Noname manuscript No.
(will be inserted by the editor)

A conjugate direction based simplicial decomposition framework for solving a specific class of dense convex quadratic programs

Enrico Bettiol · Lucas Létocart · Francesco Rinaldi · Emiliano Traversi

Received: date / Accepted: date

Abstract Many real-world applications can usually be modeled as convex quadratic problems. In the present paper, we want to tackle a specific class of quadratic programs having a dense Hessian matrix and significantly more variables than constraints. We hence carefully analyze a simplicial decomposition like algorithmic framework that handles those problems in an effective way. We introduce a new master solver, called Adaptive Conjugate Direction Method, and embed it in our framework. We also analyze the interaction of some techniques for speeding up the solution of the pricing problem. We report extensive numerical experiments based on a benchmark of almost 1400 instances from specific and generic quadratic problems. We show the efficiency and robustness of the method when compared to a commercial solver (`Cplex`).

Keywords Simplicial Decomposition · Convex Quadratic Programming · Dense Hessian Matrix · Column Generation

Mathematics Subject Classification (2010) 65K05 · 90C20 · 90C25

E. Bettiol
LIPN, CNRS, (UMR7030), Université Paris 13, Sorbonne Paris Cité,
99 av. J. B. Clément, 93430 Villetaneuse, France
E-mail: bettiol@lipn.univ-paris13.fr

L. Létocart
LIPN, CNRS, (UMR7030), Université Paris 13, Sorbonne Paris Cité,
99 av. J. B. Clément, 93430 Villetaneuse, France
E-mail: lucas.letocart@lipn.univ-paris13.fr

F. Rinaldi
Dipartimento di Matematica, Università di Padova
Via Trieste, 63, 35121 Padua, Italy
Tel.: +39-049-8271424
E-mail: rinaldi@math.unipd.it

E. Traversi
LIPN, CNRS, (UMR7030), Université Paris 13, Sorbonne Paris Cité,
99 av. J. B. Clément, 93430 Villetaneuse, France
E-mail: emiliano.traversi@lipn.univ-paris13.fr

1 Introduction

We consider the following problem:

$$\begin{aligned} &\min\ f(x) = x^\top Q x + c^\top x \\ &\text{s. t. } Ax \geq b, \\ &\quad Cx = d, \\ &\quad l \leq\ x \leq u, \end{aligned} \tag{1}$$

with $Q \in \mathbb{R}^{n\times n}$, $c, l, u \in \mathbb{R}^{\mathrm{n}}$, $A \in \mathbb{R}^{m_1\times n}$, $b \in \mathbb{R}^{\mathrm{m_1}}$, $C \in \mathbb{R}^{m_2\times n}$, $d \in \mathbb{R}^{\mathrm{m_2}}$, $n, m_1, m_2 \in \mathbb{N}$.

Moreover, we assume that the polyhedral set

$$X = \{x \in \mathbb{R}^n :\ Ax\ \geq\ b,\ \ Cx\ =\ d,\ l \leq\ x \leq u\}$$

is non-empty and bounded and that the Hessian matrix Q is positive semidefinite. Among all possible problems of type (1), we are particularly interested in the ones with the following additional properties:

- The number of equality/inequality constraints is considerably smaller than the number of variables in the problem, i.e. $m = m_1 + m_2 \ll n$ (notice that bounds are not considered in here);
- the Hessian matrix Q is dense.

A significant number of real-world problems arising in Computational Geometry, Communications, Statistics, Economics, Control and Machine Learning present a structure similar to the one described above (see, e.g., [5, 9, 52]).

Solution methods for this class of problems can be mainly categorized into either interior point methods or active set methods [51]. In interior point methods, a sequence of parameterized barrier functions is (approximately) minimized using Newton's method. The main computational burden is represented by the calculation of the Newton system solution (used to get the search direction). Even if those methods are relatively recent (they started becoming populare in the 1990s), a large number of papers and books exist related to them (see, e.g., [27, 48, 62–64]).

In active set methods, at each iteration, a working set that estimates the set of active constraints at the solution is iteratively updated. This gives a subset of constraints to watch while searching the solution (which obviously reduces the complexity of our search in the end). Those methods, which have been widely used since the 1970s, turn out to be effective when dealing with small- and medium-sized problems. They usually guarantee efficient detection of unboundedness and infeasibility (other than returning an accurate estimate of the optimal active set). An advantage of active set methods over interior points is that they are well-suited for warmstarts, where a good estimate of the optimal active set or solution is used to initialize the algorithm. This turns out to be extremely useful in applications where a sequence of QP problems is solved, e.g., in a sequential quadratic programming method. A quite large number of active set methods have been developed in recent years (see, e.g., [11–13, 20, 38]). A detailed overview of active set methods can be found in [51].

In this paper, we develop a *simplicial decomposition type* approach (see, e.g., [53, 60]) specifically tailored to tackle problems with the aforementioned features. However, it is worth noting that the algorithm proposed can handle any problem of type (1) and can also be easily modified in order to deal with problems having a general convex objective function. We want to use simplicial decomposition for two main reasons. First of all, simplicial decomposition like methods are well suited to deal with (large-scale) structured problems and, secondly, they can be used in applications where sequences of QPs need to be solved (since they can take advantage of warmstarts). Those tools can thus be fruitfully used in, e.g., Branch and Price like schemes for convex quadratic discrete problems.

Simplicial decomposition is related to cutting plane/column generation approaches (see, e.g., [3,53] for further details). Those methods are well suited to solve large scale structured convex programs, since they can efficiently exploit the structure in the problem. Many different cutting plane approaches have been proposed in the literature, including: *center of gravity method* [43, 49], *maximum volume ellipsoid cutting plane method* [58], *Chebyshev center cutting plane method* [19] and *analytic center cutting plane method* [25, 26]. In the last two decades, some in depth computational studies related to the last class of methods, which seems to give a good trade-off between simplicity and practical performance, have been conducted (see, e.g., [24, 32, 33]). In some recent papers, a good number of structured problems has been analyzed from the column-generation (primal) perspective, and a Primal-Dual Column Generation Method (PDCGM) has been developed (see, e.g., [28–30]). PDCGM has also been embedded into a Branch-Price-and-Cut algorithmic framework for solving problems with integer variables (see, e.g., [47]).

Despite the vast literature in the context of cutting plane/column generation approaches, no in-depth computational and methodological analysis has been conducted so far to investigate the behaviour of simplicial decomposition on a statistically significant set of instances. In this work, we hence show how a well designed simplicial decomposition framework can efficiently handle convex quadratic instances with a dense Hessian matrix and with significantly more variables than constraints. To obtain good performances, two important features are considered:

- a new ad-hoc method for solving the master problem, called Adaptive Conjugate Direction Method;
- some pricing strategies that help speeding up the solution process.

In particular, the new master solver represents the first attempt to embed and wisely reuse conjugate directions into a simplicial decomposition framework. We will describe in depth the idea behind the algorithm and show how it works in practice. We then analyze the connections between some pricing strategies we introduce to improve the efficiency of our approach and classic features/ideas in cutting plane approaches. We also show how those strategies can be embedded into the algorithmic framework without affecting its convergence in the end.

The rest of the paper is organized as follows. In Section 2, we describe the classic simplicial decomposition framework. In Section 3, we present some strategies to improve the efficiency of the framework itself. In Section 4, we report our numerical experience. Finally, in Section 5, we draw some conclusions.

2 Simplicial Decomposition

Simplicial Decomposition (SD) represents a class of methods used for dealing with convex problems. It was first introduced by Holloway in [40] and then further studied in other papers like, e.g., [39, 59, 60]. A complete overview of this kind of methods can be found in [53].

The method basically uses an iterative *inner approximation* of the feasible set X. The method can be viewed as a special case of *column generation* applied to a non linear problem (we refer the reader to [15] for an extensive analysis of such a method). In practice, the feasible set X is approximated with the convex hull of an ever expanding finite set $X_k = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k\}$ where $\tilde{x}_i$, $i = 1, \ldots,\ k$ are extreme points of X. We denote this set with $conv(X_k)$:

$$conv(X_k) = \{x \mid x = \sum_{i=1}^{k} \lambda_i \tilde{x}_i, \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \geq 0\} \tag{2}$$

At each iteration, it is possible to add new extreme points to X_k in such a way that a function reduction is guaranteed when minimizing the objective function over the convex hull of the new (enlarged) set of extreme points. If the algorithm does not find at least one new point, the solution is optimal and the algorithm terminates.

The use of the proposed method is particularly indicated when the following two conditions are satisfied:

1. Minimizing a linear function over X is much simpler than solving the original nonlinear problem;
2. Minimizing the original objective function over the convex hull of a relatively small set of extreme points is much simpler than solving the original nonlinear problem (i.e. tailored algorithms can be used for tackling the specific problem in our case).

The first condition is needed due to the way a new extreme point is generated. Indeed, this new point is the solution of the following linear programming problem

$$\begin{array}{l} \min \nabla f(x_k)^\top (x - x_k) \\ \text{s.t. } x \in X \end{array} \tag{3}$$

where a linear approximation calculated at the last iterate x_k (i.e. the solution obtained by minimizing f over $conv(X_k)$) is minimized over the original feasible set X.

Below, we report the detailed scheme related to the classical simplicial decomposition algorithm [2,53,60] (see Algorithm 1). At a generic iteration k of the simplicial decomposition algorithm, given the set of extreme points X_k, we first minimize f over the set $conv(X_k)$ (Step 1), thus obtaining the new iterate x_k then, at Step 2, we generate an extreme point $\tilde{x}_k$ by solving the linear program (5). Finally, at Step 3, we update X_k.

Algorithm 1 Simplicial Decomposition Algorithm

Initialization: Choose a starting set of extreme points X_0.

For $k = 1, 2, \ldots$

Step 1) Generate iterate x_k by solving the **master problem**

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in conv(X_k) \end{array} \tag{4}$$

Step 2) Generate an extreme point $\tilde{x}_k$ by solving the **subproblem**

$$\begin{array}{ll} \min & \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \end{array} \tag{5}$$

Step 3) If $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, **Stop**. Otherwise Set $X_{k+1} = X_k \cup \{\tilde{x}_k\}$

End For

Finite convergence of the method is stated in the following Proposition (see, e.g., [2,60]):

Proposition 1 *Simplicial Decomposition algorithm obtains a solution of Problem* (1) *in a finite number of iterations.*

As explained also in [60], a *vertex dropping rule* is also used to get rid of those vertices in X_k whose weight is zero in the expression of the solution x_k (Step 1). This dropping phase does not change the theoretical properties of the algorithm (finiteness still remains), but it can guarantee significant savings in terms of CPU time since it keeps the dimensions of the master problem small.

