrix $\mathbf{A}$ to pre-group certain variables as a single "super-variable" (a group of variables which always stay together in one block). Following the block structure

shown in (16), we make one super-variable $\hat{\mathbf{x}}_i$ for each group $\mathbf{x}_i$, $i = 1, \ldots, v$. Primal variables $\mathbf{x}_{v+1}$ stay shared and are randomly assigned to sub-problems to complement super-variables to which they are coupled with via block-matrices $\mathbf{W}_i$, $i = 1, \ldots, v$. More than one super-variable can be assigned to a single sub-problem, dependent upon the maximum size of a sub-problem, if defined. Note that matrix partitioning based on $\mathbf{H} + \mathbf{A}^T\,\mathbf{A}$ may result in a better grouping, but is unpractical and thus not considered as a viable approach.

2.3.3 Partial Lagrangian

The idea of smart-grouping described in the previous section can be further extended by the means of the partial Lagrangian approach. Consider a LCQP (6) having the constraint matrix $\mathbf{A}$ structure as shown in (16). Now consider the scenario in which we split the matrix $\mathbf{A}$ such that the block $\mathbf{W} = [\mathbf{W}_1, \ldots, \mathbf{W}_{v+1}]$ is admitted by the augmented Lagrangian while the rest of the constraints (blocks $\mathbf{V}_i$) are solved exactly as a part of a sub-problem, i.e. a sub-problem i is solved as

$$\mathbf{x}_i^{k+1} = \arg\min\{L_{\mathcal{P}}(\mathbf{x}_1^{k+1}, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_p^k; \mathbf{y}^k) \mid \mathbf{V}_j\,\hat{\mathbf{x}}_j = \mathbf{b}_j,\ j \in \mathcal{J},\ \mathbf{x}_i \in \mathcal{X}_i\},$$

where $\mathcal{J}$ is a set of indices of super-variables $\hat{\mathbf{x}}_j$ constituting sub-problem i at any given iteration. The partial augmented Lagrangian is defined with

$$L_{\mathcal{P}}(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} - \mathbf{y}^T(\mathbf{W}\,\mathbf{x} - \mathbf{b}_{v+1}) + \frac{\beta}{2}\|\,\mathbf{W}\,\mathbf{x} - \mathbf{b}_{v+1}\,\|^2.$$

There are two advantages of the partial Lagrangian approach. First, the rank of the constraint matrix used for the *global* constraints (matrix $\mathbf{W}$) is lower than the rank of $\mathbf{A}$, and the empirical results (Sect. 4) suggest a strong correlation between a rank of a matrix and the stability of the algorithm and its rate of convergence. Next, *local* constraints (matrices $\mathbf{V}_i$) imply there is a feasibility region in which $\mathbf{x}_i$ exist, and that region may not be infinite. In other words, even when the variables themselves are unbounded (i.e. $\mathbf{x} \in \mathbb{R}^n$), local constraints may put implicit bounds on maximum variation of values of $\mathbf{x}_i$.

Empirical results of the partial Lagrangian applied to mixed integer problems (Sect. 4.2) show the approach to be very useful. In such a scenario, local constraints are sets of rules that relate integer variables, while constraints between continuous variables are left global. In the case of a problems where such straight separation does not exist, or when problems are purely integer, a problem structure is let to guide the local/global constraints decision.

Although shown to be useful, the partial Lagrangian method suffers form being a mostly heuristic approach that depends on quality of solution methods applied to sub-problems—in the case of continuous problems, a simple barrier based methodology can be applied, but for the mixed integers problems (MIP), sub-problems require a more complex solution (e.g. an external MIP solver).

Example 4 To illustrate the usefulness of the smart grouping and partial Lagrange approaches, consider the following experiments done on selected instances taken from

the Mittelmann LP test set [73] augmented with a diagonal quadratic objective to form a standard LCQP (9)).

For each instance, a constraint matrix ($\mathbf{A}_{eq}$, $\mathbf{A}_{ineq}$ or $\mathbf{A} = [\mathbf{A}_{eq}; \mathbf{A}_{ineq}]$) was subjected to graph-partitioning procedure outlined in Sect. 2.3.1, and then solved using the smart grouping ("s_grp") and the partial Lagrangian approach ("partial_L"). Table 2 reports on the number of iterations required by RAC-ADMM algorithm to find a solution satisfying the primal/dual residual tolerance of $\epsilon = 10^{-4}$. If the solution was not found the reason is noted ("time limit" for exceeding sub-problem maximum runtime and "iter. limit" for exceeding maximum number of iterations). Fields showing "divergence" or "oscillation" mark experiments for which RAC-ADMM algorithm experienced an unstable behavior. The baseline for the comparison is the *default* approach (sub-problems created at random) shown in column "Default RAC".

The partial Lagrangian approach has a potential to help stability and rate of convergence. However, before generalizing, one needs to consider the following: stability (i.e. convergence) of RAC-ADMM algorithm, is a function, among other factors, of mapping operators (matrices $\mathbf{M}_\sigma$, Eq. 11) which are in turn functions, among other factors, of the constraint matrix of a problem being solved. In the case of partial Lagrangian methodology, this matrix is the matrix $\mathbf{W}$, meaning that if $\mathbf{W}$ produces an unstable system (e.g. conditions set by Theorem 3 not met), no implicit bounding can help to stabilize it. At the same time, a "correct" $\mathbf{W}$ can stabilize a problem which is unstable in its original form. Consider a problem that is not convergent in its original formulation. Amending $\mathbf{A}$ by moving "bad" rows to sub-problems, thus constructing $\mathbf{W}$ that produces mapping matrices satisfying $\rho(\mathbb{E}[\mathbf{M}_\sigma \otimes \mathbf{M}_\sigma]) < 1$ makes such a problem stable. Theoretical work on structures of $\mathbf{W}$ and conditions that stabilize and improve RAC-ADMM are works in progress.

Using smart grouping alone does not make RAC-ADMM unstable, but in some cases increases the number of iterations needed to satisfy feasibility tolerance, a consequence of having less randomness as described by Corollary 2.

3 RAC-ADMM quadratic programming solver

In this section we outline the implementation of the RAC-ADMM algorithm for linearly constrained quadratic problems as defined below:

$$\begin{aligned} \min_{\mathbf{x}} \ & \tfrac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \\ \text{s.t.} \ & \mathbf{A}_{eq}\,\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{A}_{ineq}\,\mathbf{x} \le \mathbf{b}_{ineq} \\ & \mathbf{x} \in \mathcal{X} \end{aligned} \tag{17}$$

where symmetric positive semidefinite matrix $\mathbf{H} \in \mathbb{R}^{n\times n}$ and vector $\mathbf{c} \in \mathbb{R}^n$ define the quadratic objective while matrix $\mathbf{A}_{eq} \in \mathbb{R}^{m\times n}$ and the vector $\mathbf{b}_{eq} \in \mathbb{R}^m$ describe equality constraints and matrix $\mathbf{A}_{ineq} \in \mathbb{R}^{s\times n}$ and the vector $\mathbf{b}_{ineq} \in \mathbb{R}^s$ describe inequality constraints. Primal variables $\mathbf{x} \in \mathcal{X}$, can be integer or continuous, thus the constraint set $\mathcal{X}$ is the Cartesian product of nonempty sets $\mathcal{X}_i \subseteq \mathbb{R}$ or $\mathcal{X}_i \subseteq \mathbb{Z}$,

Table 2 Number of iterations until termination criteria is met for various benchmark instances

Instance	Num. col.	Num. rows (A)	Default RAC	Partitioning on $\mathbf{A}_{eq}$		Partitioning on $\mathbf{A}_{ineq}$		Partitioning on $\mathbf{A}$	
				partial_L	s_grp	partial_L	s_grp	partial_L	s_grp
qap10	4150	1820	182	102	550	–	–	102	550
pds-02	7535	2953	2872	577	1530	Iter. limit	798	1287	3878
n370a	10,000	5150	4797	Time limit	852	211	1819	148	1298
supportcase10	14,770	165,684	5535	240	349	1370	2622	1480	2496
ex10	17,680	69,608	324	281	630	Oscillation	410	Oscillation	410
nug08-3rd	20,448	19,728	963	437	414	Oscillation	516	624	624
brazil3	23,968	14,646	1083	Divergence	1103	Oscillation	2207	Divergence	1126

$i = 1, \ldots, n$. QP problems arise from many important applications themselves, and are also fundamental in general nonlinear optimization.

Introducing auxiliary variables $\mathbf{s}$ and $\hat{\mathbf{x}}$, results in the following equivalent of (17):

$$\begin{aligned} \min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{s}} \ & \tfrac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \\ \text{s.t.} \ & \mathbf{A}_{eq}\,\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} = \mathbf{b}_{ineq} \\ & \mathbf{x} - \hat{\mathbf{x}} = \mathbf{0} \\ & \hat{\mathbf{x}} \in \mathcal{X}, \ \mathbf{s} \geq \mathbf{0}, \mathbf{x}\,\text{free} \end{aligned} \tag{18}$$

where the augmented Lagrangian, $L_\beta(\mathbf{x}; \mathbf{s}; \mathbf{y}_{eq}; \mathbf{y}_{ineq}; \mathbf{z})$, is given as

$$\begin{aligned} L_\beta(\cdot) := \ & \tfrac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \\ & - \mathbf{y}_{eq}^T(\mathbf{A}_{eq}\,\mathbf{x} - \mathbf{b}_{eq}) - \mathbf{y}_{ineq}^T(\mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} - \mathbf{b}_{ineq}) - \mathbf{z}^T(\mathbf{x} - \hat{\mathbf{x}}) \\ & + \tfrac{\beta}{2}(\|\,\mathbf{A}_{eq}\,\mathbf{x} - \mathbf{b}_{eq}\,\|^2 + \|\,\mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} - \mathbf{b}_{ineq}\,\|^2 + \|\,\mathbf{x} - \hat{\mathbf{x}}\|^2) \end{aligned} \tag{19}$$

RAC-ADMM, or simply RAC, quadratic programming (RACQP) solver admits continuous, binary and mixed integer problems. Algorithm 1 outlines the solver: the solution vector is initialized to $-\infty$ at the beginning of the algorithm, and the main RAC-ADMM loop described (lines 2–24). The main loop calls different procedures to optimize blocks of $\mathbf{x}$ (lines 4–16), followed by updates of slack and then dual variables.

Types of the block optimizing procedure being called to update the blocks depend on the structure of the problem being solved. The default, multi-block implementation for continuous problems is based on the Cholesky factorization, with a specialized one-block variant for very sparse problems that solves the iterates using the LDL factorization. Continuous problems that exhibit a structure (see Sect. 2.3.1) can be addressed using the partial Lagrangian approach. In such a case, sub-problems are solved using either a simple interior point method based methodology, or, when sub-problems include only equality constraints, by employing Cholesky for solving KKT conditions. In addition to the aforementioned methods, the solver supports calls to external solver(s) and specialized heuristic solution to handle hard sub-problem instances.

Binary and mixed integer problems require specialized optimization techniques (e.g. branch-and-bound), that require implementations which are beyond the scope of this paper, so we have decided to delegate optimizing of the blocks with mixed variables to an external solver. Mixed integer problems are addressed by using the partial Lagrangian to solve for primal variables and a simple procedure that helps to escape local optima, as described by Algorithm 2.

Note that Algorithms given in this section are pseudo-algorithms which describe functionality of the solver rather than actual implementation. The implementation can be downloaded from [61].

Algorithm 1 RACQP

Require: Problem *model* (Eq. 18), *run-time parameters*† with *termination criteria*††
Ensure: The optimal solution $\mathbf{x}^*$ or the best solution found before termination met

1: $\mathbf{x} \leftarrow -\infty$
2: **while** *termination criteria* not met **do**
3: $\Omega \leftarrow$ construct blocks at random; use smart grouping if applicable (Sect. 2.3.2)
4: **for all** vectors $\omega_i \in \Omega$ of block indices **do** ▷ Solve $\mathbf{x}_{\omega_i}$ blocks
5: Prepare $\mathbf{Q}_{\omega_i,\omega_i}$, $\mathbf{q}_{\omega_i}$ and $\hat{\mathbf{q}}$ following Eqs. (20) and (22)
6: $\mathbf{x}_{\omega_i} \leftarrow$ solve $\mathbf{Q}_{\omega_i,\omega_i}\,\mathbf{x}_{\omega_i} = -(\mathbf{q}_{\omega_i} + \hat{\mathbf{q}})$ using:
7: **if** (sub-problem is mixed integer) **then**
8: An external solver
9: **else if** (partial Lagrangian (26) used and sub-problem includes inequalities) **then**
10: Interior point method based procedure or an external solver
11: **else if** (partial Lagrangian used) **then**
12: Cholesky factorization and back substitution solving KKT conditions
13: **else**
14: Cholesky factorization and back substitution
15: **end if**
16: **end for**
17: $\mathbf{s}^* \leftarrow \max(0, \frac{1}{\beta}\,\mathbf{y}_{ineq} + \mathbf{b}_{ineq} - \mathbf{A}_{ineq}\,\mathbf{x})$ ▷ Update slack variables
18: **if** (bounds on $\mathbf{x}$ not addressed by the partial Lagrangian) **then**
19: $\hat{\mathbf{x}} \leftarrow \min(\max(\mathbf{l}, \mathbf{x} - \frac{1}{\beta}\,\mathbf{z}), \mathbf{u})$ ▷ Update auxiliary split variables $\hat{\mathbf{x}}$
20: $\mathbf{z} \leftarrow \mathbf{z} - \beta(\mathbf{x} - \tilde{\mathbf{x}})$ ▷ Update dual variables for split variables
21: **end if**
22: $\mathbf{y}_{eq} \leftarrow \mathbf{y}_{eq} - \beta(\mathbf{A}_{eq}\,\mathbf{x} - \mathbf{b}_{eq})$ ▷ Update dual variables for equality constraints
23: $\mathbf{y}_{ineq} \leftarrow \mathbf{y}_{ineq} - \beta(\mathbf{A}_{ineq}\,\mathbf{x} - \mathbf{b}_{ineq})$ ▷ Update dual variables for inequality constraints
24: **end while**
25: **return** $\mathbf{x}$

† Number of groups p, penalty parameter β, initial point $\mathbf{x}_0$, pre-grouped vars set $\mathcal{V}$.
†† Termination criteria may include maximum run-time, number of attempts to find a better solution, solution quality and so on.

3.1 Solving continuous problems

For the continuous QP problems, we consider (18) where $\mathcal{X}$ are possible simple lower and upper bounds on each individual variable:

$$\mathbf{l}_i \leq \hat{\mathbf{x}}_i \leq \mathbf{u}_i, \; i = 1, \ldots, n.$$

Continuous problems are solved as described by Algorithm 1, which repeats three steps until termination criteria is met: first update or optimize primal variables $\mathbf{x}$ in the RAC fashion, then update $\hat{\mathbf{x}}$ and $\mathbf{s}$ in close forms and finally update dual variables $\mathbf{y}_{eq}$, $\mathbf{y}_{ineq}$ and $\mathbf{z}$.
Step 1 Update primal variables $\mathbf{x}$ Let $\omega_i \in \Omega$ be a vector of indices of a block i, $i = 1, \ldots, p$, where p is the number of blocks. The set of vectors Ω is randomly generated (with smart grouping when applicable as described in Sect. 2.3.2) at each iteration of the Algorithm 1 (lines 2–24). Let $\mathbf{x}_{\omega_i}$ be a sub-vector of $\mathbf{x}$ constructed of components of $\mathbf{x}$ with indices ω_i, and let $\mathbf{x}_{-\omega_i}$ be the sub-vector of $\mathbf{x}$ with indices not chosen by ω_i. Algorithm 1 uses either Cholesky factorization or partial Lagrangian to

solve each block of variables $\mathbf{x}_{\omega_i}$ while holding $\mathbf{x}_{-\omega_i}$ fixed. By rewriting (19) to reflect the sub-vectors, we get

$$\begin{aligned} L_\beta(\cdot) &= [\mathbf{x}_{\omega_i}; \mathbf{x}_{-\omega_i}]^T (\tfrac{1}{2}\mathbf{H} + \tfrac{\beta}{2}\mathbf{A}_{eq}^T \mathbf{A}_{eq} + \tfrac{\beta}{2}\mathbf{A}_{ineq}^T \mathbf{A}_{ineq} + \tfrac{\beta}{2}\mathbf{I})[\mathbf{x}_{\omega_i}; \mathbf{x}_{-\omega_i}] \\ &\quad + (\mathbf{c} - \mathbf{A}_{eq}^T \mathbf{y}_{eq} - \mathbf{A}_{ineq}^T \mathbf{y}_{ineq} - \beta \mathbf{A}_{eq}^T \mathbf{b}_{eq} - \beta \mathbf{A}_{ineq}^T \mathbf{b}_{ineq})^T [\mathbf{x}_{\omega_i}; \mathbf{x}_{-\omega_i}] \\ &\quad + (\beta \mathbf{A}_{ineq}^T \mathbf{s} - \mathbf{z} - \beta\hat{\mathbf{x}})^T [\mathbf{x}_{\omega_i}; \mathbf{x}_{-\omega_i}] \\ &= \tfrac{1}{2}[\mathbf{x}_{\omega_i}; \mathbf{x}_{-\omega_i}]^T \mathbf{Q}[\mathbf{x}_{\omega_i}; \mathbf{x}_{-\omega_i}] + \mathbf{q}^T [\mathbf{x}_{\omega_i}; \mathbf{x}_{-\omega_i}] \end{aligned} \tag{20}$$

where $\mathbf{Q} = (\mathbf{H} + \beta \mathbf{A}_{eq}^T \mathbf{A}_{eq} + \beta \mathbf{A}_{ineq}^T \mathbf{A}_{ineq} + \beta I)$. Then we can minimize in $\mathbf{x}_{\omega_i}$ by solving

$$\mathbf{Q}_{\omega_i,\omega_i} \mathbf{x}_{\omega_i} = -(\mathbf{q}_{\omega_i} + \hat{\mathbf{q}}) \tag{21}$$

using Cholesky factorization and back substitution. The linear term resulting from $\mathbf{Q}$, $\hat{\mathbf{q}}$, is given as

$$\hat{\mathbf{q}} = (\mathbf{H}\,\hat{\mathbf{x}})_{\omega_i} + \beta \mathbf{A}_{\omega_i}^T (\mathbf{A}\,\tilde{\mathbf{x}}) - (\mathbf{H}_{\omega_i,\omega_i} + \beta \mathbf{A}_{\omega_i}^T \mathbf{A}_{\omega_i})\,\mathbf{x}_{\omega_i} \tag{22}$$

where $\mathbf{A} = [\mathbf{A}_{eq}, 0; \mathbf{A}_{ineq}, \mathbf{I}]$ and $\tilde{\mathbf{x}} = [\mathbf{x}; \mathbf{s}]$. A square sub-matrix $\mathbf{H}_{\omega_i,\omega_i}$ and column sub-matrix $\mathbf{A}_{\omega_i}$ are constructed by extracting ω_i rows and columns from $\mathbf{H}$ and $\mathbf{A}$ respectively.

