Branch-and-bound algorithm for efficient resolution of sparse optimization problems

Gwenaël Samain

COLLEGE SCIENCES DOCTORAL DE L'INGENIERIE PAYS DE LA LOIRE ET DES SYSTEMES

CENTRALE NANTES

MÉMOIRE DE DOCTORAT DE

L'ÉCOLE CENTRALE DE NANTES

ÉCOLE DOCTORALE N° 602
Sciences de l'Ingénierie et des Systèmes
Spécialité : *Signal, Image, Vision*

Par
Gwenaël SAMAIN

Branch-and-bound algorithm for efficient resolution of sparse optimization problems

Projet de recherche doctoral présenté et soutenu à l'Ecole Centrale de Nantes, le 08 février 2024
Unité de recherche : UMR 6004, Laboratoire des Sciences du Numérique de Nantes (LS2N)

Rapporteurs avant soutenance :

Nelly PUSTELNIK — Directrice de Recherche CNRS, École Normale Supérieure de Lyon
Charles SOUSSEN — Professeur des universités, CentraleSupelec, Gif-sur-Yvette

Composition du Jury :

Président :	Xavier GANDIBLEUX	Professeur des universités, Université de Nantes
Examinateurs :	Nelly PUSTELNIK	Directrice de Recherche CNRS, École Normale Supérieure de Lyon
	Charles SOUSSEN	Professeur des universités, CentraleSupelec, Gif-sur-Yvette
	Joseph SALMON	Professeur des universités, Université de Montpellier
	Emmanuel SOUBIES	Chargé de recherche CNRS, CNRS Toulouse
Directeur de recherches doctorales :	Sébastien BOURGUIGNON	Maître de conférences HDR, Ecole Centrale de Nantes
Co-enc. de recherches doctorales :	Jordan NININ	Maître de conférences de l'ENSTA Bretagne, ENSTA Bretagne, Brest

Remerciements

J'aimerais commencer par remercier cette Terre qui nous porte et nous supporte. Merci aux personnes ayant travaillé dans les mines qui ont produit les métaux nécessaires à la fabrication de l'ordinateur qui m'a été acheté neuf lors de mon entrée en thèse. Merci à l'air que je respire. Merci à la vie qui m'a été offerte.

Cette thèse n'aurait pas vu le jour sans mes encadrants, j'ai nommé Sébastien et Jordan, que je remercie conjointement pour leur humanité et leur humour. Merci Jordan pour ta joie, ta gentillesse. L'avantage de Brest, c'est qu'on s'habille toute l'année pareil ! Je me rappelle de nos heures à gratter des maths, ou des diagrammes UML, sur un tableau dans une "désorganisation planifiée". Merci Sébastien pour ta capacité à dégager beaucoup de temps malgré un agenda fort chargé. Merci pour ton empathie, ta sensibilité aux impacts de la recherche, ainsi que ton attention à l'environnement social dans lequel tu évolues.

Merci à mes membre de CSI, Matthieu et Emmanuel. Des rencontres intéressantes, humaines, efficaces, porteuses de perspectives, vous avez joué votre rôle à la perfection, et êtes allés bien au delà !

Merci à mes coloc's successifs de bureau: Aurélie, Constance, Andrea, Oswaldo. Quelques parties de rire encore ? Merci à Yvan et Durgesh pour vos points de vue différents, enrichissants. Merci à toute l'équipe MATRIX du LabSTICC (je partage avec Jordan la tâche d'exploser vos tympans en salle de pause), ainsi qu'à toute l'équipe SIMS du LS2N. Merci en particulier à l'équipe badminton (ne te fais pas mal Louise, il y aura d'autres volants à aller chercher dans cette partie), à l'équipe pause (Maria, Younes, je compte sur votre restau, et Lolee et Antoine, ces transfuges de STR, je vous ai vu !), merci à cette flopée de doctorant qui s'organise pour faire des choses ensembles. Merci à Vanessa, Bastien, Bruno, Bertrand et toute l'équipe des référents développement durable de Centrale Nantes pour les moments de réflexion, les fresques, et les formations à l'animation. Coucou Théo, j'espère que tu continues ta thèse sur de bons rails, en tout cas en plus de trouver sur mon chemin quelqu'un qui est sur un sujet très proche, j'ai aussi trouvé une chouette personne ;)

Merci à toute ma famille, notamment à Michel (les petits oiseaux de la campagne !), Annelise (nos mercredis soirs, ce rituel de mamy), Claudine (on mouline !), Fanny (ce setup vidéo-proj), Chloé (enfin se revoir, on va y arriver !), pour tout ce soutien here and there, ces amours généreux. Jean-Luc et tes discrètes attentions, j'espère savoir te les rendre un jour. Merci à la bande de pote "village global", aux ex-élus, au groupe Warsow, j'espère bien maintenir ces liens ! Merci à Noémie, Lyse, Pierre, Arthur et Arthur pour votre présence au long cours.

Merci à Franck Ghitalla, à qui je dédie cette thèse, et à Antoine Jouglet, sans qui je n'aurais jamais eu l'idée d'en faire une.

Table of Contents

Chapter 1

Abrégé des contributions de la thèse

Contexte

Le traitement numérique du signal possède de nombreux domaines d'applications, parmi lesquels l'imagerie biomédicale, l'astronomie, la météorologie et le contrôle non-destructif. Les travaux de cette thèse sont à l'intersection entre le traitement numérique du signal et les problèmes inverses. À partir d'un vecteur de mesures $\boldsymbol{y} \in \mathbb{R}^N$, le but est de retrouver une solution d'intérêt $\boldsymbol{x} \in \mathbb{R}^Q$ telle que:

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{x} + \boldsymbol{\epsilon}, \tag{1.1}$$

où $\boldsymbol{\epsilon} \in \mathbb{R}^N$ est un terme de bruit, et la matrice $\mathbf{A} \in \mathbb{R}^{N \times Q}$ est un modèle linaire connu à l'avance.

Selon la loi de probabilité suivie par le terme de bruit, ce problème mathématique se traduit par des problèmes d'optimisation différents. Si le terme de bruit est supposé gaussien, indépendant et identiquement distribué ($\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$), trouver le meilleur ajustement pour $\boldsymbol{x}$ dans l'Équation (1.1) revient à résoudre le problème d'optimisation suivant:

$$\hat{\boldsymbol{x}}_{\text{LS}} = \underset{\boldsymbol{x} \in \mathbb{R}^Q}{\arg\min}\, \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2. \tag{1.2}$$

Le Problème (1.2) est le problème de moindres-carrés standard.

Dans bien des cas, le modèle est sous-déterminé, entraînant une infinité de solutions à l'Équation (1.1). Sinon, dans le cas d'un modèle sur-déterminé, si $\mathbf{A}$ est de rang plein, le Problème (1.2) a une solution analytique qui vaut:

$$\hat{\boldsymbol{x}}_{\text{LS}} := (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{y}. \tag{1.3}$$

Malheureusement, les matrices $\mathbf{A}$ utilisées en traitement du signal amènent très souvent à des matrices $\mathbf{A}^T\mathbf{A}$ avec un très mauvais conditionnement, ce qui signifie que l'estimée $\hat{\boldsymbol{x}}_{\text{LS}} = \boldsymbol{x}_{\text{vrai}} + (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{\epsilon}$ est numériquement instable, dû au terme $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{\epsilon}$ possédant une forte variance. Pour résoudre ces difficultés, un terme additionnel est ajouté,

soit comme une contrainte du problème, soit comme une pénalité dans la fonction objectif. Ce terme correspond à une connaissance *a priori* sur la structure de la solution $\boldsymbol{x}$. Dans un cadre Bayésien, ce terme correspond à la loi de probabilité attribuée à $\boldsymbol{x}$.

Dans cette thèse, la structure de la solution $\boldsymbol{x}$ est choisie parcimonieuse, ce qui veut dire qu'un grand nombre de composantes x_i sont supposées valoir 0, ce que l'on mesure par la fonction ℓ_0: $\|\boldsymbol{x}\|_0 := \text{Card}(\{i \mid x_i \neq 0\})$. De tels problèmes ont émergé dans le champ du traitement du signal il y a plusieurs décennies, avec des exemples d'application en imagerie astronomique [Singer et al. 1979], en séismographie [Kormylo et al. 1982] et en contrôle non-destructif [Zala 1992] (voir Section 3.2.1 pour plus de détails à propos de ces exemples). Dans le champ des statistiques, les tâches de sélection de variables peuvent être retracées au moins jusqu'à 1901 avec la méthode d'Analyse en Composante Principale [Pearson 1901]. Depuis plusieurs décennies également, des algorithmes d'optimisation parcimonieuse ont été développés [Alliney et al. 1994; Efroymson 1960; Kormylo et al. 1982; Miller 1990; Zala 1992]. À cause de l'aspect NP-difficile du problème ℓ_0 sous-jacent [Natarajan 1995], résoudre exactement un problème ℓ_0 a longtemps été considéré comme une tâche impossible. Ainsi, une vaste litérature s'est développée autour de méthodes (polynomiales) approchées, qui peuvent être retracées jusque les années 1960 [Efroymson 1960]. Ces méthodes approchées peuvent être classifiées en trois catégories:

— Les méthodes utilisant une relaxation convexe du problème ℓ_0 originel [Ben Mhenni 2020; Ben Mhenni, Bourguignon, Mongeau, et al. 2020; Condat et al. 2019; Daubechies et al. 2004; Efron et al. 2004; Friedman et al. 2010; Lee et al. 2006; Loth 2011; Massias 2017; Massias et al. 2020; Osborne et al. 2000; Qian et al. 2019; Wainwright 2009].

— Les méthodes utilisant une recherche heuristique dans le problème ℓ_0 originel [Ben Mhenni, Bourguignon, and Idier 2020; Blumensath et al. 2008; Chen et al. 1988; Pati et al. 1993; Soussen et al. 2011; Tropp 2004].

— Les méthodes utilisant une relaxation non-convexe du problème ℓ_0 [Fan et al. 2001; Soubies et al. 2017; C.-H. Zhang 2010].

Par la suite, des méthodes pour résoudre exactement des problèmes d'optimisation parcimonieuse ℓ_0 ont été proposées, d'abord il y a une dizaine d'année [Bertsimas, King, et al. 2016; Bourguignon, Ninin, et al. 2016], via la reformulation de ces problèmes comme des problèmes MIP (Mixed Integer Programming) résolus via des solveurs génériques comme CPLEX, GUROBI, CBC... Pour dépasser les limites de passage à l'échelle des solveurs génériques, des méthodes dédiées ont ensuite été développées [Ben Mhenni 2020; Bertsimas and Shioda 2009; Guyard et al. 2022; Hazimeh et al. 2021], entre autre le précédent travail de thèse [Ben Mhenni 2020]. La présente thèse se fonde sur les travaux de [Ben Mhenni 2020], et propose des techniques d'accélérations de l'algorithme de séparation-

évaluation utilisé, ainsi que son extension au cas de la parcimonie structurée.

Résumé des contributions

Ce rapport de thèse est organisé comme suit.

Le Chapitre 3 présente le contexte de l'optimisation parcimonieuse, ses champs d'applications ainsi que des algorithmes de résolution de l'état de l'art, qui sont généralement des méthodes obtenant une solution approchée du problème d'optimisation combinant un terme d'ajustement aux données quadratique avec un terme ℓ_0. Les algorithmes d'optimisation résolvant exactement le problème d'optimisation sus-mentionné sont ensuite exposés, et l'algorithme utilisé comme point de départ de cette thèse, qui est un algorithme de séparation-évaluation (*branch-and-bound* en anglais) dédié à la résolution de problèmes de moindres carrés parcimonieux [Ben Mhenni 2020; Ben Mhenni, Bourguignon, and Ninin 2021], est détaillé.

Un algorithme de séparation-évaluation approxime une séquence de sous-problèmes en calculant des bornes inférieures et supérieures sur le minimum de la fonction objectif de ces sous-problèmes. Un algorithme de séparation-évaluation a quelques éléments clefs, entre autre la stratégie d'exploration, pour savoir dans quel ordre borner les sous-problèmes, et la méthode pour obtenir les bornes inférieures des sous-problèmes considérés. Afin d'étudier comment accélérer un algorithme de séparation-évaluation dédié à l'optimisation parcimonieuse, de nouvelles stratégies d'exploration sont proposées et leur performance comparées par rapport à des stratégies d'exploration standard dans le Chapitre 4. Également, dans le même objectif d'accélération, différentes propriétés tirées de l'analyse convexe sont utilisées pour proposer trois techniques d'accélérations du calcul de la borne inférieure. Ces trois techniques sont exposées dans le Chapitre 5.

Le cadre précédent, qui sera appelée dans la suite parcimonie scalaire, a pour objectif de mettre la majorité des variables de la solution à zéro, ou réciproquement de n'avoir qu'un petit nombre de variables non nulles. Le cadre de la parcimonie structurée étend cette approche en ne se focalisant plus sur les variables individuellement, mais sur des *groupes* de variables. Une extension de l'algorithme précédent est proposée pour résoudre des problèmes d'optimisation parcimonieuse structurée, où la solution est supposée comprendre de nombreux groupes de variables valant conjointement 0, au lieu d'être supposée comprendre des variables valant 0 de manière indépendante les unes des autres. Le Chapitre 6 introduit le sujet de l'optimisation parcimonieuse structurée, avec des champs d'applications ainsi que des algorithmes de résolution de l'état de l'art. Le Chapitre 7 est dédié à l'extension de l'algorithme de séparation-évaluation de [Ben Mhenni 2020; Ben Mhenni, Bourguignon, and Ninin 2021] au cas de la parcimonie structurée, en détaillant les défis rencontrés en chemin, et les solutions trouvées. Ce faisant, plusieurs problèmes d'optimisation différents, proposant différentes bornes inférieures sur les sous-

problèmes créés, apparaissent. Ces différentes formulations pour obtenir une borne inférieure d'un sous-problème sont détaillées et comparées dans le Chapitre 8. Les trois techniques d'accélération du calcul des bornes inférieures introduites précédemment dans le Chapitre 5 sont ensuite adaptées au cas de la parcimonie structurée dans le Chapitre 9. À notre connaissance, l'algorithme de séparation-évaluation développé dans cette thèse est le premier algorithme dédié résolvant exactement des problèmes de parcimonie structurée ℓ_0. Une comparaison avec des méthodes de l'état de l'art est proposée dans le Chapitre 10 afin d'apporter un éclairage sur les avantages et les limites des travaux de cette thèse.

Enfin, le code de l'algorithme de séparation-évaluation, développé depuis plus de 6 ans maintenant (ce code étant le résultat de cette thèse et du travail de thèse précédent de [Ben Mhenni 2020]), est fourni comme un outil de recherche aux communautés du traitement du signal et de l'optimisation. Ainsi, le Chapitre 11 présente l'architecture logicielle utilisée, avec ses extensions possibles et ses limites, ainsi que la manière de prendre en compte la structure de groupe (dans le cas de la parcimonie structurée), avant de dessiner des perspectives sur des manières de dépasser les limites de performance actuelles.

Les contributions de cette thèse ont été publiées dans différentes conférences et journaux d'optimisation et de traitement du signal:

Article de journal

1. G. Samain, S. Bourguignon, J. Ninin, "Techniques for accelerating branch-and-bound algorithms dedicated to sparse optimization", *Optimization Methods and Software*, *1-38*, 2023, URL: `https://www.tandfonline.com/doi/full/10.1080/10556788.2023.2241154`. Accepté pour publication.

Conférence avec actes expertisés

1. G. Samain, S. Bourguignon, J. Ninin, "Techniques d'accélération d'une méthode de Branch-and-bound pour l'optimisation parcimonieuse", *GRETSI'22 XXVIIIème Colloque Francophone de Traitement du Signal et des Images*, Sep 2022, Nancy, France.

Conférence avec résumés

1. G. Samain, J. Ninin, S. Bourguignon, "Branch-and-bound algorithm applied to sparse optimization problems: a study of some exploration strategies", *EUROPT 18th Workshop on Advances in Continuous Optimization*, Juil 2021, Toulouse, France.
2. G. Samain, S. Bourguignon, J. Ninin, "Techniques for accelerating branch-and-bound algorithms dedicated to sparse optimization", *EUROPT 19th Workshop on Advances in Continuous Optimization*, Juil 2022, Lisbonne, Portugal.

3. G. Samain, S. Bourguignon, J. Ninin, "La dualité convexe comme accélération d'un algorithme de Branch-and-Bound dédié à l'optimisation parcimonieuse", *23e édition du congrès ROADEF, Recherche Opérationnelle et de l'Aide à la Décision en France*, Fév 2022, Villeurbanne - Lyon, France.
4. G. Samain, S. Bourguignon, J. Ninin, "Un algorithme branch-and-bound pour résoudre exactement des problèmes d'optimisation parcimonieuse structurée", *24e édition du congrès ROADEF, Recherche Opérationnelle et de l'Aide à la Décision en France*, Fév 2023, Rennes, France.

Également, le code associé est mis à disposition sous licence libre:

Code

1. G. Samain, M. Latif, R. Ben Mhenni, J. Ninin, S. Bourguignon (2024) *Mimosa* (2.0.2), source code. `https://gitlab.univ-nantes.fr/samain-g/mimosa-solver/-/tree/structured_sparsity?ref_type=heads`

Chapter 2

General introduction

Context

The work of this thesis is at the intersection of numerical signal processing and inverse problems. From a vector of measurements $\boldsymbol{y} \in \mathbb{R}^N$, the goal is to retrieve some solution of interest $\boldsymbol{x} \in \mathbb{R}^Q$ such that:

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{x} + \boldsymbol{\epsilon}, \tag{2.1}$$

where vector $\boldsymbol{\epsilon} \in \mathbb{R}^N$ is a noise term, and matrix $\mathbf{A} \in \mathbb{R}^{N \times Q}$ is a matrix known in advance.

Depending on the probability distribution followed by the noise term, this mathematical problem translates into different optimization problems. If the noise samples are assumed to be Gaussian, independent, identically distributed ($\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$), then finding the best fit for $\boldsymbol{x}$ in Equation (2.1) in the sense of the Maximum Likelihood estimate amounts to solving the following optimization problem:

$$\hat{\boldsymbol{x}}_{\text{LS}} = \underset{\boldsymbol{x} \in \mathbb{R}^Q}{\arg\min} \, \tfrac{1}{2} \|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2. \tag{2.2}$$

Problem (2.2) is the standard least-square problem.

In many cases, the model is underdetermined, which means there are infinitely many solutions to Equation (2.1). Otherwise, in the overdetermined case, if $\mathbf{A}$ is full rank, an analytical solution to Problem (2.2) is available and reads:

$$\hat{\boldsymbol{x}}_{\text{LS}} := (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{y}. \tag{2.3}$$

Unfortunately, in numerical signal processing, the matrix $\mathbf{A}$ at hand often leads to very badly conditioned $\mathbf{A}^T\mathbf{A}$ matrices, meaning the previous estimate $\hat{\boldsymbol{x}}_{\text{LS}} = \boldsymbol{x}_{\text{truth}} + (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{\epsilon}$ is numerically unstable, due to the term $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{\epsilon}$ having a high variance. To overcome these issues, an additional term is added, either as a constraint of the problem, or a penalty in the objective function. This term corresponds to some *a priori* knowledge of the structure of the solution $\boldsymbol{x}$. In a Bayesian framework, it corresponds to the prior probabilistic distribution given to $\boldsymbol{x}$.