3 Strategies to improve the efficiency of a simplicial decomposition framework

In this section, we describe a few strategies that, once embedded in the simplicial decomposition framework, can give a significant improvement of the performances, especially when dealing with quadratic problems with a polyhedral feasible set described by a number of equations much smaller than the number of variables.

We first present and discuss two tailored strategies that efficiently solve the master problem by exploiting the special structure of the generated simplices. Then, we analyze some techniques for speeding up the solution of the pricing problem.

3.1 Strategies for efficiently solving the master problem

Here, we describe two different ways for solving the master problem. The first one is an ad-hoc method that tries to exploit the properties of the simplices and the information gathered from previous iterations (i.e., to reuse conjugate directions). To the best of our knowledge, this is the first time that such an algorithm is introduced and analyzed. The second one is a method of the *projected gradient* type that could eventually allow us to efficiently handle the more general problem of minimizing a convex function over a simplex.

3.1.1 Restriction of the domain

First of all, we notice that a generic master problem has the following form:

$$\begin{array}{ll} \min & f(x) \;=\; x^\top Q x + c^\top x \\ \text{s.t.} & x = \sum_{i=1}^{k} \lambda_i \tilde{x}_i \\ & \sum_{i=1}^{k} \lambda_i = 1 \\ & \lambda_i \geq 0 \quad \forall i = 1, \ldots, k. \end{array} \tag{6}$$

By using the matrix $B := [\tilde{x}_1 \ldots \tilde{x}_k]$ it is possible to rewrite the master in the following reduced form:

$$\begin{array}{ll} \min & \lambda^\top \tilde{Q} \lambda + \tilde{c}^\top \lambda \\ \text{s.t.} & \sum_{i=1}^{k} \lambda_i = 1 \\ & \lambda_i \geq 0 \quad \forall i = 1, \ldots, k, \end{array} \tag{7}$$

where $\tilde{Q} := B^\top Q B$ and $\tilde{c} := B^\top c$. It is easy to notice that the above problem uses only $k \ll n$ variables. Thanks to the way matrix B is defined, we just need to add one column for each new extreme point. This matrix update is much cheaper than the one described in [60], where the matrix had to be completely recalculated at each iteration.

3.1.2 An Adaptive Conjugate Direction Method for solving the master

In order to ease the description of the algorithm, in this first part of the section we assume that the objective function in (6) is strictly convex. Before giving the details of the Adaptive Conjugate Direction Method (ACDM), we preliminarily report a result related to the conjugate direction method (see, e.g., [50]) that helps us to better understand the rationale behind our algorithm:

Proposition 2 *Conjugate direction method converges to the minimum point of a strictly convex quadratic function $f(x) : \mathbb{R}^n \to \mathbb{R}$ in at most n steps.*

The result reported in Proposition 2 relies on the fact that the objective function is successively minimized along the individual directions in a conjugate set. This fact is heavily exploited when solving the master problem with ACDM. First of all, we assume that

- at each iteration the solution we get when solving the master is in the interior of the simplex approximating the feasible set (this is pretty reasonable since we can just remove from master problem those components that have zero weight in the solution);
- a suitable set of conjugate directions is available in the affine hull described by those vertices.

Hence, if the master solution at iteration k is in the interior of the simplex, ACDM easily calculates it by picking the master solution obtained at the previous iteration $k-1$ and applying a minimization step along a new suitably chosen conjugate direction (keep in mind that pricing phase only adds one dimension at each iteration). In case the master solution is on the boundary of the simplex, once the method applies the minimization along the new conjugate direction, it hits the boundary, reduces the variable space and needs to get a new set of conjugate directions. Luckily, this case does not happen so often in practice, thus making ACDM a viable option. In the rest of this section, we formally describe all the steps we need to implement the method.

Let $\Delta_k := conv(X_k)$ be the domain of the current master, Δ_{k-1} be the domain of the previous master, x_{k-1} the optimum of the previous master and $\tilde{x}_k \notin \Delta_{k-1}$ the new extreme point generated with the pricing. At any SD iteration, the master problem we want to solve (Step 1 of Algorithm 1) has the form in (7) and the following property holds:

Proposition 3 *The master solution at iteration $k-1$ lies in the relative interior of a facet of the set Δ_k.*

As we already noticed, if the master solution we get at iteration $k-1$ is on the boundary of the simplex, we can restrict the master problem to a smaller dimensional face (which is always a simplex) by means of the *column dropping rule*. Furthermore, the simplex of the current master is obtained by adding up the point provided by the pricing to the set of vertices describing that reduced face.

We now consider the descent direction $\bar{d}_{k-1} := \tilde{x}_{k-1} - x_{k-1}$. Furthermore, we assume that a set of conjugate directions $D = \{d_1, \ldots, d_{k-2}\}$ is available from previous iterations. We then use a Gram-Schmidt like procedure to turn direction $\bar{d}_{k-1}$ into a new direction d_{k-1} conjugate with respect to the set D. More specifically, we compute:

$$d_{k-1} = \bar{d}_{k-1} - \sum_{h=1}^{k-2} \delta_{k-1}^h d_h, \quad \text{where} \quad \delta_{k-1}^h = \frac{\bar{d}_{k-1}^\top Q d_h}{d_h^\top Q d_h}. \tag{8}$$

It is worth noticing that d_{k-1} is a descent direction too (since x_{k-1} is optimal, $\nabla f(x_{k-1})^\top d_h = 0$, $\forall\ h = 1, \ldots, k-2$). We use the basis $B = [\tilde{x}_1, \ldots, \tilde{x}_k]$ to express points $x^s = x_{k-1}$ and $x^t = x_{k-1} + d_{k-1}$ thus obtaining respectively points λ^s and λ^t. We intersect the halfline emanating from x^s (and passing by

x^t) with the boundary of Δ_k by solving the following problem:

$$\begin{array}{l} \max\ \alpha \\ \text{s.t.}\ \ (1-\alpha)\lambda^s + \alpha\lambda^t \geq 0. \end{array} \tag{9}$$

The solution of problem (9) can be directly written as

$$\alpha^* = \left(\max_i \frac{\lambda_i^s - \lambda_i^t}{\lambda_i^s} \right)^{-1}.$$

We finally define point $\lambda^p = (1-\alpha^*)\lambda^s + \alpha^*\lambda^t$ and solve the following problem

$$\min_{\beta \in [0,1]} f(B[(1-\beta)\lambda^s + \beta\lambda^p]).$$

If the optimal value $\beta^* < 1$ we get, by Proposition 2, an optimal solution for the master. Otherwise, $\beta^* = 1$ and we are on the boundary of the simplex. In this case, we just drop those vertices whose associated coordinates are equal to zero, and get a new smaller basis B. If B is a singleton, we can stop our procedure, otherwise we minimize $f(x)$ in the new subspace defined by B. In order to get a new set of conjugate directions in the considered subspace, we use directions connecting point $x^* = B\lambda^* = B\lambda^p$ with each vertex $\tilde{x}_j$ in B (that is $\bar{d}_j = \tilde{x}_j - x^p$) and then use the same Gram-Schmidt like procedure to make them conjugate (we want to remark that all directions $\bar{d}_j$ need to be expressed in terms of the new basis B). We report the algorithmic scheme below (see Algorithm 2).

Algorithm 2 Adaptive Conjugate Direction Method (ACDM)

Data: Basis B, conjugate directions D, and point x_{k-1}

Step 1) Set $x^s = x_{k-1}$ and $D^s = \{\bar{d}_{k-1}\}$

Step 2) Select a $\bar{d} \in D^s$ and set $D^s = D^s \setminus \{\bar{d}\}$

Step 3) Use a Gram-Schmidt like procedure to turn $\bar{d}$ into a conjugate direction d^s with respect to D

Step 4) Express points x^s and $x^t = x^s + d^s$ in terms of B (that is $x^s = B\lambda^s$ and $x^t = B\lambda^t$)

Step 5) Set

$$\alpha^* = \left(\max_i \frac{\lambda_i^s - \lambda_i^t}{\lambda_i^s} \right)^{-1}$$

Step 6) Calculate point $\lambda^p = (1-\alpha^*)\lambda^s + \alpha^*\lambda^t$ and find solution β^* of the problem

$$\min_{\beta \in [0,1]} f(B[(1-\beta)\lambda^s + \beta\lambda^p])$$

Step 7) If $\beta^* < 1$ then set $x^* = B[(1-\beta^*)\lambda^s + \beta^*\lambda^p]$ and $D = D \cup \{d^s\}$ go to Step 9
Else drop vertices with $\lambda^* = 0$ from B

Step 8) If B is a singleton then STOP
Else set $D = \emptyset$ and for each $\tilde{x}_j \in B$ set $\bar{d}_j = \tilde{x}_j - x^*$ (direction represented using coordinates in B) to get a set of directions D^s and go to Step 2

Step 9) If $D^s = \emptyset$ then STOP
Else go to Step 2

Finite convergence of an SD scheme that embeds Algorithm 2 for solving the master can be obtained by using same arguments as in [60]. The proof is based on the fact that our polyhedral feasible set contains a finite number of simplices (whose vertices are extreme points of the feasible set). Since the interior of each simplex has at most one relative minimum and the objective function strictly decreases between two consecutive points x_k and x_{k+1} (keep in mind that $\nabla f(x_k)^\top(\tilde{x}^k - x^k) < 0$), no simplex can recur. Now, observing that at each iteration we get a new simplex, we have that the number of iterations must be finite.

We notice that the result given in Proposition 2 also holds when dealing with quadratic convex functions (see, e.g., [54]). Hence it is easy to see that our method still works if the matrix Q is positive semidefinite. Indeed, let d_k be the first conjugate direction which is in the null space of Q; then $d_1, \ldots, d_{k-1}$ generate a space where Q is positive definite. We have that the function is linear along the line described by that descent direction, hence we have $\beta^* = 1$ and the search for the optimum is restricted to a face of the simplex that intersects this line. Since the line is not parallel to the face, Q is positive definite on the affine hull described by the vertices of that face and the algorithm can proceed.

