Simplicial Decomposition for Large-Scale Quadratic Convex Programming

E. Bettiol[1], L. Létocart[1], F. Rinaldi[2], E. Traversi[1]

[1] LIPN UMR CNRS 7030 Université Paris 13, Sorbonne Paris Cité, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France
{bettiol,lucas.letocart,emiliano.traversi}@lipn.univ-paris13.fr

[2] Università di Padova, Dipartimento di Matematica, Via Trieste, 63, 35121 Padova, Italia
rinaldi@math.unipd.it

Mots-clés : *Nonlinear optimization, Large-scale optimization, Simplicial Decomposition.*

1 Introduction

Quadratic programming is the optimization of a quadratic objective function subject to linear or quadratic constraints. We will deal with minimization problems with convex quadratic objective function, linear constraints and continuous variables. When the size of the problem is large, very often it is more convenient to take advantage of smart or ad-hoc strategies to tackle the problem. Column generation, described for example in [3], represents one of the most important ways to deal with large-scale problems. We will analyze the use of Simplicial Decomposition, a column generation algorithm: we developed some techniques in order to make it more efficient and we compared our algorithm against the state-of-the-art software *CPLEX*. We will present our algorithm and show our results, obtained on portfolio optimization problems and on more general convex quadratic problems.

2 Simplicial Decomposition

The idea behind the Simplicial Decomposition (SD) algorithm is described in [5]: in order to solve the original problem, it is decomposed into simpler ones, which are called respectively pricing and master programs, and are solved alternatively and repeatedly. The pricing solves the original problem with a linear objective function and the master program, instead, is a problem with the original objective function, but with lower dimension and simplified constraints.

More specifically, starting from a single point, the domain of each master program is the convex hull of a finite set of affinely independent points, i.e. a simplex, and these points are the solution of the previous pricing problems: if $B_k := \{x_1, \dots x_k\}$ is the set of points after the k^{th} iteration, the simplex S_k generated by these points can be represented as a convex combination of the generators, so the dimension of the master is $k \ll n$.

On the other hand, each pricing program is linear because it minimizes the gradient of the original objective function in the optimal point of the previous master. In this way, it obtains a descent direction, so the new master will be able to find points with lower cost; otherwise, if no new points are found, the algorithm terminates.

The most significant advantage is that the dimension of the master programs are always much lower than the original one, so the overall computing time can be reduced with this decomposition. An other extremely important characteristic of this technique is that no dual information is necessary.

3 Instances

We wanted to tackle problems with high number of variables and relatively small number of constraints. So, the first instances in which we have tested SD are portfolio optimization problems in the formulation proposed by Markowitz in [4]. It has a dense positive semidefinite objective function (the risk) and only two constraints: a lower bound μ on the expected return and a simplex constraint:

$$\begin{aligned} \min \;& f(x) = x^T \Sigma x \\ \text{s. t.} \;& r^T x \geq \mu, \\ & e^T x = 1, \\ & x \geq 0, \end{aligned} \tag{1}$$

with $e = \{1, \dots, 1\}^T$ and $x \in \mathbb{R}^n$. We started using some literature data and then we stressed them, adding new values, in order to get higher dimensional instances and to be able to analyze the behaviour of the algorithm in large-size portfolio-like instances.

Then, we decided to tackle more general problems, with higher number of constraints: regarding the objective function, we generated dense, positive definite matrices for the quadratic part and randomly generated linear terms. Concerning the constraints, in order to generate feasible problems, we chose them in two different ways: step-wise sparse constraints or random dense ones. For each of these types, three policies for additional constraints are stated: a simplex constraint, or "relaxed" simplex constraints (sum of all the variables between a minimum and a maximum value), or no additional constraints. For each of these six types of problems, we generated five instances with five different seeds.

The number of variables n for Portfolio Optimization Problems varies from 225 to 10980 and for the generic quadratic problems it varies between 2000 and 10000. For these problems, moreover, the number m of constraints goes from 2 up to $n/2$. In particular, we divided these problems into two categories: the ones with "low number of constraints" are made with 2, 22 and 42 constraints. The second category contains the problems with high number of constraints and they are generated with $m = n/32$, $n/16$, $n/8$, $n/4$, $n/2$.

4 Setting

We compared the SD algorithm with the *state-of-the-art* solver *CPLEX*. We made some preliminary tests in order to find the best options for *CPLEX* and we selected the sifting solver, which is much faster than the other solvers, for our instances.

Furthermore, we introduced some modifications in the SD algorithm, both for the pricing and for the master problems, in order to improve the convergence.

4.1 Master program

The master program can be solved in various ways: the first one that we have selected is directly using *CPLEX* itself (namely *SD - Cplex*). Then, we developed two other methods in order to speed up the computational time of the master program, based respectively on the Conjugate Directions and on the Projected Gradient Methods.

We introduced an Adaptation of the (unconstrained) Conjugate Directions Method (SD - ACDM) in order to exploit the particular structure of the simplices that are generated. Indeed, they are nested one in the following, so the informations used to solve each master program can help to solve the next one in a single iteration. So, we introduced a significant warm start, that could not be present with *CPLEX*.