When $p = 1$, i.e. we are solving a problem using a single-block approach, then we solve the block utilizing LDL factorization to avoid calculating $\mathbf{A}^T \mathbf{A}$. Although the factorization can be relatively expensive if the problem size is large as we then factorize a large matrix, the factorization is done only once and re-used in each iteration of the algorithm. From (19), we find minimizer $\mathbf{x}$ by solving

$$\mathbf{Q}\,\mathbf{x} = -\,\mathbf{q} \tag{23}$$

where

$$\mathbf{q} = \mathbf{c} - \mathbf{A}_{eq}^T \mathbf{y}_{eq} - \mathbf{A}_{ineq}^T \mathbf{y}_{ineq} - \mathbf{z} - \beta \mathbf{A}_{eq}^T \mathbf{b}_{eq} - \beta \mathbf{A}_{ineq}^T \mathbf{b}_{ineq} + \beta \mathbf{A}_{ineq}^T \mathbf{s} - \beta\hat{\mathbf{x}}.$$

With $\mathbf{A} = [\mathbf{A}_{eq}; \mathbf{A}_{ineq}]$ we can express the equivalent condition to (23) with

$$\begin{bmatrix} (\mathbf{H} + \beta\,\mathbf{I}) & \sqrt{\beta}\,\mathbf{A}^T \\ \sqrt{\beta}\,\mathbf{A} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} -\mathbf{q} \\ \mathbf{0} \end{bmatrix}. \tag{24}$$

We factorize the left hand side of the above expression and use the resulting matrices to find $\mathbf{x}$ by back substitution at each iteration of the algorithm. For single-block RACQP, LDL approach described above replaces lines 3–16 in Algorithm 1.

Furthermore, if $\mathbf{H}$ is diagonal, one can rewrite the system as

$$\begin{bmatrix} \mathbf{I} & (\mathbf{H} + \beta I)^{-1}\sqrt{\beta}\,\mathbf{A}^T \\ \sqrt{\beta}\,\mathbf{A} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} -\mathbf{q} \\ \mathbf{0} \end{bmatrix}. \tag{25}$$

Then we can factorize matrix $(\mathbf{I}+\beta\,\mathbf{A}(\mathbf{H}+\beta\,\mathbf{I})^{-1}\,\mathbf{A}^T)$ to solve the system, which would be extremely effective when the number of constraints is very small and/or sparse, since $(\mathbf{H}+\beta\,\mathbf{I})^{-1}$ is diagonal and it does not change sparsity of $\mathbf{A}$.

Partial Lagrangian approach to solving $\mathbf{x}$ blocks, described in Sect. 2.3.3, uses the same implementation as Cholesky approach described above, with additional steps that build local constraints which reflect free and fixed components of $\mathbf{x}$, $\mathbf{x}_{\omega_i}$ and $\mathbf{x}_{-\omega_i}$ respectively. The optimization problem of partial Lagrangian is formulated as

$$\begin{aligned} \mathbf{x}^*_{\omega_i} = \arg\min\ & \tfrac{1}{2}\,\mathbf{x}^T_{\omega_i}\,\mathbf{Q}^L_{\omega_i,\omega_i}\,\mathbf{x}_{\omega_i} + (\mathbf{q}^L_{\omega_i} + (\hat{\mathbf{q}}^L)^T)\,\mathbf{x}_{\omega_i} \\ \text{s.t.}\ & \mathbf{A}^L_{eq,\,\omega_i}\,\mathbf{x}_{\omega_i} = \mathbf{b}^L_{eq} - \mathbf{A}^L_{eq,\,-\omega_i}\,\mathbf{x}_{-\omega_i} \\ & \mathbf{A}^L_{ineq,\,\omega_i}\,\mathbf{x}_{\omega_i} \le \mathbf{b}^L_{eq} - \mathbf{A}^L_{ineq,\,-\omega_i}\,\mathbf{x}_{-\omega_i} \\ & \mathbf{l}_{\omega_i} \le \mathbf{x}_{\omega_i} \le \mathbf{u}_{\omega_i} \end{aligned} \tag{26}$$

with $\mathbf{Q}^L$, $\mathbf{q}^L$, $\hat{\mathbf{q}}^L$ $\mathbf{A}^L_{eq}$, $\mathbf{b}^L_{eq}$ and $\mathbf{A}^L_{ineq}$, $\mathbf{b}^L_{ineq}$ describing local objective, equality and inequality constraints, respectively. Note that partial Lagrangian procedure is used by both continuous and mixed integer problems. In the case of the former we set $\mathcal{X} = \mathbb{R}^n$, while when we solve the latter we let $\mathcal{X}_i \subseteq \mathbb{R}$ and implicitly enforce the bounds. The blocks are solved by either an external solver (e.g. Gurobi) or by using Cholesky to solve KKT conditions when $\mathbf{x}_{\omega_i}$ is unbounded.

Step 2 Update auxiliary variables $\hat{\mathbf{x}}$ With all variables but $\hat{\mathbf{x}}$ fixed, from augmented Lagrangian (19) we find that the optimal vector $\mathbf{l} \le \hat{\mathbf{x}} \le \mathbf{u}$ can be found by solving the optimization problem

$$\arg\min_{\mathbf{l}\le\hat{\mathbf{x}}\le\mathbf{u}} \frac{\beta}{2}\hat{\mathbf{x}}^T\hat{\mathbf{x}} + (\mathbf{z} - \beta\,\mathbf{x}^T)\hat{\mathbf{x}}.$$

The problem is separable and $\hat{\mathbf{x}}$ has a closed form solution given by

$$\hat{\mathbf{x}} = \min\left\{\max\{\mathbf{l}, \mathbf{x} - \frac{1}{\beta}\,\mathbf{z}\}, \mathbf{u}\right\}$$

Step 3 Update slack variables $\mathbf{s}$ Similarly to the previous step, with all variables but $\mathbf{s}$ fixed, the optimal vector $\mathbf{s}$ is found by solving

$$\arg\min_{\mathbf{s}\ge 0} \frac{\beta}{2}\,\mathbf{s}^T\,\mathbf{s} + (-\,\mathbf{y}_{ineq} + \beta(\mathbf{A}_{ineq}\,\mathbf{x} - \mathbf{b}_{ineq}))^T\,\mathbf{s}\,.$$

The problem is separable and $\mathbf{s}$ has a closed form solution given by

$$\mathbf{s} = \max\left\{0, \frac{1}{\beta}\,\mathbf{y}_{ineq} + \mathbf{b}_{ineq} - \mathbf{A}_{ineq}\,\mathbf{x}\right\}.$$

3.1.1 Termination criteria for continuous problems

Termination criteria for continuous problems include maximum run-time limit settings, maximum number of iterations and primal-dual solution (found up to some

tolerance). RACQP terminates when at least one criterion is met. For primal-dual solution criterion RACQP uses the optimality conditions of problem (18) to define primal and dual relative residuals at iteration k,

$$\begin{aligned} r^k_{\text{prim}} &:= \max(r^k_{\mathbf{A}_{eq}}, r^k_{\mathbf{A}_{ineq}}, r^k_{bounds}), \\ r^k_{\text{dual}} &:= \frac{\| \mathbf{H}\,\mathbf{x}^k + \mathbf{c} - \mathbf{A}^T_{eq}\,\mathbf{y}^k_{eq} - \mathbf{A}^T_{ineq}\,\mathbf{y}^k_{ineq} - \mathbf{z}^k \|_\infty}{1 + \max(\| \mathbf{H}\,\mathbf{x}^k \|_\infty, \| \mathbf{c} \|_\infty,\ \| \mathbf{A}^T_{eq}\,\mathbf{y}^k_{eq} \|_\infty,\ \| \mathbf{A}^T_{ineq}\,\mathbf{y}^k_{ineq} \|_\infty,\ \| \mathbf{z}^k \|_\infty)} \end{aligned} \tag{27}$$

where

$$\begin{aligned} r^k_{\mathbf{A}_{eq}} &= \frac{\| \mathbf{A}_{eq}\,\mathbf{x}^k - \mathbf{b}_{eq} \|_\infty}{1 + \max(\| \mathbf{A}_{eq}\,\mathbf{x}^k \|_\infty,\ \| \mathbf{b}_{eq} \|_\infty)} \\ r^k_{\mathbf{A}_{ineq}} &= \frac{\| \mathbf{A}_{ineq}\,\mathbf{x}^k + \mathbf{s}^k - \mathbf{b}_{ineq} \|_\infty}{1 + \max(\| \mathbf{A}_{ineq}\,\mathbf{x}^k + \mathbf{s}^k \|_\infty,\ \| \mathbf{b}_{ineq} \|_\infty)} \\ r^k_{bounds} &= \frac{\| \mathbf{x}^k - \hat{\mathbf{x}}^k \|_\infty}{1 + \max(\| \mathbf{x}^k \|_\infty,\ \| \hat{\mathbf{x}}^k \|_\infty)} \end{aligned}$$

and set RACQP to terminate when the residuals become smaller than some tolerance level $\epsilon > 0$.

$$\max(r^k_{\text{p}},\ r^k_{\text{d}}) < \epsilon. \tag{28}$$

Note that the aforementioned residuals are similar to those used in [10,65] with relative and absolute residual tolerance $(\epsilon_{abs}, \epsilon_{rel})$ set to be equal.

3.2 Mixed integer problems

For mixed integer problems we tackle (17) without introducing $\hat{\mathbf{x}}$, where augmented Lagrangian, $L_\beta(\mathbf{x}; \mathbf{s}; \mathbf{y}_{eq}; \mathbf{y}_{ineq})$, is given by

$$\begin{aligned} L_\beta(\cdot) := \tfrac{1}{2}\,\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} - \mathbf{y}^T_{eq}(\mathbf{A}_{eq}\,\mathbf{x} - \mathbf{b}_{eq}) - \mathbf{y}^T_{ineq}(\mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} - \mathbf{b}_{ineq}) \\ + \tfrac{\beta}{2}(\| \mathbf{A}_{eq}\,\mathbf{x} - \mathbf{b}_{eq} \|^2 + \| \mathbf{A}_{ineq}\,\mathbf{x} + \mathbf{s} - \mathbf{b}_{ineq} \|^2) \end{aligned}$$

where slack variables $\mathbf{s} \geq 0$, and $x_i \in \mathcal{X}_i$, $\mathcal{X}_i \subseteq \mathbb{R}$ or $\mathcal{X}_i \subseteq \mathbb{Z}$, $i = 1, \ldots, n$. Mixed integer problems (MIP) are addressed by using the partial Lagrangian to solve for primal variables and a simple procedure that helps to escape local optima, as shown in Algorithm 2. Note that MIP and continuous problems share the same main algorithm (Algorithm 1), but the former ignores the update to $\hat{\mathbf{x}}$ as the bounds on $\mathbf{x}$ are explicitly set through $\mathcal{X}$, and thus $\hat{\mathbf{x}} = \mathbf{x}$ always.

RACQP-MIP Solver, outlined in Algorithm 2, consists of a sequence of steps that work on improving the current (or initial) solution which is then "destroyed" to be possibly improved again. This solve-perturb-solve sequence (lines 2–13) is repeated until termination criteria is met. The criteria for RACQP-MIP is usually set to be maximum run-time, maximum number of attempts to find a better solution, or a solution quality (assuming primal feasibility is met within some $\epsilon > 0$). The algorithm can be seen as a variant of a neighborhood search technique usually associated with meta-heuristic algorithms for combinatorial optimization.

Algorithm 2 RACQP-MIP

Block Optimization with Integer Variables

Require: Problem *model* (Eq. 18), *run-time parameters*† , *termination criteria*††
Perturbation parameters κ, number of trials before perturbing nP
Ensure: The optimal solution $\mathbf{x}^*$ or the best solution found before termination criteria met

1: $\mathbf{x}_{best} \leftarrow -\infty,\ k \leftarrow 0$
2: **while** *termination criteria* not met **do**
3: $\mathbf{x}^* \leftarrow$RACQP(*model*, *run-time parameters*)
4: **if** $f(\mathbf{x}^*) < f(\mathbf{x}_{best})$ **then**
5: $\mathbf{x}_{best} \leftarrow \mathbf{x}^*$
6: **else**
7: $k \leftarrow k + 1$
8: **end if**
9: **if** $k = nP$ **then**
10: $k \leftarrow 0$
11: $\mathbf{x}_0 \leftarrow$ perturb $(\mathbf{x}_{best}, \kappa)$
12: **end if**
13: **end while**
14: **return** $\mathbf{x}_{best}$

† RACQP run-time parameters (number of groups p, penalty parameter β, initial point $\mathbf{x}_0$, pre-grouped vars set $\mathcal{V}$, termination criteria).
†† RACQP-MIP termination criteria (e.g. maximum run-time, number of attempts to find a better solution, solution quality and so on).

After being stuck at some local optimum solution, the algorithm finds a new initial point $\mathbf{x}_0$ by perturbing the best known solution $\mathbf{x}_{best}$ and continues from there. The new initial point does not need to be feasible, but in some cases it may be beneficial to be constructed that way. To detect a local optimum we use a simple approach that counts number of times a "feasible" solution is found without improvement in objective value. A solution is considered to be feasible if

$$\max(\| \mathbf{A}_{eq}\, \mathbf{x} - \mathbf{b}_{eq} \|_\infty,\ \| \mathbf{A}_{ineq}\, \mathbf{x} - \mathbf{b}_{ineq} \|_\infty) \leq \epsilon,$$

$\epsilon > 0$. Perturbation (line 11) can done, for example by choosing a random number (chosen from a truncated exponential distribution) of components of $\mathbf{x}_{best}$ and assigning them new values, or a more sophisticated approach can be used (see Sect. 4.2 for some implementation details). Parameters of permutation are encapsulated in a generic term κ.

4 Computational studies

The Alternating Direction Method of Multipliers (ADMM) has nowadays gained a lot of attention for solving many problems of practical importance (e.g. large-scale machine learning and signal processing, image processing, portfolio-management, to name a few). Unfortunately, the two most popular approaches, namely the two-block classical ADMM and the variable-splitting multi-block [10], both characterized by convergence speed and scaling issues somehow hindered a wide acceptance of

ADMM as the solution method of choice for ML problems. RAC-ADMM offers the multi-block solution that may help to overcome the problem of ADMM acceptance.

The goal of this section is twofold: (1) to show that RAC-ADMM is a versatile algorithm that can be directly applied to a wide range of LCQP problems and compete with commercial solvers and (2) get an insight on specific ML problems and devise a RAC-ADMM based solution that outperforms or matches the performance of the best tailored solution method(s) in both solution time and quality. To address the former, in Sects. 4.1 and 4.2 we compare RACQP with the state of the art commercial solvers, Gurobi [34] and Mosek [55], and the academic OSQP which is a ADMM-based solver developed by [65]. To address the latter, we focus on Linear Regression (Elastic-Net) and Support Vector Machine (SVM), machine learning algorithm used for classification and regression analysis, and in Sect. 4.1.7 compare RACQP with glmnet [30,64] and LIBSVM [14].

We conduct multiple numerical tests, solving randomly constructed problems and problems from benchmark test sets. Data we collect include run-time, number of iterations until termination criteria is met and quality of a solution, defined differently for continuous, mixed-integer and machine learning problems (described in corresponding subsections). Note that in some sections, due to space concerns we report on a subset of instances. Experiments using larger sets are available together with RACQP solver code online [61] in "demo" directory.

The experiments were done on MacBook Pro with 2.8 GHZ Intel Core i7 and 16Gb memory running macOS High Sierra, v 10.13.2 (Sect. 4.1.7) and 16-core Intel Xeon CPU E5-2650 machine with 96Gb memory running Debian linux 3.16.0-4-amd64 (all other sections).

4.1 Continuous problems

The section starts with the analysis of the l_2 regularized regularized Markowitz mean–variance model applied to 2018 CSRP Quarterly Stock data [78] followed by randomly generated convex quadratic problems (QP) with coupled blocks. Next three sets of benchmark problems are addressed: relaxed QAPLIB [60] (binary constraint on variables removed), Maros and Meszaros Convex QP [72], and the Mittelmann LP test set [73] expanded to QP by adding a diagonal Hessian to the problem model.

The goal of the section is to show that the multi-block ADMM approach adapted by RACQP can significantly reduce solution time compared to commercial solvers and two-block ADMM (used by OSQP) for most of the problems we addressed. Results obtained in this section are all done with a single RACQP run, using fixed random number generator seed. Performance of the solver when subjected to different seeds is described in Sect. 4.1.8. The run-time settings applied to solvers to produce results reported in this section, unless noted otherwise, are shown in Table 3.

Authors are aware that either commercial solver can be tuned for maximum performance by adjusting run-time parameters to fit a specific problem structure, which is the same with RACQP and OSQP but to the much smaller extent. In addition, the latter do not have the access to a large number of real-world instances used by the former to fine-tune algorithms to exploit "known" problem structures nor manpower to build

Table 3 Termination criteria used in this section by all solvers

Termination criteria	Parameter value
Max time	3h
Max. num. iterations (OSQP, RACQP)	4000
Primal residual (feasibility) tolerance	$\epsilon_{prim} = \epsilon = 10^{-5}$
Dual residual (optimality) tolerance	$\epsilon_{dual} = \epsilon = 10^{-5}$
Relative residual tolerance (OSQP)	$\epsilon_{rel} = \epsilon = 10^{-5}$
Barrier convergence tolerance (Gurobi, Mosek)	$\epsilon_{comp} = \epsilon = 10^{-5}$

heuristics and/or preconditioners that boost solver performance. However, in order to create a more "fair" working conditions, we decided to let Mosek and Gurobi use their default settings, except for disabling multi-threading support and aforementioned optimality termination criteria (Table 3). Although allowing the solvers to execute presolve routines seems to be unfair to RACQP (which does not implement any presolving technique except for a very simple row scaling), disabling it would be even more unfair to the opposing solvers as their performance heavily depends on finesses of the presolve algorithm(s). Multi-threading is disabled for Mosek and Gurobi because both RACQP and OSQP are single-threaded, and leaving it on would be unfair. Finally, to make RACQP and OSQP comparison more fair, and because our target is to compare two ADMM variants, RAC-ADMM and operator splitting two-block ADMM, rather than solvers' implementations, the advanced option that OSQP uses to post-process results, "Polish results", was turned off. Note that such an option is relatively easy to implement and a variant of thereof will be added to a future RACQP version.

For continuous problems described in this section, performance is measured in terms of run-time, number of iterations and quality of solution, expressed via primal and dual residuals. Terminating a run after residual(s) have been met (Table 3, rows 2–4) is one way of ensuring quality of a solution. However, this criteria could be misleading. To start with, some solvers use absolute residuals as termination criteria (e.g. Gurobi), some depend on relative residuals (e.g. Mosek, RACQP), and some are adjustable like QSQP.

Next, solvers usually scale problems (e.g. row and column scaling of a constraint matrix) to avoid numerical problems and make matrices with favorable condition numbers. Residuals are then calculated and checked against these scaled models, meaning that a solver may prematurely terminate unless the results are periodically re-scaled and residuals recalculated on the actual model—a "good" scaled solution can actually have a very bad "actual" residual. As each solver performs different scaling (and algorithms are not usually known as it is case with Gurobi and Mosek), direct comparison of residuals reported by the solvers is not possible.