3.1.3 A fast gradient projection method for solving the master

The second approach is a Fast Gradient Projection Method (FGPM) and belongs to the family of gradient projection approaches (see, e.g., [4] for an overview of gradient projection approaches). The detailed scheme is reported below (See Algorithm 3). At each iteration of the method, the new point we generate is

$$\lambda_{k+1} = \lambda_k + \beta_k(p[\lambda_k - s_k \nabla f(\lambda_k)]_{\Delta} - \lambda_k),$$

where $\beta_k \in (0, \rho_k]$, $\rho_k, s_k > 0$ and $p[\lambda_k - s_k \nabla f(\lambda_k)]_{\Delta}$ is the projection over Δ_k of the point $\lambda_k - s_k \nabla f(\lambda_k)$, chosen along the antigradient. When $p[\lambda_k - s_k \nabla f(\lambda_k)]_{\Delta} \neq \lambda_k$, it is easy to see that the direction we get is a feasible descent direction.
The method can be used in two different ways:

a) we fix s_k to a constant value and use a line search technique to get β_k;
b) we fix β_k and make a search changing s_k (thus getting a curvilinear path in the feasible set).

In our algorithm we consider case *a*) where $s_k = s > 0$.

At each iteration, projecting the point $y_k = \lambda_k - s\nabla f(\lambda_k)$ over the simplex corresponds to solve the following problem:

$$\min_{x \in \Delta} \|x - y\|_2.$$

A fast projection over the simplex is used to generate the search direction [10]. This particular way of projecting a point over the simplex is basically a Gauss-Seidel-like variant of Michelot's variable fixing algorithm [46]; that is,

Algorithm 3 Fast Gradient Projection Method (FGPM)

Data: Set point $\lambda_0 \in \mathbb{R}^{k-1}$, $\rho_0 \in [\rho_{min}, \rho_{max}]$ and a scalar value $s > 0$.

For $k = 0, 1, \ldots$

Step 1) Generate point

$$\hat{\lambda}_k = p[\lambda_k - s\nabla f(\lambda_k)]_\Delta$$

Step 2) If $\hat{\lambda}_k = \lambda_k$ STOP; otherwise set $d_k = \hat{\lambda}_k - \lambda_k$

Step 3) Choose a stepsize $\beta_k \in (0, \rho_k]$ along d_k and maximum stepsize ρ_{k+1} by means of a line search

Step 4) Set $\lambda_{k+1} = \lambda_k + \beta_k d_k$

End For

the threshold used to fix the variables is updated after each element is read, instead of waiting for a full reading pass over the list of non-fixed elements (See [10] for further details).

A nonmonotone line search [35–37] combined with a spectral steplength choice is then used at Step 3 (see [4] for further details) to speed up convergence. In Algorithm 4 we report the detailed scheme of the line search. Convergence of the FPGM algorithm to a minimum follows from the theoretical results in [4]. Therefore, the convergence of an SD method that uses FPGM to solve the master problem directly follows from the results in the previous sections.

Algorithm 4 Non-monotone Armijo line-search (with spectral steplength choice)

0 Set $\delta \in (0,1)$, $\gamma_1 \in (0, \frac{1}{2})$, $M > 0$

1 Update

$$\bar{f}_k = \max_{0 \le i \le \min\{M,k\}} f(\lambda_{k-i})$$

2 Set starting stepsize $\alpha = \rho_k$ and set $j = 0$

3 **While** $f(\lambda_k + \alpha d_k) > \bar{f}_k + \gamma_1 \, \alpha \, \nabla f(\lambda_k)^\top d_k$

4 set $j = j + 1$ and $\alpha = \delta^j \alpha$.

5 **End While**

6 Set $y_k = \nabla f(\lambda_k + \alpha d_k) - \nabla f(\lambda_k)$ and $b_k = \alpha d_k^\top y_k$

7 If $b_k \le 0$ set $\rho_{k+1} = \rho_{max}$ else set $a_k = \alpha^2 \|d_k\|^2$ and

$$\rho_{k+1} = min\{\rho_{max}, \max\{\rho_{min}, a_k/b_k\}\}$$

In the FGPM Algorithm too, we exploit the particular structure of the feasible set in the master, thus getting a very fast algorithm in the end. We will see later on that the FGPM based SD framework is even competitive with the ACDM based one, when dealing with some specific quadratic instances.

3.2 Strategies for efficiently solving the pricing problem

Now we describe two different strategies for speeding up the solution of the pricing problem (also called subproblem). The first one is an *early stopping* strategy that allows us to approximately solve the subproblem while guaranteeing finite convergence. The second one is the use of suitably generated inequalities (the so called *shrinking cuts*) that both cut away a part of the feasible set and enable us to improve the quality of extreme points picked in the pricing phase.

3.2.1 Early stopping strategy for the pricing

When we want to solve problem (1) using simplicial decomposition, efficiently handling the subproblem is, in some cases, crucial. Indeed, the total number of extreme points needed to build up the final solution can be small for some real-world problem, hence the total time spent to solve the master problems is negligible when compared to the total time needed to solve subproblems. This is the reason why we may want to approximately solve subproblem (5) in such a way that finite convergence is guaranteed (a similar idea was also suggested in [2]). In order to do that, we simply need to generate an extreme point $\tilde{x}_k$ satisfying the following condition:

$$\nabla f(x_k)^\top (\tilde{x}_k - x_k) \leq -\varepsilon < 0, \tag{10}$$

with $\varepsilon > 0$. Roughly speaking, we want to be sure that, at each iteration k, $d_k = \tilde{x}_k - x_k$ is a descent direction. Below, we report the detailed scheme related to the simplicial decomposition algorithm with early stopping (see Algorithm 5).

Algorithm 5 Simplicial Decomposition with Early Stopping Strategy for the Subproblem

Initialization: Choose a starting set of extreme points X_0

For $k = 0, 1, \ldots$

Step 1) Generate iterate x_k by solving the **master problem**

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in conv(X_k) \end{array}$$

Step 2) Generate an extreme point $\tilde{x}_k \in X$ such that

$$\nabla f(x_k)^\top (\tilde{x}_k - x_k) \leq -\varepsilon < 0.$$

In case this is not possible, pick $\tilde{x}_k$ as the optimal solution of (5)

Step 3) If $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, **Stop**. Otherwise set $X_{k+1} = X_k \cup \{\tilde{x}_k\}$

End For

At a generic iteration k we generate an extreme point $\tilde{x}_k$ by approximately solving the linear program (5). This is done in practice by stopping the algorithm used to solve problem (5) as soon as a solution satisfying constraint (10) is found. In case no solution satisfies the constraint, we simply pick the optimal solution of (5) as the new vertex to be included in the simplex at the next iteration. We would like to notice that the choice of the parameter ε is crucial, since it gives a tradeoff between solving the pricing problem quickly and obtaining a good descent direction. The details related to the parameter setting are given in the Appendix A.2.3.

Finite convergence of the method can be proved in this case as well:

Proposition 4 *Simplicial decomposition with early stopping strategy for the subproblem obtains a solution of Problem* (1) *in a finite number of iterations.*

The proof of Proposition 4 is provided in Appendix A.1.

We would like to highlight that the early stopping strategy is somehow related to the use of ε-subgradients in nonsmooth optimization (see [42] and references therein for further details). When using decomposition schemes like, e.g., cutting plane schemes, it is indeed possible to weaken the optimality requirements in subproblems and get ε-subgradients, obtaining usually an improvement in terms of the overall computing time of the algorithm (see, e.g., [31, 34]). Anyway, if a shallow cut (i.e., a cut that does not exclude the current query point) is generated, the convergence of a cutting plane approach might fail. Hence, a check is needed in order to ensure that the cut is deep enough (i.e., shallow cut is discarded, ε is suitably reduced and a new hopefully better cut is generated). In practice, as the cutting plane algorithm approaches the solution, the accuracy level with which the subproblem is solved should increase. In our simplicial decomposition framework, generating a good column (by approximately solving the pricing) while guaranteeing convergence is in general an easier task. Indeed, by taking a look at the proof of Proposition 4, we can notice that, in order to guarantee convergence, we only need the new column $\tilde{x}_k$ to be an extreme point of X (actually might be enough getting a point $\tilde{x}_k$ from a finite subset $\tilde{X} \subset X$ s.t. $conv(\tilde{X}) = X$) and to satisfy $\nabla f(x_k)^\top (\tilde{x}_k - x_k) < 0$. Thus, we can generate new columns for the master problem in a simple and natural way, without the need to check if those columns are nearly optimal (we only need to guarantee that the objective function can improve with respect to $f(x_k)$ in the new extended master).

3.2.2 Shrinking cuts

It is worth noticing that, at each iteration k, the objective function values of the subsequent iterates $x_{k+1}, x_{k+2}, \ldots$, generated by the method will be not greater than the objective function value obtained in x_k, hence the following condition will be satisfied:

$$\nabla f(x_k)^\top (x - x_k) \leq 0. \tag{11}$$

This can be easily seen by taking into account convexity of f. Indeed, choosing two points $x, y \in \mathbb{R}^n$, we have:

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x).$$

Thus, if $\nabla f(x)^\top (y - x) > 0$, we get $f(y) > f(x)$. Hence, $f(y) \leq f(x)$ implies $\nabla f(x)^\top (y - x) \leq 0$.

We remark that all those vertices $\tilde{x}_i \in X_k$ not satisfying condition (11) have the related coefficient $\lambda_i = 0$ in the convex combination (2) giving the master solution x_k at iteration k. Indeed, if we assume that a vertex $\tilde{x}_i$ is such that $\nabla f(x_k)^\top (\tilde{x} - x_k) > 0$ and the related $\lambda_i \neq 0$, then we can build a feasible descent direction in x_k thus contradicting its optimality.

We can take advantage of this property, as also briefly discussed in [2], by adding the cuts described above. The basic idea is the following: let x_k be the optimal point generated by the master at a generic iteration k, we can hence add the following *shrinking cut* c_k to the next pricing problems:

$$(c_k) \qquad \nabla f(x_k)^\top (x - x_k) \leq 0.$$

More precisely, let $\{x_1, \ldots, x_k\}$ be the set of optimal points generated by the master problems up to iteration k; then, for $k > 0$, we identify as C_k the polyhedron defined by all the associated shrinking cuts as follows:

$$C_k = \{x \in \mathbb{R}^n : \ \nabla f(x_i)^\top (x - x_i) \leq 0, \ i = 0, \ldots, k-1\}.$$

(We are assuming $x_0 := \tilde{x}_0$). Therefore, at Step 2, we generate an extreme point $\tilde{x}_k$ by minimizing the linear function $\nabla f(x_k)^\top (x - x_k)$ over the polyhedral set $X \cap C_k$. Finally, at Step 3, if $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, the algorithm stops, otherwise we update X_k by adding the point $\tilde{x}_k$ and C_k by adding the cut $\nabla f(x_k)^\top (x - x_k) \leq 0$. After a considerably large number of iterations $\bar{k}$, no more shrinking cuts are added to the pricing. This is done to ensure the convergence of the Algorithm.