In this thesis, the structure of the solution $\boldsymbol{x}$ is assumed to be sparse, which means a lot of components x_i are assumed to value 0, measured by the ℓ_0 function: $\|\boldsymbol{x}\|_0 := \text{Card}(\{i \mid x_i \neq 0\})$. Such problems emerged in signal processing several decades ago, with applications for example in astronomical imaging [Singer et al. 1979], seismography [Kormylo et al. 1982] and non-destructive testing [Zala 1992] (see Section 3.2.1 for more details about these examples). In the field of statistics, variable selection tasks can be dated back to 1901 with Principal Component Analysis [Pearson 1901]. To solve these tasks, sparse optimization algorithms were developed during several decades now [Alliney et al. 1994; Efroymson 1960; Kormylo et al. 1982; Miller 1990; Zala 1992]. Due to the NP-hardness of the underlying ℓ_0 problems [Natarajan 1995], exactly solving ℓ_0 sparse optimization problems was deemed impossible for a long time. As such, there is a vast literature about approximate (polynomial) methods, which can be dated back to the 1960s [Efroymson 1960]. Those approximate methods can be classified into three categories:

— Methods dealing with a convex relaxation of the problem instead of the original one [Ben Mhenni 2020; Ben Mhenni, Bourguignon, Mongeau, et al. 2020; Condat et al. 2019; Daubechies et al. 2004; Efron et al. 2004; Friedman et al. 2010; Lee et al. 2006; Loth 2011; Massias 2017; Massias et al. 2020; Osborne et al. 2000; Qian et al. 2019; Wainwright 2009].

— Methods doing some heuristic search in the original problem [Ben Mhenni, Bourguignon, and Idier 2020; Blumensath et al. 2008; Chen et al. 1988; Pati et al. 1993; Soussen et al. 2011; Tropp 2004].

— Methods using a non-convex relaxation of the problem instead of the original one [Fan et al. 2001; Soubies et al. 2017; C.-H. Zhang 2010].

Then, around a decade ago, methods for exactly solving ℓ_0 sparse optimization problems emerged [Bertsimas, King, et al. 2016; Bourguignon, Ninin, et al. 2016], which were relying on generic Mixed Integer Programming (MIP) solvers such as CPLEX, GUROBI, CBC, ... To lift up the scaling limits of generic solvers, dedicated methods were then developed [Ben Mhenni 2020; Bertsimas and Shioda 2009; Guyard et al. 2022; Hazimeh et al. 2021], among which the former thesis work [Ben Mhenni 2020]. This present thesis is based on the work of [Ben Mhenni 2020], and proposes some acceleration techniques of the branch-and-bound algorithm used, as well as an extension to the structured sparsity case.

Outline of the contributions

This report is organized as follows.

In Chapter 3, we present the context of sparse optimization, with its application fields as well as state-of-the-art solving algorithms, which are generally searching for an

approximate solution to an optimization problem combining a quadratic data-fitting term with a ℓ_0 term. The existing optimization algorithms tackling the exact resolution of such optimization problem are then discussed, and the algorithm used as the starting basis of this thesis, which is a branch-and-bound algorithm tailored for sparse ℓ_0 least-squares optimization problems [Ben Mhenni 2020; Ben Mhenni, Bourguignon, and Ninin 2021], is detailed.

A branch-and-bound algorithm sequentially approximates the solution of subproblems by computing lower bounds and upper bounds on the minimum of the objective value of these subproblems. Such algorithm has some key building blocks, among which the exploration strategy, to know which subproblems should be bounded first, and the method to compute lower bounds, the goal being to fastly compute a lower bound providing a good approximation of the subproblem at hand. To study how to accelerate a branch-and-bound algorithm dedicated to sparse optimization, we design some new exploration strategies and assess their performance against standard ones in Chapter 4, and we leverage different convex analysis properties to propose three techniques for accelerating the computation of lower bounds in Chapter 5.

Standard sparsity, which will be called scalar sparsity, is about setting the majority of the variables to zero, or conversely to get a small number of variables to some non-zero value. Structured sparsity extends this approach by targeting not just individual variables but *groups* of variables. We propose an extension of the previous branch-and-bound algorithm for tackling structured sparsity optimization problems, where the solution is sought to have groups of variables which jointly value 0, instead of having variables valuing 0 in an independent manner. Chapter 6 broadly introduces the subject of structured sparsity, with application problems as well as state-of-the-art algorithms. Chapter 7 is dedicated to extending the branch-and-bound algorithm of [Ben Mhenni 2020; Ben Mhenni, Bourguignon, and Ninin 2021] to the structured sparsity setup, detailing the challenges faced in the process, as well as the solution provided. This leads to several optimization problems providing different lower bounds for the subproblems created inside the branch-and-bound algorithm. These different optimization problems are detailed, and algorithmic solutions are proposed and compared in Chapter 8. Lower bound accelerations developed in Chapter 5 are then adapted to the structured sparsity case in Chapter 9. As far as we know, the branch-and-bound algorithm developed in this thesis is the first dedicated algorithm tackling exactly ℓ_0 structured sparsity problems. We propose some comparison with state-of-the art methods in Chapter 10 to highlight the benefits and limits of this work.

Finally, the branch-and-bound algorithm code, developed for more than 6 years now (being the result of this thesis and the former one [Ben Mhenni 2020]), is given as a research tool for the signal processing and optimization communities. To this end, Chapter 11 presents the software architecture used, with its possible extensions and its current limits,

as well as how the group structure is taken into account, before giving future paths for lifting up the current performance limits.

The contributions of this thesis were published in optimization and signal processing conferences and journals:

Journal article

1. G. Samain, S. Bourguignon, J. Ninin, "Techniques for accelerating branch-and-bound algorithms dedicated to sparse optimization", *in: Optimization Methods and Software*, *1-38*, 2023, URL: `https://www.tandfonline.com/doi/full/10.1080/10556788.2023.2241154`. Accepted for publication.

Conference with peer-reviewed proceedings

1. G. Samain, S. Bourguignon, J. Ninin, "Techniques d'accélération d'une méthode de Branch-and-bound pour l'optimisation parcimonieuse", *in: GRETSI'22 XXVIIIème Colloque Francophone de Traitement du Signal et des Images*, Sep 2022, Nancy, France.

Conference with peer-reviewed abstracts

1. G. Samain, J. Ninin, S. Bourguignon, "Branch-and-bound algorithm applied to sparse optimization problems: a study of some exploration strategies", *in: EUROPT 18th Workshop on Advances in Continuous Optimization*, Jul 2021, Toulouse, France.
2. G. Samain, S. Bourguignon, J. Ninin, "La dualité convexe comme accélération d'un algorithme de Branch-and-Bound dédié à l'optimisation parcimonieuse", *in: 23e édition du congrès ROADEF, Recherche Opérationnelle et de l'Aide à la Décision en France*, Feb 2022, Villeurbanne - Lyon, France.
3. G. Samain, S. Bourguignon, J. Ninin, "Techniques for accelerating branch-and-bound algorithms dedicated to sparse optimization", *in: EUROPT 19th Workshop on Advances in Continuous Optimization*, Jul 2022, Lisbon, Portugal.
4. G. Samain, S. Bourguignon, J. Ninin, "Un algorithme branch-and-bound pour résoudre exactement des problèmes d'optimisation parcimonieuse structurée", *in: 24e édition du congrès ROADEF, Recherche Opérationnelle et de l'Aide à la Décision en France*, Feb 2023, Rennes, France.

Moreover, the source code of the thesis software is available under a free software licence at:

Code

1. G. Samain, M. Latif, R. Ben Mhenni, J. Ninin, S. Bourguignon (2024) *Mimosa* (2.0.2), source code. `https://gitlab.univ-nantes.fr/samain-g/mimosa-solver/-/tree/structured_sparsity?ref_type=heads`

Part I

Scalar sparsity

Chapter 3

Panorama of scalar sparsity optimization problems

Contents

3.1 Introduction

What is a sparse optimization problem? This is an optimization problem where we look for a solution with only a small number of active *features*. In the simplest case, these

features are just the variables of the solution. This will be the case for this Part I. More complex cases do exist. In Part II we will see how we can tackle features which are *groups* of variables. For a regression task, this gives rise for example to the following bi-objective problem :

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \{\tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2, \|\boldsymbol{x}\|_0\}. \tag{3.1}$$

with $\|\cdot\|_0$ the ℓ_0 counting function: $\|\boldsymbol{x}\|_0 := \text{Card}(\{i|x_i \neq 0\})$. In Problem (3.1), we are looking for solutions which are at the same time fitting the data $\boldsymbol{y} \in \mathbb{R}^N$ and with a small amount of non-zero variables, which we will call active variables. The set of nonzero variables of a solution is called the *support* of the solution. The matrix $\mathbf{A} \in \mathbb{R}^{N\times Q}$ can be of arbitrary shape and content.

Notations In the remainder of this report, vectors will be denoted as $\boldsymbol{x}$ (italic, bold, lowercase letters), matrices will be written as $\mathbf{A}$ (standard, bold, uppercase letters) and scalars will have italic non-bold fonts (like x_i of M). The notation $\boldsymbol{a}_i$ with i a scalar refers to the i-th column of matrix $\mathbf{A}$, while the notation $\mathbf{A}_S$ where S is a set designates the submatrix formed by concatenating the columns indexed by the members of S.

In the remainder of this report, unless otherwise stated, $\mathbf{A}$ is assumed to be normalized: $\forall i \in \{1,..,Q\}, \|\boldsymbol{a}_i\|_2 = 1$. Solving Problem (3.1) amounts to sampling its Pareto front, which is expensive. From that bi-objective problem, we will formulate simpler mono-objective problems.

An intuitive way to formulate a mono-objective problem from (3.1) is to seek the best possible solution given a budget of active variables (with $K \in \mathbb{N}^*$ representing the budget of variables):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 \text{ subject to (s.t.) } \|\boldsymbol{x}\|_0 \leq K. \tag{$\mathcal{P}_{2/0}$}$$

The penalized formulation of Problem ($\mathcal{P}_{2/0}$) is also used (with $\mu \in \mathbb{R}^+$ which represents a trade-off between data fitting and sparsity):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \mu\|\boldsymbol{x}\|_0. \tag{$\mathcal{P}_{2+0}$}$$

We can also look for the sparsest solution given an approximation quality target, which looks like (with $\epsilon \in \mathbb{R}^+$ the allowed approximation error):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \|\boldsymbol{x}\|_0 \text{ s.t. } \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 \leq \epsilon. \tag{$\mathcal{P}_{0/2}$}$$

Problems ($\mathcal{P}_{2/0}$), ($\mathcal{P}_{2+0}$) and ($\mathcal{P}_{0/2}$) are all NP-hard [Natarajan 1995]. Problems ($\mathcal{P}_{2/0}$) and ($\mathcal{P}_{0/2}$) enjoy some easy interpretation of their parameter, while Problem ($\mathcal{P}_{2+0}$) is an unconstrained problem, which enjoys specific algorithms (see [Ben Mhenni 2020] for examples of such specific algorithms). This chapter is a broad introduction to the field of sparse optimization problems such as ($\mathcal{P}_{2/0}$), ($\mathcal{P}_{2+0}$) or ($\mathcal{P}_{0/2}$). We will link these

formulations to some practical applications in Section 3.2. After that, we will move to a general review of algorithms dedicated to the resolution of such problems in Section 3.3 before exposing the specific framework this thesis is built on in Section 3.4. Contributions of this thesis will be summarized in Section 3.5, and the datasets used throughout this thesis for numerical experiments are detailed in Section 3.6.

3.2 Applications involving sparsity

Sparse optimization problems have found numerous applications in domains such as signal processing, machine learning, statistics and finance. We highlight here some of these use cases of sparsity (this list is not exhaustive).

3.2.1 Sparse deconvolution

In the general deconvolution context, we wish to retrieve a "true" signal ($\boldsymbol{x}$) from twisted and noisy measurements ($\boldsymbol{y}$), the twisted part being modelled by the convolution of a linear filter with the true signal (matrix $\mathbf{A}$) [Stéphane 2009]. In the sparse deconvolution context, we are looking for true signals which are sparse by nature. Such signals arise for example in non-destructive testing [Zala 1992], where the anomalies to detected are rarely present, seismography [Kormylo et al. 1982], where it is assumed that only a small amount of changes of material should be detected, and astronomical imaging [Singer et al. 1979], where stars are bright spots on a black background. Depending on the application, we may look for a specific number of components, using Problem ($\mathcal{P}_{2/0}$) (see for example [Gilbert et al. 2003]), a prescribed approximation error using Problem ($\mathcal{P}_{0/2}$) (see for example [G. M. Davis et al. 1997]), or for a given trade-off between data fidelity and sparsity, leading us to use Problem ($\mathcal{P}_{2+0}$) (see for example [Soussen et al. 2011]).

3.2.2 Source separation

In the source separation context, we measure some data which are a mixture of some specific sources. In our context where the matrix $\mathbf{A}$ is fixed, these sources are known a priori. The goal is to retrieve and to untangle the different sources from the measurements. An example of source separation is spectral unmixing [Iordache et al. 2011]. In this case, each pixel is associated with a measured reflectance. Each pixel is a mixture of elementary spectral signatures. Each source has its own specific spectral signature. The goal is to recover the sources inside each pixel, knowing that there are only a few of them per pixel. The model $\mathbf{A}$ is then the concatenation of the spectral signatures of the different sources, and $\boldsymbol{x}$ represent the abundance of a given source in the pixel. This can be typically modelled by a problem of the form of ($\mathcal{P}_{2/0}$), where the best fit to data is sought given a

budget of spectral signatures, or ($\mathcal{P}_{0/2}$), where the sparsest approximation is sought given an approximation quality.

3.2.3 Variable selection

In statistics, we often deal with models containing a huge number of predictors. As such, tools like Principal Component Analysis [Pearson 1901] have been developed to help taking a grasp on the model at hand. Thus, we simplify the model itself before using the simplified model to fit some data. That way, we obtain a lower number of predictors, and it is easier to interpret the model parameters estimated from the data. Basically, we can see variable selection as having the same goal (obtain a lower number of predictors) without compromising the model quality (keeping the complex model).

It is generally formulated using Problem ($\mathcal{P}_{2+0}$) [Bonnefoy et al. 2014; Friedman et al. 2010; Hazimeh et al. 2021], and solved using several values of the trade-off parameter μ, in an attempt to both sample the Pareto front of Problem (3.1) and ease the computation and numerical issues when dealing with small values of μ. Problem ($\mathcal{P}_{2/0}$) can also be used, the optimization task is called subset selection in that case [Miller 1990]. In this application, $\boldsymbol{y}$ typically represents raw data, and $\mathbf{A}$ is a concatenation of the predictors impact on measured data.

3.2.4 Portfolio optimization

In the field of finance, sparse problems have enjoyed a fruitful use for asset management. Sparse portfolio optimization [Bertsimas and Shioda 2009; Cui et al. 2013], in its standard form, wishes to maximize profit using *few* assets, and is a (developed) variant of problem ($\mathcal{P}_{2/0}$):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\boldsymbol{x}^T \underbrace{\mathbf{A}^T\mathbf{A}}_{\boldsymbol{\Sigma}}\boldsymbol{x} - \underbrace{\boldsymbol{y}^T\mathbf{A}}_{\boldsymbol{r}}\boldsymbol{x} + \tfrac{1}{2}\|\boldsymbol{y}\|_2^2 \text{ s.t. } \|\boldsymbol{x}\|_0 \leq K$$

where $\boldsymbol{r}$ represents the expected rates of return of the different assets, and $\boldsymbol{\Sigma}$ the covariance of these rates. Some additional constraints are added to reflect asset management practices, most notably a minimum amplitude constraint for assets in use (if we are going to invest in an asset, we want to invest a minimal amount of money).

3.2.5 Compression

Compressing data can be thought as a two-step procedure:

1. Transform the data to exhibit repetitive patterns.
2. Efficiently encode the patterns.

The first step often uses zeros as a repetitive pattern. The goal is then to transform and/or approximate the data $\boldsymbol{y}$ to get a lot of zeros. Compression standards such as JPEG [Joint Photographic Experts Group 1992] and JPEG 2000 [Joint Photographic Experts Group 2000] are actually formulating problems such as $(\mathcal{P}_{2/0})$, K being related to the compression rate.

In the case of JPEG, the model $\mathbf{A}$ represents the inverse discrete cosine transform operator, and in the case of JPEG2000 the model $\mathbf{A}$ is an inverse wavelet transform. In both cases, the model $\mathbf{A}$ is an orthogonal matrix, which dramatically simplifies the problem because the term $\frac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2$ is equal to $\frac{1}{2}\|\mathbf{A}^T\boldsymbol{y} - \boldsymbol{x}\|_2^2$, leading to the following problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\mathbf{A}^T\boldsymbol{y} - \boldsymbol{x}\|_2^2 \text{ s.t. } \|\boldsymbol{x}\|_0 \leq K. \tag{3.2}$$

this problem is separable in each variable x_i, and its solution reads:

$$S = \arg\max^K |\mathbf{A}^T\boldsymbol{y}|, \boldsymbol{x}_S^\star = \mathbf{A}_S^T\boldsymbol{y}, \forall i \notin S, x_i^\star = 0.$$

where $\arg\max^K$ returns a set containing the indices of the K largest components of a given vector (see Appendix A.1 on page 163 for a sketch of the proof of this analytical solution). However, we can also aim for better compression results with non-orthogonal $\mathbf{A}$ [Candès et al. 2011], and in this case we get back to problem $(\mathcal{P}_{2/0})$ or $(\mathcal{P}_{0/2})$.

3.3 State-of-the-art algorithms

Sparse optimization algorithms were developed during several decades now [Alliney et al. 1994; Efroymson 1960; Kormylo et al. 1982; Miller 1990; Zala 1992]. Due to the NP-hardness of the underlying problems [Natarajan 1995], exactly solving sparse optimization problems was deemed impossible for a long time. As such, there is a vast literature about approximate (polynomial) methods, which can be dated back to the 1960s [Efroymson 1960]. Those approximate methods can be classified into three categories:

— Methods dealing with a convex relaxation of the problem instead of the original one. These will be detailed in Section 3.3.1.

— Methods doing some heuristic search in the original problem. These methods will be detailed in Section 3.3.2.

— Methods using a non-convex relaxation of the problem instead of the original one. These will be detailed in Section 3.3.3.

Even though the ℓ_0 original problems are NP-hard, in practice we can still solve some of them in a tractable time. This has been highlighted almost a decade ago [Bertsimas, King, et al. 2016; Bourguignon, Ninin, et al. 2016], and further research will be detailed

in Section 3.3.4 before describing the algorithm used as a starting basis of this thesis in Section 3.4.