To circumvent the issue, we re-calculate primal and dual residuals using the solutions (primal and dual variables), returned by the solvers as follows:

$$
\begin{aligned}
r_{\text{prim}} &:= \max(r_{\mathbf{A}_{eq}}, r_{\mathbf{A}_{ineq}}, r_{bounds}) \\
r_{\text{dual}} &:= \frac{\| \mathbf{H}\,\mathbf{x}^* + \mathbf{c} - \mathbf{A}^T\,\mathbf{y}^* - \mathbf{y}^*_{bounds} \|_\infty}{1 + \max(\| \mathbf{H}\,\mathbf{x}^* \|_\infty, \| \mathbf{c} \|_\infty,\ \| \mathbf{A}^T\,\mathbf{y}^* \|_\infty,\ \| \mathbf{y}^*_{bounds} |_\infty)}
\end{aligned}
$$

where $\mathbf{A} = [\mathbf{A}_{eq}; \mathbf{A}_{ineq}]$, $\mathbf{y}^*$ is a vector of dual variables related to equality and inequality constraints, $\mathbf{y}^*_{bounds}$ is a vector of dual variables related to primal variable bounds, and $\mathbf{x}^*$ is a vector of primal variables. Residuals due to equality and inequality constraints and bounds are defined with

$$\begin{aligned}
r_{\mathbf{A}_{eq}} &= \frac{\| \mathbf{A}_{eq}\, \mathbf{x}^* - \mathbf{b}_{eq} \|_\infty}{1 + \max(\| \mathbf{A}_{eq}\, \mathbf{x}^* \|_\infty,\ \| \mathbf{b}_{eq} \|_\infty)} \\
r_{\mathbf{A}_{ineq}} &= \frac{\max(0, \| \mathbf{A}_{ineq}\, \mathbf{x}^* - \mathbf{b}_{ineq} \|_\infty)}{1 + \max(\| \mathbf{A}_{ineq}\, \mathbf{x}^* \|_\infty,\ \| \mathbf{b}_{ineq} \|_\infty)} \\
r_{bounds} &= \max\left(\frac{\| \max(0, \mathbf{l} - \mathbf{x}^*) \|_\infty}{1 + \max(\| \mathbf{x} \|_\infty,\ \| \mathbf{l} \|_\infty)}, \frac{\| \max(0, \mathbf{x}^* - \mathbf{u}) \|_\infty}{1 + \max(\| \mathbf{x} \|_\infty,\ \| \mathbf{u} \|_\infty)}\right).
\end{aligned}$$

Note that Gurobi does not provide dual variables for bounds ($\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$) directly. To get around we convert the bounds into inequality constraints, what makes Gurobi to produce the dual variables. This introduces negligible run-time cost as the additional constraints are discovered as bounds during presolve phase and consequently removed. The initial point $\mathbf{x}^0$ for all instances addressed by RACQP is $\max(\mathbf{0}, \mathbf{l})$.

4.1.1 Choosing RACQP solver working mode

To address differences in problem structure, the following simple rules are used to decide on the RACQP solver mode:

1. If $\mathbf{H}$ is non-diagonal and $\mathbf{A}$ is non-structural or the problem is large, use multi-block mode (Eq. 21).
2. If $\mathbf{H}$ is non-diagonal and $\mathbf{A}$ is structural, which implies that $\mathbf{A}$ has non-zero entries that follow some pattern and problem structure is easy to detect, use multi-block mode with smart-grouping as described in Sect. 2.3.2.
3. If $\mathbf{H}$ is diagonal, $m << n$ or $\mathbf{H}$ and $\mathbf{A}$ are very sparse, and the problem is of moderate size, use single-block mode (group all primal variables $\mathbf{x}$ together in one block) with localized equality constraints for the sub-problem and apply (Eq. 25).
4. If $\mathbf{H}$ is non-diagonal, both $\mathbf{H}$ and $\mathbf{A}$ are very sparse, and the problem is of moderate size, use single-block ADMM. If only a subset of primal variables is bounded, solve the block using an external solver (e.g. Gurobi or Mosek) with localized bounds. Otherwise, solve the block using (Eq. 24).

4.1.2 Regularized Markowitz mean–variance model

The Markowitz mean–variance model describes N assets characterized by a random vector of returns $\mathbf{R} = (\mathbf{R}_1, \ldots, \mathbf{R}_N)$ with known expected value $\mathbf{m}_i$ of each random variable R_i and covariance σ_{ij} for all pairs of random variables $\mathbf{R}_i$ and $\mathbf{R}_j$. Given some portfolio asset $\mathbf{x} = (x_1, \ldots, x_N)$, where x_i is the fraction of resources invested in asset i, an investor chooses a portfolio $\mathbf{x}$, satisfying two objectives: expected value of the portfolio return $\mathbf{m}_\mathbf{x} = E(\mathbf{R}_\mathbf{x}) = \langle \mathbf{m}, \mathbf{x} \rangle$ is maximized and portfolio risk, measured by variance $\sigma^2_\mathbf{x} = \mathrm{Var}(\mathbf{R}_\mathbf{x}) = \langle \mathbf{x}, \mathbf{V}\mathbf{x} \rangle$, $\mathbf{V} = (\sigma_{ij})$ is minimized [25]. The problem of

finding the optimal portfolio can be formulated as a quadratic optimization problem,

$$\begin{aligned} \min_{\mathbf{x}} \ & \mathbf{x}^T \mathbf{V} \mathbf{x} - \tau \, \mathbf{m}^T \mathbf{x} + \kappa \| \mathbf{x} \|_2^2 \\ \text{s.t.} \ & \mathbf{e}^T \mathbf{x} = 1 \\ & \mathbf{x} \in \mathbb{R}_+^n \end{aligned} \tag{29}$$

where $\tau \geq 0$ is *risk tolerance* parameter, and $\mathbf{e}$ is a vector of all ones. The above problem formulation includes the regularization term with parameter κ.

The raw data was collected by the Center for Research in Security Price (CRSP), and provided through Wharton Research Data Services [78] covering daily prices of 4628 assets from Jan 01 to Dec 31, 2018, and monthly prices for 7958 stocks from Jan 31 to Dec 31, 2018. Missing data was filled using the yearly average price. The model uses risk tolerance parameter $\tau = 1$, and is regularized with $\kappa = 10^{-5}$. For the formulation (29), because Hessian ($\mathbf{V}$) is dense and non-diagonal, the multi-block ADMM is used, following the rules on choosing the RACQP solver mode (rule 1, Sect. 4.1.1). The number of groups p is 50, and the augmented Lagrangian penalty parameter $\beta = 1$. Default run settings (Table 3) are used by all solvers, except for OSQP that had max iteration number set to 20,000.

The performance comparison between the solvers, given in Table 4, shows that multi-block RAC finds the solution of high quality in a fraction of time needed by the commercial solvers. In addition, the results show that OSQP requires many iterations to converge to a solution meeting primal/dual tolerance criteria ($\epsilon = 10^{-5}$), confirming the slow convergence issue of a 2-block ADMM approach.

Low-rank re-formulation Noting that the number of observations k is not large and that the covariance matrix $\mathbf{V}$ is of low rank and thus can be expressed as $\mathbf{V} = \mathbf{B}^T \mathbf{B}$, where

$$\mathbf{B} = \frac{1}{\sqrt{k-1}} (\mathbf{R} - \frac{1}{k} \mathbf{e} \, \mathbf{e}^T \mathbf{R}) \tag{30}$$

and $\mathbf{R} \in \mathbb{R}^{k \times N}$, with rows corresponding to time series observations, and columns corresponding to different assets, we reformulate the problem as

$$\begin{aligned} \min_{\mathbf{x}} \ & \| \mathbf{y} \|_2^2 - \tau \, \mathbf{m}^T \mathbf{x} + \kappa \| \mathbf{x} \|_2^2 \\ \text{s.t.} \ & \mathbf{e}^T \mathbf{x} = \mathbf{1} \\ & \mathbf{B} \mathbf{x} - \mathbf{y} = \mathbf{0} \\ & \mathbf{x} \in \mathbb{R}_+^n \end{aligned} \tag{31}$$

Since the Hessian of (31) is diagonal, and number of constraints is relatively small, the problem is solved using the single-block ADMM (rule 3, Sect. 4.1.1). Run-time settings are identical to those used for the regular model described previously, with the exception of the augmented Lagrangian penalty parameter which is set to $\beta = 0.1$. The performance comparison between the solvers, given in Table 5, shows that RACQP is also competitive in low-rank formulation of the problem. Run-time is given in seconds.

Table 4 Markowitz min–variance model (29)

Solver	Quarterly Data ($n = 7958$)				Monthly Data ($n = 7958$)				Daily Data ($n = 4628$)			
	Run time	Num. iter	Res. primal	Res dual	Run time	Num.iter	Res. primal	Res dual	Run time	Num. iter	Res. primal	Res dual
Gurobi	2305	11	$6 \cdot 10^{-14}$	$1 \cdot 10^{-8}$	2525	12	$9 \cdot 10^{-15}$	$8 \cdot 10^{-7}$	731	15	$6 \cdot 10^{-15}$	$5 \cdot 10^{-7}$
Mosek	162	4	$2 \cdot 10^{-5}$	$1 \cdot 10^{-8}$	188	5	$3 \cdot 10^{-5}$	$2 \cdot 10^{-8}$	68	10	$5 \cdot 10^{-5}$	$5 \cdot 10^{-6}$
OSQP	3599	20,000	$2 \cdot 10^{-3}$	$4 \cdot 10^{-7}$	4359	20,000	$6 \cdot 10^{-5}$	$5 \cdot 10^{-7}$	639	11,475	$1 \cdot 10^{-5}$	$1 \cdot 10^{-8}$
RACQP	97	356	$7 \cdot 10^{-6}$	$9 \cdot 10^{-6}$	314	1191	$7 \cdot 10^{-7}$	$1 \cdot 10^{-5}$	38	576	$2 \cdot 10^{-7}$	$1 \cdot 10^{-5}$

CRSP 2018 data [78], run-time in seconds

Table 5 Low-rank reformulation Markowitz min–variance model (31)

Solver	Quarterly Data ($n = 7958$)				Monthly Data ($n = 7958$)				Daily Data ($n = 4628$)			
	Run time	Num. iter	Res. primal	Res dual	Run time	Num.iter	Res. primal	Res dual	Run time	Num. iter	Res. primal	Res dual
Gurobi	0.1	11	$6 \cdot 10^{-14}$	$3 \cdot 10^{-9}$	0.1	10	$4 \cdot 10^{-13}$	$3 \cdot 10^{-7}$	2.2	17	$2 \cdot 10^{-15}$	$1 \cdot 10^{-6}$
Mosek	0.6	5	$4 \cdot 10^{-5}$	$4 \cdot 10^{-9}$	0.2	5	$6 \cdot 10^{-4}$	$6 \cdot 10^{-8}$	1.3	3	$6 \cdot 10^{-4}$	$3 \cdot 10^{-8}$
OSQP	1.5	2600	$3 \cdot 10^{-9}$	$3 \cdot 10^{-6}$	1.0	1175	$9 \cdot 10^{-6}$	$9 \cdot 10^{-8}$	12.5	1900	$1 \cdot 10^{-5}$	$1 \cdot 10^{-8}$
RACQP	0.6	350	$2 \cdot 10^{-12}$	$9 \cdot 10^{-6}$	0.8	767	$1 \cdot 10^{-12}$	$9 \cdot 10^{-6}$	5.8	561	$2 \cdot 10^{-13}$	$1 \cdot 10^{-5}$

CRSP 2018 data [78]

4.1.3 Randomly generated linearly constrained quadratic problems (LCQP)

In this section we analyze RACQP performance for different problem structures and run-time settings (number of blocks p, penalty parameter β, tolerance ϵ). In order to have more control over problem structure we generate synthetic problem instances starting with a simple one row Markowitz-like problem to multi-row problems of large sizes. Note that although we compare RACQP with Gurobi and Mosek on randomly generated instances, which may be considered to be unfair to the latter, our goal is not to diminish the importance of barrier type solution methods those solvers utilize, but to show that multi-block ADMM can be an approach to argument these methods when instances are large and/or dense. In this section we solve linearly constrained quadratic problems LCQP, described by (17), with $\mathbf{x} \in \mathbb{R}^n$.

Similarly to [80] we construct a positive definite Hessian matrix $\mathbf{H}$ from a random ($\sim U(0,1)$) matrix $\mathbf{U} \in \mathbb{R}^{n \times n}$ and a normalized diagonal matrix $\mathbf{V} \in \mathbb{R}^n_+$ whose elements were chosen from a log-uniform distribution to have a specific condition number:

$$\begin{aligned} \mathbf{U}_\eta &= \eta\,\mathbf{U} + (1-\eta)\,\mathbf{I} \\ \mathbf{H} &= \mathbf{U}_\eta\,\mathbf{V}\,\mathbf{U}_\eta^T + \zeta\,\mathbf{e}\,\mathbf{e}^T \end{aligned} \tag{32}$$

where parameters $\eta \in (0,1)$ and $\zeta \geq 0$ induce different types of orientation bias. For convenience we normalize matrix H and construct vector $\mathbf{c}$ as a random vector ($\sim U(0,1)$). Jacobian matrices $\mathbf{A}_{eq}$ and $\mathbf{A}_{ineq}$ are constructed in a way that the desired sparsity is met and $a_{i.j} \sim N(0,1)$ for both matrices. Our analysis of LCQP is based on extensive experimentation using different problem structure embedded in the matrix $\mathbf{H}$, by varying its orientation, condition number and the random seed used to construct $\mathbf{H}$ (and vector $\mathbf{c}$).

Markowitz-like Problem Instances RACQP implementation allows solving optimization problems by multi-block ADMM. A question that arises is the optimal number of blocks p (i.e. sub-problems) to use. The optimal number, it turns out, is related to structure and density of both Hessian and Jacobian matrices. For any $\mathbf{H}$ that is not a block matrix, and a dense $\mathbf{A}$, as is the case with the Markowitz model, the number of blocks is related to the problem size—having more blocks leads to having more iterations before the process meets the tolerance on residual error ϵ and more sub-problems to construct and solve. However, a sub-problem of a smaller size can be constructed and solved in less time than a larger sub-problem. Total time (t_T) is thus a function of opposing arguments. To show this interdependence, we solve simple Markowitz-like problem instances, with randomly generated $\mathbf{Q}$ and $\mathbf{c}$, and with $\mathbf{A}_{eq} = \mathbf{e}^T$, $\mathbf{b} = \mathbf{1}$, and $\mathbf{x} \in \mathbb{R}^n_+$ (inequity constraints are not used). Following (29), we add a regularization term to the objective function with $\kappa = 10^{-5}$.

Table 6 presents the aggregate results collected over a set of experiments (10 for each group size) using random problems constructed using (32). The reason for constructing problems in such a way is to emulate a real-world situation when a problem model (Hessian, Jacobian, $\mathbf{x}$ upper and lower bounds) do not change, but coefficients do. The results confirm that there exist a "right" number of blocks which minimizes overall run-time. For now, choosing that number is based on experience, but we are working on formalizing the procedure. In addition to run-time cost per iteration, Table 6

Table 6 RACQP performance with respect to number of blocks p for randomly generated problems of type (29)

Num blocks	Block size	Number of iterations (k)				Cost per iteration [s]		
		μ	σ^2	min	max	μ	min	max
50	180	43.2	1.87	40	46	0.147	0.133	0.152
100	90	46.6	1.51	44	49	0.095	0.094	0.095
150	60	49.0	1.25	47	51	0.091	0.090	0.092
200	45	50.6	0.97	49	52	0.108	0.107	0.109

Problem size $n = 9000$, density$_Q = 0.05$, $\epsilon = 10^{-5}$

Table 7 A typical RACQP performance with respect to primal/dual residual tolerance ϵ for a randomly generated problems of type (29)

ϵ	Number of iterations (k)				Residuals (mean values)	
	μ	σ^2	Min	Max	Primal	Dual
10^{-4}	30.4	1.43	28	33	$4.7 \cdot 10^{-7}$	$9.3 \cdot 10^{-5}$
10^{-5}	46.6	1.51	44	49	$4.7 \cdot 10^{-8}$	$9.5 \cdot 10^{-6}$
10^{-6}	63.4	1.65	60	65	$3.6 \cdot 10^{-9}$	$9.2 \cdot 10^{-7}$
10^{-7}	79.9	2.02	76	83	$3.5 \cdot 10^{-10}$	$9.5 \cdot 10^{-8}$

Problem size $n = 9000$, density$_Q = 0.05$. Number of blocks $p = 100$

reports number of iterations until convergence (k) for different number of blocks. It is interesting to observe is that k is very mildly affected by the choice of p, if tolerance ϵ is kept the same. This leads to another interesting question on how much a change in ϵ affect run-time. Table 7 gives an answer to this question. The table lists RACQP performance over the same problem set, but with different residual tolerances. As expected, results show that the number of iterations increases as the tolerance gets tighter.

General LCQP Building on the results from the previous section, we expand the QP model to include general equality and inequality constraints with unbounded variables $\mathbf{x}$. We analyze RACQP when solving sparse problems (dense problems are covered in the next section where we address relaxed QAP) for problems of size $n = 6000$ and $n = 9000$. The number of rows in both constraint matrices is equal ($m = m_{eq} = m_{ineq}$), and set to be a function of a problem size, $m = r \cdot n$, with $r = \{0.1, 0.5\}$. The number of blocks used by RACQP is related to size of a block, $p_n = n/b_{\text{size}}$, with the optimal block size b_{size} empirically determined to be 60. The penalty parameter $\beta = 1$ was found to produce the best results.

Tables 8 and 9 give comparative analysis of performance of the solvers with respect to run-time and primal/dual residuals. Although both OSQP and RACQP did well in terms of primal and dual residuals, the results show that multi-block RACQP converges to solutions much faster (4–10×) then OSQP. Both solvers outperform Gurobi and Mosek in run-time, even though the tolerance on residual error is set to the same value

Table 8 Run-time comparison between the solvers for LCQP

Problem size	Num rows	Run-time [s]			
		Gurobi	Mosek	OSQP	RACQP
6000	600	1082	208	84	9
	3000	1861	143	98	26
	900	4222	365	293	22
9000	4500	6308	408	304	65

Density = 0.05, $p_{n=6000}$ = 100, $p_{n=9000}$ = 150

($\epsilon = 10^{-5}$). Another observation is that Mosek produces solutions of inferior quality to all aforementioned solvers—dual residuals are of 10^{-3} and 10^{-4} levels, far below the requested ϵ threshold. Investigation of the log files produced by Mosek reveled two problems: (1) Mosek terminates as soon as primal or dual or complementary gap residual criteria is met (unlike the other solvers which terminate when *all* the residual criteria are met); (2) residuals are not periodically checked on a re-scaled model, resulting in a large discrepancy between internally evaluated residuals (scaled data) and the actual one.

4.1.4 Relaxed QAP

As of this section we continue the study of RACQP but, instead of randomly generating problems, we use benchmark test sets compiled by other authors which reflect real-world problems. We start by addressing large scale instances from the QAPLIB benchmark library [60] compiled by [11] and hard problems of large size, described in [21]. The quadratic assignment problem (QAP) problem is a binary problem, but for the purpose of more realistic comparison between the solvers, we relax it to a continuous problem. The numerical tests solving the binary problem formulation will be given later in Sect. 4.2.3.