Below, we report the detailed scheme related to the simplicial decomposition algorithm with shrinking cuts (see Algorithm 6).

In practice, we implemented the algorithm with the following variant: at the end of Step 2, after the solution of the pricing problem, we remove all shrinking cuts that are not active. In this way we are sure to have a pricing problem that is computationally tractable by keeping its size under control.

Finite convergence of the method is stated in the following Proposition:

Proposition 5 *Simplicial decomposition algorithm with shrinking cuts obtains a solution of Problem* (1) *in a finite number of iterations.*

The proof of Proposition 5 is provided in Appendix A.1.

Shrinking cuts are connected to cutting plane approaches as well. In fact, those constraints are somehow related to classic objective cuts (see, e.g., [42]). The main difference in this case is that the cuts are not added directly to the original problem in order to cut away a part of the feasible set, but instead

Algorithm 6 Simplicial Decomposition with Shrinking Cuts

Initialization: Choose a starting set of extreme points X_0

For $k = 0, 1, \ldots, \bar{k}$

Step 1) Generate iterate x_k by solving the **master problem**

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in conv(X_k) \end{array}$$

Step 2) Generate an extreme point $\tilde{x}_k$ by solving the **subproblem**

$$\begin{array}{ll} \min & \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \cap C_k \end{array} \tag{12}$$

Step 3) If $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, **Stop**. Otherwise set $X_{k+1} = X_k \cup \{\tilde{x}_k\}$ and set $C_{k+1} = \{x \in \mathbb{R}^n : \ \nabla f(x_i)^\top (x - x_i) \leq 0, \ i = 0, \ldots, k\}$

End For

they are used in the pricing problem in order to shrink the original feasible set and hopefully generate better columns. An additional interesting feature of shrinking cuts is that, when solving the pricing problem, we always have a feasible point to warmstart the solver we use. It is indeed easy to notice that the master solution x_k both satisfies the original constraints and the shrinking cuts in C_k. This comes from the fact that the objective function value $f(x_k)$ is lower or equal than the objective functions of the master solutions generated at previous iterations. On the other hand, guaranteeing feasibility while including objective cuts might be an issue in some cutting plane strategies.

As a final remark, we would like to notice that combining the shrinking cuts with the early stopping strategy is possible (this is a part of what we actually do in practice) and finite convergence still holds for the simplicial decomposition framework.

4 Computational results

In this section, we give a detailed description of the computational results obtained with the SD based algorithmic framework we analyzed in the previous sections. In order to test our software, we need instances with the suitable features (i.e., a dense Hessian matrix Q and a number of constraints considerably smaller than the number of variables). To the best of our knowledge, it is not possible to find in the literature a significant number of instances satisfying such requirements. Indeed, instances in standard QP libraries (like, e.g., Maros and Mészáros library [45] or QPLIB [21]) do not have both a large number of variables and a sufficiently dense Hessian matrix. For this reason, we introduce a benchmark that includes both generic and real quadratic programming problems. We use instances with a moderately large number of variables (up to 10000). This choice aims at making the problems hard enough, without

anyway requiring specific techniques for storing the Hessian matrix related to the objective function.

Due to the specific features of the given problems, we use `Cplex` version 12.6.3 (see [41] for further details) as the baseline software in our tests. This tool includes several solvers for quadratic programming: primal simplex, dual simplex, network simplex, barrier, sifting and concurrent. They perform differently and some of them can efficiently handle the specific classes of problems we consider in the paper.

We point out that cutting plane based algorithms might not be the best choice in this case, since no structure can be exploited when solving the subproblem.

As we will see later on, one among sifting, network and the default optimizer from `Cplex` is used as solver for linear subproblems in our experiments. Furthermore, we decided to use a tolerance of $10E-6$ for FGPM. We refer to Appendix A.2 for more details about the choice of these preliminary settings.

4.1 Instance description

In our tests we use three different sets of instances as test-bed: generic quadratic instances, portfolio instances and continuous relaxations of combinatorial instances. In the following subsections we describe their mathematical formulation, while we refer the reader to Appendix A.3 for a detailed description on how such instances are created.

4.1.1 Generic quadratic problems

The first set of instances is of the form:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\ & Ax \geq b, \\ & l \leq x \leq u. \end{aligned} \tag{13}$$

with $Q \in \mathbb{R}^{n\times n}$, $c \in \mathbb{R}^{\mathrm{n}}$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{\mathrm{m}}$, $l, u \in \mathbb{R}^{\mathrm{n}}$ and both finite.

The benchmark of Generic instances is split into two sets: the first one consists of 450 instances *with Small number of constraints* (GS) and the second one of 750 instances *with Large number of constraints* (GL).

4.1.2 Portfolio optimization problems

We consider the formulation for portfolio optimization problems (POP) proposed by Markowitz in [44]. These instances have a quadratic objective function (the risk, i.e. the portfolio return variance) and only two constraints: one giving a lower bound μ on the expected return and one representing the so call "budget" constraint. The problem we want to solve is then described as follows

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \ & f(x) = x^\top \Sigma x \\ \text{s.t.} \ & r^\top x \geq \mu, \\ & e^\top x = 1, \\ & x \geq 0, \end{aligned} \tag{14}$$

where $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix, r is the vector of the expected returns, and e is the n-dimensional vector of all ones. The complete portfolio benchmark consists of 40 instances.

4.1.3 Continuous relaxations of combinatorial instances

Continuous problems can obviously be viewed as a way to obtain valid dual bounds for combinatorial problems (to be used in a Branch-and-Bound framework). For this reason, it is interesting to analyze how SD performs when solving models related to the continuous relaxation of some combinatorial problems with a quadratic objective function.

In our tests, we used continuous relaxations of quadratic multidimensional knapsack problems (see, e.g., [16,23,57]) and quadratic shortest path problems (see, e.g., [6,55,56]).

Quadratic multidimensional knapsack problem (QMKP). The instances belonging to this class of problems take the following form:

$$\begin{aligned} \max \ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.} \ & Ax \leq b, \\ & 0 \leq x \leq 1. \end{aligned} \tag{15}$$

where $Q \in \mathbb{R}^{n \times n}$ is negative definite, $c \in \mathbb{R}^{\mathrm{n}}$, $A \in {\mathbb{R}_+}^{m \times n}$ and $b \in \mathbb{R}_+^{\mathrm{m}}$.

The set of quadratic multidimensional knapsack problems consists of 54 instances.

Quadratic shortest path problems (QSPP). The instances belonging to this class of problems take the following form:

$$\begin{aligned} \min \ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.} \ & \sum_{e \in \delta^+(s)} x_s = 1, \\ & \sum_{e \in \delta^+(v)} x_v - \sum_{e \in \delta^-(v)} x_v = 0, \quad \forall v \neq s, t \\ & \sum_{e \in \delta^-(t)} x_t = 1 \\ & 0 \leq x \leq 1. \end{aligned} \tag{16}$$

with $c \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$. s, t are the source and termination nodes, respectively; $\delta^+(v)$ are the outgoing arcs and $\delta^-(v)$ are the incoming arcs in the node v.

The benchmark consists of 30 instances based on grid graphs (QGSPP) and 72 instances based on random graphs (QRSPP).

4.2 Numerical results related to the complete testbed

In this section, we report the numerical results of our SD framework.

In the first part of the analysis, we investigate how the use of different options for solving the master problem influences the overall performances of the algorithm. We show the results concerning the following three different settings for the master problem:

- ACDM: the new conjugate direction method, presented in Section 3.1.2.
- FGPM: the gradient projection method explained in Section 3.1.3.
- `Cplex`: the continuous optimizer of `Cplex`, default settings.

In the second part, we test the impact of the following pricing options:

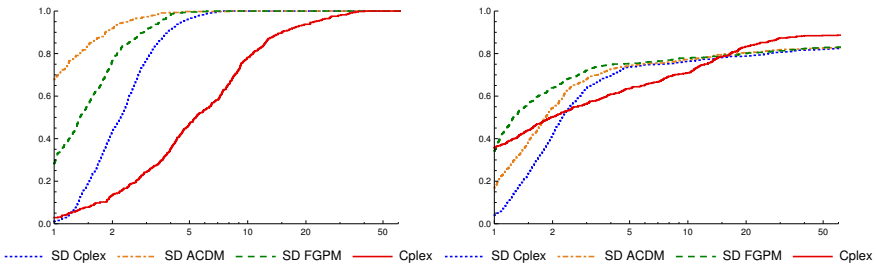
- Default: the pricing problem is solved with the Linear Programming optimizers of `Cplex`, default settings.
- Cuts: we add to the Default option the Shrinking cuts, described in Section 3.2.2.
- Early stopping: we add to the Default option the Early stopping technique described in Section 3.2.1 .
- Cuts + Early stopping: both techniques are added to the Default option.

We further compare the default option of `Cplex` with more specific options like the Sifting optimizer and the Network optimizer. In our analysis, we produced the performance profiles according to [17] and using the software *Mathematica* version 11.3 (see [61] for further details).

4.2.1 Master solvers

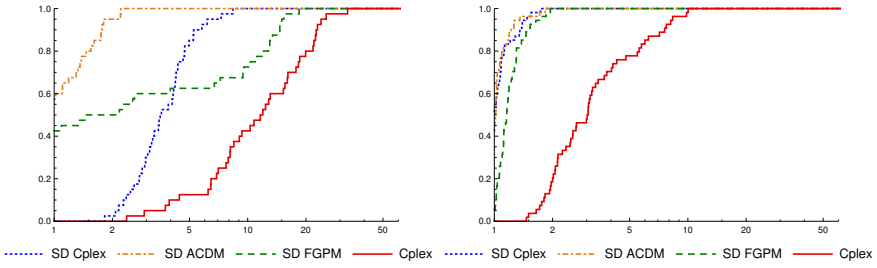
Now, we focus on the computational analysis of the different methods used for solving the master problem in the SD framework. Figure 1 provides the results concerning all the classes of problems previously introduced. We indicate with SD-Cplex, SD-ACDM and SD-FGPM the results concerning SD using respectively `Cplex`, ACDM and FGPM for solving the master problem. For the sake of comparison, we also include the performances of `Cplex`.

These plots show that the SD framework significantly outperforms `Cplex` in the vast majority of the cases. We can further see that SD-ACDM is the most efficient and robust for almost all the classes of problems considered (more precisely, GS, POP, QMKP, QGSPP and QRSPP). As regards the GL instances, we notice that SD-FGPM has better performances than both SD-ACDM and SD-Cplex and that the Cplex solver is competitive with it.