3.3.1 Convex relaxation: ℓ_1-norm methods

The original ℓ_0 problems are discontinuous, non-convex and non-differentiable, and their NP-hardness comes from those properties. One way to deal with these difficulties is to ease them by finding a problem as close as possible to the original one while having nicer properties. This is exactly the principle behind ℓ_1 norm methods, the ℓ_1 norm being the closest continuous and convex lower approximation of the ℓ_0 function (on a bounded domain, see Figure 3.1 for an illustration). A lot of attention [Daubechies et al. 2004; Efron et al. 2004; Friedman et al. 2010; Massias 2017; Massias et al. 2020; Qian et al. 2019; Wainwright 2009] has been paid to the relaxed version of Problem $(\mathcal{P}_{2+0})$, which reads:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1. \qquad (\mathcal{P}_{2+1})$$

Due to the convexity of the problem, the constrained relaxations of Problem $(\mathcal{P}_{2/0})$ and $(\mathcal{P}_{0/2})$ can be reformulated as Problem $(\mathcal{P}_{2+1})$ with a correct choice of λ. Moreover, under conditions such as the Exact Recovery Condition [Tropp 2004] or the Restricted Isometry Property [Candès et al. 2006], the support of the global minimizer of $(\mathcal{P}_{2+1})$ coincides with the one of Problem $(\mathcal{P}_{2+0})$. As the ℓ_1 norm is convex, a descent algorithm is able to find the global minimum of the problem.

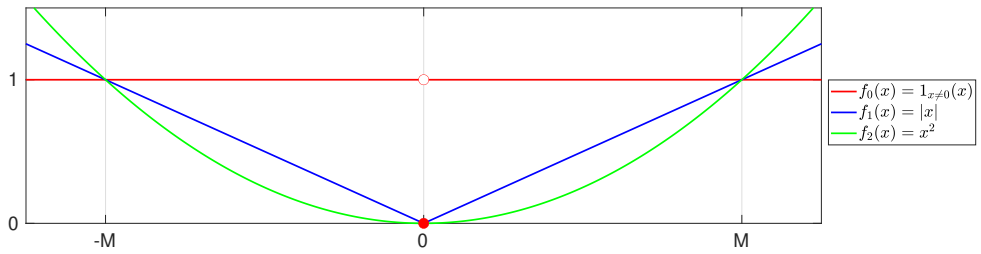


Figure 3.1 – The ℓ_1 norm is the best convex under-approximation of the ℓ_0 term, provided a bounded domain

Still, some points like $\mathbf{0}$ being non-differentiable points of the ℓ_1 norm, the relaxed problems need dedicated algorithms, as no gradient is directly available. Different algorithms have been developed to tackle the non-differentiable nature of the problem. A non-exhaustive list of the different classes of such algorithms is detailed below, with the first two (proximal algorithms and coordinate descent ones) having an asymptotic convergence guarantee, while the last two (active set algorithm, homotopy continuation method) enjoy a stronger exact convergence guarantee (convergence in a finite number of iterations).

Proximal algorithms Proximal algorithms [Briceño-Arias et al. 2023; Combettes et al. 2019; Condat et al. 2019; Komodakis et al. 2014; J.-J. Moreau 1962] rely on the monotone operator theory [Bauschke et al. 2011] to iteratively minimize the problem by doing alternate direction descent. We separate the objective function in two halves. On one hand, we have the smooth, differentiable and non-separable term $h(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2$. We can use its gradient to have a descent direction on this term. On the other hand, we have the non-differentiable but separable term $g(\boldsymbol{x}) = \lambda\|\boldsymbol{x}\|_1$. We will use the proximal operator (3.3) to get a descent direction on this term:

$$\underset{\lambda f}{\text{prox}}(\boldsymbol{z}) = \arg\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{z} - \boldsymbol{x}\|_2^2 + \lambda f(\boldsymbol{x}). \tag{3.3}$$

Overall, we will take descent steps in h and in g alternately, the whole scheme being proved convergent to $\min_{\boldsymbol{x}} h(\boldsymbol{x}) + g(\boldsymbol{x})$ under some technical considerations on the descent steps, which are mainly bounded between zero and a constant depending on the spectral norm of $\mathbf{A}$ [Condat et al. 2019]. This splitting scheme for the minimization of a composite function can be used for two [Chambolle and Pock 2011; Daubechies et al. 2004; Komodakis et al. 2014; Malitsky et al. 2018], three [Condat et al. 2019; Komodakis et al. 2014] or an arbitrary number [Condat et al. 2019] of terms, requiring a differentiable term such as our function h [Daubechies et al. 2004] or not [Chambolle and Pock 2011; Malitsky et al. 2018], with accelerations [Beck et al. 2009] or not [Daubechies et al. 2004] (see [Condat et al. 2019] for a review). Proximal algorithms also enjoy some interesting computational benefits. It is possible to parallelize them, as the proximal operator is separable if the targeted function (f in Problem (3.3)) is separable, and we only need to know how to multiply a vector with $\mathbf{A}$ and $\mathbf{A}^T$. In particular, we do not need to construct, or even to have the knowledge of the matrix $\mathbf{A}$. When the matrix $\mathbf{A}$ corresponds to some orthogonal transform, then we only need to be able to compute the transform $(\mathbf{A}\cdot)$ and the inverse transform ($\mathbf{A}^T\cdot$ for an orthgonal transform). This property is particularly used in signal and image processing [Chambolle, DeVore, et al. 1998], where the matrix $\mathbf{A}$ often corresponds to some transforms such as Fourier transform or a wavelet transform, and fast algorithms (e.g. FFT) are available to compute the transform $(\mathbf{A}\cdot)$ and its inverse $(\mathbf{A}^T\cdot)$.

One famous example of proximal algorithms is ISTA (*Iterative Soft Thresholding Algorithm*) [Daubechies et al. 2004], which is briefly described in Algorithm 1, where ST denotes the soft-thresholding operator, defined as:

$$\text{ST}_\nu(\boldsymbol{u}) := \text{sign}(\boldsymbol{u}) \circ [|\boldsymbol{u}| - \nu\mathbf{1}]_+ \tag{3.4}$$

where $\mathbf{1}$ is a vector full of 1, $[\cdot]_+ := \max(0, \cdot)$ (where max is a component-wise maximum), and $\circ$ is the Hadamard-Schur (term-by-term) product. Figure 3.2 gives the shape of the ST operator for a scalar. ISTA is dedicated to the resolution of Problem ($\mathcal{P}_{2+1}$). However,

its principle generalizes to other convex problems involving a differentiable function h and a non-differentiable function g.

Algorithm 1 Iterative Soft Thresholding

procedure ISTA($\mathbf{A}, \boldsymbol{y}, \lambda, \tau, \boldsymbol{x}^0$)
  $k \leftarrow 0$
  while not convergence **do**
    $\boldsymbol{x}^{k+\frac{1}{2}} \leftarrow \boldsymbol{x}^k + \tau \mathbf{A}^T(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^k)$
    $\boldsymbol{x}^{k+1} \leftarrow \mathrm{ST}_{\tau\lambda}(\boldsymbol{x}^{k+\frac{1}{2}})$
    $k \leftarrow k + 1$
  end while
end procedure

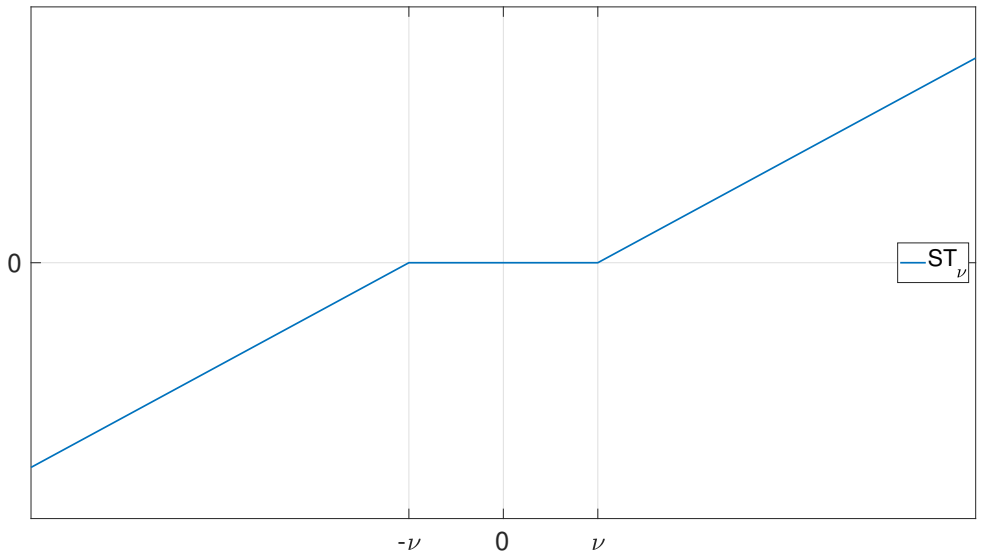


Figure 3.2 – The soft-thresholding operator ST.

Coordinate descent algorithms Coordinate descent algorithms aim to minimize a given objective function by iteratively minimizing over one coordinate at a time. In the general case, they are deemed poorly performing when compared to full gradient descent. However, in the case of sparse optimization, including ℓ_1-norm problem minimization, they perform well [Bach et al. 2011; Friedman et al. 2010], partly due to their ability to leverage the sparse nature of the sought solution by skipping some iterations involving coordinates at 0. A coordinate descent algorithm applied on Problem $(\mathcal{P}_{2+1})$ will successively and iteratively solve scalar subproblems where only one coordinate is allowed to move at each iteration k:

$$\begin{aligned} x_i^{k+1} &= \arg\min_{x_i} \tfrac{1}{2}\| \underbrace{\boldsymbol{y} - \mathbf{A}_{\{1,..,Q\}\setminus\{i\}}\boldsymbol{x}^k_{\{1,..,Q\}\setminus\{i\}}}_{\text{constant}} - \boldsymbol{a}_i x_i\|_2^2 + \underbrace{\lambda\|\boldsymbol{x}_{\{1,..,Q\}\setminus\{i\}}\|_1}_{\text{constant}} + \lambda|x_i|, \\ &= \arg\min_{x_i} \tfrac{1}{2}\|\boldsymbol{e}_i - \boldsymbol{a}_i x_i\|_2^2 + \lambda|x_i|. \end{aligned} \tag{3.5}$$

In Problem (3.5), $\boldsymbol{e}_i$ is a constant in the minimization, equal to: $\boldsymbol{y} - \sum_{j \neq i} \boldsymbol{a}_j x_j$. If the different coordinates are visited regularly and if all instances of Problem (3.5) are solved exactly to minimality, the algorithm converges to the minimum or Problem $(\mathcal{P}_{2+1})$ (see [Tseng 2001] for all the details and technical conditions). The solution of (3.5) is analytical, and reads (proof in Appendix A.2):

$$x_i = \frac{1}{\|\boldsymbol{a}_i\|_2^2} \mathrm{ST}_\lambda(\boldsymbol{a}_i^T \boldsymbol{e}_i). \tag{3.6}$$

A coordinate descent algorithm dedicated to the resolution of Problem $(\mathcal{P}_{2+1})$ is given in Algorithm 2. An interesting property of coordinate descent algorithms for Problem $(\mathcal{P}_{2+1})$

Algorithm 2 Iterative coordinate descent algorithm for Problem $(\mathcal{P}_{2+1})$

procedure ICD($\mathbf{A}, \boldsymbol{y}, \lambda, \boldsymbol{x}^0$)
    $k \leftarrow 0$
    $\boldsymbol{x} \leftarrow \boldsymbol{x}^0$
    while not convergence **do**
        for $i \in \{1, .., Q\}$ **do**
            $\boldsymbol{e}_i \leftarrow \boldsymbol{y} - \sum_{j \neq i} \boldsymbol{a}_j x_j$
            $x_i \leftarrow \frac{1}{\|\boldsymbol{a}_i\|_2^2} \mathrm{ST}_\lambda(\boldsymbol{a}_i^T \boldsymbol{e}_i)$
        end for
        $k \leftarrow k + 1$
    end while
end procedure

is that each coordinate must be visited regularly, but some can be visited more often than others without compromising convergence. As such, we can rely on the sparse nature of the target solution to visit frequently coordinates which are currently non-zero in the iterates, and visit the zero coordinates only from time to time. This technique of partial visits (which is an instance of *spacer steps* [Bertsekas 1982]) can greatly speed-up the algorithm [Bourguignon, Mary, et al. 2011; Friedman et al. 2010; Johnson et al. 2017].

Active set algorithms Active set algorithms are originally designed to handle differentiable problems with linear inequality constraints [Nocedal et al. 2006]. These algorithms solve partial problems where some of these constrained are removed, called the *inactive constraints*, and some are equated, called the *active constraints*. In other words, we solve partial problems where we decided beforehand which constraints are saturated and which are not. If the guess is wrong, we adjust the set of active constraints and solve a partial problem with the updated *active set*. When optimality conditions are satisfied, the algorithm converges.

As the algorithm retrieves the minimum of the objective function on a given active set, and as it decreases the objective function at each iteration, it cannot visit the same active set several times. The number of active sets corresponds to 2^N, with N being the

number of inequality constraints. This number is potentially huge, but finite. Thus, an active set algorithms converges in a finite number of iterations.

Problem ($\mathcal{P}_{2+1}$) is non-differentiable, and has no inequality constraints. A vanilla active set algorithm cannot handle this problem. However we can adapt it using the support of the solution: we will decide beforehand which variables are non-zero, the active variables, and which are not, the inactive variables. Minimizing over the active variable can be done through a gradient/Newton descent, as the ℓ_1 norm is differentiable everywhere except on zero-valued coordinates. This scheme has been described in length in [Ben Mhenni 2020; Ben Mhenni, Bourguignon, Mongeau, et al. 2020; Lee et al. 2006; Loth 2011]. Note that in this scheme, inactive variables are variables *constrained* to be zero. Compared to the standard use of active set algorithms, the meaning of active and inactive sets relative to constraints is swapped.

Homotopy continuation The homotopy continuation method [Ben Mhenni 2020; Osborne et al. 2000] is dedicated to solving Problem ($\mathcal{P}_{2+1}$) and uses the structure of the minimizer of this problem. Indeed, the minimizer is a piecewise linear function of the penalty parameter λ [Osborne et al. 2000]. This means the support of the minimizer is piecewise constant over the values of λ. The homotopy continuation algorithm starts from a high value $\lambda^0 := \max_{i\in\{1,..,Q\}} |\boldsymbol{a}_i^T \boldsymbol{y}|$ such that the solution is identically zero. Then, it iteratively identifies the different breakpoint values of λ^k where the support of the minimizer changes. The solution corresponding to the different breakpoint values is known analytically. When the algorithm finds λ^k and λ^{k+1} bounding the target parameter, the minimizer is retrieved through a linear combination of the solutions attached to λ^k and λ^{k+1}. The homotopy algorithm is particularly suited for very sparse problems as it starts for the zero solution and mainly adds components to the iterates, and performs well even in the case of very correlated columns of $\mathbf{A}$ (see [Bach et al. 2011] for some numerical experiments).

3.3.2 ℓ_0 heuristics

Heuristics are doing local search: they guarantee to reach a local minimum. If the matrix $\mathbf{A}$ is "nice" enough, as characterized by conditions such as ERC [Tropp 2004], they can even be proved to reach the global minimum in the low noise regime. Several classes of heuristics exist, we will describe here two classes, proximal methods and greedy algorithms.

Proximal methods Proximal methods such as Iterative Hard Thresholding (IHT) [Blumensath et al. 2008], borrow the main ideas from ℓ_1 norm minimization. As such, they are primarily targeted to solve the penalized problem ($\mathcal{P}_{2+0}$). Let's note that due to the non-convex nature of the ℓ_0 term, a proximal gradient descent algorithm such as IHT

does converge to a local minimum only, without any guarantee about the global quality of this local minimum if no assumption on $\mathbf{A}$ is taken. The proximal operator on the ℓ_0 term $\mu\|\boldsymbol{z}\|_0$ is called the *hard-thresholding* operator and reads:

$$\mathrm{HT}_\mu(\boldsymbol{z}) := \boldsymbol{x} \mid \forall i, x_i = \begin{cases} z_i & \text{if } |z_i| > \mu \\ 0 & \text{if } |z_i| \leq \mu \end{cases}. \tag{3.7}$$

The shape of the hard-thresholding operator is given in Figure 3.3.

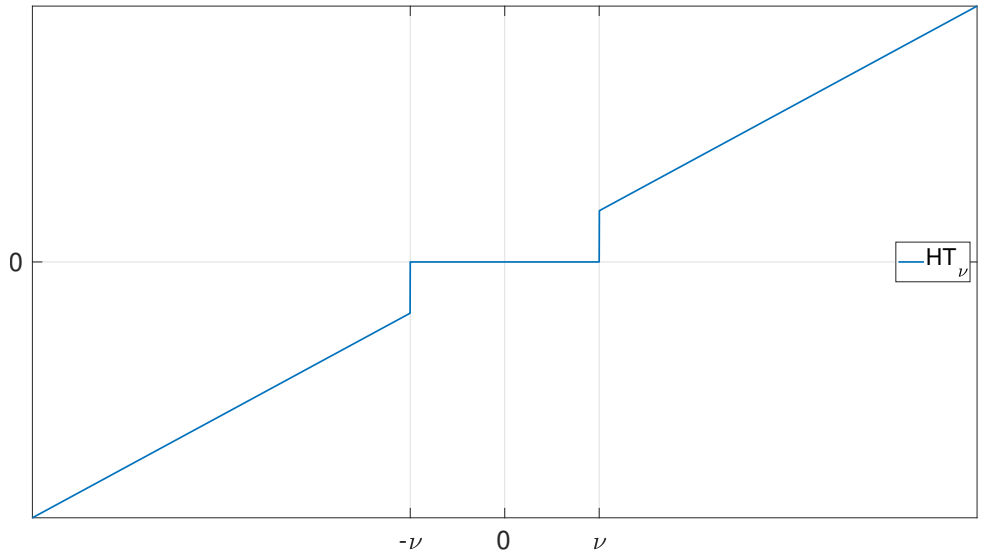


Figure 3.3 – The hard-thresholding operator

Greedy algorithms In greedy algorithms, we build a solution by increasingly adding or removing components from it. This recursive construction fits nicely for solving Problem $(\mathcal{P}_{2/0})$ and Problem $(\mathcal{P}_{0/2})$, and classical algorithms such as Orthogonal Matching Pursuit (OMP) [Pati et al. 1993] and Orthogonal Least Squares [Chen et al. 1988] are built specifically for solving instances of these problems, the parameter K or ϵ being the stopping criterion of the algorithm. Other algorithms in the "greedy" category are optimizing Problem $(\mathcal{P}_{2+0})$ instead [Ben Mhenni, Bourguignon, and Idier 2020; Soussen et al. 2011].