The quadratic assignment problem belongs to a class of combinatorial optimization problems that arise from problems of practical interest. The QAP objective is to assign n facilities to n locations in such a way that the assignment cost is minimized. The assignment cost is the sum, over all pairs, of a weight or flow between a pair of facilities multiplied by the distance between their assigned locations. Mathematically, the QAP can be presented as follows:

$$\begin{aligned} \min_{\mathbf{X}} \ & \mathrm{vec}(\mathbf{X})^T \, \mathbf{H} \, \mathrm{vec}(\mathbf{X}) \\ \text{s.t.} \ & \textstyle\sum_{i=1}^{r} x_{ij} = 1, \ \forall j = 1, \dots r & \text{(a)} \\ & \textstyle\sum_{j=1}^{r} x_{ij} = 1, \ \forall i = 1, \dots r & \text{(b)} \\ & 0 \leq x_{ij}, \ \forall i, j = 1, \dots r & \text{(c)} \end{aligned} \tag{33}$$

where x_{ij} is the entry of the permutation matrix $\mathbf{X} \in \mathbb{R}^{r \times r}$. To make the problem convex and be admitted by Cholesky factorization, we make $\mathbf{H} \in \mathbb{R}^{n \times n}$ strict diagonally dominant, $\mathbf{H} = \hat{\mathbf{H}} + d \cdot \mathbf{I}$, where $\hat{\mathbf{H}} = (\mathbf{A} \otimes \mathbf{B})$ and $d = \max(\sum_{i=1, i \neq j}^{n} \hat{h}_{i,j}) + \delta$,

Table 9 Primal and dual residuals comparison between the solvers for LCQP

Problem size	Num rows	Residuals							
		Gurobi		Mosek		OSQP		RACQP	
		Primal	Dual	Primal	Dual	Primal	Dual	Primal	Dual
6000	600	$2.8 \cdot 10^{-13}$	$1.5 \cdot 10^{-10}$	$1.4 \cdot 10^{-6}$	$5.1 \cdot 10^{-5}$	$1.5 \cdot 10^{-9}$	$2.3 \cdot 10^{-8}$	$3.4 \cdot 10^{-7}$	$9.8 \cdot 10^{-6}$
	3000	$1.2 \cdot 10^{-12}$	$1.4 \cdot 10^{-10}$	$9.3 \cdot 10^{-6}$	$8.2 \cdot 10^{-3}$	$1.4 \cdot 10^{-9}$	$4.0 \cdot 10^{-8}$	$1.4 \cdot 10^{-6}$	$9.6 \cdot 10^{-6}$
	900	$5.7 \cdot 10^{-13}$	$1.9 \cdot 10^{-9}$	$7.0 \cdot 10^{-6}$	$6.9 \cdot 10^{-4}$	$4.9 \cdot 10^{-9}$	$4.3 \cdot 10^{-7}$	$1.6 \cdot 10^{-7}$	$8.7 \cdot 10^{-6}$
9000	4500	$1.9 \cdot 10^{-12}$	$7.8 \cdot 10^{-9}$	$8.8 \cdot 10^{-6}$	$3.0 \cdot 10^{-5}$	$2.8 \cdot 10^{-9}$	$5.5 \cdot 10^{-8}$	$9.5 \cdot 10^{-7}$	$9.4 \cdot 10^{-6}$

Density = 0.05. $p_{n=6000} = 100$, $p_{n=9000} = 150$

with δ being some small positive number and $n = r^2$. The "flow" matrix $\mathbf{A} \in \mathbb{R}^{r \times r}$ and the "distance" matrix $\mathbf{B} \in \mathbb{R}^{r \times r}$.

For QAP we apply a method for variance reduction as described in Sect. 2.3 since the assignment constraints are highly structured and observable. We group variables following a simple reasoning—given that the permutation matrix $\mathbf{X}$ is doubly stochastic, each row (or column) can be seen as a single super-variable, an integer representing a permutation order. Thus, it makes sense to make one super-variable, $\mathbf{x}_i$ for each row i of $\mathbf{X}$, so that each super-variable is of size r. For each of the experiments shown we set number of groups $p = r$ (thus we solve for one super-variable per block), and penalty parameter β to the best we found by running multiple experiments with different parameter values. We found that $\beta = r$ offered the best run-time.

The results showing performance of solvers on a selected set of large QAP instances are summarized in Tables 10 and 11. The instances were chosen in such a way to cover a variety of problem densities (Hessian) and sizes. Table 10 shows run-time and number of iterations. Note that any comparison between barrier based solvers (Gurobi and Mosek) and ADMM solvers (RACQP, OSQP) is not possible, as the solution methods are completely different, but giving the number of iterations allow us to compare performances within each class of the solvers.

Similarly to results presented previously, RACQP is the fastest solver. Solution quality (primal and dual residual tolerance) is achieved in a fraction of time required by the other solvers. The average speedup is 214×, 86× and 83× with respect to Gurobi, Mosek and OSQP respectively. OSQP, although performing a similar number of iterations as RACQP does, is much slower—splitting a large problem into two parts (OSQP executes 2-block ADMM) still leaves two large matrices to solve. On the positive side, OSQP finds better solutions (primal residual smaller by the order of magnitude). Mosek is the worst performing solver—run-time-wise it is close to OSQP, only one returned solution satisfies the dual residual (tai125e01). The other instances report the dual to be as low as 10^{-1}. Gurobi found the best solutions, except for tai125e01 and tho150 instances, when max run-time limit (3h) was reached.

4.1.5 Maros and Meszaros convex QP

The Maros and Meszaros test set [72] is a collection of convex quadratic programming examples from a variety of sources [49] of the following form

$$\begin{aligned} \min_{\mathbf{x}} \ & \tfrac{1}{2}\mathbf{x}^T \mathbf{H}\mathbf{x} + \mathbf{c}^T \mathbf{x} + c_0 \\ \text{s.t.} \ & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

with $\mathbf{H} \in \mathbb{R}^{n \times n}$ symmetric positive definite, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n$, meaning that some of components of $\mathbf{l}$ and $\mathbf{u}$ may be $-\infty$ and $+\infty$ respectively. Constant c_0 is assumed to be $|c_0| < \infty$.

As in the previous section, only a subset of instances is used in experiments. The instances were chosen in such a way to cover a variety of problem models (density, size) but also to point to strengths and weaknesses of ADMM-based algorithms. Problem

Table 10 Relaxed QAP [11,21] instances

Instance name	Problem size (n)	Density (H)	Run-time [s]				Num. iterations			
			Gurobi	Mosek	OSQP	RACQP	Gurobi	Mosek	OSQP	RACQP
dre110	12,100	0.03	375	587	1259	12	10	6	50	45
sko100a	10,000	0.68	4305	408	401	13	8	7	50	22
sko100f	10,000	0.67	4694	396	405	14	9	7	50	23
tai100a	10,000	0.96	4214	419	416	12	8	7	50	20
tai125e01	15,625	0.29	Limit	1820	1544	16	5	8	50	22
tho150	22,500	0.42	Limit	4088	4586	91	1	7	50	26
wil100	10,000	0.88	4529	497	409	13	8	9	50	19

Run-time and iteration count comparison between the solvers

Table 11 Relaxed QAP [11,21] instances. Primal and dual residuals comparison between the solvers

Instance name	Gurobi		Mosek		OSQP		RACQP	
	Primal	Dual	Primal	Dual	Primal	Dual	Primal	Dual
dre110	$9.0 \cdot 10^{-11}$	$2.6 \cdot 10^{-8}$	$2.1 \cdot 10^{-6}$	$9.7 \cdot 10^{-2}$	$8.7 \cdot 10^{-10}$	$2.1 \cdot 10^{-7}$	$7.0 \cdot 10^{-7}$	$9.0 \cdot 10^{-6}$
sko100a	$1.7 \cdot 10^{-13}$	$4.5 \cdot 10^{-8}$	$1.5 \cdot 10^{-6}$	$1.8 \cdot 10^{-2}$	$8.6 \cdot 10^{-10}$	$1.6 \cdot 10^{-7}$	$1.3 \cdot 10^{-6}$	$6.0 \cdot 10^{-6}$
sko100f	$2.5 \cdot 10^{-12}$	$4.8 \cdot 10^{-8}$	$5.4 \cdot 10^{-6}$	$4.9 \cdot 10^{-2}$	$8.6 \cdot 10^{-10}$	$1.6 \cdot 10^{-7}$	$1.6 \cdot 10^{-6}$	$8.9 \cdot 10^{-6}$
tai100a	$7.5 \cdot 10^{-13}$	$1.2 \cdot 10^{-9}$	$1.5 \cdot 10^{-7}$	$3.0 \cdot 10^{-3}$	$8.6 \cdot 10^{-10}$	$1.6 \cdot 10^{-7}$	$6.4 \cdot 10^{-6}$	$4.7 \cdot 10^{-6}$
tai125e01	NA	NA	$2.1 \cdot 10^{-7}$	$8.4 \cdot 10^{-5}$	$8.7 \cdot 10^{-10}$	$2.1 \cdot 10^{-7}$	$1.9 \cdot 10^{-6}$	$9.1 \cdot 10^{-6}$
tho150	NA	NA	$2.3 \cdot 10^{-6}$	$2.3 \cdot 10^{-2}$	$8.7 \cdot 10^{-10}$	$2.5 \cdot 10^{-7}$	$1.2 \cdot 10^{-6}$	$7.4 \cdot 10^{-6}$
wil100	$3.2 \cdot 10^{-13}$	$4.5 \cdot 10^{-9}$	$9.3 \cdot 10^{-6}$	$1.5 \cdot 10^{-1}$	$8.6 \cdot 10^{-10}$	$1.6 \cdot 10^{-7}$	$9.8 \cdot 10^{-6}$	$8.1 \cdot 10^{-6}$

sizes n range from $4 \cdot 10^3$ to almost 10^5 with the number of constraints m up to 10^5. The Hessian matrices are either diagonal, with number of non-zero diagonal elements less or equal to n, or symmetric with no nonzero diagonal elements. The constraint matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ are very sparse across the problems; for most of the instances density is below 10^{-3}. In addition to being sparse, the Jacobian matrices are not block separable.

RACQP mode was set to a single-block mode according to the rules 3 and 4 of Sect. 4.1.1, with $\beta = 1$ for all instances except for CONT* and UBH1 which use $\beta = 350$ and $\beta = 12{,}000$ respectively. Residual tolerance of $\epsilon = 10^{-4}$ was used in producing the results, reported in Tables 12 and 13. The tolerance is lower than the default one (10^{-5}) because ADMM methods had hard time converging on CONT* and CVXQP* instances for tighter residuals (max number of iterations limit is 4000). Positive-definite instances (i.e. $\mathbf{H} \succ 0$) are marked with †.

Overall, for solving sparse and Hessian-diagonal problems, both Gurobi and Mosek seem more robust than OSQP and RACQP, probably due to the linear programming structure. The latter two are of the comparable performance. The results, in terms of the gap are of similar quality, and run-time is approximately the same, except for a couple of instances, where self-adjusting methodology used by OSQP for penalty parameter estimation, gives OSQP speed advantage. Also, some of the run-time variation can also be contributed to different languages used to implement solvers; OSQP is implemented in c/c++ while RACQP uses Matlab.

RACQP solved more instances than OSQP, which in addition to not being able to meet primal/dual residuals for 25% of instances, it also could not find a feasible solution for HUES-MOD and HUETIS instances. Mosek residual issue reported in the previous section continues to persists on these problem instances. For example AUG2DQP instance solution has dual residual of $5.5 \cdot 10^{-2}$, the value that does not meet the requested tolerance.

4.1.6 Convex QP based on the Mittelmann LP test set

In this section we report on the performance of solvers when applied to very large quadratic problems. Instances are taken from the Mittelmann LP test set [73] augmented with a diagonal Hessian H to form a standard LCQP (17). The results are shown in Tables 14 and 15. Residual tolerance was set to 10^{-4} (OSQP could not solve any instance but i_n13 when default tolerance of 10^{-5} was used, and RACQP had hard time with nug30). Other default termination criteria apply (Table 3). For all instances the number of blocks was set to $p = 200$ and penalty parameter, to $\beta = 5$ except for nug30 that used $\beta = 50$.

RACQP solved very large ($n > 750{,}000$) quadratic problems to the required accuracy ($\epsilon = 10^{-5}$) very fast. The results were obtained using different solution strategies: multi-block Cholesky factorization approach for wide15, square15 and long15 instances, and the partial Lagrangian approach for nug30 (localized lower and upper bound of sub-problem primal variables). The best set of parameters were found by a brute-force approach, which implies that additional research work needs to be done to identify algebraic methods to characterize instances so that run-time parameters can be chosen automatically. RACQP was unable to find a solution satisfying both

Table 12 Large Maros and Meszaros [72] instances

Instance name	Problem size (n)	Num. rows	Density (A)	Run-time [s]				Num. iterations			
				Gurobi	Mosek	OSQP	RACQP	Gurobi	Mosek	OSQP	RACQP
AUG2DC†	20,200	10,000	$2{\cdot}10^{-4}$	0.1	0.8	0.1	0.6	1	4	50	1
AUG2DQP	20,200	10,000	$2{\cdot}10^{-4}$	0.3	0.6	1.6	3.8	15	12	800	238
AUG3DC†	3873	1000	$2{\cdot}10^{-3}$	0.0	0.1	0.0	0.0	1	4	50	1
AUG3DQP	3873	1000	$2{\cdot}10^{-3}$	0.0	0.1	0.0	0.1	13	10	100	154
BOYD1†	93,261	18	$3{\cdot}10^{-1}$	0.8	2.0	31.6	10.3	21	19	3325	826
CONT-050†	2597	2401	$2{\cdot}10^{-3}$	0.1	0.2	1.2	2.4	10	10	2100	2058
CONT-100†	10,197	9801	$5{\cdot}10^{-4}$	0.4	1.0	15.4	10.5	10	13	Limit	852
CONT-101	10,197	10,098	$5{\cdot}10^{-4}$	0.3	0.8	15.8	40.0	9	11	Limit	2839
CONT-300	90,597	90,298	$1{\cdot}10^{-4}$	5.5	12.0	278.0	877.8	10	13	Limit	3405
CVXQP1_L	10,000	5000	$3{\cdot}10^{-4}$	23.5	15.3	77.6	67.9	10	8	Limit	Limit
CVXQP2_L	10,000	2500	$3{\cdot}10^{-4}$	6.4	15.4	23.0	4.3	9	15	1475	248
DTOC3	14,999	9998	$2{\cdot}10^{-4}$	0.0	0.2	0.2	0.1	1	4	275	65
HUES-MOD†	10,000	2	~ 1	0.0	0.1	0.1	0.0	10	6	200	34
HUESTIS†	10,000	2	~ 1	0.0	0.1	0.0	0.0	11	8	75	34
UBH1	18,009	12,000	$2{\cdot}10^{-4}$	0.1	0.1	0.1	0.5	5	4	75	1

Run-time and iteration count comparison between the solvers
†Positive-definite instances ($H \succ 0$)

Table 13 Large Maros and Meszaros [72]

Instance name	Gurobi		Mosek		OSQP		RACQP	
	Primal	Dual	Primal	Dual	Primal	Dual	Primal	Dual
AUG2DC†	$8.0 \cdot 10^{-13}$	$9.7 \cdot 10^{-7}$	$9.4 \cdot 10^{-8}$	$6.8 \cdot 10^{-8}$	$3.6 \cdot 10^{-10}$	$3.9 \cdot 10^{-11}$	$1.6 \cdot 10^{-12}$	$1.0 \cdot 10^{-16}$
AUG2DQP	$1.1 \cdot 10^{-14}$	$1.5 \cdot 10^{-8}$	$7.5 \cdot 10^{-5}$	$5.5 \cdot 10^{-2}$	$2.3 \cdot 10^{-6}$	$5.1 \cdot 10^{-5}$	$8.9 \cdot 10^{-8}$	$9.8 \cdot 10^{-5}$
AUG3DC†	$1.4 \cdot 10^{-14}$	$9.5 \cdot 10^{-7}$	$4.8 \cdot 10^{-10}$	$8.3 \cdot 10^{-11}$	$7.9 \cdot 10^{-10}$	$4.4 \cdot 10^{-10}$	$1.2 \cdot 10^{-14}$	$8.8 \cdot 10^{-17}$
AUG3DQP	$8.9 \cdot 10^{-16}$	$3.0 \cdot 10^{-7}$	$3.1 \cdot 10^{-4}$	$9.2 \cdot 10^{-3}$	$4.2 \cdot 10^{-5}$	$5.5 \cdot 10^{-6}$	$9.6 \cdot 10^{-5}$	$2.6 \cdot 10^{-6}$
BOYD1†	$9.4 \cdot 10^{-15}$	$1.5 \cdot 10^{-11}$	$1.4 \cdot 10^{-6}$	$8.9 \cdot 10^{-4}$	$1.7 \cdot 10^{-8}$	$9.9 \cdot 10^{-5}$	$6.7 \cdot 10^{-5}$	$9.9 \cdot 10^{-5}$
CONT-050†	$2.2 \cdot 10^{-15}$	$2.5 \cdot 10^{-7}$	$4.0 \cdot 10^{-6}$	$9.6 \cdot 10^{-11}$	$4.1 \cdot 10^{-6}$	$2.2 \cdot 10^{-6}$	$9.9 \cdot 10^{-5}$	$1.1 \cdot 10^{-5}$
CONT-100†	$2.6 \cdot 10^{-14}$	$5.3 \cdot 10^{-7}$	$1.4 \cdot 10^{-6}$	$2.3 \cdot 10^{-9}$	$2.6 \cdot 10^{-4}$	$5.0 \cdot 10^{-8}$	$9.8 \cdot 10^{-5}$	$8.3 \cdot 10^{-5}$
CONT-101	$4.3 \cdot 10^{-10}$	$3.6 \cdot 10^{-7}$	$9.8 \cdot 10^{-6}$	$4.6 \cdot 10^{-7}$	$1.8 \cdot 10^{-3}$	$6.2 \cdot 10^{-7}$	$9.9 \cdot 10^{-5}$	$9.8 \cdot 10^{-5}$
CONT-300	$9.4 \cdot 10^{-9}$	$9.6 \cdot 10^{-7}$	$4.9 \cdot 10^{-8}$	$3.0 \cdot 10^{-6}$	$8.8 \cdot 10^{-3}$	$1.1 \cdot 10^{-5}$	$9.9 \cdot 10^{-5}$	$9.1 \cdot 10^{-5}$
CVXQP1_L	$4.5 \cdot 10^{-8}$	$3.0 \cdot 10^{-8}$	$4.7 \cdot 10^{-5}$	$1.5 \cdot 10^{-3}$	$1.2 \cdot 10^{-4}$	$1.1 \cdot 10^{-5}$	$7.6 \cdot 10^{-3}$	$2.5 \cdot 10^{-5}$
CVXQP2_L	$9.2 \cdot 10^{-12}$	$2.8 \cdot 10^{-11}$	$9.7 \cdot 10^{-9}$	$6.7 \cdot 10^{-5}$	$6.2 \cdot 10^{-5}$	$5.3 \cdot 10^{-8}$	$4.0 \cdot 10^{-6}$	$9.9 \cdot 10^{-5}$
DTOC3	$5.2 \cdot 10^{-11}$	$6.3 \cdot 10^{-9}$	$6.7 \cdot 10^{-10}$	$2.4 \cdot 10^{-13}$	$9.2 \cdot 10^{-5}$	$2.6 \cdot 10^{-6}$	$7.2 \cdot 10^{-13}$	$9.9 \cdot 10^{-5}$
HUES-MOD†	$2.8 \cdot 10^{-15}$	$3.4 \cdot 10^{-7}$	$7.2 \cdot 10^{-5}$	$1.0 \cdot 10^{-1}$	NA*	NA*	$8.5 \cdot 10^{-5}$	$8.6 \cdot 10^{-6}$
HUESTIS†	$9.8 \cdot 10^{-15}$	$4.8 \cdot 10^{-9}$	$3.9 \cdot 10^{-6}$	$1.1 \cdot 10^{-2}$	NA*	NA*	$8.5 \cdot 10^{-5}$	$8.6 \cdot 10^{-6}$
UBH1	$1.6 \cdot 10^{-10}$	$8.2 \cdot 10^{-9}$	$2.8 \cdot 10^{-4}$	$1.3 \cdot 10^{-4}$	$9.8 \cdot 10^{-5}$	$7.8 \cdot 10^{-6}$	$1.5 \cdot 10^{-5}$	$6.0 \cdot 10^{-8}$