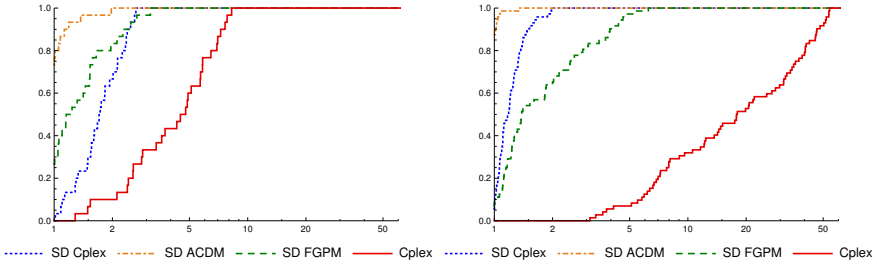
(a) GS

(b) GL



(c) POP

(d) QMKP



(e) QGSPP

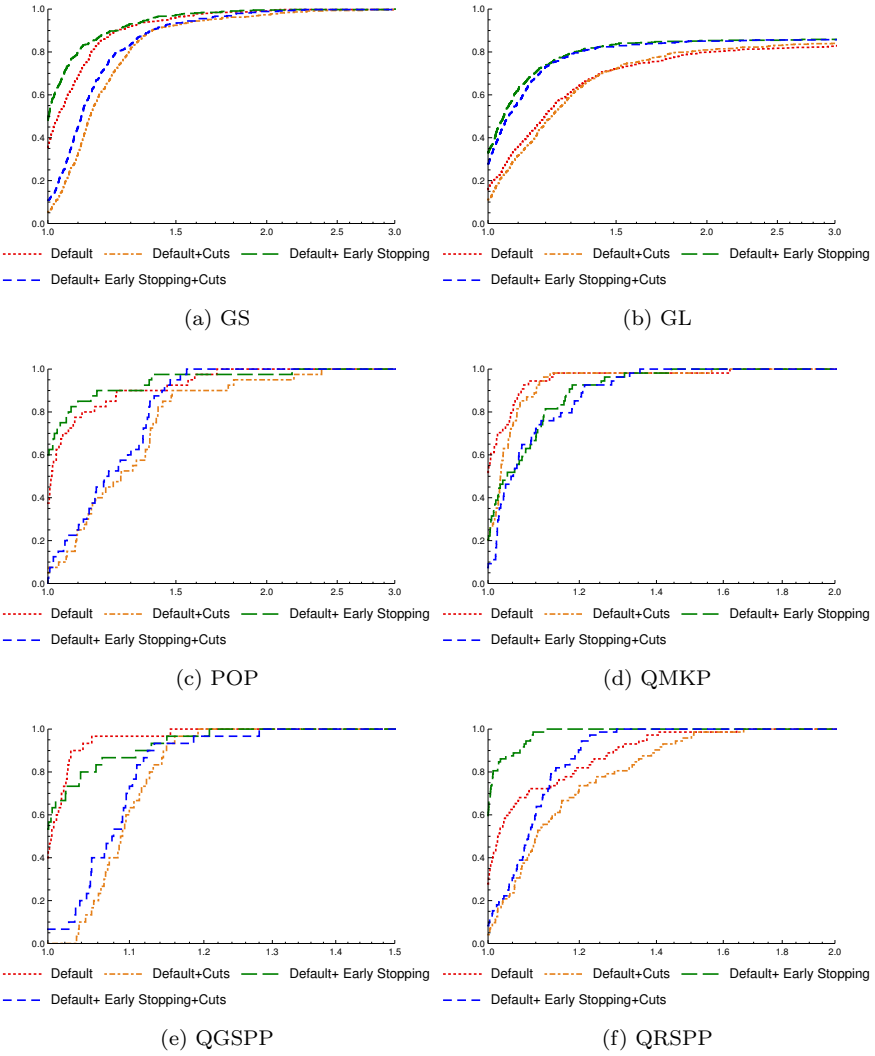
(f) QRSPP

Fig. 1: Performance analysis of the different methods used for solving the master problem in the SD framework.

4.2.2 Pricing Options

Here, we analyze the impact of the different pricing options in the SD framework. For each class of problems, we use as master solver the most effective method, according to the results of the previous section. Hence, ACDM is used for the GS, POP, QMKP, QGSPP and QRSPP instances and FGPM is used for the GL instances. Figure 2 shows the results concerning all the classes of problems considered. We indicate with Default the results concerning SD using `Cplex` with default settings to solve the pricing problem. Furthermore, we use Default+Cuts, Default+Early Stopping and Default + Early Stopping

+ Cuts to indicate the results obtained when respectively adding to Default the shrinking cuts, the early stopping procedure and both at the same time.



(a) GS (b) GL

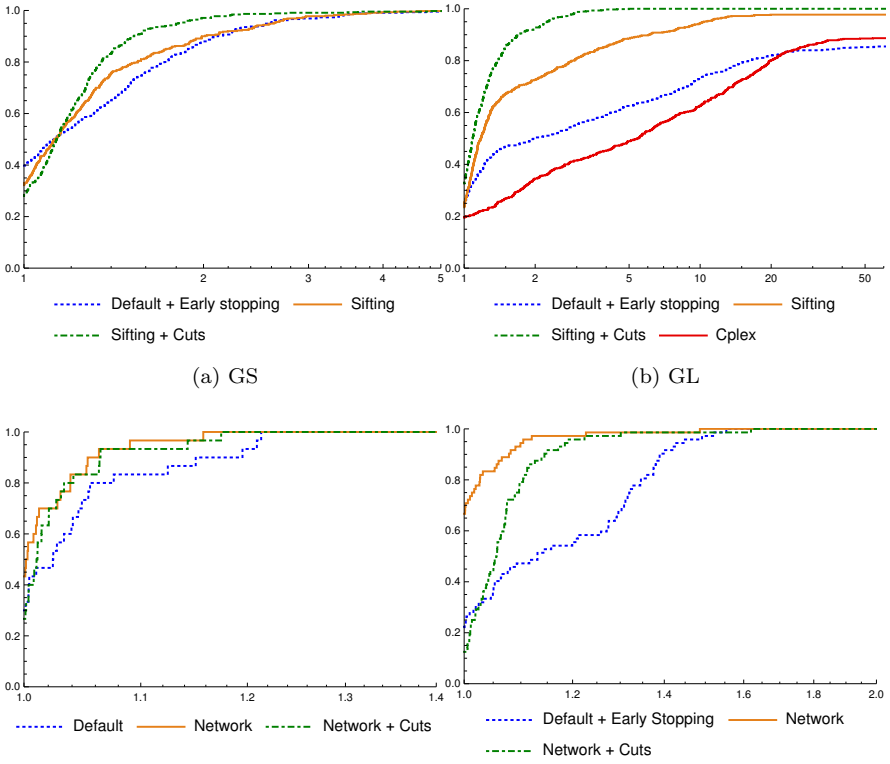
(c) POP (d) QMKP

(e) QGSPP (f) QRSPP

Fig. 2: Performance analysis of the different options used for solving the pricing problem in the SD framework.

We notice that the option Default + Early Stopping shows the best performances (both in terms of efficiency and robustness) for the GL, GP, POP and QRSPP instances. With respect to QGSPP, Default + Early Stopping is still the most efficient, but the Default version is slightly better in terms

of robustness. We further notice that the option Default + Early Stopping + Cuts is competitive with the option Default + Early Stopping on the GL instances, and it is as robust as the option Default + Early Stopping on the POP instances. Finally, if we consider the QMKP instances, the option Default is the most efficient, while the option Default + Early Stopping + Cuts is the most robust.



(a) GS (b) GL

(c) QGSPP (d) QRSPP

Fig. 3: Additional performance analysis of the different options used for solving the pricing problem in the SD framework.

Figure 3 finally shows the effects of replacing the Default `Cplex` solver for the pricing problem with the Sifting/Network optimizer. We only focus on two specific classes of problems where it makes sense to use such taylored approaches. More specifically, we considered generic quadratic instances to test the Sifting and quadratic shortest path problems to test the Network solver. We compare the best pricing option obtained from the analysis carried out in Figure 2 with the different sifting variants. We would like to highlight that, when using those taylored solvers in the pricing, early stopping can only be

implemented by means of callback functions. Since this would surely worsen the performances of the framework, we decided not to include the option in the analysis. As we can easily see by taking a look at the plots, the option Sifting+Cuts is the best one when dealing with GS and GL instances. In particular, for the GL instances the SD framework significantly outperforms also the baseline `Cplex` solver and finds a solution for all the instances within the time limit. The Network option, on the other hand, guarantees good results on both QGSPP and QRSPP instances.

4.2.3 Average computational time

In our experiments, we fixed a time limit of 900 seconds for all the algorithms. For each class of problems, we report in Table 1 the number of available instances (*N inst*), the number of instances solved within the time limit and the average computational time in seconds spent by `Cplex` (*NS Cplex* and *T Cplex*) and by SD (*NS SD* and *T SD*). We consider the best master/pricing options for SD in the analysis. The average is done by taking into account only the instances solved by both the algorithms. Furthermore, we add the average number of SD iterations (*N it*) needed to solve the problems.

Class	N inst	NS Cplex	T Cplex	NS SD	T SD	N it
GS	450	450	11.7	450	2.4	171.4
GL	750	666	63.8	750	16.7	90.7
POP	40	40	9.6	40	0.7	116.6
QMKP	54	54	36.5	54	11.7	31.7
QGSPP	30	30	77.6	30	15.0	290.4
QRSPP	72	72	2.2	72	0.1	19.7

Table 1: Solved instances and average CPU time.

In particular, we see that in GL problems, 84 instances out of 750 are not solved by `Cplex` within the time limit, while SD with the improving tools for both master and pricing solves all of them. We finally highlight that the most time consuming part in each SD cycle is the solution of the pricing problem (detailed results on the CPU time distribution are included in Appendix A.4).

5 Conclusions

We presented an SD framework to solve continuous convex quadratic problems. In particular, we focused on solving instances with significantly more variables than constraints and with an objective function having a dense Hessian.

We introduced a new adaptive conjugate direction method (ACDM) that is specifically designed to repeatedly solve the master problem of a SD algorithm; we also used a method that conveniently adapts the projected gradient

approach to this framework. Furthermore, two different strategies to speed up the pricing were included: an early stopping technique and a method to shrink the feasible region based on ad-hoc cuts. Finally, specific options for solving the pricing problem were tested, namely the sifting and the network optimizer.