To give an example of how greedy algorithms work, we will give a brief description of the OMP algorithm. The OMP algorithm uses an approximation which is exact if $\mathbf{A}$ is orthogonal. If $\mathbf{A}$ is orthogonal, as we have seen in Section 3.2.5, we are trying to solve Problem (3.2). This problem has an analytical solution, which is:

$$\begin{cases} S^* = \arg\max^K |\boldsymbol{a}_i^T \boldsymbol{y}| \\ \boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\mathbf{A}_{S^*}^T \boldsymbol{y} - \boldsymbol{x}\|_2^2 = \mathbf{A}_{S^*}^T \boldsymbol{y}, \end{cases} \tag{3.8}$$

where $\arg\max^K$ is the "argmax K" operator, giving the indices of the K maximum components. In other words the solution (3.8) is composed of the K components which correlate

the most with the measurements $\boldsymbol{y}$. This quantity, $\boldsymbol{y}$, can also be viewed as the residual of the zero solution: $\boldsymbol{r} = \boldsymbol{y} - \mathbf{A}\boldsymbol{x}^0$, with $\boldsymbol{x}^0 = \mathbf{0}$. The OMP algorithm constructs a solution progressively using the correlation to a residual quantity which is the error term linked to the current approximation $\mathbf{A}\boldsymbol{x}$. Then, OMP will add the component which correlates the most with the residual (error) term. Once the component is added, the current solution and the residual are updated. This way, OMP does not choose all the components from $\boldsymbol{y}$ only, but looks at the outcome of adding features which are influencing each other. Algorithm 3 describes the different steps of OMP. Note that step 8 is written with an

Algorithm 3 Orthogonal Matching Pursuit for Problem $(\mathcal{P}_{2/0})$

1: **procedure** OMP($\boldsymbol{y}$, $\mathbf{A}$, K)
2: $\quad\boldsymbol{r} \leftarrow \boldsymbol{y}$ ▷ Initialize residual.
3: $\quad\boldsymbol{x} \leftarrow \mathbf{0}$
4: $\quad S \leftarrow \emptyset$
5: $\quad$**while** $|S| < K$ **do**
6: $\quad\quad j \leftarrow \arg\max_{i \notin S} |\boldsymbol{a}_i^T r|$ ▷ Find the best atom to add.
7: $\quad\quad S \leftarrow S \cup \{j\}$ ▷ Add it to the support.
8: $\quad\quad \boldsymbol{x}_S \leftarrow \arg\min_{\boldsymbol{z}_S} \frac{1}{2}\|\boldsymbol{y} - \mathbf{A}_S \boldsymbol{z}_S\|_2^2$ ▷ Estimate amplitudes of the new solution.
9: $\quad\quad r \leftarrow \boldsymbol{y} - \mathbf{A}_S \boldsymbol{x}_S$
10: $\quad$**end while**
11: $\quad$**return** $\boldsymbol{x}, S$
12: **end procedure**

arg min for the sake of clarity, but the solution is analytical and reads:

$$\boldsymbol{x}_S = (\mathbf{A}_S^T \mathbf{A}_S)^{-1} \mathbf{A}_S^T \boldsymbol{y}. \tag{3.9}$$

3.3.3 Non-convex relaxations

While finding the global minimum of the ℓ_1-norm relaxation and finding a local minimum of the original ℓ_0 problem are now pretty known, another class of methods relies on a non-convex relaxation of the original problem. It is in some sense a middle ground, because these methods are doing a local search like ℓ_0 heuristics, but are using a relaxation of the original problem like ℓ_1-norm problems. The relaxed problem being non-convex, practical approaches rely on some descent algorithm (which can be, for example, a proximal algorithm [X. Zhang et al. 2023], a coordinate descent algorithm [Chouzenoux et al. 2016], or a reweighted ℓ_1-norm iterative scheme [Lazzaro et al. 2015; Soubies et al. 2015]), able to find a local minimum, without knowing if this minimum is the global one of not. The theoretical side is quite loose, in the sense that there is no guarantee of converging to the global minimum of the problem which is a relaxation of the ℓ_0 problem and not the original one. However, we can start descent algorithms from multiple initial points, attempting to escape local minima. In practice, these methods lead to satisfactory solu-

tions [Fan et al. 2001; C.-H. Zhang 2010]. For a review of this field, the reader is referred to [Soubies et al. 2017].

3.3.4 Exact ℓ_0 optimization

Finally, we can also prefer to solve exactly the original problem, for contexts where the quality of the minimizer is crucial. Deemed impossible in the past (and still advertised as impossible by some papers in the present !), finding the global minimum of ℓ_0 problems is possible. Clearly, the related algorithms have a worst-case complexity which is exponential in the size of the unknowns, compared to the previously described algorithms which are all polynomial in the worst case. Indeed, in the worst case, the related algorithms are doing an exhaustive combinatorial search, exploring all the $\binom{N}{k}$ possible support configurations. Consequently, no theoretical result tells us the related algorithms terminate in a "reasonable" time. In practice, we are rarely in the worst case, and the related algorithms converge much faster. What are these algorithms? They are mainly branch-and-bound and branch-and-cut algorithms, either generic ones [Bertsimas, King, et al. 2016; Bourguignon, Ninin, et al. 2016], or dedicated ones [Ben Mhenni 2020; Bertsimas and Shioda 2009; Guyard et al. 2022; Hazimeh et al. 2021].

For the generic algorithms, these are generally *Mixed Integer Programming* (MIP) solvers, such as CBC, CPLEX, GUROBI, ... Using them to solve one of the problem $(\mathcal{P}_{2/0})$, $(\mathcal{P}_{2+0})$, $(\mathcal{P}_{0/2})$ amounts to rewrite it in a MIP form, which means rewriting the ℓ_0 term as a sum of binary variables, adding constraints such that those binary variables are indicating if a given x_i is zero or non-zero. For example, using the so called "big-M" constraints, one can formulate Problem $(\mathcal{P}_{2+0})$ in a MIP form as (see [Bertsimas, King, et al. 2016; Bienstock 1996; Bourguignon, Ninin, et al. 2016]):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q, \boldsymbol{b}\in\{0,1\}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \mu\sum_{i=1}^{Q} b_i \text{ s.t. } \forall i\in\{1,..,Q\}, -Mb_i \leq x_i \leq Mb_i. \tag{3.10}$$

In Problem (3.10), M must be chosen big enough so that the global minimum of Problem $(\mathcal{P}_{2+0})$ lies inside the feasible space. Generic solvers allow one to do rapid prototyping, and are also suited to solve sparse problems with arbitrary linear inequality and equality constraints. They are also especially suited when designing accelerations such as cuts, which are reducing the search space by adding some non-trivial linear constraints. One drawback of generic solvers is their poor performance when scaling up the problems dimensions. Dedicated algorithms can do much better in this respect, see for example [Ben Mhenni 2020; Ben Mhenni, Bourguignon, and Ninin 2021] for a comparison.

On the dedicated algorithms side, some are dedicated to the penalized formulation $(\mathcal{P}_{2+0})$ [Guyard et al. 2022; Hazimeh et al. 2021], some target the cardinality constrained one $(\mathcal{P}_{2/0})$ [Bert-

simas and Shioda 2009], and some are suited for all three formulations [Ben Mhenni 2020; Bourguignon, Ninin, et al. 2016]. The behaviour of one of these dedicated branch-and-bound algorithms, which used as the starting basis of this thesis, will be described in Section 3.4.

3.4 Branch-and-bound basis of this thesis

This section is dedicated to describing the branch-and-bound algorithm this thesis is based upon. This can be seen as a digest of earlier publications linked to Ramzi Ben Mhenni's works [Ben Mhenni 2020; Ben Mhenni, Bourguignon, and Ninin 2021]. No new contribution is claimed within this section. Also, while the original work [Ben Mhenni 2020] is suited to the three problems $(\mathcal{P}_{2/0})$, $(\mathcal{P}_{2+1})$, $(\mathcal{P}_{0/2})$, we will focus on the penalized problem $(\mathcal{P}_{2+0})$ for the remainder of this report.

3.4.1 Main ideas

Branch-and-bound algorithms [Nemhauser et al. 1988] rely on a divide-and-conquer strategy: the search space is iteratively divided into smaller blocks, represented by nodes in a tree. A node is then a subproblem, which is simpler than the original one because its search space is reduced. Indeed, the search space corresponding to a node is smaller than the one of the node's parent. For each node, we compute upper and lower bounds on the minimum of the corresponding subproblem. As we progress through the algorithm, the search space in each node becomes smaller, refining the bounds.

If we iterate the node division process so that the resulting nodes have a search space containing only one point, we will compute exactly the minimum of the original function in these nodes, but we will also have an exponential number of nodes, hitting the worst-case complexity of the algorithm. Instead, we will use the bounds we have for each node. The lowest upper bound across all nodes $\overline{\text{ub}}$ represents the best solution found so far. If a given node $\mathbf{N}_i$ has a lower bound $\text{lb}_{\mathbf{N}_i}$ greater than $\overline{\text{ub}}$, it means that the corresponding subproblem minimum value exceed $\overline{\text{ub}}$, so this node cannot improve the currently best solution. Consequently, this node is not worth being explored, and we can stop dividing it: we prune the node.

This node pruning is a key element in making a branch-and-bound method more effective than its worst-case complexity. A synthetic version of a branch-and-bound method is given in Algorithm 4.

Search space Before detailing the main steps of Algorithm 4, we need to know what is the search space at hand. A naive way would be to search over the space of $\boldsymbol{x}$: $\mathbb{R}^Q$. However, if we look more in depth at problem $(\mathcal{P}_{2+0})$, we can see that if we fix the *support*

Algorithm 4 Branch & Bound algorithm for optimization of $(\mathcal{P}_{2+0})$.

procedure BRANCHANDBOUND($\mathcal{L}$) with $\mathcal{L}$ a data structure holding nodes
  $\overline{\text{lb}} \leftarrow -\infty$, $\overline{\text{ub}} \leftarrow +\infty$ and $\hat{\boldsymbol{x}} \leftarrow \mathbf{0}$
  while $\mathcal{L}$ is not empty or *another stopping condition is not met* **do**
    Pop a node $\mathbf{N}$ from $\mathcal{L}$. ▷ Exploration strategy
    Divide $\mathbf{N}$ into two sub-nodes $\mathbf{L}$ and $\mathbf{R}$. ▷ Branching strategy
    for each sub-node $\mathbf{N}_i$ in $\{\mathbf{L}, \mathbf{R}\}$ **do**
      Compute a lower bound $\text{lb}_{\mathbf{N}_i}$ of $\mathbf{N}_i$ with solution $\boldsymbol{x}_{\text{lb}}^{\mathbf{N}_i}$. ▷ Bounding
      if $\text{lb}_{\mathbf{N}_i} < \overline{\text{ub}}$ **then**
        Compute an upper bound $\text{ub}_{\mathbf{N}_i}$ of $\mathbf{N}_i$ with solution $\boldsymbol{x}_{\text{ub}}^{\mathbf{N}_i}$. ▷ Bounding
        if $\text{ub}_{\mathbf{N}_i} < \overline{\text{ub}}$ **then**
          Update the best solution found: $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}_{\text{ub}}^{\mathbf{N}_i}$ and $\overline{\text{ub}} \leftarrow \text{ub}_{\mathbf{N}_i}$.
          Prune nodes in $\mathcal{L}$ which are sub-optimal with respect to the new $\overline{\text{ub}}$ (such that $\text{lb}^{\mathbf{N}} > \overline{\text{ub}}$).
        end if
        Push $\mathbf{N}_i$ in $\mathcal{L}$. ▷ Exploration strategy
      end if
    end for
  end while
  if $\mathcal{L}$ is not empty **then** ▷ Case of a truncated branch-and-bound
    Compute the lowest lower bound: $\overline{\text{lb}} \leftarrow \min_{\mathbf{N} \in \mathcal{L}} \text{lb}_{\mathbf{N}}$
  else
    $\overline{\text{lb}} \leftarrow \overline{\text{ub}}$
  end if
  return $(\overline{\text{lb}}, \overline{\text{ub}}, \hat{\boldsymbol{x}})$ ▷ The minimum value lies in $[\overline{\text{lb}}, \overline{\text{ub}}]$
end procedure

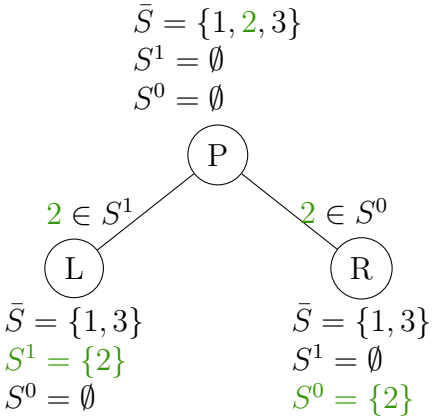


Figure 3.4 – Node division example, where we branch on the green variable.

of the solution $S = \{i | x_i \neq 0\}$, the ℓ_0 term becomes constant and we end up with a simple least-squares problem:

$$\boldsymbol{x}_S^* = \arg\min_{\boldsymbol{x}_S} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_S \boldsymbol{x}_S\|_2^2 + \mu|S| = (\mathbf{A}_S^T \mathbf{A}_S)^{-1} \mathbf{A}_S^T \boldsymbol{y}. \tag{3.11}$$

This means the difficult task in problem $(\mathcal{P}_{2+0})$ is finding the correct support. Once the support is optimal, finding the optimal amplitudes is easy. Consequently, the branch-and-bound procedure described in Algorithm 4 focuses on finding the optimal support. We do so by using the space of supports as a search space. Each node identifies with a subspace of supports: a set where some components of $\boldsymbol{x}$ must be nonzero, some other components must be zero, and some components remain free. We denote a node by $\mathbf{N}(S_1, S_0, \bar{S})$ (simplified to $\mathbf{N}$ when there is no ambiguity), where:

— S_1 contains the indices of the nonzero components,

— S_0 contains the indices of the zero components,

— $\bar{S}$ contains the indices of the undecided components.

In this context, dividing a node $\mathbf{N}(S_1, S_0, \bar{S})$ into two children means taking an index i from $\bar{S}$ and putting it to S_1 for the left child and to S_0 for the right child (the left and right side being arbitrary conventions), as illustrated by Figure 3.4. Now, let's write the subproblem corresponding to a node. When a components is put to S_0, it is set at 0, so it will be removed from the data-fitting term as well as from the ℓ_0 term. When a component is put to S_1, it must be part of the solution, so it is kept in the data-fitting term, but its contribution to the ℓ_0 term is replaced by 1: it must be part of the support, so the ℓ_0 term must count it. This means that inside a node $\mathbf{N}(S_1, S_0, \bar{S})$, the ℓ_0 problem reads:

$$\min_{\boldsymbol{x} \in \mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1} \boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}} \boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|S_1| + \mu\|\boldsymbol{x}_{\bar{S}}\|_0 \text{ s.t. } \boldsymbol{x}_{S_0} = 0. \tag{3.12}$$

Replacing $\mu\|\boldsymbol{x}_{S_1}\|_0$ by $\mu|S_1|$ assumes that every component in S_1 will be non-zero at the

optimum. If it is not the case, in other words if some components in S_1 are at 0 at the optimum, this means the guess for S_1 is wrong: these components should not be in S_1. In this case, Problem (3.12) for the node considered will have a higher objective function than for a node with the same components in S_1 except the ones at 0 (which can be either in $\bar{S}$ or S_0), so the wrong guess will be discarded by sub-optimality.

Key points We are now ready to detail the main steps of the branch-and-bound algorithm 4. They are three building blocks for a branch-and-bound:

— the lower and upper bounding methods (steps 7 and 9 respectively),

— the index $i \in \bar{S}$ to pick to divide a node into two children, called the *branching strategy* (step 5),

— the method to choose the next node to be divided, called the *exploration strategy* (steps 4 and 14).

These building blocks are now detailed in Sections 3.4.2, 3.4.3, 3.4.4, respectively.

3.4.2 Bounds

Bounds are crucial for the branch-and-bound performance. We wish to have bounds which are at the same time as tight as possible and cheap to compute.

Lower bound A tight lower bound will be helpful to prune a lot of sub-optimal nodes. At the same time, we wish to have something computationally cheap. To this end, we use a convex relaxation, and in particular an ℓ_1-norm relaxation. However, the ℓ_1 norm is a convex lower approximation of the ℓ_0 function only on a bounded domain (see Figure 3.1). To overcome this, we add a box constraint to our original problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2+\mu\|\boldsymbol{x}\|_0 \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M \qquad (\mathcal{P}_{2+0}^M)$$

where $M \in \mathbb{R}^{+*}$ is fixed and $\|\cdot\|_\infty$ stands for the ℓ_∞-norm: $\|\boldsymbol{x}\|_\infty := \max_{i\in\{1,..,Q\}} |x_i|$. This box constraint leads to nodes subproblems of the form:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}_{S_1}\boldsymbol{x}_{S_1}-\mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2+\mu|S_1|+\mu\|\boldsymbol{x}_{\bar{S}}\|_0 \text{ s.t. } \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0}=0 \end{cases}. \qquad (\mathcal{P}_{2+0}^{\mathbf{N}})$$

At any feasible point of problem $(\mathcal{P}_{2+0}^{\mathbf{N}})$, we have:

$$\|\boldsymbol{x}\|_0 = \sum_{i\in\{1..Q\}} \mathbf{1}_{x_i\neq 0} \geq \sum_{i\in\{1..Q\}} \tfrac{|x_i|}{M} = \tfrac{1}{M}\sum_{i\in\{1..Q\}} |x_i| = \tfrac{1}{M}\|\boldsymbol{x}\|_1. \qquad (3.13)$$

This means that a valid lower bound $\mathrm{lb}^{\mathbf{N}}$ of Problem $(\mathcal{P}^{\mathbf{N}}_{2+0})$ is:

$$\mathrm{lb}^{\mathbf{N}} = \min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|S_1| + \tfrac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_1 \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M, \boldsymbol{x}_{S_0} = 0. \quad (\mathcal{P}^{\mathbf{N}}_{2+1})$$

This is the lower bound problem we are going to focus on in the remaining of Part I. The minimizer of this problem will be denoted as $x^{\mathbf{N}}_{\mathrm{lb}}$. Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$ can be solved by any algorithm presented in Section 3.3.1 with some minor adaptations. Computing such lower bound at each node takes approximately 90% of the total running time of the algorithm. Accelerating those computations is one contribution of this thesis, detailed in Chapter 5.

Upper bound We would like to have an upper bound both cheap to compute and able to retrieve the global minimum of the corresponding subproblem when the node has no component in $\bar{S}$ (the support is fully decided). As such, we will compute the optimal solution of Problem $(\mathcal{P}^{\mathbf{N}}_{2+0})$ *when restricted to* S_1:

$$\mathrm{ub}^{\mathbf{N}} = \min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1}\|_2^2 + \mu|S_1| \text{ s.t. } \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = 0 \\ \boldsymbol{x}_{\bar{S}} = 0 \end{cases}. \tag{3.14}$$

Problem (3.14) is just a box-constrained least-squares problem. Though quite simple, no analytical solution can be given, and we rely on an iterative algorithm to optimize it, such as an active set algorithm (see Section 3.3.1), or an interior point algorithm (see [Nocedal et al. 2006] for a description of interior point algorithms).