Primal and dual residuals comparison between the solvers
†Positive-definite instances ($H \succ 0$)
*No feasible solution found

Table 14 Convex QP based on the Mittelmann LP test set [73]

Instance name	Problem size (n)	Num. rows	Density (A)	Run-time [s]				Num. iterations			
				Gurobi	Mosek	OSQP	RACQP	Gurobi	Mosek	OSQP	RACQP
nug30	753,687	32,769	$7.91 \cdot 10^{-5}$	9109	6738	Limit	3976	13	7	1101	1057
wide15	753,526	32,762	$6.10 \cdot 10^{-5}$	9194	5267	Limit	345	18	13	1407	136
square15	753,690	32,769	$6.10 \cdot 10^{-5}$	9158	3733	Limit	363	18	9	1573	142
long15	379,350	52,260	$6.10 \cdot 10^{-5}$	8959	5238	Limit	332	18	13	1534	136
i_n13	741,455	8192	$2.44 \cdot 10^{-4}$	71	40	156	Limit	28	10	275	2171
16_n14	262,144	16,384	$1.22 \cdot 10^{-4}$	24	13	267	2847	38	10	Limit	Limit

Run-time and iteration count comparison between the solvers

Table 15 Convex QP based on the Mittelmann LP test set [73]

Instance name	Gurobi		Mosek		OSQP		RACQP	
	Primal	Dual	Primal	Dual	Primal	Dual	Primal	Dual
nug30	$6.3 \cdot 10^{-15}$	$5.5 \cdot 10^{-9}$	$5.1 \cdot 10^{-2}$	$1.7 \cdot 10^{-4}$	$1.7 \cdot 10^{-4}$	$9.6 \cdot 10^{-6}$	$9.5 \cdot 10^{-6}$	$9.3 \cdot 10^{-5}$
wide15	$1.9 \cdot 10^{-13}$	$5.5 \cdot 10^{-13}$	$3.1 \cdot 10^{-5}$	$2.2 \cdot 10^{-3}$	$4.1 \cdot 10^{-9}$	$2.4 \cdot 10^{-3}$	$1.5 \cdot 10^{-5}$	$7.1 \cdot 10^{-5}$
square15	$8.3 \cdot 10^{-16}$	$6.4 \cdot 10^{-13}$	$4.6 \cdot 10^{-5}$	$1.4 \cdot 10^{-3}$	$2.1 \cdot 10^{-9}$	$7.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-5}$	$9.4 \cdot 10^{-5}$
long15	$1.9 \cdot 10^{-13}$	$5.5 \cdot 10^{-13}$	$3.1 \cdot 10^{-5}$	$2.2 \cdot 10^{-3}$	$4.0 \cdot 10^{-9}$	$1.1 \cdot 10^{-3}$	$1.5 \cdot 10^{-5}$	$7.1 \cdot 10^{-5}$
i_n13	$1.5 \cdot 10^{-13}$	$2.1 \cdot 10^{-12}$	$3.1 \cdot 10^{-4}$	$1.5 \cdot 10^{-1}$	$6.7 \cdot 10^{-6}$	$1.0 \cdot 10^{-5}$	$2.3 \cdot 10^{-3}$	$8.3 \cdot 10^{-5}$
16_n14	$1.1 \cdot 10^{-14}$	$4.4 \cdot 10^{-12}$	$2.0 \cdot 10^{-4}$	$2.2 \cdot 10^{-1}$	$1.0 \cdot 10^{-4}$	$7.4 \cdot 10^{-5}$	$1.4 \cdot 10^{-3}$	$4.2 \cdot 10^{-5}$

Primal and dual residuals comparison between the solvers

primal and dual residual tolerances for two instances (i_n13 and 16_n14), no matter of what run-time settings we used.

OSQP solved only one instance (i_n13) within given run-time and number of iterations limitations, while Gurobi solved all the instances to a high precision, regardless of having termination criteria, Table 3, set to $\epsilon = 10^{-5}$. Mosek did not find a single solution meeting the residual criteria, due to the aforementioned scaling and termination criteria issue.

4.1.7 Selected machine learning problems

In this section we apply RAC method and RP method to few selected machine learning (ML) problems related to convex quadratic optimization, namely Linear Regression (Elastic-Net) and Support Vector Machine (SVM). To solve the former we apply a specialized implementation of RAC-ADMM (available for download at [61]), while for the latter we use RACQP solver. RAC-ADMM is compared with the specialized methods, Glmnet [30,64] for Elastic-Net and LIBSVM [14] for SVM problems. The results show that our general-purpose solver matches and under certain circumstances exceeds the performance of those tailored methods.

Linear Regression using Elastic Net For a classical linear regression model, with observed features $\mathbf{X} \in \mathbb{R}^{n \times p}$ and labels $\mathbf{y} \in \mathbb{R}^n$, where n is number of observations and p is number of features, one solves the following unconstrained optimization problem

$$\min_{\boldsymbol{\beta}} \ \frac{1}{2n}(\mathbf{y} - \mathbf{X}\,\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\,\boldsymbol{\beta}) + P_{\lambda,\alpha}(\boldsymbol{\beta}) \tag{34}$$

with $P_{\lambda,\alpha}(\boldsymbol{\beta}) = \lambda\{\frac{1-\alpha}{2}\| \boldsymbol{\beta} \|_2^2 + \alpha\| \boldsymbol{\beta} \|_1\}$ used for Elastic Net model. By adjusting α and λ, one could obtain different models: for ridge regression, $\alpha = 0$, for lasso $\alpha = 1$, and for classic linear regression, $\lambda = 0$. For the problem to be solved by ADMM, we use variable splitting and reformulate the problem as follows

$$\begin{aligned} &\min_{\boldsymbol{\beta}} \ \tfrac{1}{2n}(\mathbf{y} - \mathbf{X}\,\beta)^T(\mathbf{y} - \mathbf{X}\,\beta) + P_{\lambda,\alpha}(\mathbf{z}) \\ &\text{s.t.} \ \ \boldsymbol{\beta} - \mathbf{z} \ = \ \mathbf{0} \end{aligned} \tag{35}$$

Note that in (35) we follow the standard machine learning Elastic Net notation in which β is the decision variable in the optimization formulation, rather than $\mathbf{x}$.

Let $\mathbf{c} = -\frac{1}{n}\,\mathbf{X}^T\,\mathbf{y}$, $\mathbf{A} = \frac{\mathbf{X}}{\sqrt{n}}$, and let γ denote the augmented Lagrangian penalty parameter with respect to constraint $\boldsymbol{\beta} - \mathbf{z}$, and $\boldsymbol{\xi}$ be the dual with respect to constraint $\boldsymbol{\beta} - \mathbf{z} \ = \ \mathbf{0}$. The augmented Lagrangian could then be written as

$$L_\lambda = \tfrac{1}{2}\,\boldsymbol{\beta}^T(\mathbf{A}^T\,\mathbf{A} + \gamma\,\mathbf{I})\,\boldsymbol{\beta} + (\mathbf{c} - \boldsymbol{\xi})^T\,\boldsymbol{\beta} + (\boldsymbol{\xi} - \gamma\,\boldsymbol{\beta})^T\,\mathbf{z} + \tfrac{\gamma}{2}\,\mathbf{z}^T\,\mathbf{z} + P_{\lambda,\alpha}(\mathbf{z})$$

We apply RAC-ADMM algorithm by partitioning β into multi-blocks, but solve z as one block. For any given $\boldsymbol{\beta}_{k+1}$, optimizer $\mathbf{z}^*_{k+1}$ has the closed form solution.

$$\mathbf{z}^*_{k+1}(i)(\boldsymbol{\beta}_{k+1}(i), \boldsymbol{\xi}_k(i)) = \frac{S(\boldsymbol{\xi}_k(i) - \gamma\,\boldsymbol{\beta}_{k+1}(i), \lambda\alpha)}{(1-\alpha)\lambda + \gamma},$$

where $\boldsymbol{\xi}_i$ is the dual variable with respect to constraint $\boldsymbol{\beta}_i - \mathbf{z}_i = 0$, and $S(a, b)$ is soft-threshold operation [29].

$$S(a,b) = \begin{cases} -(a-b), & \text{if } b < |a|,\ a > 0 \\ -(a+b), & \text{if } b < |a|,\ a \leq 0 \\ 0, & \text{if } b \geq |a| \end{cases}$$

In order to solve classic linear regression directly, $\mathbf{X}^T\mathbf{X}$ must be positive definite which can not be satisfied for $p > n$. However, RAC-ADMM only requires each sub-block $\mathbf{X}_{sub}^T\mathbf{X}_{sub}$ to be positive definite, so, as long as block size $s < n$, RAC-ADMM can be used to solve the classic linear regression.

It is worth pointing out that although the objective function here is non-smooth, we could still reformulate the problem as convex quadratic programming with bounded constraints. To see this, consider the case where $\alpha = 1$, and elastic net problem becomes a classic lasso regression, with (35) becomes

$$\begin{aligned} \min_{\boldsymbol{\beta}} \ & \tfrac{1}{2N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda||\mathbf{z}||_1 \\ \text{s.t.} \ & \boldsymbol{\beta} - \mathbf{z} = \mathbf{0} \end{aligned} \tag{36}$$

Optimization problem 35 is equivalent as

$$\begin{aligned} \min_{\boldsymbol{\beta},\mathbf{z}',\mathbf{z}''} \ & \tfrac{1}{2N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda(\mathbf{z}' + \mathbf{z}'') \\ \text{s.t.} \ & \boldsymbol{\beta} - \mathbf{z}' + \mathbf{z}'' = \mathbf{0} \\ & \mathbf{z}', \mathbf{z}'' \geq 0 \end{aligned}$$

Let $\boldsymbol{\xi}$ be the dual with respect to constraint $\boldsymbol{\beta} - \mathbf{z}' + \mathbf{z}'' = \mathbf{0}$. The optimal $(\mathbf{z}')^k$ and $(\mathbf{z}'')^k$ at each iteration satisfies $(\mathbf{z}')^k - (\mathbf{z}'')^k = \frac{1}{\gamma}S(\boldsymbol{\xi}_k(i) - \gamma\,\boldsymbol{\beta}_{k+1}(i), \lambda)$, which equals to $\mathbf{z}^k$ when we solves problem (36). Essentially, the non-smooth objective here could be reformulated as a smooth convex quadratic programming with non-negative constraints. Using the absolute value update is just a compact way to implement the algorithm, so that the expected convergence is guaranteed from our theorem.

We compare our solver with Glmnet [30,64] and Matlab lasso implementation on synthetic data (sparse and dense problems) and benchmark regression data from LIBSVM [14].

Synthetic Data The data set for dense problems $\mathbf{X}$ is generated uniform randomly with $n = 10,000$, $p = 50,000$, with zero sparsity, while for the ground truth β^* we use standard Gaussian and set sparsity of β^* to 0.1. Due to the nature of the problem, estimation requires lower feasibility precision, so we fix number of iterations to 10 and 20. Glmnet solver benefits from having a diminishing sequence of λ, but given that many applications (e.g. see [3]) require a fixed λ value , we decided to use fixed λ for all solvers. Note that the computation time of RAC-ADMM solver is invariant regardless of whether λ is decreasing or fixed.

Table 16 Comparison on solver performance, dense elastic net model

λ	Num. iterations	Absolute l_2 loss				Total time [s]			
		RAC	RP	glmnet	Matlab	RAC	RP	glmnet	Matlab
0.01	10	204.8	204.6	213.9	249.1	396.5	227.6	2465.9	1215.2
	20	208.1	230.2	213.9	237.1	735.2	343.9	3857.9	2218.2
0.1	10	217.8	215.6	220.5	213.1	388.7	212.5	4444.3	2125.9
	20	272.6	202.4	220.5	212.4	739.7	337.2	4452.4	2434.6
1	10	213.6	209.0	203.1	210.5	415.3	213.6	3021.1	1138.9
	20	213.8	212.4	210.5	203.1	686.3	392.1	5295.5	1495.6

Dense problem, $n = 10,000,\ \ p = 50,000$

Table 16 reports on the average cross-validation run-time and the average absolute l_2 loss for all possible pairs (α, λ) with parameters chosen from $\alpha = \{0,\ 0.1, \ldots, 1\}$ and $\lambda = \{1,\ 0.01\}$. Without specifying, RAC-ADMM solver run-time parameters were identical across the experiments, with augmented Lagrangian penalty parameter $\gamma = 0.1\lambda$ for sparsity < 0.995, $\gamma = \lambda$ for sparsity > 0.995, and block size $s = 100$. Large scale sparse data set $\mathbf{X}$ is generated uniform randomly with $\{n = 40,000,\ \ p = 4,000,000\}$, using sparsity $= 0.998$. For ground truth β^*, the standard Gaussian with sparsity $\beta^* = 0.5$ and fixed λ. Noticing from the previous experiment that increasing a step size from 10 to 20 didn't significantly improve prediction error, we fix number of iteration to 10.

Table 17, report on the average cross-validation run time and the average absolute l_2 loss for all possible pairs (α, λ) with parameters chosen from $\alpha = \{0,\ 0.1, \ldots, 1\}$ and $\lambda = \{1,\ 0.01\}$. The table also shows the best l_2 loss for each solver. Because it took more than 10,000 seconds for Matlab lasso to solve even one estimation, the table reports only comparison between glmnet and RAC.

Experimental results on synthetic data show that RAC-ADMM solver outperforms significantly all other solvers in total time while being competitive in absolute l_2 loss. Further RAC-ADMM speedups could be accomplished by fixing block-structure (RP-ADMM). In terms of run-time, for dense problem, RAC-ADMM is 3 times faster compared with Matlab lasso and 7 times faster compared with glmnet. RP-ADMM is 6 times faster compared with Matlab lasso, and 14 times faster compared with glmnet. For sparse problem, RAC-ADMM is more than 30 times faster compared with Matlab lasso, and 3 times faster compared with glmnet. RP-ADMM is 4 times faster compared with glmnet.

Following Corollary 2 RP-ADMM is slower that RAC-ADMM when convergence is measured in number of iterations, and experimental evidence (Table 1) show that it also suffers from slow convergence to a high precision level on L1-norm of equality constraints. However, the benefit of RP-ADMM is that it could store pre-factorized sub-block matrices, as block structure is fixed at each iteration,in contrast to RAC-ADMM which requires reformulation of sub-blocks at each iteration, what it turn makes each iteration more time-wise costly . In many machine learning problems, including regression, due to the nature of problem, a less precision level is required.

Table 17 Comparison on solver performance, elastic net model

λ	Num. iterations	Avg Absolute l_2 loss			Best Absolute l_2 loss			Total time [s]		
		RAC	RP	glmnet	RAC	RP	glmnet	RAC	RP	glmnet
0.01	10	1293.3	1356.7	8180.3	745.2	703.52	4780.2	4116.1	2944.8	17,564.2
0.1	10	777.31	717.92	4050.4	613.9	611.79	3125.6	3756.3	2989.1	12,953.7
1	10	676.17	671.23	3124.5	615.7	614.79	1538.9	3697.8	3003.8	8290.5

Sparse problem, $n = 40{,}000$, $p = 4{,}000{,}000$

Table 18 E2006-tfidf performance summary for Lasso problem ($\alpha = 1, \lambda = 0.01$)

Solver	Training ME	Total time [s]
OSQP	64.0	1482.5
Matlab	61.1	3946.6

Table 19 E2006-tfidf performance summary

λ	Training ME			Total time [s]		
	RAC	RP	glmnet	RAC	RP	glmnet
0.01	22.4	22.4	29.9	106.5	50.9	653.2
0.1	22.1	22.1	22.7	100.5	51.9	269.3
1	25.7	25.7	23.5	102.5	54.2	282.9

This makes RP-ADMM an attractive approach, as it could converge within fewer steps and potentially be faster than RAC-ADMM. In addition, while performing simulations we observed that increasing number of iteration does not significantly improve performance of prediction. In fact, absolute l_2 loss remains similar even when number of iteration is increased to 100. This further gives an advantage to RP-ADMM, as it benefits the most when number of iteration is relatively small.

LIBSVM Benchmark instances Regression data E2006-tfidf feature size is 150,360 with number of training and testing data points of 16,087 and 3308 respectively. The null training error of test set is 221.8758. Following the findings from the section on synthetic problems and noticing that this dataset is sparse (density=0.991), this setup uses fixed number of iterations to 10, and vary $\lambda = \{1, 0.01\}$ and $\alpha = \{0, 0.1, 0.2, \ldots, 1\}$. The training set is used to predict β^*, and the model error (ME) of test set is compared across different solvers.

Table 18 shows the performance of OSQP and Matlab lasso for $\alpha = 1$ and $\lambda = 0.01$, and Table 19 compares compare RAC-ADMM with glmnet. The reason for splitting the results in two tables is related to inefficiency of factorizing a big matrix by OSQP solver and Matlab lasso implementation. Each solver requires more than than 1000 seconds to solve the problem for even 10 iterations, making them impractical to use. On the other hand, glmnet, which uses a cyclic coordinate descent algorithm on each variable, performs significantly faster than OSQP and Matlab lasso. However, glmnet can still be inefficient, as a complete cycle through all p variables requires $O(pN)$ operations [30]. Results given in Table 19 are the averages over run-time and training error collected from experiments with $\alpha = \{0, 0.1, \ldots, 1\}$. The results show that RAC-ADMM is faster than glmnet for all different parameters and that it achieves the best training model error, 22.0954, among all the solvers. In terms of run-time, RAC-ADMM is 14 times faster than OSQP, 38 times faster than Matlab lasso, and 4 times faster than glmnet. RP-ADMM is 28, 18 and 8 times faster than OSQP, Matlab lasso and glmnet, respectively.

For log1pE2006 benchmark , feature size is 4,272,227, number of training data is 16,087 and number of testing data is 3308. The null training error of test set is 221.8758 and sparsity of data is 0.998. Similarly to the previous benchmark, the performance

Table 20 log1pE2006 performance summary for Lasso problem ($\alpha = 1, \lambda = 0.01$)

Solver	Training ME	Total time [s]
OSQP	66.6	11,437.4
Matlab	–	> 3 days

Table 21 log1pE2006 performance summary

λ	Training ME			Total time [s]		
	RAC	RP	glmnet	RAC	RP	glmnet
0.01	43.0	41.8	22.0	962.2	722.5	7639.6
0.1	30.8	31.8	22.5	978.7	721.4	4945.2
1	32.1	35.5	29.3	958.5	749.2	1889.5

results are split into two tables. Table 20 shows the performance of OSQP and Matlab lasso, while Table 21 compares RAC-ADMM and glmnet.