We carefully analyzed the impact of the different master and pricing settings and we showed that our algorithm is significantly better than `Cplex`. In particular, ACDM proved to be a key ingredient to obtain an effective SD framework. Finally, the pricing options allowed to further enhance the performances of our method.

References

1. Beasley, J.E.: Portfolio optimization data (2016). URL `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/`
2. Bertsekas, D.P.: Convex optimization algorithms. Athena Scientific Belmont (2015)
3. Bertsekas, D.P., Yu, H.: A unifying polyhedral approximation framework for convex optimization. SIAM Journal on Optimization **21**(1), 333–360 (2011)
4. Birgin, E.G., Martínez, J.M., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. SIAM Journal on Optimization **10**(4), 1196–1211 (2000)
5. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
6. Buchheim, C., Traversi, E.: Quadratic combinatorial optimization using separable underestimators. INFORMS Journal on Computing **30**(3), 424–437 (2018)
7. Cesarone, F., Tardella, F.: Portfolio datasets (2010). URL `http://host.uniroma3.it/docenti/cesarone/datasetsw3_tardella.html`
8. Chu, P.C., Beasley, J.E.: A genetic algorithm for the multidimensional knapsack problem. Journal of heuristics **4**(1), 63–86 (1998)
9. Clarkson, K.L.: Coresets, sparse greedy approximation, and the frank-wolfe algorithm. ACM Transactions on Algorithms (TALG) **6**(4), 63 (2010)
10. Condat, L.: Fast projection onto the simplex and the l1-ball. Mathematical Programming **158**(1), 575–585 (2016)
11. Cristofari, A., De Santis, M., Lucidi, S., Rinaldi, F.: A two-stage active-set algorithm for bound-constrained optimization. J. Optim. Theory Appl. **172**(2), 369–401 (2017)
12. Curtis, F.E., Han, Z., Robinson, D.P.: A globally convergent primal-dual active-set framework for large-scale convex quadratic optimization. Computational Optimization and Applications **60**(2), 311–341 (2015)
13. De Santis, M., Di Pillo, G., Lucidi, S.: An active set feasible method for large-scale minimization problems with bound constraints. Computational Optimization and Applications **53**(2), 395–423 (2012)
14. Demetrescu, C., Goldberg, A.V., Johnson, D.S.: The Shortest Path Problem: Ninth DIMACS Implementation Challenge, vol. 74. American Mathematical Soc. (2009)
15. Desaulniers, G., Desrosiers, J., Solomon, M.M.: Column generation, vol. 5. Springer Science & Business Media (2006)
16. Djerdjour, M., Mathur, K., Salkin, H.: A surrogate relaxation based algorithm for a general quadratic multi-dimensional knapsack problem. Operations Research Letters **7**(5), 253–258 (1988)
17. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical Programming **91**(2), 201–213 (2002)
18. Drake, J.: Benchmark instances for the multidimensional knapsack problem (2015)
19. Elzinga, J., Moore, T.G.: A central cutting plane algorithm for the convex programming problem. Mathematical Programming **8**(1), 134–145 (1975)
20. Ferreau, H.J., Kirches, C., Potschka, A., Bock, H.G., Diehl, M.: qpoases: A parametric active-set algorithm for quadratic programming. Mathematical Programming Computation **6**(4), 327–363 (2014)

21. Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H., et al.: Qplib: A library of quadratic programming instances. Optimization Online **5846** (2017)
22. Glover, F., Kochenberger, G.: Critical event tabu search for multidimensional knapsack problems. In: Meta-Heuristics, pp. 407–427. Springer (1996)
23. Glover, F., Kochenberger, G., Alidaee, B., Amini, M.: Solving quadratic knapsack problems by reformulation and tabu search: Single constraint case. In: Combinatorial and global optimization, pp. 111–121. World Scientific (2002)
24. Goffin, J.L., Gondzio, J., Sarkissian, R., Vial, J.P.: Solving nonlinear multicommodity flow problems by the analytic center cutting plane method. Mathematical Programming **76**(1), 131–154 (1997)
25. Goffin, J.L., Vial, J.P.: Cutting planes and column generation techniques with the projective algorithm. Journal of optimization theory and applications **65**(3), 409–429 (1990)
26. Goffin, J.L., Vial, J.P.: On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm. Mathematical Programming **60**(1), 81–92 (1993)
27. Gondzio, J.: Interior point methods 25 years later. European Journal of Operational Research **218**(3), 587–601 (2012)
28. Gondzio, J., González-Brevis, P.: A new warmstarting strategy for the primal-dual column generation method. Mathematical Programming **152**(1-2), 113–146 (2015)
29. Gondzio, J., González-Brevis, P., Munari, P.: New developments in the primal–dual column generation technique. European Journal of Operational Research **224**(1), 41–51 (2013)
30. Gondzio, J., González-Brevis, P., Munari, P.: Large-scale optimization with the primal-dual column generation method. Mathematical Programming Computation **8**(1), 47–82 (2016)
31. Gondzio, J., Kouwenberg, R.: High-performance computing for asset-liability management. Operations Research **49**(6), 879–891 (2001)
32. Gondzio, J., du Merle, O., Sarkissian, R., Vial, J.P.: Accpm a library for convex optimization based on an analytic center cutting plane method. European Journal of Operational Research **94**(1), 206 – 211 (1996)
33. Gondzio, J., Sarkissian, R., Vial, J.P.: Using an interior point method for the master problem in a decomposition approach. European Journal of Operational Research **101**(3), 577–587 (1997)
34. Gondzio, J., Vial, J.P., et al.: Warm start and -subgradients in a cutting plane scheme for block-angular linear programs. Computational Optimization and Applications **14**, 17–36 (1999)
35. Grippo, L., Lampariello, F., Lucidi, S.: A nonmonotone line search technique for newtons method. SIAM Journal on Numerical Analysis **23**(4), 707–716 (1986)
36. Grippo, L., Lampariello, F., Lucidi, S.: A truncated newton method with nonmonotone line search for unconstrained optimization. Journal of Optimization Theory and Applications **60**(3), 401–419 (1989)
37. Grippo, L., Lampariello, F., Lucidi, S.: A class of nonmonotone stabilization methods in unconstrained optimization. Numerische Mathematik **59**(1), 779–805 (1991)
38. Hager, W.W., Zhang, H.: A new active set algorithm for box constrained optimization. SIAM J. Optim. **17**(2), 526–557 (2006)
39. Hearn, D.W., Lawphongpanich, S., Ventura, J.A.: Restricted simplicial decomposition: Computation and extensions. Computation Mathematical Programming pp. 99–118 (1987)
40. Holloway, C.A.: An extension of the frank and wolfe method of feasible directions. Mathematical Programming **6**(1), 14–27 (1974)
41. IBM: Cplex (version 12.6.3) (2015). URL `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`
42. Kiwiel, K.C.: Methods of descent for nondifferentiable optimization, vol. 1133. Springer (2006)
43. Levin, A.Y.: On an algorithm for the minimization of convex functions. In: Soviet Mathematics Doklady, vol. 160, pp. 1244–1247 (1965)
44. Markowitz, H.: Portfolio selection. The Journal of Finance **7**(1), 77–91 (1952)
45. Maros, I., Mészáros, C.: A repository of convex quadratic programming problems. Optimization Methods and Software **11**(1-4), 671–681 (1999)

46. Michelot, C.: A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. Journal of Optimization Theory and Applications **50**(1), 195–200 (1986)
47. Munari, P., Gondzio, J.: Using the primal-dual interior point algorithm within the branch-price-and-cut method. Computers & Operations Research **40**(8), 2026–2036 (2013)
48. Nesterov, Y., Nemirovskii, A.: Interior-point polynomial algorithms in convex programming. SIAM (1994)
49. Newman, D.J.: Location of the maximum on unimodal surfaces. Journal of the ACM (JACM) **12**(3), 395–398 (1965)
50. Nocedal, J., Wright, S.J.: Conjugate gradient methods. Numerical optimization pp. 101–134 (2006)
51. Nocedal, J., Wright, S.J.: Sequential quadratic programming. Springer (2006)
52. Palomar, D.P., Eldar, Y.C.: Convex optimization in signal processing and communications. Cambridge university press (2010)
53. Patriksson, M.: The traffic assignment problem: models and methods. Courier Dover Publications (2015)
54. Pshenichnyĭ, B.N., Danilin, I.M.: Numerical methods in extremal problems. Mir Publishers (1978)
55. Rostami, B., Chassein, A., Hopf, M., Frey, D., Buchheim, C., Malucelli, F., Goerigk, M.: The quadratic shortest path problem: complexity, approximability, and solution methods. European Journal of Operational Research **268**(2), 473–485 (2018)
56. Rostami, B., Malucelli, F., Frey, D., Buchheim, C.: On the quadratic shortest path problem. In: International Symposium on Experimental Algorithms, pp. 379–390. Springer (2015)
57. Syam, S.S.: A dual ascent method for the portfolio selection problem with multiple constraints and linked proposals. European Journal of Operational Research **108**(1), 196 – 207 (1998)
58. Tarasov, S., Khachiian, L., Erlikh, I.: The method of inscribed ellipsoids. Doklady Akademii Nauk SSSR **298**(5), 1081–1085 (1988)
59. Ventura, J.A., Hearn, D.W.: Restricted simplicial decomposition for convex constrained problems. Mathematical Programming **59**(1), 71–85 (1993)
60. Von Hohenbalken, B.: Simplicial decomposition in nonlinear programming algorithms. Mathematical Programming **13**(1), 49–68 (1977)
61. WolframAlpha: Mathematica (version 11.3) (2018). URL `http://www.wolfram.com/mathematica/`
62. Wright, M.: The interior-point revolution in optimization: history, recent developments, and lasting consequences. Bulletin of the American mathematical society **42**(1), 39–56 (2005)
63. Wright, S.J.: Primal-dual interior-point methods. SIAM (1997)
64. Ye, Y.: Interior point algorithms: theory and analysis, vol. 44. John Wiley & Sons (2011)

A Appendices

A.1 Proofs of the propositions in Section 3

A.1.1 Proof of Proposition 4

Proof. Extreme point $\tilde{x}_k$, obtained approximately solving subproblem (5), can only satisfy one of the following conditions

1. $\nabla f(x_k)^\top(\tilde{x}_k - x_k) \geq 0$, and subproblem (5) is solved to optimality. Hence we get
$$\min_{x \in X} \nabla f(x_k)^\top(x - x_k) = \nabla f(x_k)^\top(\tilde{x}_k - x_k) \geq 0,$$
that is necessary and sufficient optimality conditions are satisfied and x_k minimizes f over the feasible set X;
2. $\nabla f(x_k)^\top(\tilde{x}_k - x_k) < 0$, whether the pricing problem is solved to optimality or not, that is direction $d_k = \tilde{x}_k - x_k$ is descent direction and
$$\tilde{x}_k \notin conv(X_k). \tag{17}$$
Indeed, since x_k minimizes f over $conv(X_k)$ it satisfies necessary and sufficient optimality conditions, that is $\nabla f(x_k)^\top(x - x_k) \geq 0$ for all $x \in conv(X_k)$.