3.4.3 Exploration strategy

The exploration strategy defines the way to pick the next node to be divided. Standard exploration strategies include depth-first search, breadth-first search and best-first search [Nemhauser et al. 1988].

Depth-first search seeks to unroll a branch (a path) in the branch-and-bound tree as deep as possible, before unrolling another branch. In other words, when we divide a node, the next node to be divided will be one of its children, as long as no pruning happens. In case of pruning, we backtrack to the closest ancestor and start again unrolling. Depth-first search is deemed to quickly give good upper bounds, by quickly following the choices of the branching strategy.

Breadth-first search is quite the opposite of depth-first search. While in depth-first search, we seek to get as deep as possible, in breadth-first search we seek to be as close to the root node as possible. In other words, we explore the branch-and-bound tree by dividing all the nodes at a given height before dividing nodes deeper in the tree. Breadth-first search is believed to be poorly performing in general when it comes to refining the

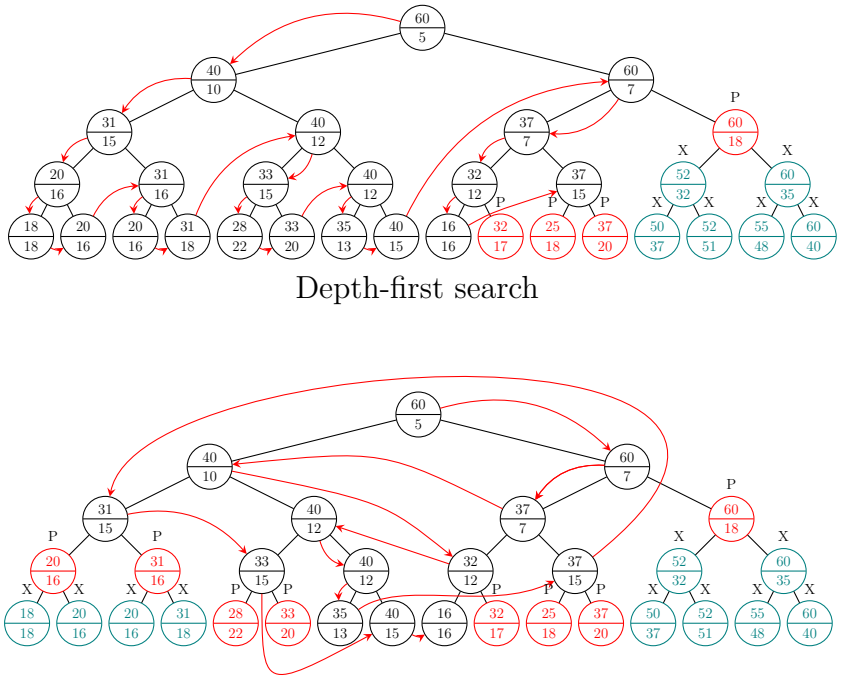


Figure 3.5 – Two different exploration strategies operating on a branch-and-bound tree. Depth-first search (top) and best-first search (bottom). Numerical values in the upper (respectively lower) part of each node **N** indicates its corresponding upper bound $\text{ub}_{\mathbf{N}}$ (respectively its lower bound $\text{lb}_{\mathbf{N}}$). The red arrows illustrate the path followed by the exploration strategy. A node denoted by P is a pruned node: its children are not created, saving all the nodes marked by an X.

bounds, even though it can provide quite diverse solutions when using a truncated branch-and-bound, that is to say when the branch-and-bound is stopped before convergence.

Best-first search is quite different from the previous two strategies because it does not rely on the structural position of the nodes, but on the bounds computed at each node. More precisely, best-first search divides in priority the node with the lowest lower bound. As such, it is deemed to quickly increase the lower bounds, which can be thought as the dual behaviour of depth-first search.

Empirically, the choice of the exploration strategy has an impact on the overall performance. A small hand-crafted example is given in Figure 3.5 to illustrate how two different exploration strategies applied on the same branch-and-bound tree lead to a different number of explored nodes. In this small example, 25 nodes are explored using depth-first search, while only 21 nodes are explored using best-first search.

One contribution of this thesis is the design of tailored exploration strategies, and their comparison with standard ones. This will be detailed in Chapter 4.

3.4.4 Branching strategy

The branching strategy is responsible for dividing a node into two children. An important aspect of this division is to be able to create a "favorable" node and a "defavorable" one. The "favorable" node would be the node which has a high chance to contain the optimal support. A favorable node should have a good (i.e. a low) upper bound, challenging the current $\overline{\text{ub}}$ as often as possible. The "defavorable" node would be a node with a low chance to contain the optimal support. A defavorable node should have a high lower bound, leading it to be pruned as soon as possible. As a recall, in our case, dividing a node $\mathbf{N}(S_1, S_0, \bar{S})$ means taking a component index i from $\bar{S}$. This component is then put into S_1 for its left child, and into S_0 for its right child (the left and right side being arbitrary conventions). The branching strategy is about which $i \in \bar{S}$ should be picked.

Standard branching strategies include strong branching, maximum infeasibility and minimum infeasibility [Ben Mhenni 2020; Bienstock 1996]. Strong branching has a look-before-leaping behaviour: at a given node, all the possible divisions will be tested, and we retain the one giving the best improvement of bounds. This branching rule is really costly, as we compute a high number (in the order of $2Q$) of lower bounds for each node. Some variations try to avoid testing every combination through some heuristic decision (see for example [Achterberg et al. 2005]).

Maximum infeasibility originally comes from the resolution of MIP problems. When looking at the MIP problem (3.10), a generic solver will compute lower bounds on it by computing relaxations, where $b_i \in \{0, 1\}$ is relaxed to $b_i \in [0, 1]$. Maximum infeasibility then chooses the variable b_i which is closest to $\frac{1}{2}$. In our case, this means taking the minimizer of the lower bound problem $x_{\text{lb}}^{\mathbf{N}}$, and picking the variable in $\bar{S}$ whose absolute value is the closest to $\frac{M}{2}$ (see [Ben Mhenni 2020]):

$$i = \underset{i \in \bar{S}}{\arg\max} \left(\min(|(\boldsymbol{x}_{\text{lb}}^{\mathbf{N}})_i|, M - |(\boldsymbol{x}_{\text{lb}}^{\mathbf{N}})_i|) \right) \tag{3.15}$$

Minimum infeasibility is the opposite: it chooses the variable in $\bar{S}$ whose absolute value is the closest to either 0 or M:

$$i = \underset{i \in \bar{S}}{\arg\min} \left(\min(|(\boldsymbol{x}_{\text{lb}}^{\mathbf{N}})_i|, M - |(\boldsymbol{x}_{\text{lb}}^{\mathbf{N}})_i|) \right) \tag{3.16}$$

Both maximum infeasibility and minimum infeasibility are standard choices for generic solvers. However, they give equal importance to zero variables and high amplitude ones. Here, we will use the maximum of amplitude rule developed in [Ben Mhenni 2020] as-is, which reads:

$$i = \underset{i \in \bar{S}}{\arg\max} \, |(\boldsymbol{x}_{\text{lb}}^{\mathbf{N}})_i| \tag{3.17}$$

This rule, which favours components close to $\pm M$ (whereas minimum infeasibility favours

components close to 0 or $\pm M$ in a symetric manner) shows better performance than standard choices for sparse optimization problems [Ben Mhenni 2020]. Intuitively, it selects components which have a high amplitude in the lower bound, meaning they have a good chance to be in the optimal support. Therefore, including these components to the support lead to "favorable" nodes, while excluding them lead to "defavorable" nodes.

3.5 Contributions

Three classes of contributions were made during this thesis. All tried to improve or generalize the branch-and-bound algorithm described previously. The first class of contributions keeps Problem ($\mathcal{P}^M_{2+0}$) as the target problem, and try to make the branch-and-bound quicker, either by choosing an appropriate exploration strategy (detailed in Chapter 4) or by making lower bound computations more efficient (detailed in Chapter 5).

The second class of contributions extends Problem ($\mathcal{P}^M_{2+0}$) to the more general class of structured sparsity, which requires some changes about the search space and node division framework (detailed in Chapter 7) as well as new formulations and algorithms to compute the lower bounds (detailed in Chapter 8). Accelerations for the lower bound computations are proposed in Chapter 9, and comparisons with state of the art methods are proposed in Chapter 10.

Finally, the third class of contributions are transverse, and corresponds to contributions related to the implementation of the algorithms of the previous chapters. They cover modularization of the branch-and-bound code to ease further developments, as well as adaptations for the structured sparsity case, which are both covered in Chapter 11.

These contributions are tested, whenever applicable, against synthetic datasets whose generation is described in Section 3.6.

3.6 Data generation protocol

For the different numerical experiments investigated in the following of this thesis report, datasets were generated in a similar vein to [Ben Mhenni, Bourguignon, and Ninin 2021; Bertsimas, King, et al. 2016]. The goal is to generate random matrices $\mathbf{A}$ of varying difficulty. To this end, the matrices are generated with rows following a multivariate Gaussian distribution $\underline{\boldsymbol{a}_i} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$. The covariance matrix $\mathbf{\Sigma}$ is set such that $\Sigma_{ii} = 1$ and $\forall i, \forall j \neq i, \Sigma_{ij} = \rho^{|i-j|}, \rho \in [0, 1[$. When ρ is close to 1, neighbouring columns of $\mathbf{A}$ are highly correlated, which gives a difficult optimization problem. Conversely, when ρ is close to 0, neighbouring columns of $\mathbf{A}$ are lowly correlated, and the optimization problem is easier. In both cases, two columns of $\mathbf{A}$ are more correlated if they are close to each other. The columns of matrix $\mathbf{A}$ are normalized: $\forall i \in \{1, .., Q\}, \|\boldsymbol{a}_i\|_2 = 1$.

To simulate diverse situations, we will use different values of ρ. To keep tractable instances, we will adjust problems dimensions with the ρ chosen. The values for ρ, Q, N used in this thesis are summarized in Table 3.1. $\rho = 0.92$ corresponds to very difficult problems, $\rho = 0.8$ and $\rho = 0.7$ corresponds to moderately correlated problems, and $\rho = 0$ corresponds to easy (but high-dimensional) problems.

	N	Q
$\rho = 0.92$	500	100
$\rho = 0.8$	500	100
$\rho = 0.7$	500	1 000
$\rho = 0$	1 000	100 000

Table 3.1 – Values for ρ, N, Q

Once the matrix $\mathbf{A}$ is generated, we create instances by generating a ground truth value $\boldsymbol{x}_{\text{truth}}$ with a support S_{truth} such that $\forall i \in S_{\text{truth}}, x_i = 1, \forall i \notin S_{\text{truth}}, x_i = 0$. Measurements $\boldsymbol{y} \in \mathbb{R}^N$ are generated with the relation $\boldsymbol{y} := \mathbf{A}\boldsymbol{x}_{\text{truth}} + \boldsymbol{\epsilon}$, with additive noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_\epsilon^2)$. The noise variance σ_ϵ^2 is set such that the signal-to-noise ratio (SNR) $\|\mathbf{A}\boldsymbol{x}_{\text{truth}}\|_2^2/(N\sigma_\epsilon^2)$ is equal to 6 (or equivalently a SNR of about 7.78 dB). The μ parameter of Problem $(\mathcal{P}_{2+0}^M)$ is tuned empirically to retrieve a solution $\hat{\boldsymbol{x}}$ in Algorithm 4 such that $\|\hat{\boldsymbol{x}}\|_0 = \|\boldsymbol{x}_{\text{truth}}\|_0$ [1]. Parameter M is set to $1.1 \max |\mathbf{A}^T\boldsymbol{y}|$. Authors in [Ben Mhenni, Bourguignon, and Ninin 2021] used similar datasets, and investigated the performance of different methods, including ℓ_1-norm methods as well as the branch-and-bound algorithm used as a starting basis of this thesis with different time limits. As shows in [Ben Mhenni, Bourguignon, and Ninin 2021], the branch-and-bound algorithm improves the quality of the estimated solution, compared to state-of-the-art approaches, particularly the ℓ_1-norm method, as soon as the branch-and-bound algorithm is left running for more than 10 seconds.

1. The different datasets used in this thesis for $\rho \in \{0.7, 0.8, 0.92\}$ are available at `https://gitlab.univ-nantes.fr/samain-g/mimosa-random-matrices-dataset`

Chapter 4

Exploration strategies study in the branch-and-bound setting

Contents

4.1 Introduction

Exploration strategies have an impact on the overall performance of a branch-and-bound algorithm, as Section 3.4.3 (on page 37) showed. The performance here can be defined in two ways. Indeed, a branch-and-bound algorithm retrieves the global minimizer of the problem at hand when it terminates. More precisely, when no more nodes can be explored, the current estimate is proved to be the global minimizer. However, the branch-and-bound algorithm may retrieve this global minimizer while still having many nodes to explore. This leads to the use of a truncated branch-and-bound as a heuristic on the problem [Ben Mhenni, Bourguignon, and Ninin 2021]. Therefore, one may be interested either in the speed of a branch-and-bound algorithm run until termination (which finds and proves the global minimizer) or the quality of the solution retrieved by a truncated branch-and-bound algorithm (which hopefully finds, without any proof, the global minimizer). In this chapter, several exploration strategies are studied and compared in order to define the best choice when solving sparse optimization problems, for both use cases. To cast the different exploration strategies in an uniform way, a

parallel is made between an exploration strategy and a data structure containing nodes in Section 4.2. Building on this analogy, standard explorations strategies are expressed as a choice of data structures in Section 4.3 before designing new exploration strategies in Section 4.4. Finally, the performance of the different choices considered is assessed in Section 4.5 before concluding and drawing out some future work in Section 4.6.

4.2 Exploration strategies as data structures

Data structures and algorithms are both fundamental aspects of computer science. On the micro viewpoint, the need for different ways to organize information leads to the creation of trees, heaps, double-ended queues, stacks (Last In First Out), FIFOs (First In First Out). On the macro viewpoint, we can see database storage schemes such as Relational databases, triplet-store databases, Document databases, Graph databases as instances of the same need.

In the mathematical optimization field, we are especially focused in the algorithmic part, trying to get the best out of data which are matrices and vectors. Matrices and vectors are simple data structures: they are lists. As we generally do not store a lot of information, we are fine with variables and lists. However, when designing a branch-and-bound algorithm, we have to choose how to store the nodes we will create along the way. We may need to store hundreds of thousands of nodes at a given time, so a correct choice of data structure is needed.

What defines a correct choice of data structure? This would be a data structure suited to the tasks at hand. Some data structures are better suited for raw access, others for inserting data at arbitrary positions. Some are fast to add data, others are suited to delete data at an arbitrary position. The correct choice depends on the frequency of those operations. In our case, what are the requirements for this data structure? It must store (Algorithm 4 step 14, on page 34) and give (Algorithm 4 step 4, same page) nodes fast. Iterating over the whole data structure is rare, and we will never need to directly access a node (we do not need an indexation or a key for a given node). When we give a node (step 4), it is removed from the data structure. Arbitrary deletion will seldom happen. Several data structures meet these overall light requirements. Additionally, the scheduling of the nodes given by the `Pop` function (Algorithm 4 step 4) should match the order of the exploration strategy used. Consequently, this is the exploration strategy which will drive the choice of the data structure, such that the prescribed scheduling of nodes is implemented through operations with low complexity on the data structure.

Our proposition is to merge both concepts: one exploration strategy = one data structure. The behaviour of this exploration strategy (and data structure) is completely encoded in the `push` (step 14) and `pop` (step 4) functions of Algorithm 4. Consequently, in the branch-and-bound algorithm developed in this thesis, the search tree is never stored

entirely. Only leaf nodes of this search tree are stored. The exploration strategy, which is a particular walk in the tree, is then defined by the order in which leaf nodes are analyzed. This order is implemented through the choice of a given data structure.

4.3 Standard exploration strategies viewed as data structures

Three standard strategies were seen in Section 3.4.3 (page 37): Depth-first search, Breadth-first search, Best-first search.

Depth-first search completely unrolls a branch before unrolling another one. In other words, when a node is divided, one of its children is immediately explored. When the sub-tree given by this child is completely explored, the second child is explored. A stack, also called LIFO (*Last In First Out*), implements this behaviour with $O(1)$ operations. The behaviour of a stack can be sketched as a tower of bricks. The lower bricks cannot be touched, only the highest brick in the tower can be taken, or another brick can be put on top of the tower. Depth-first search is implemented by replacing bricks by nodes in the previous analogy: nodes are stored by putting them on top of the tower, and the next node to be explored is the one at the top of the tower.

Breadth-first search explores in priority nodes which are closest to the root. This means we do not go straight to newly created nodes, but rather explore the older ones. This behaviour is implemented with $O(1)$ operations on a FIFO (*First In First Out*). Supermarket racks are recharged using a FIFO method: new products (with far expiry date) are put at the rear of the rack, while older ones (with closer expiry date) are moved to the front of the rack.

Best-first search explores the node with the lowest lower bound first. This corresponds to a heap. A heap is a partially ordered data structure such that the element with the minimum value according to a given metric is put at the top of the heap (insertion is done in $O(\log(n))$). The `pop` function always takes the top element of the heap in $O(\log(n))$. Compared to a sorted list, insertions and deletions are faster, $O(\log(n))$ compared to $O(n \log(n))$ in contiguous memory for deletion. To get Best-first search, the metric should be the lower bound value. We will say this is a heap *sorted* on the lower bound, to emphasize the metric used (keep in mind the heap is only partially sorted, not fully sorted). Other metrics will lead to other exploration strategies. For example, the Limited Discrepancy Search (LDS) [Harvey et al. 1995] is a less common exploration strategy where we would like to get good upper bounds without the bias of the depth-first search, which treats differently branching choices according to the moment they were made. Indeed, to get good upper bounds, we should look in priority at nodes where the fewest "defavorable" branching happened. This is what is done in depth-first search when a branch is unrolled.

But when backtracking, the nodes explored in priority are those which are closer to the branch we just unrolled, regardless or whether these nodes are "defavorable" or not. The LDS strategy, on the other hand, explicitly explore the least "defavorable" nodes first. In our data structure framework, the LDS strategy corresponds to a heap sorted on the size of S_0: the first nodes to be explored are those with the least number of variables set to 0.

4.4 Propositions of tailored exploration strategies

We propose novel exploration strategies, tailored to the structure of Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$) (on page 37) structure. Indeed, the lower bound of a node is the minimum of Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$), which is the sum of the data-fitting least-squares term, and of the ℓ_1-norm term. Thus, a first idea is to see what happens when we focus only on one of these two terms. Consequently, as Best-first search is a heap sorted on the lower bound, we propose Least-square-first search, a heap sorted on the least-square (data fitting) term, and ℓ_1-first search, a heap sorted on the ℓ_1 norm term. These two new strategies will respectively favor nodes matching data (without looking at sparsity) and nodes with sparse solution on $\bar{S}$ components (without looking at data fidelity).