The results show that RAC-ADMM and RP-ADMM are still competitive and are of same level as glmnet with respect to model error, and all outperform OSQP and Matlab. In terms of run-time, RAC-ADMM is 12 times faster than OSQP, and 5 times faster than glmnet. RP-ADMM is 16 and 7 times faster than OSQP and glmnet, respectively.

Support Vector Machine A Support Vector Machine (SVM) is a machine learning method for classification, regression, and other learning tasks. The method learns a mapping between the features $\mathbf{x}_i \in \mathbb{R}^r$, $i = 1, \ldots n$ and the target label $y_i \in \{-1, 1\}$ of a set of data points using a *training set* and constructs a hyperplane $\mathbf{w}^T \phi(\mathbf{x}) + b$ that separates the data set. This hyperplane is then used to predict the class of further data points. The objective uses Structural Risk Minimization principle which aims to minimize the empirical risk (i.e. misclassification error) while maximizing the confidence interval (by maximizing the separation margin) [74,75].

Training an SVM is a convex optimization problem, with multiple formulations, such as C-support vector classification (C-SVC), υ-support vector classification (υ-SVC), ϵ-support vector regression (ϵ-SVR), and many more. As our goal is to compare RACQP, a general QP solver, with specialized SVM software and not to compare SVM methods themselves, we decided on using C-SVC [8,18], with the dual problem formulated as

$$\begin{aligned} \min_{\mathbf{z}} \ & \tfrac{1}{2}\mathbf{z}^T \mathbf{Q}\,\mathbf{z} - \mathbf{e}^T \mathbf{z} \\ \text{s.t.} \ & \mathbf{y}^T \mathbf{z} = 0 \\ & \mathbf{z} \in [0, C] \end{aligned} \tag{37}$$

with $\mathbf{Q} \in \mathbb{R}^{n \times n}$, $\mathbf{Q} \succeq 0$, $q_{i,j} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, where $K(\mathbf{x}_i, \mathbf{x}_j) := \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ is a kernel function, and regularization parameter $C > 0$. The optimal $\mathbf{w}$ satisfies $\mathbf{w} = \sum_{i=1}^n y_i\, \mathbf{z}_i\, \phi(\mathbf{x}_i)$, and the bias term b is calculated using the support vectors that lie on the margins (i.e. $0 < \mathbf{z}_i < C$) as $b_i = \mathbf{w}^T \phi(\mathbf{x}_i) - y_i$. To avoid numerical stability issues, b is then found by averaging over b_i. The decision function is defined with $f(\mathbf{x}) = \text{Sign}(\mathbf{w}^T \phi(\mathbf{x}) + b)$.

We compare RACQP with LIBSVM [14], due its popularity, and with Matlab-SVM, due to its ease of use. These methods implement specialized approaches to address the SVM problem (e.g. LIBSVM uses a Sequential Minimal Optimization, SMO, type decomposition method [9,26]), while our approach solves the optimization problem (37) directly.

The LIBSVM benchmark library provides a large set of instances for SVM, and we selected a representative subset: training data sets with sizes ranging from 20,000 to 580,000; number of features from eight to 1.3 million. We use the test data sets when provided, otherwise, we create test data by randomly choosing 30% of testing data and report cross-validation accuracy results.

In Table 22 we report on model training run-time and accuracy, defined as (num. correctly predicted data)/(total testing data size)$\times$100%. RAC-ADMM parameters were as follows: max block size $s = 100, 500$, and 1000 for small, medium and large instances, respectively and augmented Lagrangian penalty $\beta = 0.1p$, where p is the number of blocks, which in this case is found to be $p = \lceil n/s \rceil$ with n being the size of training data set. In the experiments we use Gaussian kernel, $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2\sigma^2} \| \mathbf{x}_i - \mathbf{x}_j \|^2)$. Kernel parameters σ and C were estimated by running a *grid-check* on cross-validation. We tried different pairs (C, σ) and picked those that returned the best cross-validation accuracy (done using randomly choose 30% of train data) when instances were solved using RAC-ADMM. Those pairs were then used to solve the instances with LIBSVM and Matlab. The pairs were chosen from a relatively coarse grid, $\sigma, C \in \{0.1, 1, 10\}$ because the goal of this experiment is to compare RAC-ADMM with heuristic implementations rather than to find the best classifier. Termination criteria were either primal/dual residual tolerance ($\epsilon_p = 10^{-1}$ and $\epsilon_d = 10^{-0}$) or maximum number of iterations, $k = 10$, whichever occurs the first. Dual residual was set to such a low value because empirical observations showed that restricting the dual residual does not significantly increase accuracy of the classification but effects run-time disproportionately. Maximum run-time was limited to 10 hours for mid-size problems, and unlimited for the large ones. Run-time is shown in seconds, unless noted otherwise.

The results show that RACQP produces classification models of competitive quality as models produced by specialized software implementations in a much shorter time. RACQP is in general faster than LIBSVM (up to 27$\times$) except for instances where ratio of number of observations n with respect to number of features r is very large. It is noticeable that while producing (almost) identical results as LIBSVM, the Matlab implementation is significantly slower.

For small and mid-size instances (training test size $<$ 100K) we tried, the difference in accuracy prediction is less than 2%, except for problems where test data sets are much larger than the training sets. In the case of "rcv1.binary" instance test data set is 5$\times$ larger than the training set, and for "cod_rna" instance is 4$\times$ larger. In both cases RACQP outperforms LIBSVM (and Matlab) in accuracy, by 20% and 9%, respectively.

All instances except for "news20.binary" have $n >> r$ and the choice of the Gaussian kernel is the correct one. For instances where the number of features is larger than the number of observations, linear kernel is usually the better choice as the separability of the model can be exploited [79] and problem solved to similar accuracy in a fraction of time required to solve it with the non-linear kernel. The

Table 22 Model training performance comparison for SVM

Instance name	Training set size	Testing set size	Num. features	Accuracy [%]			Training run-time [s]		
				RAC	LIBSVM	Matlab	RAC	LIBSVM	Matlab
a8a	22,696	9865	122	76.3	78.1	78.1	91	250	2653
w7a	24,692	25,057	300	97.1	97.3	97.3	83	133	2155
rcv1.binary	20,242	135,480	47,236	73.6	52.6	–	78	363	10+h
news20.binary*	19,996	5998	1,355,191	99.9	99.9	–	144	3251	NA
a9a	32,561	16,281	122	76.7	78.3	78.3	211	485	5502
w8a	49,749	14,951	300	97.2	99.5	99.5	307	817	20,372
ijcnn1	49,990	91,701	22	91.6	91.3	91.3	505	423	0
cod_rna	59,535	271,617	8	79.1	73.0	73.0	381	331	218
real_sim*	72,309	21,692	20,958	69.5	69.5	–	1046	9297	10+h
skin_nonskin*	245,057	73,517	3	99.9	99.9	–	2.6h	0.5h	NA
webspam_uni*	350,000	105,000	254	64.3	99.9	–	13.8h	11.8h	NA
covtype.binary*	581,012	174,304	54	91.3	99.9	–	16.2h	45.3h	NA

*No test set provided, using 30% of randomly chosen data from the training set. Reporting cross-validation accuracy results

reason we used the Gaussian kernel on "news20.binary' instance is that we wanted to show that RACQP is only mildly affected by the feature set size. Instances of similar sizes but different number of features are all solved by RACQP in approximately the same time, which is in contrast with LIBSVM and Matlab that are both affected by the feature space size. LIBSVM slows down significantly while Matlab, in addition to slowing down could not solve "news.binary"—the implementation of fitcsvm() function that invokes Matlab-SVM algorithm requires full matrices to be provided as the input which in the case of "news.binary" requires 141.3GB of main memory.

"Skin_nonskin" benchmark instance "marks" a point where our direct approach starts showing weaknesses—LIBSVM is 5× faster than RACQP because of the fine-tuned heuristics which exploit very small feature space (with respect to number of observations). The largest instance we addressed is "covtype.binary", with more than half of million observations and the (relatively) small feature size ($p = 54$). For this instance, RACQP continued slowing down proportionately to the increase in problem size, while LIBSVM experienced a large hit in run-time performance, requiring almost two days to solve the full size problem. This indicates that the algorithms employed by LIBSVM are put to the limit and specialized algorithms (and implementations) are needed to handle large-scale SVM problems. RACQP accuracy is lower than that of LIBSVM, but can be improved by tightining residual tolerances under the cost of increased run-time.

For large-size problems RACQP performance degraded, but the success with the mid-size problems suggests that a specialized "RAC-SVM" algorithm could be developed to address very large problems. Such a solution could merge RAC-ADMM algorithm with heuristic techniques to (temporarily) reduce the size of the problem (e.g. [42]), smart kernel approximation techniques, probabilistic approach(es) to shrinking the support vector set (e.g. [62]), and similar.

4.1.8 Changing random seed for RACQP

When it comes to algorithms that are stochastic in nature, as RAC-ADMM is, the question that always comes onto mind is about robustness of the algorithm. More precisely, how much is RAC-ADMM sensitive to variations in problem data for a given problem model, and to variations arising from differences in sub-problems due to from randomness of block building procedure (Algorithm 1, line 3). The answer to the former question has been provided in Sect. 4.1.3, and this section tackles the latter.

To answer the question on RACQP sensitivity to sub-problem structure we subject RACQP to different *random seeds*—each sub-problem is solving minimization problem defined by Lagrangian (Eq. 20), which is, in turn, a function of blocks of primal variables constructed using a stochastic process, following the procedure outlined in Sect. 3.1, Step 1. This stochastic process is guided by values drawn from a pseudo random number generator, which is initialized using a *random seed* number. For different seeds the generator produces different sequences of numbers, what in turn produces different sub-problems addressed by RACQP.

Table 23 shows results over a selected set of instances chosen to represent each problem type addressed so far. The table aggregates statistical data collected by solving each instances using ten different seeds per primal/dual tolerance ϵ. Note that CuteR

instances (Sect. 4.1.5) are not included in the analysis as all the instances are solved using a single-block approach. The results show that RACQP is a robust algorithm and that using a single run was a correct choice to make, at least when it comes to problem instances reported in this section. To generalize the claim about RACQP robustness with respect to randomness of block building scheme would require much more experiments and theoretical analysis, what we delegate to our future work.

4.2 Binary and mixed integer problems

The RAC-ADMM multi-block approach can be applied directly to binary (and mixed integer) problems without any adaptation. However, when dealing with combinatorial problems, a divide-and-conquer approach does not necessary lead to a good solution, because solver may get stuck in some local optima. To mitigate this problem RACQP, we introduce additional randomness into the implementation: a simple perturbation scheme shown in Algorithm 2 that helps the solver to "escape" the local optimum and to continue search for another one (and possibly find the global optimum). Thus, in addition to the run-time parameters used for continuous problems, for MIP we need to specify perturbation parameters such as probability distribution to use when choosing how many variables are perturbed (N_p) and the parameters thereof. As a default, RACQP implements truncated exponential distribution, $N_p \sim \text{Exp}(\lambda)$ with parameter $\lambda = 0.4n$, minimum number of variables $N_{p,\min} = 2$, and maximum number of variables $N_{p,\max} = n$, based on the observation that for most of the problems "good" solutions tend to be grouped. Variables are chosen at random, and in the general case, perturbation is done by assigning "new" values (within bounds) to the chosen variables. Default number of trials before perturbation, $N_{trial} = \min(2, 0.005n)$. For all binary problems presented in this section the primal residual error was zero, i.e. the problems were solved to feasibility.

As the default solver for sub-problems, RACQP uses Gurobi, but any other solver that admits mixed integer quadratic problem would suffice. The results reported in this section are based on Gurobi 7.5, and may be outdated. However, since we use Gurobi as the sub-solver, we expect RACQP to implicitly gain by the improvements made to Gurobi. Gurobi was ran using its default run-time settings (e.g. presolve option was turned on).

In [66] the authors present a mixed integer quadratic solver, MIQPs, which uses OSQP solver for solving sub-problems resulting from branch-and-bound strategy. Since the solver is built for small and medium size problems that occur in embedded applications, we do not include it in our current study. However, given that MIQPs showed a promising numerical performance (3× faster than Gurobi) even though being implemented in Python, it would be interesting to use it within RACQP as the external solver for MIP (Algorithm 1, line 8) instead of our default solver (Gurobi) and compare performance. We defer this comparison to future work.

To solve MIP problems RACQP uses the partial Lagrangian approach, described in Sect. 2.3.3, to handle bounds on variables $\mathcal{X}_i$, $\mathbf{x}_i \in \mathcal{X}_i$. Additionally, depending on a problem structure, equality and inequality constraints can also be moved to the local constraint set. Our experiments show that moving some (as it done for QAP), or

Table 23 RACQP performance—number of iterations over different random seeds

Instance	$\epsilon = 10^{-4}$				$\epsilon = 10^{-5}$				$\epsilon = 10^{-6}$			
	μ	σ	min	max	μ	σ	min	max	μ	σ	min	max
regular_monthly	575.7	0.8	574	577	1181.8	30.9	1094	1192	2591.1	1.1	2589	2593
regular_daily	184.6	1.6	183	188	579.0	3.9	576	587	1242.7	2.7	1240	1248
sko100a	17.5	0.5	17	18	21.3	0.5	21	22	25.2	0.6	24	26
tai125e01	18.6	0.5	18	19	22.5	0.5	22	23	26.6	0.5	26	27
tai150b	27.8	0.6	27	29	34.8	0.6	34	36	41.4	0.8	40	42
tho150	21.8	0.8	21	23	26.4	0.7	25	27	30.6	0.8	29	32
wil100	15.0	0.0	15	15	19.4	0.5	19	20	24.0	0.0	24	24
square15	145.0	5.0	140	157	195.3	13.1	178	207	241.3	9.4	215	247

Ten experiments per instance per ϵ

all constraints (e.g. graph cut problems) to a local set is beneficial in terms of block sizes, run-time, and overall solution quality. By using local constraints we help the sub-solver (e.g. Gurobi) reduce the size of the problem and tighten its formulation (using presolve and cutting plane algorithms).

Rather than solving the binary QP problem exactly, our goal is to find a (randomized or deterministic) algorithm that could find a better solution under a fixed solution time constraint. Our preliminary tests show that solving a large-scale problem using RAC-ADMM based approach can lead to a very good quality solution for an integer problem in a very limited time.

The quality of solutions is given in a form of a gap between the objective value of the optimal solution x^*_{opt} and the objective value of the solution found by a solver S, x^*_S:

$$gaps = \frac{f(x^*_S) - f(x^*_{\text{opt}})}{1 + \text{abs}(f(x^*_{\text{opt}}))} \tag{38}$$

For the instances for which the optimal solution remains unknown (e.g. QAPLIB and GSET instances), we use the best known results from the literature. Note that for maximization problems (e.g. Max-Cut, Max-Bisection) gap is the negative of (38). All binary problems are solved with primal residual equal to zero (i.e. the solutions are feasible and integer).

4.2.1 Randomness helps

We start the analysis of RACQP for binary problems with a short example showing that having blocks that are randomly constructed at each iteration, as done by RAC-ADMM, is the main feature that makes RACQP work well for combinatorial problems, without a need for any special adaptation of the algorithm for the discrete domain.

RAC-ADMM can be easily adapted to execute classical ADMM or RP-ADMM algorithms, so here we compare these three ADMM variants when applied to combinatorial problems. We use a small size problem ($n = 1000$) and construct a problem using (32) applied to problem of Markowitz type (39),

$$\begin{aligned} \min_{\mathbf{x}} \ & \mathbf{x}^T \mathbf{V} \mathbf{x} + \tau \mathbf{m}^T \mathbf{x} + \kappa \| \mathbf{x} \|_2^2 \\ \text{s.t.} \ & \mathbf{e}^T \mathbf{x} = r \\ & \mathbf{x} \in \{0, 1\}^n \end{aligned} \tag{39}$$

with $\kappa = 10^{-5}$ and a positive integer number $r \in \mathbb{Z}_+$, $r \in (1, n)$ that defines how many stocks from a given portfolio must be chosen.

A typical evaluation of the algorithms is shown in Fig. 3. Results show that RAC-ADMM is much better suited for binary optimization problems than either cyclic ADMM or RP-ADMM or distributed-ADMM, which is not surprising since more randomness is adapted into the algorithm making it more likely to escape local optima. All the algorithms are quick to find a local optimum, but besides RAC-ADMM stay at that first found point, while RAC-ADMM continues to find local optima, which could be better or worse than previously found. Because of this behavior, one can keep track

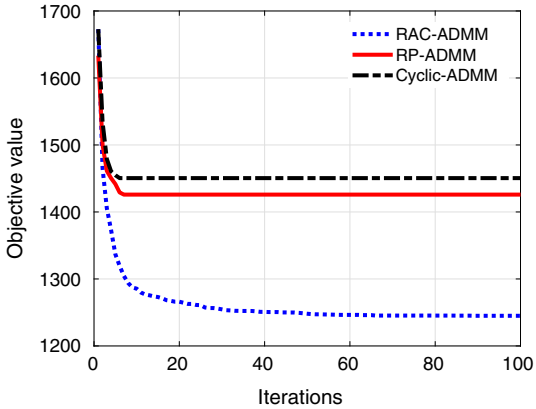


Fig. 3 A typical evaluation of the objective function value of (39) for different ADMM algorithms. $n = 1000$, $r = n/2, p = 50$, $\beta = 50$

the best solution found ($\mathbf{x}_{best}$, Algorithm 2). The algorithms seem robust with respect to the structure of the Hessian and choice of initial point. For completeness of the comparison, we have implemented distributed-ADMM (Eq. 4) for binary problems and ran the algorithm on the same data. The results are omitted from the Fig. 3 because the algorithm was unable to find a single feasible solution in 500 iterations.

4.2.2 Markowitz portfolio selection

Similarly to the section on continuous problems, we compare RACQP performance with that of Gurobi on Markowitz cardinality constrained portfolio selection problem (39) using real data coming from CRSP 2018 [78]. In the experiments, we set $r = n/2$ with all other settings identical to those used in Sect. 4.1.2, including $\mathbf{V}$ and $\mathbf{m}$, estimated from CRSP 2018 data. The default perturbation RACQP settings with $\beta = 0.05$, $p = 100$ were used in the experiments. Gap is measured from the "Optimal" objective values of the solutions found by Gurobi in about 1 hour run-time after relaxing MIPGAP parameter to 0.1.

From the results (Table 24) it is noticeable that RACQP finds relatively good solutions (gap $10^{-2} - 10^{-4}$) in a very short time, in some cases even before Gurobi had time to finalize root relaxation step of its binary optimization procedure. Maximal allowed run-time of 1 min was far too short for Gurobi to find any solution, so it returned a heuristic ones. Note that those solutions (third column of the table) are extremely weak, suggesting that a RAC-ADMM based solution could be implemented and used instead.