From (17) we thus have $\tilde{x}_k \notin X_k$. Since our feasible set X has a finite number of extreme points, case 2) occurs only a finite number of times, and case 1) will eventually occur. □

A.1.2 Proof of Proposition 5

Proof. We first show that at each iteration the method gets a reduction of f when suitable conditions are satisfied. Since at Step 2 we get an extreme point $\tilde{x}_k$ by solving subproblem (12), if $\nabla f(x_k)^\top(\tilde{x}_k - x_k) < 0$, we have that $d_k = \tilde{x}_k - x_k$ is a descent direction and there exists an $\alpha_k \in (0, 1]$ such that $f(x_k + \alpha_k d_k) < f(x_k)$. Since at iteration $k+1$, when solving the master problem, we minimize f over the set $conv(X_{k+1})$ (including both x_k and $\tilde{x}_k$), then the minimizer x_{k+1} must be such that

$$f(x_{k+1}) \leq f(x_k + \alpha_k d_k) < f(x_k).$$

Extreme point $\tilde{x}_k$, obtained solving subproblem (12), can only satisfy one of the following conditions

1. $\nabla f(x_k)^\top(\tilde{x}_k - x_k) \geq 0$. Hence we get
$$\min_{x \in X \cap C_k} \nabla f(x_k)^\top(x - x_k) = \nabla f(x_k)^\top(\tilde{x}_k - x_k) \geq 0,$$
that is necessary and sufficient optimality conditions are satisfied and x_k minimizes f over the feasible set $X \cap C_k$. Furthermore, if $x \in X \setminus C_k$, we get that there exists a cut c_i with $i \in \{0, \ldots, k-1\}$ such that
$$\nabla f(x_i)^\top(x - x_i) > 0.$$
Then, by convexity of f, we get
$$f(x) \geq f(x_i) + \nabla f(x_i)^\top(x - x_i) > f(x_i) > f(x_k)$$
so x_k minimizes f over X.
2. $\nabla f(x_k)^\top(\tilde{x}_k - x_k) < 0$, that is direction $d_k = \tilde{x}_k - x_k$ is descent direction and
$$\tilde{x}_k \notin conv(X_k). \tag{18}$$
Indeed, since x_k minimizes f over $conv(X_k)$ it satisfies necessary and sufficient optimality conditions, that is we have $\nabla f(x_k)^\top(x - x_k) \geq 0$ for all $x \in conv(X_k)$.

Since from a certain iteration $\bar{k}$ on we do not add any further cut (notice that we can actually reduce cuts by removing the non-active ones), then case 2) occurs only a finite number of times. Thus case 1) will eventually occur. □

A.2 Preliminary tests

In this section, we describe the way we chose the `Cplex` optimizer for solving our convex quadratic instances. Then, we explain how we set the parameters in the different algorithms used to solve the master problem in the SD framework.

A.2.1 Choice of the `Cplex` optimizer

As already mentioned, we decided to benchmark our algorithm against `Cplex`. The aim of our first test was to identify, among the seven different options for the LP solver, the most efficient in solving instances with a dense Q and $n \gg m$.

In Table 2, we present the results concerning instances with 42 constraints and three different dimensions n: 2000, 4000 and 6000. We chose problems with a small number of constraints in order to be sure to pick the best `Cplex` optimizer for those problems where the SD framework is supposed to give very good performances. For a fixed n, three different instances were solved of all six problem types. So, each entry of Table 2 represents the averages computing times over 18 instances. A time limit of 1000 seconds was imposed and in parenthesis we report (if any) the number of instances that reached the time limit.

n	Default	Primal	Dual	Network	Barrier	Sifting	Concurrent
2000	72.2	1.6	1.6	1.6	84.2	2.0	89.0
4000	641.8 (2)	12.7	13.9	13.9	618.0 (2)	11.5	689.4 (2)
6000	1000.0 (18)	31.5	30.7	30.5	1000.0 (18)	26.3	1000.0 (18)

Table 2: Comparison among the different Cplex optimizers

The table clearly shows that the default optimizer, the barrier and the concurrent methods give poor performances when dealing with the quadratic programs we previously described. On the other side, the simplex type algorithms and the sifting algorithm seem to be very fast for those instances. In particular, sifting gives the overall best performance. Taking into account these results, we decided to use the `Cplex` sifting optimizer as the baseline method in our experiments. It is worth noting that the sifting algorithm is specifically conceived by `Cplex` to deal with problems with $n \gg m$, representing an additional reason for comparing our algorithmic framework against this specific `Cplex` optimizer. When dealing with Quadratic Shortest Path problems, we used the quadratic Network optimizer, more suited for this type of problems.

A.2.2 Tolerance setting when solving the master problem

We have three options available for solving the master problem in the SD framework: ACDM, FGPM and `Cplex`. In order to identify the best choice, we need to properly set tolerances for those methods. When using `Cplex` as the master solver, we decided to keep the tolerance to its default value (that is $1E10-6$). The peculiar aspect of ACDM is that no tolerance needs to be fixed a priori. On the other hand, with FGPM, the tolerance setting phase is very importance since, as we will see, it can significantly affect the performance of the algorithm in the end.

In Table 3, we compare the different behaviours of our SD framework for the three different choices of master solver. Each line of the table represents the average values concerning the 54 instances used in the previous experiment. Column "T" represents the time (in seconds) spent by the algorithms. "Er" and "Max Er" represent the average and maximum relative errors with respect to the value found by `Cplex` (using sifting optimizer). "Ei" and "Max Ei" represent the average and maximum distance (calculated using ℓ_∞ norm) from

the solution found by `Cplex`. In the last column, "Dim" represents the dimension of the final master program.

Solver	Tol	T (s)	Er	Max Er	Ei	Max Ei	Dim
	1E-02	0.25	8.64E-02	2.67E-01	2.24E-02	5.04E-02	9.9
	1E-04	1.15	2.21E-04	6.79E-04	7.80E-04	1.44E-03	55.6
SD FGPM	1E-06	2.46	5.65E-07	2.63E-06	5.72E-05	1.86E-04	102.2
	1E-08	6.09	5.98E-09	1.15E-07	4.61E-06	1.88E-05	114.0
	1E-10	9.81	2.35E-09	4.59E-08	3.48E-06	2.16E-05	113.4
SD Cplex	1E-06	4.66	8.86E-09	4.26E-08	5.50E-06	2.46E-05	156.0
SD ACDM	None	3.63	1.53E-09	1.97E-08	2.65E-06	1.99E-05	113.1
Cplex		4.29					

Table 3: Comparison for the three different choices of master solver (Cplex indicates the results obtained with sifting optimizer).

From this table, one can observe that the ACDM based SD framework gets the best results in terms of errors with respect to `Cplex`. One can also see that the performance of the FGPM based one really changes depending on the tolerance chosen. If we want to get for FGPM the same errors as ACDM, we need to set the tolerance to very low values, thus considerably slowing down the algorithm. In the end, we decided to use a tolerance of $10E-6$ for FGPM, which gives a good trade-off between computational time and accuracy. This means anyway that we need to give up precision to keep FGPM competitive in terms of time with respect to ACDM.

A.2.3 Choice of the ε parameter for the early stopping pricing option

In this section we discuss how to fix the threshold ε used in equation (10) for the Early Stopping option. We decided to fix the value of ε as a fraction ε_0 of the quantity $|\nabla f(x_k)^\top x_k|$:

$$\varepsilon = -\varepsilon_0 |\nabla f(x_k)^\top x_k|. \tag{19}$$

The value of ε_0 has been chosen after testing three different values on a subset of instances. We chose the subset of the randomly generated instances with random dense constraints and budget constraint, where SD has the worst behavior with respect to `Cplex`. The results are presented in Table 4, where we compare the average computational time T and the number of SD iterations $N\ its$. The table presents the results on the 67 instances solved by all the algorithms.

Solver	ε_0	T (s)	N its
SD	0.0	77.4	165.0
	0.5	80.3	188.2
	1.0	70.7	165.0
	1.5	73.0	165.0
Cplex		9.4	

Table 4: Test on the ε_0 parameter for the Early Stopping technique.

One can see that, with $\varepsilon_0 = 0.5$, the time and number of iterations are larger. On the other hand, if $\varepsilon_0 = 1.5$, the threshold is too weak and the early stopping is never used.

Hence, we chose the value of $\varepsilon_0 = 1.0$, which improves the computational time while keeping the number of iterations unchanged.

A.3 Instances details

A.3.1 Generic quadratic instances

These are randomly generated quadratic programming instances. In particular, Q was built starting from its singular value decomposition using the following procedure:

- the n eigenvalues were chosen in such a way that they are all positive and equally distributed in the interval $[10^{-4}, 3]$;
- the $n \times n$ diagonal matrix S, containing these eigenvalues in its diagonal, was constructed;
- an orthogonal, $n \times n$ matrix U was supplied by the QR factorization of a randomly generated $n \times n$ square matrix;
- finally, the desired matrix Q was given by $Q = USU^\top$, so that it is symmetric and its eigenvalues are exactly the ones we chose.

The coefficients of the linear part of the objective function were randomly obtained in the interval $[0.05, 0.4]$, in accordance with the quadratic terms and in order to make the solution of the problem quite sparse.

The m constraints (with $m \ll n$) were generated in two different ways: step-wise sparse constraints (S) or random dense ones (R). In the first case, for each constraint, the coefficients associated to short overlapping sequences of consecutive variables were set equal to 1 and the rest equal to 0. More specifically, if m is the number of constraints and n is the number of columns, we defined $s = 2 * n/(m+1)$ and all the coefficients of each i-th constraint are zero except for a sequence of s consecutive ones, starting at the position $1 + (s/2) * (i-1)$. In the second case, each coefficient of the constraint matrix takes a uniformly generated random value in the interval $[0, 1]$. The right-hand side was generated in such a way to make all the problems feasible: for the step-wise constraints, the right hand side was set equal to $f * s/n$, with $0.4 \leq f \leq 1$ and for a given random constraint, the corresponding right-hand side b was a convex combination of the minimum a_{min} and the maximum a_{max} of the coefficients related to the constraint itself, that is $b = 0.75 * a_{min} + 0.25 * a_{max}$.