Additionally, we would like to get the benefit of the Depth-first search (stack) to quickly get good feasible solutions (good upper bounds $\overline{\text{ub}}$) and the ability of Best-first search (heap sorted on the lower bound) to efficiently increase lower bounds. In schematic terms, Depth-first search quickly *finds* the best solution, but takes time to *prove* its optimality: during most iterations, the global minimum of Problem ($\mathcal{P}^{M}_{2+0}$) (on page 36) is already found without being aware of it. On the other side, Best-first search is quite long to find the best solution, but quicker to prove its optimality. This general belief on the behaviour of Depth-first search and Best-first search is problem-agnostic. Testing that hypothesis on our actual problem is a contribution of this thesis, as well as the design of mixed strategies.

Mixed strategies (the second idea in this chapter) correspond to mixing several strategies together (see for example [Neveu et al. 2016] with the introduced `LBvUB` and `LBvXX`, randomly mixing exploration strategies). In our case, we will mix two strategies, and we will do that sequentially: we will start with a given strategy, and when a condition is met, we switch to a second strategy. Several conditions could be designed: we could use the number of $\overline{\text{ub}}$ update, the number of nodes after the last $\overline{\text{ub}}$ update, the percentage of pruned nodes, ... Here, we will use a basic condition, the number of nodes: after having created $\texttt{N}_{\texttt{switch}}$ nodes, we switch to the second strategy. Implementation-wise, we do this by using the data structure corresponding to the first strategy, and when the condition is met we `Pop` all the nodes from the first data structure and `Push` them to the second one.

If the general knowledge about Depth-first search and Best-first search is applicable to Problem ($\mathcal{P}^{M}_{2+0}$), then a mixed strategy starting with a stack and switching to a heap sorted

on the lower bound should get good performance, better than the two ones implemented separately hopefully. We will name that strategy in the data structure naming as stack then heap on lb. Moreover, we also test mixed strategies where we start with Depth-first search and then switch to Least-square-first (stack then heap on ls), and Depth-first search then ℓ_1-first (stack then heap on l1).

4.5 Numerical experiments and results

4.5.1 Experiment description

We will test all the tailored strategies exposed in Section 4.4 as well as stack (Depth-first search) and heap on lb (Best-first search). For the mixed strategies, we will use three values for the number of created nodes threshold $\texttt{N}_{\texttt{switch}}$: 10, 100, 1000. Finally, we test 13 strategies:

— stack (`stack`)

— heap on lb ($\texttt{heap}^{\texttt{lb}}$)

— heap on ls ($\texttt{heap}^{\texttt{LS}}$)

— heap on l1 ($\texttt{heap}^{\ell_1}$)

— stack then heap on lb:

  — with $\texttt{N}_{\texttt{switch}} = 20$ ($\texttt{stack}^{20}$ + $\texttt{heap}^{\texttt{lb}}$)

  — with $\texttt{N}_{\texttt{switch}} = 200$ ($\texttt{stack}^{200}$ + $\texttt{heap}^{\texttt{lb}}$)

  — with $\texttt{N}_{\texttt{switch}} = 2000$ ($\texttt{stack}^{2000}$ + $\texttt{heap}^{\texttt{lb}}$)

— stack then heap on ls:

  — with $\texttt{N}_{\texttt{switch}} = 20$ ($\texttt{stack}^{20}$ + $\texttt{heap}^{\texttt{LS}}$)

  — with $\texttt{N}_{\texttt{switch}} = 200$ ($\texttt{stack}^{200}$ + $\texttt{heap}^{\texttt{LS}}$)

  — with $\texttt{N}_{\texttt{switch}} = 2000$ ($\texttt{stack}^{2000}$ + $\texttt{heap}^{\texttt{LS}}$)

— stack then heap on l1:

  — with $\texttt{N}_{\texttt{switch}} = 20$ ($\texttt{stack}^{20}$ + $\texttt{heap}^{\ell_1}$)

  — with $\texttt{N}_{\texttt{switch}} = 200$ ($\texttt{stack}^{200}$ + $\texttt{heap}^{\ell_1}$)

  — with $\texttt{N}_{\texttt{switch}} = 2000$ ($\texttt{stack}^{2000}$ + $\texttt{heap}^{\ell_1}$)

To test the different exploration strategies, we will test them on the same datasets. This will be a synthetic one, inspired by [Bertsimas, King, et al. 2016], generated with the procedure detailed in Section 3.6 (on page 40). As a recall, the goal is to solve the ℓ_0 problem on 10 instances for each value of ρ.

We will consider $N = 500, Q = 100, \rho = 0.8$ (moderate correlation) and $N = 500, Q = 100, \rho = 0.92$ (strong correlation), and we will compare the different strategies against

two metrics: the number of created nodes to *find and prove* the global minimum, as well as the number of created nodes to *find* the global minimum, without the proof. The first metric is of interest when we want to get the global minimum at all times (we wish a guarantee of optimality), whereas the second metric is of interest to get a good solution quick, and stop prematurely (for example with an elapsed time bound) the execution of the branch-and-bound algorithm.

4.5.2 Results

We show the simulation results as performance profiles [Dolan et al. 2002]. Performance profiles were designed to give more subtle comparison than just comparing the average execution time of each strategy. The horizontal axis is a performance ratio, it measures how many times slower is, in the sense of how many times more nodes are created by, a given strategy compared to the best performing one. The vertical axis is an instance proportion (1 means 100% of instances). Consequently, a strategy which would be the best for every instances would show as an horizontal line vertically located at 1. Otherwise, it means this strategy is not the best one for *every* instance. If two curves are crossing, it means that one strategy is very good on a subset of instances but overall slower than the second strategy.

With this in mind, Figure 4.1 shows the performance of the different strategies on the first metric, namely the total number of created nodes in the branch-and-bound algorithm. For $\rho = 0.8$ (top), we can see that the $\texttt{heap}^{\texttt{lb}}$ (Best-first search) is the best for all instances, and `stack` (Depth-first search) is the worst. Note that the vertical axis is restricted to the instance proportion $[0.95, 1]$, and the performance ratio does not exceed 1.15: all strategies are close to one another. Mixed strategies are in between `stack` and their corresponding `heap`, the ones with $\texttt{N}_{\texttt{switch}} = 2000$ being closer to the `stack`, the ones with $\texttt{N}_{\texttt{switch}} = 20$ being closer to their corresponding `heap`. For $\rho = 0.92$ (bottom), the situation is more contrasted, and we can actually see all the different strategies clearly. Here again, $\texttt{heap}^{\texttt{lb}}$ is the best performing strategy. Interestingly, the poorest performing one is not the `stack` but an ℓ_1 based strategy. Actually, the ℓ_1 strategies (blue) are all on the bottom of the figure (worst strategies). Least-square based strategies (red) are not as good as their counterparts using the full lower bound instead of just the data-fitting term, but they are still better than `stack`: relying only on the least-square term to schedule node division is an acceptable choice (though not the best) when it comes to solving the problem with an optimality proof.

Figure 4.2 shows the performance of the different strategies on the second metric, namely the number of nodes required to find the optimal solution, without knowing (thus proving) it yet. For $\rho = 0.8$ (top) we can clearly see that $\texttt{heap}^{\texttt{lb}}$ is by far the worst strategy. There is no clear winner on every instance, but overall $\texttt{heap}^{\texttt{LS}}$ (Least-square

first) is the best strategy for quickly finding the global minimum without proving it. For $\rho = 0.92$, there is more contrast (similarly to the situation in Figure 4.1), though the main conclusions are the same: `heap`$^{\texttt{lb}}$ is very bad, `heap`$^{\texttt{LS}}$ is the best. Interestingly, the different `stack + heap`$^{\texttt{lb}}$ mixed strategies are way better than `heap`$^{\texttt{lb}}$ itself, but also than the `stack` strategy: there is some kind of a "best of both worlds" effect. The different `stack + heap`$^{\texttt{LS}}$ are better than the different `stack + heap`$^{\texttt{lb}}$, and `stack + heap`$^{\ell_1}$ strategies are worse than `stack`. For this metric, `heap`$^{\texttt{LS}}$ is the best performing strategy, and mixed strategies using either `heap`$^{\texttt{LS}}$ or `heap`$^{\texttt{lb}}$ are reasonable choices (though not the best possible).

We can draw some conclusions from these results. First, for Problem ($\mathcal{P}^M_{2+0}$), Best-first search is the quickest to solve the problem to optimality with proof, whereas it is very bad if we are interested in quickly finding the optimal solution without proving it (one use case here being a truncated branch-and-bound). The difference in terms of performance is especially visible when problems are hard: if the matrix A has very correlated columns, the choice of the exploration strategy has a visible impact. When A has weakly correlated columns, all strategies tend to perform similarly. For quickly finding the best solution, `heap`$^{\texttt{LS}}$, one contribution of this thesis, which explores in priority nodes with lower bounds having a small least-squares term, is the best suited one. If one looks to fix an exploration strategy without knowing in advance their goal (either quickly finding or quickly proving the optimal solution), one of `stack + heap`$^{\texttt{lb}}$, `heap`$^{\texttt{LS}}$ and one of `stack + heap`$^{\texttt{LS}}$ are reasonable choices, which highlights that mixed strategies can play a role of a by-default setting.

4.6 Discussion

Conclusion In this chapter, contributions were developed to accelerate the branch-and-bound algorithm, both for finding the optimal solution as well as proving it, through the design of exploration strategies. To this end:

- a parallel between the exploration strategy of a branch-and-bound algorithm and data structures containing nodes was detailed;
- standard and new exploration strategies were designed for the branch-and-bound algorithm of this thesis;
- the performance of the different exploration strategies was assessed through numerical experiments, with `heap`$^{\texttt{lb}}$ being the best strategy for finding and proving the optimal solution, whereas `heap`$^{\texttt{LS}}$ is the best strategy for quickly finding the optimal solution.

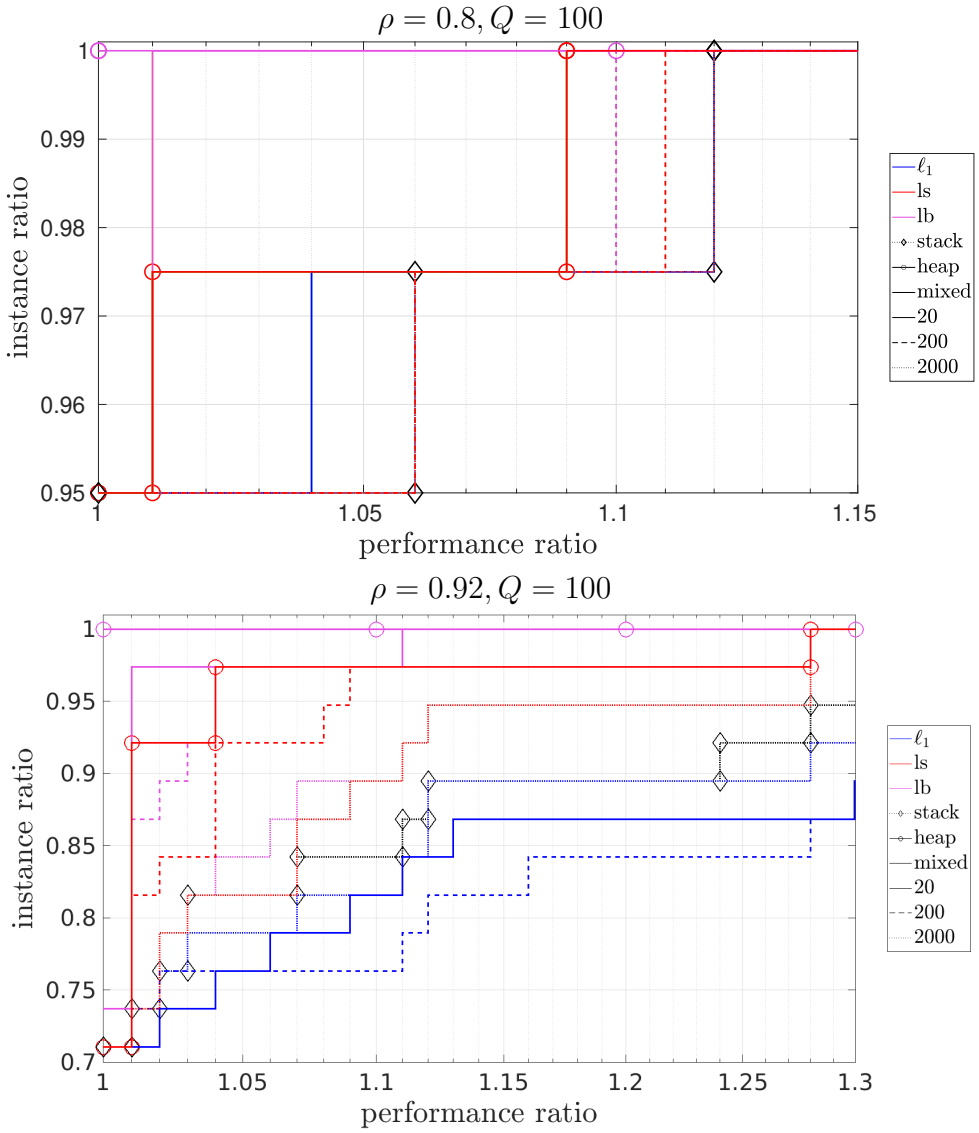


Figure 4.1 – Performance profiles comparing the number of created nodes for different exploration strategies, for moderately difficult problems ($\rho = 0.8$, top) and highly difficult problems ($\rho = 0.92$, bottom). Exploration is run until the entire search tree has been explored or the computation time exceeds one hour, in the latter case the instance is removed from the comparison. The black dotted line with diamonds represents the `stack` implementation, and the full line with circles represents the `heap` implementation, sorted on the lower bound (magenta), its least-squares term (red), or its ℓ_1-norm term (blue). Mixed strategies $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+ heap` use the same color code for the heap sorting, and are represented with full lines for $\texttt{N}_{\texttt{switch}} = 20$, dashed lines for $\texttt{N}_{\texttt{switch}} = 200$, and dotted lines for $\texttt{N}_{\texttt{switch}} = 2000$.

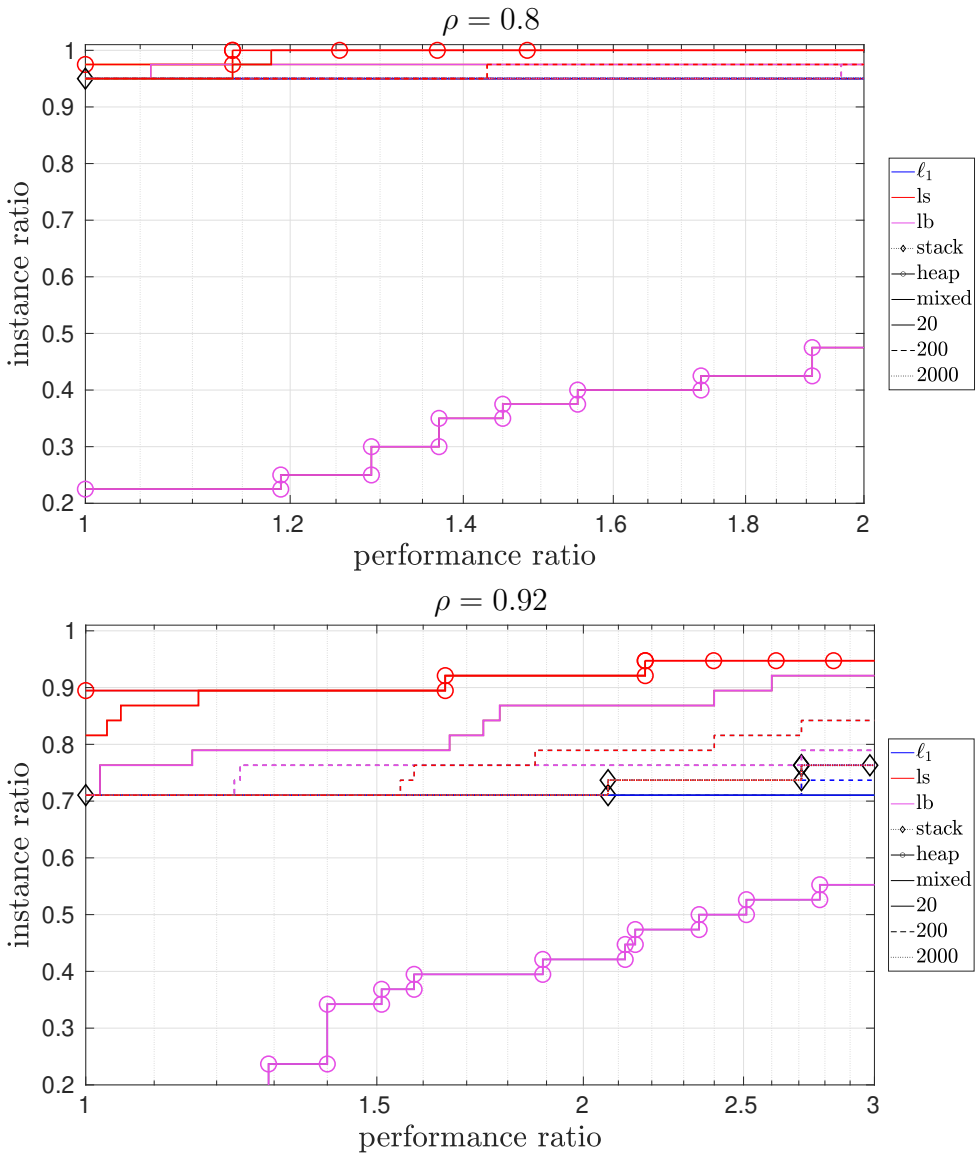


Figure 4.2 – Performance profiles comparing the number of created nodes required **to find the optimal solution without proving its optimality**, for different exploration strategies, for moderately difficult problems ($\rho = 0.8$, top) and highly difficult problems ($\rho = 0.92$, bottom). The black dotted line with diamonds represents the `stack` implementation, and the full line with circles represents the `heap` implementation, sorted on the lower bound (magenta), its least-squares term (red), or its ℓ_1-norm term (blue). Mixed strategies $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+ heap` use the same color code for the heap sorting, and are represented with full lines for $\texttt{N}_{\texttt{switch}} = 20$, dashed lines for $\texttt{N}_{\texttt{switch}} = 200$, and dotted lines for $\texttt{N}_{\texttt{switch}} = 2000$.

Perspectives Many things could be done and improved in this exploration strategy study, mainly around the mixed strategies. First, the switching condition $\texttt{N}_{\texttt{switch}}$ is very crude, and we could design smarter switching conditions, some of those being already written in Section 4.4. Second, starting with the `stack` strategy is questionable. The intent of this choice is to quickly get a good upper bound $\overline{\text{ub}}$, but LDS or other strategies not mentioned here may be better suited for that purpose. Third, the choice of starting by a good upper bound decreasing strategy and then switching to a heap is questionable. Maybe it's better to start exploring nodes fitting to data and then seek for sparse nodes first? We would start with $\texttt{heap}^{\text{LS}}$ and then switch to $\texttt{heap}^{\ell_1}$ in that case. Fourth, the choice of sequential mixed strategy, where only one exploration strategy is used at a given time, has not been discussed, but it is not the only way to mix strategies. Some authors [Neveu et al. 2016] propose to maintain several data structures in parallel and pick the next node to be divided by calling `Pop` on a randomly chosen data structure (without forgetting to maintain the consistency between all the data structures by removing that node from all of them).