More specifically, we exploited the property that, for every master program, *the search for the optimum starts from a point in the interior of a facet of its domain.* So we developed and proved the convergence of a method that proceeds in this way: it starts from the previous optimum, it finds the new conjugate direction, it determines the optimum along this new line with unconstrained techniques (described, for example, in [2]) and, if needed, it projects the point onto the simplex, in order to get the constrained optimal point. In this case the intersection with the boundary of the simplex is found and the method finds the optimal point in the intersection face.

The projected gradient method (SD - PG) that we propose is an adaptation of the Spectral Projected Gradient Method described in [1].

4.2 Pricing options

With respect to the pricing, it is always solved by *CPLEX*, which is one of the best solvers for linear programs; however, we developed some features in order to improve the performance of the classic algorithm.

We introduced some cuts in order to reduce the number of points, i.e. iterations of SD, that are to be generated in order to reach the global optimum. This is done exploiting the informations on the previous iterations.

The second strategy is the use of an early stopping in the pricing problem: this reduces the computational time of the pricing and is still sufficient to provide a descent direction at each cycle, which is enough to guarantee the convergence of the algorithm.

Finally, we tested the use of the sifting solver for *CPLEX*.

We analyzed all the eight combinations of these three options in order to find the one that solves best the instances.

5 Results

The results show, for all the different problems, that SD improves the computational time spent by *CPLEX*. More in detail, when the size is small, Cplex is fast and only a few options of SD are better. But as the size increases, the improvement is more and more evident and all the methods of SD with all the options are much better than the reference solver. Deeper classifications can be shown in the three cases of portfolio optimization instances, generic quadratic instances with low number of constraints and general quadratic problems with high number of constraints. In particular in the last case, the differences among the six different choices of constraint matrix become significant: while some classes of problems are too complicated and none of the solvers can find the optimum within the imposed timelimit, in other instances, *CPLEX* reaches the time limit and some SD algorithms reach the solution in a lower time. In Table (1) the average CPU time (in seconds) elapsed with some different solvers is shown. For the case in which the number m of constraints is high, the average results are not impressive: this is due to the fact that the problems with completely random constraints increase a lot the computing time of SD, because the pricing is much slower. In order to show the results for the other instances, we added a new line, specific for the problems generated with *stepwise* constraints with high values of m. The $*$ means the combination of the best master and pricing options for each algorithm. The number of problems that have not solved within the time limit, whose computational times have been excluded from the average, is in brackets. N is the total number of instances.

	Cplex	SD Cplex	SD Cplex*	SD ACDM*	SD PG*	N
Portfolio	9.58	1.76	1.76	0.87	0.93	40
Low m	13.50	5.36	4.76	2.30	2.34	450
High m	65.66 (146)	64.37 (215)	57.9 (218)	54.79 (100)	29.71 (136)	750
High m - Step	33.38 (31)	11.98 (26)	11.92 (26)	9.41 (29)	10.79 (33)	375

TAB. 1: CPU time (in seconds) of Cplex and of SD with the best options

Another analysis can be carried out, on the subdivision of the total computing time among the master and the pricing solvers and the preprocessing time. In Table (2) the results are shown for general quadratic problems with low number of constraints. The average times are represented. The last two columns contain the number of iterations needed to achieve the solution and the dimension of the last master program, which is the dimension of the optimal face in the original domain.

	Total	Master		Pricing		Number	Dimension
	time	updating	solving	updating	solving	iterations	last master
SD - Cplex	5.93	1.38	3.21	0.53	0.81	180.04	177.04
SD - ACDM	3.60	1.08	0.68	0.41	1.42	176.02	136.93
SD - PG	3.24	1.21	0.38	0.47	1.18	152.74	129.55

TAB. 2: Averaged CPU time (in seconds), instances with low m.

From this second table it is clear that the two methods that we have developed have improved sensibly the computational time for the master. However, the time spent for solving the pricing remains substantially unchanged, because it depends mainly on the number of iterations that are computed.

Another relevant point is that there are some differences among the number of iterations and the dimension of the last master program. Although the errors between the solutions of SD and of the reference solver are always very low (relative error on the solution lower than 10^{-5}), the two algorithms that we introduced tend to find sparser representations of the optimal face and the difference in time between SD with ACDM and SD with PG depends mostly on the difference in the number of iterations, so of pricing problems, to be solved in the two cases.

6 Conclusions and perspectives

The results that we have reported show that Simplicial Decomposition, with some suitable modifications, is a really competitive algorithm and often improves dramatically the CPU time of the state of the art solver, for large-size instances.

A possible further improvement could be the development of some techniques that allow us to solve the pricing faster: indeed, the highest part of the computational time for each iteration is due to solve this problem.

These good results suggest that this method could be used in a more general setting. For example, in mixed integer convex programming, this algorithm can be inserted profitably in a branch and bound scheme.

References

[1] E.G. Birgin, J.M. Martinez and M. Raydan. *Nonmonotone spectral projected gradient methods for convex sets.* SIAM Journal on Optimization 10, pp. 1196-1211, 2000.

[2] L. Grippo, M. Sciandrone. *Metodi di ottimizzazione non vincolata.* Springer, 2011.

[3] Larsson, Torbjörn, Athanasios Migdalas, and Michael Patriksson. *A generic column generation scheme.* 2006.

[4] Markowitz, H. *Portfolio Selection.* The Journal of Finance, Vol. 7, No. 1, pp. 77-91. March. 1952.

[5] B. Von Hohenbalken. *Simplicial decomposition in nonlinear programming algorithms.* Mathematical Programming 13 (1977) 49-68.