Each class of constraints was then possibly combined with two additional type of constraints: a budget type constraint (b) $e^\top x = 1$, and a "relaxed" budget type constraints (rb) $slb \leq e^\top x \leq sub$. Summarizing, we obtained six different classes of instances:

- S, instances with step-wise constraints only;
- S-b, instances with both step-wise constraints and budget constraint;
- S-rb, instances with both step-wise and relaxed budget constraints;
- R, instances with dense random constraints only;
- R-b, instances with both dense random constraints and budget constraint;
- R-rb, instances with both dense random and relaxed budget constraints.

For each class, we fixed $n = 2000, 3000, \ldots, 10000$, while the number of both step-wise and dense random constraints m was chosen in two different ways:

1) $m = 2$, 22, 42 for each value of n;
2) $m = n/32$, $n/16$, $n/8$, $n/4$, $n/2$ for each value of n.

In the first case, we then have problems with a small number of constraints, while, in the second case, we have problems with a large number of constraints. Finally, for each class and combination of n and m we randomly generated five instances. Hence, the total number of instances with a small number of constraints was 450 and the total number of instances with a large number of constraints was 750.

A.3.2 Portfolio instances

We used data based on time series provided in [1] and [7]. Those data are related to sets of assets of dimension $n = 226, 457, 476, 2196$. The expected return and the covariance matrix are calculated by the related estimators on the time series related to the values of the assets.

In order to analyze the behaviour of the algorithm on larger dimensional problems, we created additional instances using data series obtained by modifying the existing ones. More precisely, we considered the set of data with $n = 2196$, and we generated bigger series by adding additional values to the original ones: in order not to have a negligible correlation, we assumed that the additional data have random values close to those of the other assets. For each asset and for each time, we generate from 1 to 4 new values, thus obtaining 4 new instances whose dimensions are multiples of 2196 (that is 4392, 6588, 8784, 10980).

For each of these 8 instances, we chose 5 different thresholds for the expected return: 0.006, 0.007, 0.008, 0.009, 0.01, we thus obtained 40 portfolio optimization instances.

A.3.3 Quadratic multidimensional knapsack instances

The instances we used for the quadratic multidimensional knapsack problem are provided by J. Drake in [18]. This benchmark collects various instances, including the ORLib dataset proposed by Chu and Beasley in [8] and the GK dataset proposed by Glover and Kochenberger, mentioned in [22]. In particular, we considered only problems with n greater than 1000. Hence, we kept instances gk09, gk10 and gk11 of Glover and Kochenberger from [18], and we generated other instances using the same criteria described in [8], but using larger values of n, that is 5000, 7500 and 10000. We kept $m = 100$ in this last case and we considered two different options to obtain the right hand side. So we generated 6 instances. As regards the objective function, the coefficients of the linear part were already included in the instances and we changed their signs in order to obtain minimization problems; for the quadratic part we used again matrices generated same way as for the general problems described before. In order to get meaningful results in the end, we suitably scaled the two terms in the objective function with a parameter ρ. We used two different seeds to generate the matrix and three different values for ρ. So, we have 6 combinations for the objective function for each of the 9 linear problems (the instances gk09, gk10 and gk11 from the literature and the 6 problems generated by us) so we have 54 instances globally.

A.3.4 Quadratic shortest path instances

The directed graphs used in the experiments are related to two different kind of problems:

1. Grid shortest path problem, that is graphs represented by a squared grid;
2. Random shortest path problem, that is randomly generated graphs (obtained by the generator ch9-1-1 used in the 9th DIMACS implementation challenge [14]).

For grid shortest path instances, we considered square grids of 5 different sizes k, that is $k = 30, 40, 50, 60, 70$. We fixed the source and the sink respectively as the top-left and the bottom-right node. The number of variables n, same as the number of arcs, is $2 * k * (k - 1)$. Hence we get, respectively: $n = 1740, 3120, 4900, 7080$ and 9660. The number of constraints is the same as the number of vertices of the graph, that is k^2.

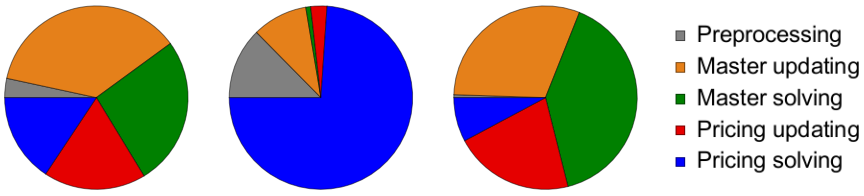
When generating random shortest path instances, we fixed three values of n: 1000, 3000 and 5000; the number of constraints m was chosen in order to get similar densities in the graphs: we respectively chose $m = 100$ and 150 for $n = 1000$, $m = 150$ and 250 for $n = 3000$ and $m = 200$ and 300 for $n = 5000$. In this way we obtained graphs with densities (number of arcs over number of arcs of a complete graph with the same number of nodes) that vary between 10% and 25%.

For both classes, we built up the objective function in this way: we defined the quadratic part with matrices generated same way as for the general problems described before; then we added linear coefficients for the linear part, generated in three different intervals: $[0.05, 0.4]$, $[0.5, 1.0]$ and $[2.0, 3.0]$. We used two different seeds for generating the quadratic part and for

the linear part we considered three different choices, so we have 6 problems for each value of n and m. In this way we obtained 30 different problems for the Grid shortest path, where m is fixed depending on n, while we got 72 instances of Random shortest path, because for each n we got 2 different values for m and for each of them we used two different seeds for generating the graph.

A.4 CPU time usage in the SD framework

Now we analyze the way CPU time is used in the SD framework, that is we show the average CPU time needed for preprocessing data, solving the master problems and solving the pricing problems with the best master and pricing settings. In Figures 4 and 5, we report the aggregated results over the first three classes of instances and on the continuous relaxations of combinatorial instances, respectively. In each figure, we report the time spent by SD in the preprocessing phase of the algorithm (*preprocessing*) and in the solution of the master and pricing problem. The solving time of both the pricing and master problem is split in the time needed to update the data structures (*updating*) and the time needed to solve the problem (*solvers*). Figures 4 clearly suggests that for generic instances the percentage of computing time of the pricing problem increases with the increase of the size of the instances. On the other hand, the subdivision of CPU times differs significantly for the three continuous relaxations of combinatorial instances considered (QGSPP, QRSPP, QMKP). First of all, we observe that for quadratic shortest path instances the percentage of computing time for the pricing is lower than the one for the quadratic multidimensional knapsack instances. This is due to the fact that the pricing problem for a quadratic shortest path instance reduces to a simple shortest path problem and thus it can be handled efficiently by the LP Network solver of `Cplex`. Finally, we notice that the percentage for the preprocessing time in random shortest path is high. This happens because the overall computing time is significantly small, hence the total time needed to prepare the initial data structures is not negligible.

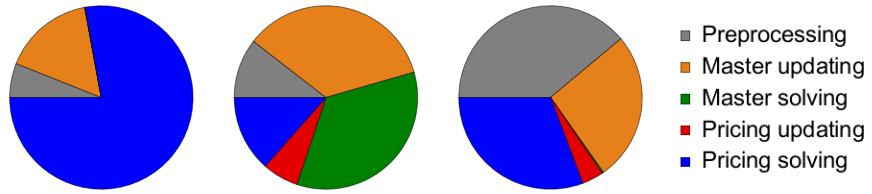


(a) GS (b) GL (c) POP (d) Legend

Fig. 4: CPU time pie charts for Portfolio and General Problems.

A.5 In-depth analysis

In order to better analyze the behavior of the SD framework, we show now how the objective function value changes with respect to the elapsed time. Since we want to get meaningful results, we only consider generic instances solved in more than 10 seconds (but always within the time limit of 900 seconds). In particular, we consider instances with random dense constraints and we take a set of 25 instances for each of the three types of additional constraints (see Appendix A.3.1). Hence, we plot

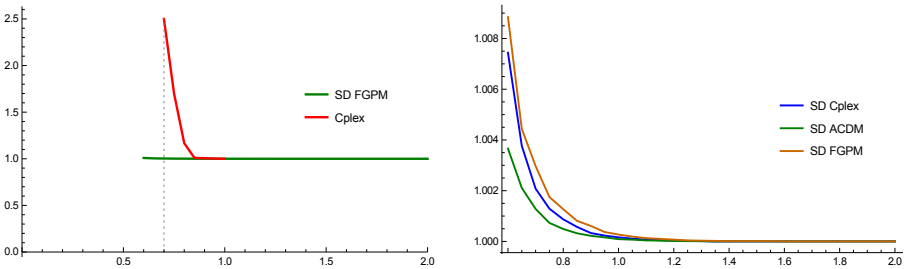


(a) QMKP (b) QGSPP (c) QRSPP (d) Legend

Fig. 5: CPU time pie charts (continuous relaxations of combinatorial instances).

- on the *x-axis* the **CPU time ratio**, that is the CPU time elapsed divided by the overall time needed by `Cplex` to get a solution on the same instance.
- on the *y-axis* the **objective function ratio**, that is the objective function value divided by the optimal value obtained by `Cplex` on the same instance.

All the results are averaged over the whole set of instances. For the SD framework, we plot the results up to twice the time needed by `Cplex` to get a solution. In the analysis, we always consider the setting that includes all the pricing options (and gives same performance as the best one). Figures 6a and 6b show the overall results for the 75 instances considered: the first figure shows the comparison between `Cplex` and SD FGPM, while the second one shows the comparison of the three different SD framework versions. From the comparison of `Cplex` and SD FGPM, it is easy to notice that SD gets a good objective function value very soon. Indeed, at a CPU time ratio 0.6 (i.e., 60% of the overall `Cplex` CPU time) corresponds an objective function ratio slightly bigger than 1 for SD FGPM, while at the same CPU time ratio `Cplex` still needs to find a feasible solution. `Cplex` gets a first feasible solution for a CPU time ratio equal to 0.7 (in this case the objective function ratio is bigger than 2.5), and it obtains an objective function ratio close to 1 only for a CPU time ratio bigger than 0.8. By taking a look at the comparison of the three different versions of our SD framework, we notice that SD FGPM actually takes longer than the others to get an objective function ratio close to 1. The better results obtained for SD FGPM hence depend, as we already noticed, on the way we choose the tolerance in the master solvers.



(a) SD FGPM vs `Cplex`. (b) SD solvers comparison.

Fig. 6: Objective function decay - Objective function ratio (y-axis) and CPU time ratio (x-axis).