Finally, when looking at how fast we find the best solution without proving it, *anytime* strategies [Luo et al. 2021] should be considered. These strategies are designed for the case of a truncated branch-and-bound, and as such they look for both favorable *and* diverse nodes, trying to somehow correctly sample the branch-and-bound tree. Testing these anytime strategies against ours in their ability to quickly find the global minimum would be a nice addition to these contributions.

Chapter 5

Accelerate lower bound estimation in the branch-and-bound

Contents

5.1 Motivation

As a recall (on page 37), lower bounds are computed by solving Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$):

$$\mathrm{lb}^{\mathbf{N}} = \min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|S_1| + \tfrac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_1 \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M, \boldsymbol{x}_{S_0} = 0, \ (\mathcal{P}^{\mathbf{N}}_{2+1})$$

which can be seen as the LASSO Problem ($\mathcal{P}_{2+1}$) (on page 25) with an additional box constraint, and a ℓ_1-norm term applied on a subset of variables only. As pointed out in Section 3.4.2 (on page 36), in our experiments, computing the lower bounds at each node evaluation takes approximately 90% of the total running time of the branch-and-bound. Consequently, as computing lower bounds is a performance bottleneck, we are aiming to compute them faster. Two main ideas have been explored in this thesis, with corresponding techniques for each idea. The first one is to accelerate *the goal* of lower bounds. Indeed, lower bounds serve two different purposes: check if a node should be pruned, and if not, use the precise lower bound value (and related minimizer) to define how to divide the node (additionally, some exploration strategies also use the lower bound value). The first use of lower bounds (prune nodes) does not require the exact computation of the lower bound. Section 5.2 investigates a technique to accelerate this pruning decision through the computation of approximations of the lower bound. The second idea is to ease the job of the optimization algorithm computing the lower bounds. To this end, we will investigate how the dual problem can be used to prematurely stop optimization algorithms, as detailed in Section 5.3. Finally, screening methods, which are reducing the problem dimension dynamically during the iterations of the optimization algorithm, are investigated in Section 5.4, before giving some concluding remarks and perspectives in Section 5.5.

5.2 Early node pruning through duality

As mentioned earlier when detailing Algorithm 4 (on page 34), one goal of lower bounds, obtained by solving Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$) at each node, is to prune sub-optimal nodes: if a node's lower bound $\text{lb}^{\mathbf{N}}$ is greater than $\overline{\text{ub}}$, then the node is pruned. Notice that we do not need the exact value of $\text{lb}^{\mathbf{N}}$, we just need to know whether we have $\text{lb}^{\mathbf{N}} \geq \overline{\text{ub}}$ or not. Therefore, the idea of what we call *early pruning* is to build a function D which under-estimates the value of the lower bound, that is the minimum of Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$): $\forall \boldsymbol{w} \in \mathbb{R}^N, D(\boldsymbol{w}) \leq \text{lb}^{\mathbf{N}}$. If at a given $\boldsymbol{w}$, we have $D(\boldsymbol{w}) \geq \overline{\text{ub}}$, then we can prune the node, since the relation $\text{lb}^{\mathbf{N}} \geq D(\boldsymbol{w}) \geq \overline{\text{ub}}$ holds. This case is illustrated in Figure 5.1 (left, node $\mathbf{N}^{\text{pruned}}$).

We can construct such a decision function $D(\boldsymbol{x})$ using the dual problem of ($\mathcal{P}^{\mathbf{N}}_{2+1}$). After describing the mathematical expression of the dual in Section 5.2.1 and the related literature on duality based pruning in Section 5.2.2, we will describe and investigate empirical performance of some optimization algorithms to this respect in Section 5.2.3 and finally investigate the performance of the early pruning technique itself in Section 5.2.4

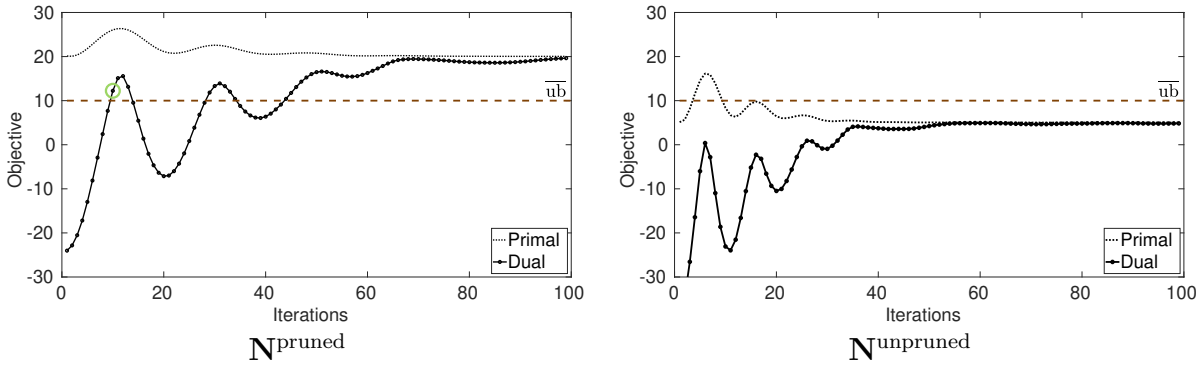


Figure 5.1 – Early pruning illustration: primal and dual iterates for the optimization of $(\mathcal{P}^{\mathbf{N}}_{2+1})$ (dashed and full lines, respectively), and the best known upper bound (dotted horizontal line). Left: pruning of the node is achieved after convergence of the primal descent algorithm minimizing $(\mathcal{P}^{\mathbf{N}}_{2+1})$ (here in 100 iterations), but the dual value after 10 iterations informs us that the node can be pruned (green circle). Right: in this case, the lower bound $\text{lb}^{\mathbf{N}}$ is too low and the node cannot be pruned.

5.2.1 Dual problem at each node

We can use some standard convex analysis properties to build a suited function $D(\boldsymbol{x})$. Indeed Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$ is convex and can be written as:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^Q} \quad & P(\boldsymbol{x}) := f(\mathbf{A}\boldsymbol{x}) + g(\boldsymbol{x}), \\ \text{with } & f(\boldsymbol{u}) := \tfrac{1}{2}\|\boldsymbol{y}-\boldsymbol{u}\|_2^2 \\ \text{and } & g(\boldsymbol{x}) := \mu|S_1| + \tfrac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_1 + I_{[-M,M]^Q}(\boldsymbol{x}) + I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0}), \end{aligned} \tag{5.1}$$

where $I_{\mathcal{C}}(\boldsymbol{x})$ is the indicator function equal to 0 if $\boldsymbol{x}\in\mathcal{C}$ and $+\infty$ otherwise. Let $\phi^*(\boldsymbol{w}) := \sup_{\boldsymbol{x}} \boldsymbol{w}^T\boldsymbol{x} - \phi(\boldsymbol{x})$ denote the Fenchel conjugate of any convex function ϕ. We use the Fenchel-Rockafellar theorem ([Rockafellar 1970], theorem 31.2) to get the dual problem associated with Problem (5.1), which reads:

$$\max_{\boldsymbol{w}\in\mathbb{R}^N} \quad D(\boldsymbol{w}) := -f^*(\boldsymbol{w}) - g^*(-\mathbf{A}^T\boldsymbol{w}). \tag{5.2}$$

For the detailed expression of this dual problem, given in Proposition 5.2.2, Lemma 5.2.1 will be used.

Lemma 5.2.1. *Let $h(x_i) := \lambda|x_i| + I_{[-M,M]}(x_i)$ where $x_i\in\mathbb{R}, \lambda>0, M>0$. The Fenchel conjugate of h is:*

$$\forall u_i\in\mathbb{R}, h^*(u_i) := M[|u_i|-\lambda]_+ = M\max(0,|u_i|-\lambda) \tag{5.3}$$

Proof. From the definition of the Fenchel conjugate, we have:

$$\begin{aligned} h^*(u_i) &= \sup_{x_i \in \mathbb{R}} \Big(u_i x_i - \lambda |x_i| - I_{[-M,M]}(x_i) \Big) = \sup_{-M \leq x_i \leq M} (u_i x_i - \lambda |x_i|) \\ &= \sup_{0 \leq |x_i| \leq M} (|u_i||x_i| - \lambda |x_i|) = \sup_{0 \leq |x_i| \leq M} |x_i|(|u_i| - \lambda) \\ &= M[|u_i| - \lambda]_+ \end{aligned}$$

□

Proposition 5.2.2. *The dual problem of Problem* $(\mathcal{P}^{\mathbf{N}}_{2+1})$ *reads:*

$$\max_{\boldsymbol{w} \in \mathbb{R}^N} \tfrac{1}{2}(\|\boldsymbol{y}\|_2^2 - \|\boldsymbol{w} + \boldsymbol{y}\|_2^2) - M \left(\sum_{i \in \bar{S}} \max(0, |\boldsymbol{a}_i^T \boldsymbol{w}| - \tfrac{\mu}{M}) + \|\mathbf{A}_{S_1}^T \boldsymbol{w}\|_1 \right) + \mu |S_1|. \quad (\mathcal{D}^{\mathbf{N}}_{2+1})$$

The objective function of this problem is $D(\boldsymbol{w})$*.*

Proof. From the dual problem expression (5.2), It comes that:

$$f^*(\boldsymbol{w}) = \tfrac{1}{2}(\|\boldsymbol{w} + \boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2) \tag{5.4}$$

and using Lemma 5.2.1 for the components in $\bar{S}$ (and in S_1) the Fenchel conjugate of g can be obtained by:

$$\begin{aligned} g^*(\boldsymbol{u}) &= \sup_{\boldsymbol{x}} \underbrace{\Big(\boldsymbol{u}^T \boldsymbol{x} - \mu |S_1| - \tfrac{\mu}{M} \|\boldsymbol{x}_{\bar{S}}\|_1 - I_{[-M,M]^Q}(\boldsymbol{x}) - I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0}) \Big)}_{\text{separable}}, \\ &= \sum_{i \in \bar{S}} \sup_{-M \leq x_i \leq M} (u_i x_i - \tfrac{\mu}{M} |x_i|) + \sum_{i \in S_1} \sup_{-M \leq x_i \leq M} u_i x_i \\ &\quad + \sum_{i \in S_0} \sup_{-M \leq x_i \leq M} \underbrace{(u_i x_i - I_{\{0\}}(x_i))}_{=0} - \mu |S_1|, \\ &= M \left(\sum_{i \in \bar{S}} [|u_i| - \tfrac{\mu}{M}]_+ + \sum_{i \in S_1} |u_i| \right) - \mu |S_1|. \end{aligned} \tag{5.5}$$

□

Let us remark that the dual problem is not constrained, and its objective function is analytical and simple to evaluate. Weak duality states that $P(\boldsymbol{x}) \geq D(\boldsymbol{w}), \forall (\boldsymbol{x}, \boldsymbol{w}) \in \mathbb{R}^Q \times \mathbb{R}^N$. Weak duality ensures that the dual objective $D(\boldsymbol{w})$ is a valid under-estimation, because it means in particular that $\text{lb}^{\mathbf{N}} = P(\boldsymbol{x}^\star) \geq D(\boldsymbol{w}), \forall \boldsymbol{w} \in \mathbb{R}^N$. In our case, strong duality applies too [1], which means that at optimality we have $P(\boldsymbol{x}^\star) = D(\boldsymbol{w}^\star)$, with

1. Slater's constraint qualifications trivially hold because the objective function is convex, and there exists a strictly feasible point, we can take for example $\mathbf{0}$ (it is strictly feasible inside all the box constraints).

$\boldsymbol{x}^\star = \arg\min_{\boldsymbol{x}\in\mathbb{R}^Q} P(\boldsymbol{x})$ and $\boldsymbol{w}^\star = \arg\max_{\boldsymbol{w}\in\mathbb{R}^N} D(\boldsymbol{w})$. Strong duality is not mandatory to build a valid under-estimator, but it is convenient, in the sense that if $D(\boldsymbol{w})$ never prunes the node, then the node should not be pruned: there is no undecidable zone where the node should be pruned but $D(\boldsymbol{w})$ cannot tell us to.

An illustration of the intended behaviour is sketched in Figure 5.1, where a node is pruned as soon as $D(\boldsymbol{w}) \geq \overline{\text{ub}}$ for some generated dual point $\boldsymbol{w}$. If such a situation occurs at many nodes during the branch-and-bound algorithm iterations, the computation time is expected to decrease.

In practice, we still need to define which dual points $\boldsymbol{w}$ will be used for evaluating $D(\boldsymbol{w})$. If we use a primal-dual algorithm, we can take the generated dual points. However, we will also use some algorithms which are only generating primal iterates. It turns out that standard convex analysis arguments give a starting basis. Indeed, the first-order Kuhn-Tucker optimality conditions read [Rockafellar 1970]:

$$\boldsymbol{w}^\star \in \partial f(\mathbf{A}\boldsymbol{x}^\star) \tag{5.6a}$$

$$-\mathbf{A}^T\boldsymbol{w}^\star \in \partial g(\boldsymbol{x}^\star) \tag{5.6b}$$

$$\mathbf{A}\boldsymbol{x}^\star \in \partial f^*(\boldsymbol{w}^\star) \tag{5.6c}$$

$$\boldsymbol{x}^\star \in \partial g^*(-\mathbf{A}^T\boldsymbol{w}^\star) \tag{5.6d}$$

where ∂f denotes the subdifferential of function f, defined as [Rockafellar 1970]:

$$\partial f(\boldsymbol{x}) := \{\boldsymbol{u} \in \mathbb{R}^N | \, \forall \boldsymbol{y} \in \mathbb{R}^N,\, f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{u}^T(\boldsymbol{y} - \boldsymbol{x})\}.$$

In our case, condition (5.6a) means we have $\boldsymbol{w}^\star = \mathbf{A}\boldsymbol{x}^\star - \boldsymbol{y}$. We will extend this relation the same way as in [Bonnefoy et al. 2014]: we link a primal iterate $\boldsymbol{x}^k$ with a dual iterate $\boldsymbol{w}^k$ by $\boldsymbol{w}^k := \mathbf{A}\boldsymbol{x}^k - \boldsymbol{y}$. This relationship, which is arbitrary, has two advantages. First, if the optimization algorithm converges to $\boldsymbol{x}^\star$, the dual iterates will converge to $\boldsymbol{w}^\star$. Note that we do not have *any* guarantee about the dynamic of the dual convergence. In particular, if the optimization algorithm monotonically decreases the primal objective function, the dual objective values $D(\boldsymbol{w}^k)$ can be non-monotonically increasing. Second, a lot of standard algorithms detailed in Section 3.3.1 (on page 25)compute at each iteration the residual, defined as $\boldsymbol{r}^k := \boldsymbol{y} - \mathbf{A}\boldsymbol{x}^k = -\boldsymbol{w}^k$, thus the point $\boldsymbol{w}^k$ comes without any additional cost.

5.2.2 Link with literature

In the literature related to exactly solving ℓ_0 problems, the dual expression of the lower bound problems of a given branch-and-bound algorithm has been investigated in [Guyard et al. 2022; Hazimeh et al. 2021]. In [Hazimeh et al. 2021], the authors use a coordinate descent algorithm to inexactly (therefore quickly) solve their lower bound problems. As

such, they only have feasible points of the primal problem, whose objective function value is not a valid lower bound. Therefore, they use the objective function of Problem ($\mathcal{D}^{\mathbf{N}}_{2+1}$) to get a valid lower bound at the end of the coordinate descent algorithm iterations.

In [Guyard et al. 2022], the authors evaluate $D(\boldsymbol{w})$ on several dual iterates $\boldsymbol{w}$ generated as in our case. However, they use it to prune *children* of the current node instead of the node itself, leveraging the fact that the dual objective function $D(\boldsymbol{w})$ of a children node can be easily obtained from its parent. Their technique is a different trade-off: as they have to test several children (and therefore evaluate several times the dual objective function), it is more costly, but it can be more effective than our early pruning strategy.

5.2.3 Algorithms dueling for increasing the dual objective value

As pointed out in Section 5.2.1, as long as an optimization algorithm converging to the optimal primal point $\boldsymbol{x}^\star$ is used, the optimal dual point $\boldsymbol{w}^\star$ will finally be reached. So, $D(\boldsymbol{w})$ will globally increase, although not monotonically (remember the dual problem (5.2) is a maximization problem) from $D(\boldsymbol{w}^0)$ to $D(\boldsymbol{w}^\star)$. However, we do not control the intermediate behaviour.

Moreover, there are several families of algorithms we can use to minimize the lower bound problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$). To know which algorithm is better suited for early pruning, we empirically study how they increase $D(\boldsymbol{w})$ over time. Indeed, if we are able to increase quickly $D(\boldsymbol{w}^k)$, we may quickly meet the condition $D(\boldsymbol{w}^k) \geq \overline{\text{ub}}$ needed to prune the node.

We will consider the different ℓ_1-norm optimization algorithms as described in Section 3.3.1. However, such standard algorithms are designed without the box constraint and support separation included in Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$) and require some specific developments. For active-set and homotopy algorithms adaptations were already introduced in [Ben Mhenni 2020]. For proximal algorithms and coordinate descent ones, we need to adapt them to Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$).