Low-rank Markowitz portfolio selection model

Similarly to (31) we formulate the model for low-rank covariance matrix V as

$$\begin{aligned} \min_{\mathbf{x}} \ & \| \mathbf{y} \|_2^2 - \tau \, \mathbf{m}^T \mathbf{x} + \kappa \| \mathbf{x} \|_2^2 \\ \text{s.t.} \ & \mathbf{e}^T \mathbf{x} = r \\ & \mathbf{B}\,\mathbf{x} - \mathbf{y} = \mathbf{0} \\ & \mathbf{x} \in \{0, 1\}^n \end{aligned} \tag{40}$$

Table 24 Markowitz portfolio selection model (39)

CRSP 2018 data	Problem size (n)	Optimal Obj. Val.	Gap					
			run time = 1 min		run time = 5 min		run time = 10 min	
			Gurobi*	RACQP	Gurobi	RACQP	Gurobi	RACQP
Quarterly	7958	0.055	43.4	$2.6 \cdot 10^{-3}$	0	$9.0 \cdot 10^{-4}$	0	$9.0 \cdot 10^{-4}$
Monthly	7958	0.144	30.9	$1.1 \cdot 10^{-2}$	$2.0 \cdot 10^{-3}$	$2.0 \cdot 10^{-3}$	$-8.7 \cdot 10^{-6}$	$2.0 \cdot 10^{-3}$
Daily	4628	1.164	4.0	$2.8 \cdot 10^{-4}$	0	$4.6 \cdot 10^{-5}$	0	0

CRSP 2018 data [78]
*Root relaxation step not finished. Gurobi returned a heuristic feasible solution

Table 25 Low-rank reformulation Markowitz portfolio selection model (39)

CRSP 2018 data	Problem size (n)	Optimal Obj. Val.	Gap	
			Gurobi	RACQP
Quarterly	7958	0.015	$-2.2 \cdot 10^{-7}$	$1.6 \cdot 10^{-3}$
Monthly	7958	0.104	$-4.2 \cdot 10^{-6}$	$-1.3 \cdot 10^{-5}$
Daily	4628	1.140	3.3	$-1.0 \cdot 10^{-2}$

CRSP 2018 data [78]. Max run-time 1 min

and solve the model for CRSP 2018 data. We use $\beta = 0.5$, $p = 50$. RACQP gap was measured from the optimal solution returned by Gurobi. In Table 25 we report on the best solutions found by RACQP with max run-time limited to 60 seconds. Results are hard to compare. When Hessian is diagonal and the number of constraints are small, as the case for this data, Gurobi has a very easy time solving the problems (monthly and daily data)—it finds good heuristic points to start with, and solves problems at a root node after a couple of hundreds of simplex iterations. On the other hand, RACQP, which does not directly benefit from diagonal Hessian, needs to execute multiple iterations of ADMM. Even though the problems are small and solved very quickly, the overhead of preparing the sub-problems and initializing Gurobi to solve sub-problems accumulates to the point of overwhelming RACQP run-time. In that light, for the rest of this section we consider problems where Hessian is a non-diagonal matrix, and address the problems that are hard to solve directly by Gurobi (and possibly other MIP QP solvers).

4.2.3 QAPLIB

The binary quadratic assignment problem (QAP) is known to be NP-hard and that binary instances of larger sizes (dimension of the permutation matrix $r > 40$) are considered to be intractable and cannot be solved exactly (though some instances of a large size with special structure have been solved). Currently, the only practical solutions for solving large QAP instances are heuristic methods.

For binary QAP we apply the same method for variance reduction as we did for relaxed QAP (Sect. 4.1.4). We group variables following the structure of constraints, which is dictated by the permutation matrix $X \in \{0, 1\}^{r \times r}$ (see Eq. 33 for QAP problem formulation)—we construct one super-variable, $\mathbf{x}_i$ for each row i of X. Next we make the use of the partial Lagrangian, and split constraints into the local constraint set consisting of (33) (a) and the global constraint set consisting of (33) (b), so that the partial Lagrangian is

$$L_\beta(\mathbf{x}, y) = \frac{1}{2}\mathbf{x}^T \mathbf{H}\mathbf{x} - \mathbf{y}^T(\mathbf{A}_{global}\mathbf{x} - \mathbf{1}) + \frac{\beta}{2}||\mathbf{A}_{global}\mathbf{x} - \mathbf{1}||^2.$$

At each iteration, we update the ith block by solving

$$\mathbf{x}_i^{k+1} = \arg\min\{L_\beta(\cdot)|\ \mathbf{A}_{local}\mathbf{x}_i = \mathbf{1},\ \mathbf{x}_i \in \{0, 1\}^n\}.$$

Table 26 Number of instances = 133

QAPLIB [11] benchmark results summary	Gurobi	RACQP
Num. instances opt/best found	3	18
Num. instances gap < 0.01 (excluding opt/best)	0	17
Num. instances gap < 0.1 (excluding opt/best and < 0.01)	3	70

Max run-time: 10 min

Next, continuing on the discussion on perturbation from the previous section, we turn the feature on and set parameters as follows: number of super-variables to perturb is drawn from truncated exponential distribution, $N_p \sim \text{Exp}(\lambda)$ with parameter $\lambda = 0.4r$, minimum number of variables $N_{p,\min} = 2$ and maximum number of variables $N_{p,\max} = r$. The number of trials before perturbation N_{trial} is set to its default value. Note that we do not perturb single variables ($x_{i,j}$), rather super-variables that we choose at random. If a super-variable $\mathbf{x}_i$ has value of '1' at one location, and '0' on all other entries, then we randomly swap location of '1' within the super variable (thus keeping the row-wise constraint on X for row i satisfied). If the super-variable is not feasible (number of '1'$\neq 1$), we flip values of a random number of variables that make $\mathbf{x}_i$. The initial point is a random feasible vector. The penalty parameter is a function of the problem size, $\beta = n$, while the number of blocks depends on the permutation matrix size and it is $p = \lceil r/2 \rceil$.

The summary of the QAPLIB benchmark [60] results is given in Table 26. Out of 133 total instances the benchmark includes, RACQP found the optimal solution (or the best known from literature as not all instances have proven optimal solution) for 18 instances within 10 min of run-time. For the rest of the instances, RACQP returned solutions with an average gap of $\mu = 0.07$. Gurobi solved only three instances to optimality. The average gap of the unsolved instances is $\mu = 12.15$, which includes heuristic solutions returned when root relaxation step was not finalized (20 instances). Removing those outliers results in the average gap of $\mu = 5.57$.

Table 27 gives detailed information on 21 large instances from QAPLIB data set. The most important takeaway from the table is that Gurobi can not even start solving very large problems as it can not finalize the root relaxation step within given maximum run time, while RACQP can.

4.2.4 Maximum cut problem

The maximum-cut (Max-Cut) problem consists of finding a partition of the nodes of a graph $G = (V, E)$, into two disjoint sets V_1 and V_2 ($V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$) in such a way that the total weight of the edges that have one endpoint in V_1 and the other in V_2 is maximized. The problem has numerous important practical applications, and is one of Karp's 21 NP-complete problems. A standard formulation of the problem is $\max_{\mathbf{y}_i \in \{-1,1\}} \frac{1}{4} \sum_{i,j} w_{i,j}(1 - y_i y_j)$, which can be re-formulated into

Table 27 QAPLIB, large problems [11,52] and RACQP/Gurobi objective values

Instance name	Problem size (n)	Density (H)	Best known Obj val	Gap		
				Gurobi*	RACQP	
				10 min	5 min	10 min
lipa80a	6400	0.96	253,195	0.15	0.01	0.01
lipa80b	6400	0.96	7,763,962	−0.96	0.23	0.22
lipa90a	8100	0.97	360,630	0.22	0.01	0.01
lipa90b	8100	0.97	12,490,441	−0.96	0.24	0.23
sko81	6561	0.69	90,998	1.11	0.03	0.03
sko90	8100	0.68	115,534	1.17	0.04	0.03
sko100a	10,000	0.68	152,002	1.34	0.04	0.03
sko100b	10,000	0.68	153,890	1.38	0.04	0.03
sko100c	10,000	0.67	147,862	1.21	0.04	0.05
sko100d	10,000	0.67	149,576	1.21	0.04	0.03
sko100e	10,000	0.67	149,150	1.17	0.04	0.04
sko100f	10,000	0.67	149,036	1.18	0.04	0.03
tai80a	6400	0.96	13,499,184	−0.98	0.05	0.05
tai80b	6400	0.43	818,415,043	−1.00	0.24	0.23
tai100a	10,000	0.96	21,043,560	−0.97	0.06	0.05
tai100b	10,000	0.43	1,185,996,137	−1.00	0.23	0.23
tai150b	22,500	0.44	498,896,643	−1.00	0.19	0.18
tho40	1600	0.38	240,516	−0.92	0.06	0.05
tho150	22,500	0.42	8,133,398	−0.89	0.09	0.06
wil50	2500	0.86	48,816	0.53	0.02	0.01
wil100	10,000	0.88	273,038	1.19	0.03	0.02

Gap between best known results
*Root relaxation step not finished. Gurobi returned heuristic feasible solutions

quadratic unconstrained binary problem

$$\begin{aligned} \min_{\mathbf{x}} \ & \mathbf{x}^T \mathbf{H} \mathbf{x} \\ \text{s.t.} \ & \mathbf{x} \in \{0, 1\}^n \end{aligned} \tag{41}$$

where $h_{i,j} = w_{i,j}$ and $h_{i,i} = -\frac{1}{2}(\sum_{j=1}^{n} w_{i,j} + \sum_{j=1}^{n} w_{j,i})$.

We use the Gset benchmark from [33], and compare the results of our experiments with the optimal solutions (found by Gurobi) and the best known solutions from the literature [4,48]. For perturbation we use default parameters and perform perturbation by choosing a random number of variables and negating their values, i.e. $x_i = 1 - x_i$. The number of blocks is equal for all instances, $p = 4$, and the initial point is set to zero ($\mathbf{x}_0 = \mathbf{0}$) for all the experiments. Note that as the max-cut problem is unconstrained,

the penalty parameter β is not used (and RACQP is doing a randomly assembled cyclic BCD).

In contrast to continuous sparse problems (rule 4, Sect. 4.1.1), sparse binary problems benefit from using a randomized multi-block approach, as shown in Table 28. The table compares RACQP and Gurobi results collected from experiments on Gset instances for three different maximum run-time limit settings, 10, 30 and 60 minutes. RACQP again outperforms Gurobi, overall, it finds better solutions when run-time is limited. Although Gurobi does better on a few problems, on average RACQP is better. Note that for large(r) problems ($n \geq 5000$) RACQP keeps improving, which can be explained by the difference in number of perturbations—for smaller problems, good points have already being visited and a chance to find a better one are small. Adaptively changing perturbation parameters could help, but this topic is out of scope of this work.

4.2.5 Maximum bisection problem

The maximum bisection problem is a variant of the Max-Cut problem that involves partitioning the vertex set V of a graph $G = (V, E)$ into two disjoint sets V_1 and V_2 of equal cardinality (i.e. $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$, $|V_1| = |V_2|$) such that the total weight of the edges whose endpoints belong to different subsets is maximized. The problem formulation follows (41) with the addition of a constraint $\mathbf{e}^T \mathbf{x} = \lfloor n/2 \rfloor$, where n is the graph size.

For Max-Bisection, at each iteration we update the ith block by solving

$$\mathbf{x}_{\omega_i}^{k+1} = \underset{\mathbf{x}_{\omega_i} \in \{0,1\}^{d_i}}{\arg\min} \{\mathbf{x}_{\omega_i}^T \mathbf{H}_{\omega_i} \mathbf{x}_{\omega_i} - y(\mathbf{e}^T \mathbf{x}_{\omega_i} - \mathbf{b}_{\omega_i}) + \frac{\beta}{2} \| \mathbf{e}^T \mathbf{x}_{\omega_i} - \mathbf{b}_{\omega_i} \|^2\}$$

where d_i is the size of block i, $\mathbf{x}_{\omega_i}$ is a sub-vector of $\mathbf{x}$ constructed of components of $\mathbf{x}$ with indices $\omega_i \in \Omega$, and $b_{\omega_i} = \lfloor n/2 \rfloor - \mathbf{e}^T \mathbf{x}_{-\omega_i}$ with $\mathbf{x}_{-\omega_i}$ being the sub-vector of $\mathbf{x}$ with indices not chosen by ω_i. Solving the sub-problems directly has shown to be very time consuming. However, noticing that Gurobi, while solving the problem as whole, makes a good use of cuts for this type of problems (matrix Q structure), we decided to reformulate the sub-problems as follows

$$\begin{aligned} \min_{\mathbf{x}} \ & \mathbf{x}_{\omega_i}^T \mathbf{H}_{\omega_i} \mathbf{x}_{\omega_i} - yr + \tfrac{\beta}{2} r^2 \\ \text{s.t.} \ & \mathbf{e}^T \mathbf{x}_{\omega_i} - r = \mathbf{b}_{\omega_i} \\ & \mathbf{x}_{\omega_i} \in \{0, 1\}^{d_i}, \ r \in \{0, 1\}. \end{aligned}$$

Note that r can be also defined as a bounded continuous or integer variable, but because the optimal value is zero and because Gurobi makes good use of binary cuts, we decided to define r as binary.

As in the previous section, we use Gset benchmark library and compare the results of our experiments with the best known solutions for max-bisection problems found in the literature [48]. The experimental setup is identical to that of Max-Cut experiments except for the use of the penalty parameter $\beta = 0.005$ and the initial point $\mathbf{x}_0$ which is

Table 28 Max-Cut, GSET instances

Instance name	Problem size	Density (**H**)	Best known Obj. val.	Gap							
				run time = 5 min		run time = 10 min		run time = 30 min		run time = 60 min	
				Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP
G1	800	$6 \cdot 10^{-2}$	11,624	− 0.006	− 0.003	− 0.005	− 0.003	− 0.005	− 0.002	− 0.005	− 0.002
G6	800	$6 \cdot 10^{-2}$	2178	− 0.015	− 0.011	− 0.014	0.000	− 0.012	− 0.008	− 0.012	− 0.007
G11	800	$6 \cdot 10^{-3}$	564	0.000	− 0.011	0.000	− 0.004	0.000	− 0.004	0.000	− 0.004
G14	800	$2 \cdot 10^{-2}$	3064	− 0.021	− 0.001	− 0.021	− 0.001	− 0.021	− 0.001	− 0.020	− 0.001
G18	800	$2 \cdot 10^{-2}$	992	− 0.081	− 0.011	− 0.081	− 0.011	− 0.081	− 0.011	− 0.081	− 0.011
G22	2000	$1 \cdot 10^{-2}$	13,359	− 0.062	− 0.009	− 0.062	− 0.008	− 0.052	− 0.008	− 0.052	− 0.007
G27	2000	$1 \cdot 10^{-2}$	3848	− 0.160	− 0.152	− 0.155	− 0.152	− 0.152	− 0.151	− 0.149	− 0.141
G32	2000	$2 \cdot 10^{-3}$	1410	0.000	− 0.016	0.000	− 0.014	0.000	− 0.014	0.000	− 0.013
G36	2000	$6 \cdot 10^{-3}$	7678	− 0.026	− 0.004	− 0.026	− 0.004	− 0.026	− 0.004	− 0.026	− 0.004
G39	2000	$6 \cdot 10^{-3}$	2408	− 0.102	− 0.016	− 0.102	− 0.010	− 0.102	− 0.007	− 0.102	− 0.006
G43	1000	$2 \cdot 10^{-2}$	6660	− 0.046	− 0.002	− 0.046	− 0.002	− 0.045	− 0.002	− 0.045	− 0.002
G50	3000	$2 \cdot 10^{-3}$	5880	0.000	− 0.001	0.000	− 0.001	0.000	− 0.001	0.000	− 0.001
G51	1000	$1 \cdot 10^{-2}$	3848	− 0.021	− 0.008	− 0.021	− 0.004	− 0.021	− 0.003	− 0.021	− 0.003
G55	5000	$1 \cdot 10^{-3}$	10,299	− 0.044	− 0.007	− 0.041	− 0.005	− 0.038	− 0.005	− 0.038	− 0.005
G56	5000	$1 \cdot 10^{-3}$	4016	− 0.112	− 0.017	− 0.112	− 0.016	− 0.112	− 0.011	− 0.112	− 0.011
G58	5000	$3 \cdot 10^{-3}$	19,276	− 0.078	− 0.008	− 0.040	− 0.007	− 0.039	− 0.005	− 0.039	− 0.004
G60	7000	$8 \cdot 10^{-4}$	14,187	− 0.120	− 0.008	− 0.098	− 0.007	− 0.096	− 0.006	− 0.091	− 0.004
G61	7000	$8 \cdot 10^{-4}$	5796	− 0.222	− 0.015	− 0.181	− 0.014	− 0.158	− 0.013	− 0.117	− 0.012
G63	7000	$2 \cdot 10^{-3}$	26,997	− 0.046	− 0.008	− 0.046	− 0.006	− 0.032	− 0.005	− 0.032	− 0.005
G67	10,000	$5 \cdot 10^{-4}$	6940	− 0.003	− 0.018	− 0.002	− 0.016	0.001	− 0.014	0.001	− 0.013

Table 28 continued

Instance name	Problem size	Density (**H**)	Best known Obj. val.	Gap							
				run time = 5 min		run time = 10 min		run time = 30 min		run time = 60 min	
				Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP
G70	10,000	$3 \cdot 10^{-4}$	9581	− 0.006	− 0.006	− 0.006	− 0.006	− 0.005	− 0.005	− 0.004	− 0.004
G77	14,000	$3 \cdot 10^{-4}$	9926	− 0.020	− 0.018	− 0.010	− 0.017	0.001	− 0.013	0.001	− 0.012
G81	20,000	$2 \cdot 10^{-4}$	14,030	− 0.119	− 0.024	− 0.031	− 0.017	− 0.023	− 0.015	0.002	− 0.014
	Average			− 0.0569	− 0.0162	− 0.0478	− 0.0142	− 0.0442	− 0.0134	− 0.0409	− 0.0124

Gap between best known results and RACQP/Gurobi objective values

a feasible random vector. Perturbation is done with a simple swap—an equal number of variables with values "1" and "0" is chosen and the new value set to be the negation of the old value.

The results are shown in Table 29. Compared to the unconstrained max-cut problem, RACQP seems to have less trouble solving max-bisection problem—adding a single constraint boosted its performance by up to $2\times$. Gurobi performance on the other worsened. Overall, RACQP outperforms Gurobi, finding better solutions when run-time is limited. Both Gurobi and RACQP continue gaining on solution quality (gap gets smaller) with longer time limits.

5 Summary

In this paper, we introduced a novel randomized algorithm, randomly assembled multi-block and cyclic alternating direction method of multipliers (RAC-ADMM), for solving continuous and binary convex quadratic problems. We provided a theoretical proof of the performance of our algorithm for solving linear-equality constrained continuous convex quadratic programming, including the expected convergence of the algorithm and sufficient condition for almost surely convergence of the algorithm. We further provided open source code of our solver, RACQP, and numerical results on demonstrating the efficiency of our algorithm.

We conducted multiple numerical tests on solving synthetic, real-world, and benchmark quadratic optimization problems, which include continuous and binary problems. We compare RACQP with Gurobi, Mosek and OSQP for cases that do not require high accuracy, but a strictly improved solution in shortest possible run-time. Computational results show that RACQP, except for a couple of instances with a special structure, finds solutions of a very good quality in a much shorter time than the compared solvers.

In addition to general linearly constrained quadratic problems we applied RACQP to few selected machine learning problems, Linear Regression, LASSO, Elastic-Net, and SVM. Our solver matches the performance of the best tailored methods such as Glmnet and LIBSVM, and often gives better results than that of tailored methods. In addition, our solver uses much less computation memory space than other ADMM based method do, so that it is suitable in real applications with big data.

The following is a quick summary of the pros and cons of RACQP, implementation of RAC-ADMM, for solving quadratic problems, and suggests the future research:

- RACQP is remarkably effective for solving continuous and binary convex QP problems when the Hessian is non-diagonal, the constraint matrix are unstructured, or the number of constraints are small. These findings are demonstrated by solving Markowitz portfolio problems with real or random data, and randomly generated sparse convex QP problems.
- RACQP, coupled with smart-grouping and a partial augmented Lagrangian, is equally effective when the structure of the constraints is known. This finding is supported by solving continuous and binary bench-mark Quadratic Assignment, Max-Cut, and Max-Bisection problems. However, efficiently deciding on group-

Table 29 Max-Bisection, GSET instances

Instance name	Problem size	Density (**H**)	Best known Obj. val.	Gap							
				run time = 5 min		run time = 10 min		run time = 30 min		run time = 60 min	
				Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP
G1	800	$6{\cdot}10^{-2}$	11,624	− 0.004	− 0.005	− 0.004	− 0.005	− 0.004	− 0.005	− 0.002	− 0.005
G6	800	$6{\cdot}10^{-2}$	2177	− 0.023	− 0.006	− 0.022	− 0.003	− 0.018	− 0.003	− 0.015	0
G11	800	$6{\cdot}10^{-3}$	564	0	− 0.014	0	− 0.011	0	− 0.011	0	− 0.007
G14	800	$2{\cdot}10^{-2}$	3062	− 0.018	− 0.008	− 0.017	− 0.008	− 0.017	− 0.002	− 0.016	− 0.002
G18	800	$2{\cdot}10^{-2}$	992	− 0.065	− 0.004	− 0.065	− 0.004	− 0.065	− 0.001	− 0.065	− 0.001
G22	2000	$1{\cdot}10^{-2}$	13,359	− 0.207	− 0.009	− 0.205	− 0.005	− 0.066	− 0.004	− 0.066	− 0.003
G27	2000	$1{\cdot}10^{-2}$	3341	− 0.050	− 0.023	− 0.050	− 0.021	− 0.043	− 0.016	− 0.042	− 0.014
G32	2000	$2{\cdot}10^{-3}$	1410	0	− 0.010	0	− 0.010	0	− 0.009	0	− 0.009
G36	2000	$6{\cdot}10^{-3}$	7678	− 0.027	− 0.004	− 0.022	− 0.004	− 0.022	− 0.004	− 0.022	− 0.004
G39	2000	$6{\cdot}10^{-3}$	2408	− 0.080	− 0.011	− 0.078	− 0.010	− 0.076	− 0.010	− 0.076	− 0.010
G43	1000	$2{\cdot}10^{-2}$	6659	− 0.077	− 0.004	− 0.072	− 0.004	− 0.064	− 0.001	− 0.063	− 0.001
G50	3000	$2{\cdot}10^{-3}$	5880	− 0.013	0	− 0.012	0	− 0.006	0	− 0.006	0
G51	1000	$1{\cdot}10^{-2}$	3847	− 0.017	− 0.005	− 0.015	− 0.005	− 0.015	− 0.004	− 0.014	− 0.004
G55	5000	$1{\cdot}10^{-3}$	10,299	− 0.128	− 0.008	− 0.038	− 0.007	− 0.037	− 0.006	− 0.036	− 0.006
G56	5000	$1{\cdot}10^{-3}$	4016	− 0.218	− 0.020	− 0.115	− 0.018	− 0.102	− 0.017	− 0.101	− 0.017
G58	5000	$3{\cdot}10^{-3}$	19,276	− 0.133	− 0.007	− 0.132	− 0.007	− 0.128	− 0.005	− 0.038	− 0.005
G60	7000	$8{\cdot}10^{-4}$	14,187	− 0.201	− 0.011	− 0.201	− 0.006	− 0.141	− 0.005	− 0.084	− 0.004
G61	7000	$8{\cdot}10^{-4}$	5796	− 0.274	− 0.020	− 0.274	− 0.019	− 0.218	− 0.019	− 0.181	− 0.018
G63	7000	$2{\cdot}10^{-3}$	26,988	− 0.263	− 0.007	− 0.160	− 0.006	− 0.160	− 0.005	− 0.037	− 0.003
G67	10,000	$5{\cdot}10^{-4}$	6938	− 0.272	− 0.016	− 0.168	− 0.014	− 0.004	− 0.011	− 0.001	− 0.010
G70	10,000	$3{\cdot}10^{-4}$	9581	− 0.009	− 0.008	− 0.007	− 0.007	− 0.006	− 0.005	− 0.003	− 0.004

Table 29 continued

Instance name	Problem size	Density (**H**)	Best known Obj. val.	Gap							
				run time = 5 min		run time = 10 min		run time = 30 min		run time = 60 min	
				Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP
G77	14,000	$3 \cdot 10^{-4}$	9918	− 0.468	− 0.015	− 0.468	− 0.013	− 0.247	− 0.012	− 0.095	− 0.010
G81	20,000	$2 \cdot 10^{-4}$	14,030	− 0.280	− 0.017	− 0.280	− 0.015	− 0.253	− 0.014	− 0.214	− 0.012
Average				− 0.1228	− 0.0101	− 0.1046	− 0.0088	− 0.0735	− 0.0073	− 0.0513	− 0.0065

Gap between best known results and RACQP/Gurobi objective values

ing strategy is also challenging. We plan to build an "automatic-smart-grouping" method as a pre-solver for unknown structured problem data.

- Computational studies done on binary problems show that RAC-ADMM approach to solving problems offers an advantage over the traditional direct approach (solving the problem as whole) when finding a good quality solution for a large-scale integer problem in a very limited time. However, exact binary QP solvers, such as Gurobi, are needed, because our binary RACQP relies on solving many small or medium sized binary sub-problems. Of course, we plan to explore more high efficiency solvers for medium-sized binary problems for RACQP.
- The ADMM-based approach, either RACQP or OSQP, is less competitive when the Hessian of the convex quadratic objective is diagonal and the constraints are sparse but structured such as a network-flow type. We believe in this case both Gurobi and Mosek can utilize more efficient Cholesky factorization that is commonly used by interior-point algorithms for solving linear programs; see more details in Sect. 3.1. In contrary, RACQP has considerable overhead cost of preparing block data and initialization time of the sub-problem solver, and the time spent on solving diagonal sub-problems was an order of magnitude shorter than time needed to prepare data. This, together with the divergence problem of multi-block ADMM, hints that there must be something connected to the problem structure that makes such instances hard for the ADMM-based approach. We plan on conducting additional research to identify problem instances that are well-suited and those that are unsuitable for ADMM.
- There are still many other open questions regarding RAC-ADMM. For example, there is little work on how to optimally choose run-time parameters to work with RAC-ADMM, including penalty parameter β, number of blocks, and so for.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Allman, A., Tang, W., Daoutidis, P.: Decode: a community-based algorithm for generating high-quality decompositions of optimization problems. Optim. Eng. **20**(4), 1067–1084 (2019)
2. Baingana, B., Traganitis, P., Giannakis, G., Mateos, G.: Big data analytics for social networks (2015)
3. Bastani, H., Bayati, M.: Online decision-making with high-dimensional covariates. SSRN 2661896 (2015)
4. Benlic, U., Hao, J.K.: Breakout local search for the max-cutproblem. Eng. Appl. Artif. Intell. **26**(3), 1162–1173 (2013)

5. Bertsekas, D.P.: Incremental aggregated proximal and augmented Lagrangian algorithms. CoRR arXiv:1509.09257 (2015)
6. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods, vol. 23. Prentice Hall Englewood Cliffs, NJ (1989)
7. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10), P10008 (2008)
8. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pp. 144–152. ACM (1992)
9. Bottou, L., Lin, C.J.: Support vector machine solvers. Large Scale Kernel Machines **3**(1), 301–320 (2007)
10. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011)
11. Burkard, R.E., Karisch, S.E., Rendl, F.: QAPLIB—a quadratic assignment problem library. J. Global Optim. **10**(4), 391–403 (1997). Revised 02.04.2003 (electronic update): http://www.seas.upenn.edu/qaplib/
12. Byrd, R.H., Hansen, S.L., Nocedal, J., Singer, Y.: A stochastic quasi-newton method for large-scale optimization. SIAM J. Optim. **26**(2), 1008–1031 (2016)
13. Cai, X., Han, D., Yuan, X.: The direct extension of ADMM for three-block separable convex minimization models is convergent when one function is strongly convex. Optim. Online **229**, 230 (2014)
14. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**, 27:1–27:27 (2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
15. Chen, C., He, B., Ye, Y., Yuan, X.: The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. Math. Program. **155**(1), 57–79 (2016). https://doi.org/10.1007/s10107-014-0826-5
16. Chen, C., Li, M., Liu, X., Ye, Y.: On the convergence of multi-block alternating direction method of multipliers and block coordinate descent method (2015). http://www.optimization-online.org/DB_HTML/2015/08/5046.html
17. Chen, C., Li, M., Liu, X., Ye, Y.: Extended ADMM and BCD for nonseparable convex minimization models with quadratic coupling terms: convergence analysis and insights. Math. Program. (2017). https://doi.org/10.1007/s10107-017-1205-9
18. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
19. Costa, O.L.V., Fragoso, M.D., Marques, R.P.: Discrete-Time Markov Jump Linear Systems. Springer, Berlin (2006)
20. Deng, W., Yin, W.: On the global and linear convergence of the generalized alternating direction method of multipliers. J. Sci. Comput. **66**(3), 889–916 (2016)
21. Drezner, Z., Hahn, P., Taillard, É.D.: Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. Ann. Oper. Res. **139**(1), 65–94 (2005)
22. Eckstein, J., Bertsekas, D.P.: On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. Math. Program. **55**(1–3), 293–318 (1992)
23. Eckstein, J., Yao, W.: Understanding the convergence of the alternating direction method of multipliers: theoretical and computational perspectives. Pac. J. Optim. **11**(4), 619–644 (2015)
24. Esser, E., Zhang, X., Chan, T.F.: A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. SIAM J. Imaging Sci. **3**(4), 1015–1046 (2010)
25. Evstigneev, I.V., Hens, T., Schenk-Hoppé, K.R.: Mean–variance portfolio analysis: the markowitz model. In: Mathematical Financial Economics, pp. 11–18. Springer, Berlin (2015)
26. Fan, R.E., Chen, P.H., Lin, C.J.: Working set selection using second order information for training support vector machines. J. Mach. Learn. Res. **6**, 1889–1918 (2005)
27. Ferris, M.C., Horn, J.D.: Partitioning mathematical programs for parallel solution. Math. Program. **80**(1), 35–61 (1998)
28. Forero, P.A., Cano, A., Giannakis, G.B.: Distributed clustering using wireless sensor networks. IEEE J. Sel. Top. Signal Process. **5**(4), 707–724 (2011)
29. Friedman, J., Hastie, T., Höfling, H., Tibshirani, R., et al.: Pathwise coordinate optimization. Ann. Appl. Stat. **1**(2), 302–332 (2007)
30. Friedman, J., Hastie, T., Tibshirani, R.: Regularization paths for generalized linear models via coordinate descent. J. Stat. Softw. **33**(1), 1 (2010)

31. Gabay, D., Mercier, B.: A dual algorithm for the solution of nonlinear variational problems via finite element approximation. Comput. Math. Appl. **2**(1), 17–40 (1976)
32. Glowinski, R.: On alternating direction methods of multipliers: a historical perspective. In: Ditzgibbon, W., Kuznetsov, Y., Neittaanmäki, P., Pironneau, O. (eds.) Modeling, Simulation and Optimization for Science and Technology, pp. 59–82. Springer, Berlin (2014)
33. Gset http://web.stanford.edu/yyye/yyye/Gset/
34. Gurobi Optimizer 8.1.1 http://gurobi.com/ (2018)
35. He, B., Tao, M., Yuan, X.: Alternating direction method with Gaussian back substitution for separable convex programming. SIAM J. Optim. **22**(2), 313–340 (2012)
36. Hestenes, M.R.: Multiplier and gradient methods. J. Optim. Theory Appl. **4**(5), 303–320 (1969)
37. Hong, M., Luo, Z.Q.: On the linear convergence of the alternating direction method of multipliers. Math. Program. **162**(1–2), 165–199 (2017)
38. Hong, M., Luo, Z.Q., Razaviyayn, M.: Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. SIAM J. Optim. **26**(1), 337–364 (2016)
39. Huang, K., Sidiropoulos, N.D.: Consensus-ADMM for general quadratically constrained quadratic programming. IEEE Trans. Signal Process. **64**(20), 5297–5310 (2016)
40. Jiang, B., Lin, T., Ma, S., Zhang, S.: Structured nonconvex and nonsmooth optimization: algorithms and iteration complexity analysis. arXiv e-prints arXiv:1605.02408 (2018)
41. Jiang, B., Ma, S., Zhang, S.: Tensor principal component analysis via convex optimization. Math. Program. **150**(2), 423–457 (2015)
42. Joachims, T.: Making large-scale SVM learning practical. Technical report, SFB 475: Komplexitätsreduktion in Multivariaten ... (1998)
43. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1998)
44. Lai, R., Osher, S.: A splitting method for orthogonality constrained problems. J. Sci. Comput. **58**(2), 431–449 (2014)
45. Lin, T., Ma, S., Ye, Y., Zhang, S.: An ADMM-based interior-point method for large-scale linear programming. arXiv e-prints arXiv:1805.12344 (2017)
46. Lin, T., Ma, S., Zhang, S.: On the global linear convergence of the ADMM with multiblock variables. SIAM J. Optim. **25**(3), 1478–1497 (2015)
47. Lin, T., Ma, S., Zhang, S.: Iteration complexity analysis of multi-block ADMM for a family of convex minimization without strong convexity. J. Sci. Comput. **69**(1), 52–81 (2016)
48. Ma, F., Hao, J.K., Wang, Y.: An effective iterated tabu search for the maximum bisection problem. Comput. Oper. Res. **81**, 78–89 (2017)
49. Maros, I., Mészáros, C.: A repository of convex quadratic programming problems. Optim. Methods Softw. **11**(1–4), 671–681 (1999)
50. Matlab R2018b https://www.mathworks.com/ (2018)
51. Mihic, K., Ryan, K., Wood, A.: Randomized decomposition solver with the quadratic assignment problem as a case study. INFORMS J. Comput. **30**(2), 295–308 (2018)
52. Misevičius, A.: New best known solution for the most difficult qap instance "tai100a". Memet. Comput. **11**(3), 331–332 (2019)
53. Mohan, K., London, P., Fazel, M., Witten, D., Lee, S.I.: Node-based learning of multiple Gaussian graphical models. J. Mach. Learn. Res. **15**(1), 445–488 (2014)
54. Monteiro, R.D., Svaiter, B.F.: Iteration-complexity of block-decomposition algorithms and the alternating direction method of multipliers. SIAM J. Optim. **23**(1), 475–507 (2013)
55. MOSEK version 8.1.0.49 https://www.mosek.com/ (2018)
56. Newman, M.E.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. **103**(23), 8577–8582 (2006)
57. Ohlsson, H., Yang, A., Dong, R., Sastry, S.: Cprl—an extension of compressive sensing to the phase retrieval problem. In: Advances in Neural Information Processing Systems, pp. 1367–1375 (2012)
58. Peng, Y., Ganesh, A., Wright, J., Xu, W., Ma, Y.: Rasl: robust alignment by sparse and low-rank decomposition for linearly correlated images. IEEE Trans. Pattern Anal. Mach. Intell. **34**(11), 2233–2246 (2012)
59. Powell, M.J.D.: Algorithms for nonlinear constraints that use Lagrangian functions. Math. Program. **14**, 224–248 (1978)
60. QAPLIB http://anjos.mgi.polymtl.ca/qaplib/
61. RACQP https://github.com/kmihic/RACQP. https://doi.org/10.5281/zenodo.4008720

62. Rudi, A., Carratino, L., Rosasco, L.: Falkon: an optimal large scale kernel method. In: Advances in Neural Information Processing Systems, pp. 3888–3898 (2017)
63. SCIP Optimization Suite https://www.scipopt.org/
64. Simon, N., Friedman, J., Hastie, T., Tibshirani, R.: Regularization paths for Cox's proportional hazards model via coordinate descent. J Stat Softw **39**(5), 1–13 (2011)
65. Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S.: OSQP: an operator splitting solver for quadratic programs. In: 2018 UKACC 12th International Conference on Control (CONTROL), pp. 339–339. IEEE (2018)
66. Stellato, B., Naik, V.V., Bemporad, A., Goulart, P., Boyd, S.: Embedded mixed-integer quadratic optimization using the OSQP solver. In: 2018 European Control Conference (ECC), pp. 1536–1541. IEEE (2018)
67. Sun, D.L., Fevotte, C.: Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6201–6205. IEEE (2014)
68. Sun, R., Luo, Z.Q., Ye, Y.: On the expected convergence of randomly permuted ADMM. Optimization for Machine Learning, OPT2015 (2015)
69. Sun, R., Luo, Z.Q., Ye, Y.: On the efficiency of random permutation for ADMM and coordinate descent. Math. Oper. Res. **45**(1), 233–271 (2020)
70. Tao, M., Yuan, X.: Recovering low-rank and sparse components of matrices from incomplete and noisy observations. SIAM J. Optim. **21**(1), 57–81 (2011)
71. Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., Goldstein, T.: Training neural networks without gradients: a scalable ADMM approach. In: International Conference on Machine Learning, pp. 2722–2731 (2016)
72. The Maros and Meszaros Convex QP Test Problem Set. http://www.cuter.rl.ac.uk/Problems/marmes.html
73. The Mittelmann LP test set. http://plato.asu.edu/ftp/lptestset/
74. Vapnik, V., Vapnik, V.: Statistical Learning Theory, pp. 156–160. Wiley, New York (1998)
75. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, Berlin (1995)
76. Wang, F., Cao, W., Xu, Z.: Convergence of multi-block Bregman ADMM for nonconvex composite problems. arXiv preprint arXiv:1505.03063 (2015)
77. Wang, Y., Yin, W., Zeng, J.: Global convergence of ADMM in nonconvex nonsmooth optimization. arXiv e-prints arXiv:1511.06324 (2017)
78. Wharton Research Data Services https://wrds-web.wharton.upenn.edu/wrds/index.cfm (2018)
79. Woodsend, K., Gondzio, J.: Exploiting separability in large-scale linear support vector machine training. Comput. Optim. Appl. **49**(2), 241–269 (2011)
80. Wright, S.J.: Coordinate descent algorithms. Math. Program. **151**(1), 3–34 (2015)
81. Zarepisheh, M., Xing, L., Ye, Y.: A computation study on an integrated alternating direction method of multipliers for large scale optimization. Optim. Lett. **12**(1), 3–15 (2018)
82. Zhang, J., Ma, S., Zhang, S.: Primal-dual optimization algorithms over Riemannian manifolds: an iteration complexity analysis. arXiv e-prints arXiv:1710.02236 (2017)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Krešimir Mihić[1,2] · Mingxi Zhu[3] · Yinyu Ye[4]

Mingxi Zhu
mingxiz@stanford.edu

Yinyu Ye
yyye@stanford.edu

1 School of Mathematics, The University of Edinburgh, Edinburgh, UK

[2] Oracle Labs, Redwood Shores, California, USA

[3] Graduate School of Business, Stanford University, Stanford, California, USA

[4] Department of Management Science and Engineering, School of Engineering, Stanford University, Stanford, California, USA