Proximal algorithms Proximal algorithms use the proximity operator of the non-differentiable function(s) to decrease the objective function. Plenty of algorithms exist, and we will pick two standard algorithms in our comparison: the forward-backward algorithm [Daubechies et al. 2004] as a reference primal-only algorithm, and the Chambolle-Pock algorithm [Chambolle and Pock 2011] as a reference primal-dual algorithm. What we just need to do to adapt these algorithms to our case is to write the analytical solution of the proximity operator problem related to the non-differentiable part of Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$):

$$\boldsymbol{x}^{k+1} = \underset{\eta g}{\text{prox}}(\boldsymbol{x}^k) = \arg\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{x}^k - \boldsymbol{x}\|_2^2 + \eta(\|\boldsymbol{x}_{\bar{S}}\|_1 + I_{\{0\}}(\boldsymbol{x}_{S_0}) + I_{[-M,M]^Q}(\boldsymbol{x})) \qquad (5.7)$$

with $\eta \in \mathbb{R}$. The proximal operator of a sum of separable terms is the sum of the individual proximal operators. Also, the proximal operator of *only* an indicator function is the projection operator Π onto the corresponding set. Separating the terms in (5.7) we get:

$$\forall i \in S_0, x_i^{k+1} = 0, \tag{5.8a}$$
$$\forall i \in S_1, x_i^{k+1} = \Pi_{[-M,M]}(x_i^k), \tag{5.8b}$$
$$\forall i \in \bar{S}, x_i^{k+1} = \arg\min_{x_i} \tfrac{1}{2}(x_i^k - x_i)^2 + \eta(|x_i| + I_{[-M,M]}(x_i)). \tag{5.8c}$$

The first-order optimality condition for (5.8c) reads: $\forall i \in \bar{S}$,

$$0 \in \{-x_i^k + x_i^{k+1}\} + \eta \left(\begin{Bmatrix} 1 & \text{if } x_i^{k+1} > 0 \\ [-1,1] & \text{if } x_i^{k+1} = 0 \\ -1 & \text{if } x_i^{k+1} < 0 \end{Bmatrix} + \begin{Bmatrix} [0,+\infty[& \text{if } x_i^{k+1} = M \\ 0 & \text{if } x_i^{k+1} \in]-M, M[\\ ]-\infty, 0] & \text{if } x_i^{k+1} = -M \end{Bmatrix} \right). \tag{5.9}$$

Separating this condition for the different possible cases, we have:

$$\underline{\text{For } x_i^{k+1} = M:}$$
$$0 \in \{-x_i^k + M\} + \{\eta\} + [0,+\infty[\iff x_i^k - M - \eta \geq 0 \iff x_i^k \geq M + \eta.$$
$$\underline{\text{For } x_i^{k+1} \in]0, M[:}$$
$$0 \in \{-x_i^k + x_i^{k+1}\} + \{\eta\} \iff x_i^{k+1} = x_i^k - \eta \in]0, M[\implies x_i^k \in]\eta, M + \eta[.$$
$$\underline{\text{For } x_i^{k+1} = 0:}$$
$$0 \in \{-x_i^k + 0\} + [-\eta, \eta] \iff x_i^k \in [-\eta, \eta].$$
$$\underline{\text{For } x_i^{k+1} \in]-M, 0[:}$$
$$0 \in \{-x_i^k + x_i^{k+1}\} - \{\eta\} \iff x_i^{k+1} = x_i^k + \eta \in]-M, 0[$$
$$\implies x_i^k \in]-M - \eta, -\eta[.$$
$$\underline{\text{For } x_i^{k+1} = -M:}$$
$$0 \in \{-x_i^k - M\} - \{\eta\} +]-\infty, 0] \iff x_i^k + \eta + M \leq 0 \iff x_i^k \leq -M - \eta.$$

Consequently, as all the conditions on x_i^{k+1} are disjoint, and as there is no value of x_i^k common to several cases, we can derive the following rules:

$$\text{If } |x_i^k| \leq \eta, \text{ then } x_i^{k+1} = 0;$$
$$\text{If } |x_i^k| \in]\eta, M + \eta[, \text{ then } x_i^{k+1} = \text{sign}(x_i^k)(|\boldsymbol{x}^k| - \eta);$$
$$\text{If } |x_i^k| \geq M + \eta, \text{ then } x_i^{k+1} = \text{sign}(x_i^k) M$$

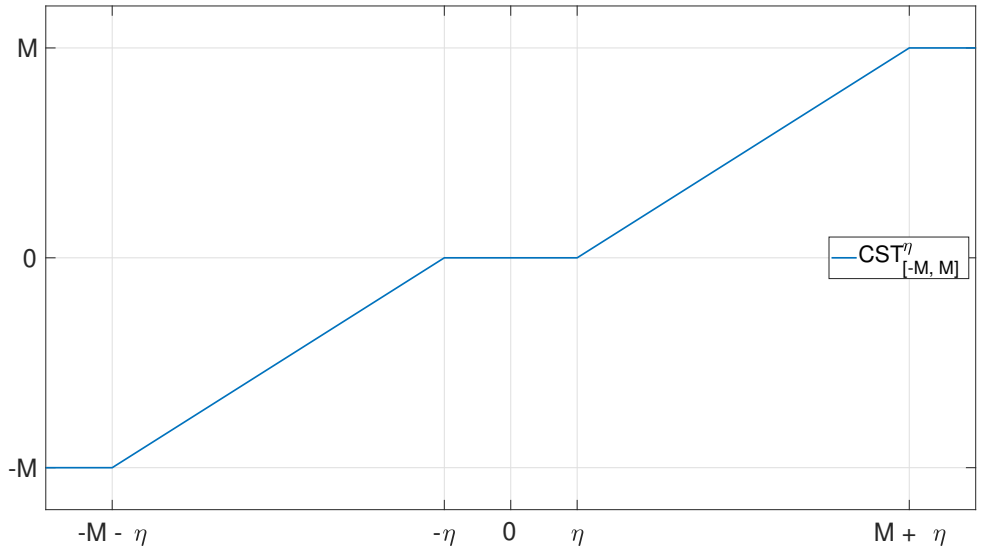


Figure 5.2 – Illustration of the capped soft-thresholding function in 1D

with $[\cdot]_+ := \max(\cdot, \mathbf{0})$, the max being applied component-wise. That is, written in a more compact form:

$$\forall i \in \bar{S}, x_i^{k+1} = \Pi_{|-M,M]}\{\text{sign}(x_i^k)[|x_i^k| - \eta]_+\} \tag{5.10}$$

This is actually just composing the projection on the box with the soft-thresholding operator [Chaux et al. 2009; Hazimeh et al. 2021], we call it the *capped soft-thresholding operator* $\text{CST}^{\eta}_{[-M,M]}$. Figure 5.2 gives an illustration of this operator on a scalar.

Algorithm 5 gives the Forward-Backward algorithm tailored to solve Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, and Algorithm 6 gives the Chambolle-Pock algorithm tailored to solve Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$.

Algorithm 5 Forward-Backward algorithm for Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$

procedure FB($\mathbf{A}, \boldsymbol{y}, \lambda, \tau, \boldsymbol{x}^0$)
  $k \leftarrow 0$
  while not convergence **do**
    $\boldsymbol{x}^{k+\frac{1}{2}} \leftarrow \boldsymbol{x}^k + \tau \mathbf{A}^T(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^k)$
    $\forall i \in S_0, x_i^{k+1} \leftarrow 0$
    $\forall i \in S_1, x_i^{k+1} \leftarrow \Pi_{|-M,M]}\left\{x_i^{k+\frac{1}{2}}\right\}$
    $\forall i \in \bar{S}, x_i^{k+1} \leftarrow \Pi_{|-M,M]}\left\{\text{sign}\left(x_i^{k+\frac{1}{2}}\right)\left[\left|x_i^{k+\frac{1}{2}}\right| - \eta\right]_+\right\}$
    $k \leftarrow k + 1$
  end while
end procedure

Algorithm 6 Chambolle-Pock algorithm for Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$)

procedure CP($\mathbf{A}, \boldsymbol{y}, \lambda, \tau, \sigma, \rho, \boldsymbol{x}^0, \boldsymbol{w}^0$)
  $k \leftarrow 0$
  while not convergence **do**
    $\boldsymbol{x}^{k+\frac{1}{2}} \leftarrow \boldsymbol{x}^k - \tau \mathbf{A}^T \bar{\boldsymbol{w}}^k$
    $\forall i \in S_0, x_i^{k+1} \leftarrow 0$
    $\forall i \in S_1, x_i^{k+1} \leftarrow \Pi_{[-M,M]} \left\{ x_i^{k+\frac{1}{2}} \right\}$
    $\forall i \in \bar{S}, x_i^{k+1} \leftarrow \Pi_{[-M,M]} \left\{ \text{sign}\left(x_i^{k+\frac{1}{2}} \right) \left[\left| x_i^{k+\frac{1}{2}} \right| - \eta \right]_+ \right\}$
    $\boldsymbol{w}^{k+1} \leftarrow \frac{\boldsymbol{w}^k + \sigma(\mathbf{A}\boldsymbol{x}^{k+1} - \boldsymbol{y})}{1+\sigma}$
    $\bar{\boldsymbol{w}}^{k+1} \leftarrow \boldsymbol{w}^{k+1} + \rho(\boldsymbol{w}^{k+1} - \boldsymbol{w}^k)$
    $k \leftarrow k + 1$
  end while
end procedure

Coordinate descent algorithms Coordinate descent algorithms solve scalar subproblems. For coordinates $i \in \bar{S}$, this scalar subproblem is:

$$x_i^\star = \arg\min_{x_i} P(x_i) = \tfrac{1}{2}\|\boldsymbol{e}_i - \boldsymbol{a}_i x_i\|_2^2 + \tfrac{\mu}{M}|x_i| + I_{[-M,M]}(x_i) \tag{5.11}$$

where $\boldsymbol{e}_i$ is a constant in this minimization problem: it corresponds to the residual generated by all the components except the x_i at hand, that is, $\boldsymbol{e}_i = \boldsymbol{y} - \sum_{j \neq i} \boldsymbol{a}_j x_j$. To get a convergent coordinate descent algorithm, we need the analytical solution of this problem. Problem (5.11) rewrites as:

$$\begin{aligned} x_i^\star &= \arg\min_{x_i} \tfrac{1}{2} \left(\boldsymbol{e}_i^T \boldsymbol{e}_i - 2(\boldsymbol{a}_i^T \boldsymbol{e}_i) x_i + \underbrace{(\boldsymbol{a}_i^T \boldsymbol{a}_i)}_{=1} x_i^2 \right) + \tfrac{\mu}{M}|x_i| + I_{[-M,M]}(x_i) \\ &= \arg\min_{x_i} \ \tfrac{1}{2}(\boldsymbol{a}_i^T e_i - x_i)^2 + \alpha_i + \tfrac{\mu}{M}|x_i| + I_{[-M,M]}(x_i). \end{aligned} \tag{5.12}$$

$\alpha_i \in \mathbb{R}$ being a constant. Problem (5.12) identifies with Problem (5.8c), using a similar method, its analytical solution reads:

$$\forall i \in \bar{S}, x_i^\star = \Pi_{[-M,M]}\{\underbrace{(\boldsymbol{a}_i^T \boldsymbol{a}_i)^{-1}}_{=1} \text{sign}(\boldsymbol{a}_i^T \boldsymbol{e}_i)[|\boldsymbol{a}_i^T \boldsymbol{e}_i| - \tfrac{\mu}{M}]_+\}. \tag{5.13}$$

The term $\text{sign}(\boldsymbol{a}_i^T \boldsymbol{e}_i)[|\boldsymbol{a}_i^T \boldsymbol{e}_i| - \frac{\mu}{M}]_+$ is once again the soft-thresholding operator [Daubechies et al. 2004].

For components $i \in S_1$, the problem reads:

$$x_i^\star = \arg\min_{x_i} P(x_i) := \tfrac{1}{2}\|\boldsymbol{e}_i - \boldsymbol{a}_i x_i\|_2^2 + I_{[-M,M]}(x_i) \tag{5.14}$$

from which it comes that:

$$\forall i \in S_1, x_i^\star = \Pi_{[-M,M]}\{\underbrace{(\boldsymbol{a}_i^T\boldsymbol{a}_i)^{-1}}_{=1}\boldsymbol{a}_i^T\boldsymbol{e}_i)\} \tag{5.15}$$

Algorithm 7 gives the coordinate descent algorithm tailored to solve Problem ($\mathcal{P}_{2+1}^{\mathbf{N}}$). The stopping criterion used is a vanishing duality gap.

Algorithm 7 Iterative coordinate descent algorithm for Problem ($\mathcal{P}_{2+1}^{\mathbf{N}}$)

procedure ICD($\mathbf{A}, \boldsymbol{y}, \lambda, \boldsymbol{x}^0$)
  $k \leftarrow 0$
  $\boldsymbol{x} \leftarrow \boldsymbol{x}^0$
  $\boldsymbol{e} \leftarrow \boldsymbol{y} - \mathbf{A}\boldsymbol{x}$
  while not convergence **do**
    for $i \in S_1$ **do**
      $\boldsymbol{e}_i \leftarrow \boldsymbol{y} - \mathbf{A}_{S_1\setminus\{i\}}\boldsymbol{x}_{S_1\setminus\{i\}} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}$
      $x_i \leftarrow \Pi_{[-M,M]}\{\boldsymbol{a}_i^T\boldsymbol{e}_i)\}$
    end for
    for $i \in \bar{S}$ **do**
      $\boldsymbol{e}_i \leftarrow \boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}\setminus\{i\}}\boldsymbol{x}_{\bar{S}\setminus\{i\}}$
      $x_i \leftarrow \Pi_{[-M,M]}\{\text{sign}(\boldsymbol{a}_i^T\boldsymbol{e}_i)[|\boldsymbol{a}_i^T\boldsymbol{e}_i| - \frac{\mu}{M}]_+\}$
    end for
    $k \leftarrow k + 1$
  end while
end procedure

Performance of the different algorithms considered for ascending dual values We will now compare the different algorithms on their ability to quickly improve the dual objective value (Problem ($\mathcal{D}_{2+1}^{\mathbf{N}}$), which is the dual of Problem ($\mathcal{P}_{2+1}^{\mathbf{N}}$)). Implementation is performed in Matlab. We compare Forward-Backward [Daubechies et al. 2004] (primal proximal algorithm), Chambolle-Pock [Chambolle and Pock 2011] (primal-dual proximal algorithm), coordinate descent [Friedman et al. 2010], active-set [Lee et al. 2006] and homotopy continuation [Osborne et al. 2000] algorithms. For the Chambolle-Pock algorithm, we use the dual points generated during the iterations. For all other algorithms except homotopy continuation, we use the formula $\boldsymbol{w}^k = \mathbf{A}\boldsymbol{x}^k - \boldsymbol{y}$ described previously, where $\boldsymbol{x}^k$ is a primal iterate. For the homotopy continuation, we introduce an additional rescaling step. Indeed, the algorithm solves a sequence of problems for different values of the penalty parameter λ, say λ^k. The different problems are solved exactly, which means we have access to some $\mathbf{A}\boldsymbol{x}^k - \boldsymbol{y}$ which is the optimal dual point for some $\lambda^k > \mu$. We propose to take benefit of the knowledge of λ^k to use a rescaled dual point: $\boldsymbol{w}^k = \frac{\mu}{\lambda^k}(\mathbf{A}\boldsymbol{x}^k - \boldsymbol{y})$.

We ran the different algorithms on a subset of instances described in Section 3.6 (on page 40). Figure 5.3 shows the typical performance at the root node, that is to say with

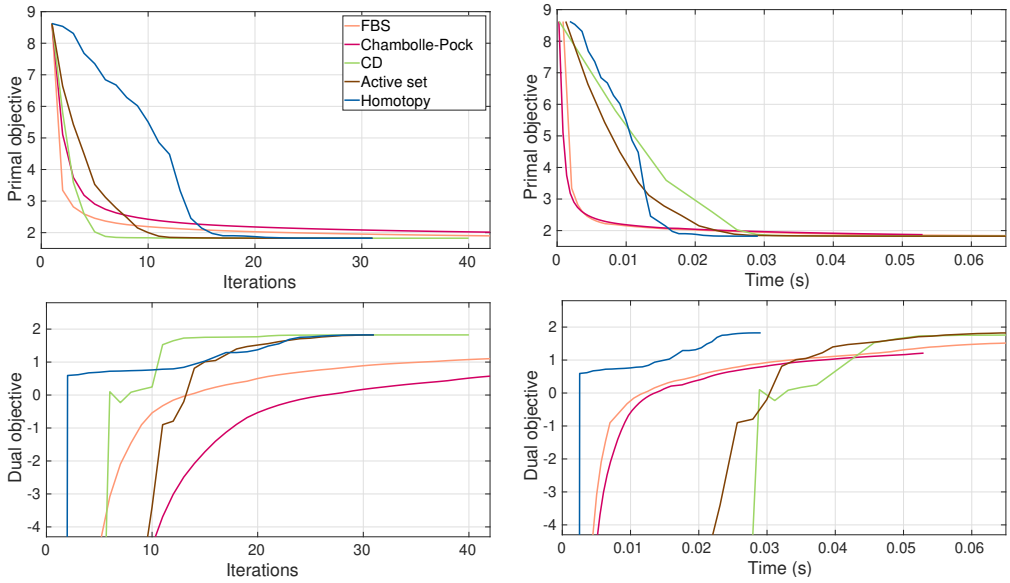


Figure 5.3 – Evolution of the primal (top) and dual (bottom) objective functions of a relaxed problem ($\rho = 0.8, Q = 100$) at the root node of the branch-and-bound algorithm, for different optimization strategies, as a function of the iteration number (left) and of the computation time (right).

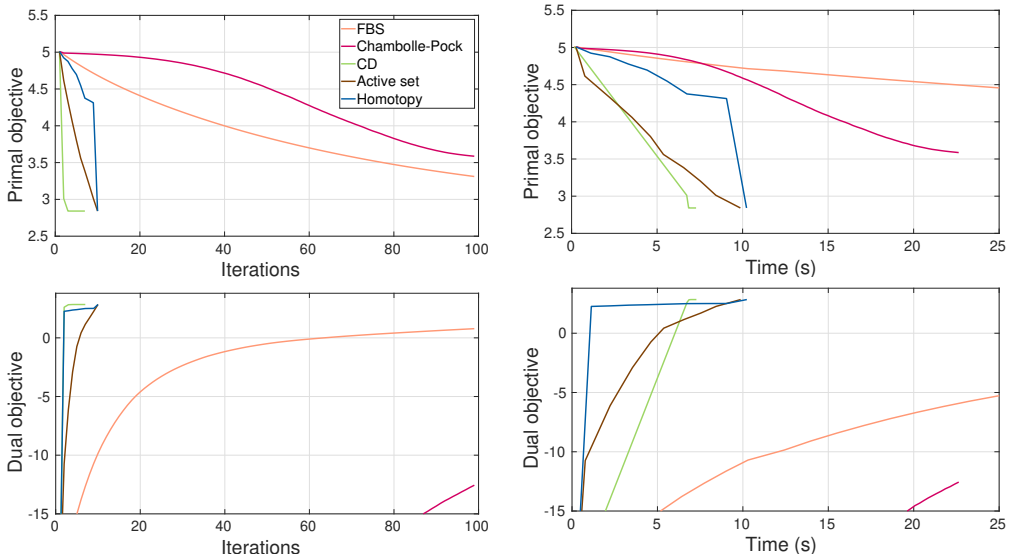


Figure 5.4 – Evolution of the primal (top) and dual (bottom) objective functions of a relaxation problem ($\rho = 0, Q = 100\,000$) at the root node of the branch-and-bound algorithm, for different optimization strategies, as a function of the iteration number (left) and of the computation time (right).

$S_1 = S_0 = \emptyset, \bar{S} = \{1, .., Q\}$, which corresponds to a ℓ_1-norm box-constrained problem, in a small ($Q = 100$), moderately correlated ($\rho = 0.8$) setting, while Figure 5.4 shows the typical performance for large ($Q = 100\,000$), decorrelated ($\rho = 0$) problems.

Looking at Figure 5.3, we can see that while the homotopy continuation does not decrease sharply the primal objective (top part) compared to other algorithms, it is the fastest to increase the dual (bottom part) in terms of elapsed time (bottom right), and

the quickest to get the dual to optimality in terms of number of iterations (bottom left). Coordinate descent is better, in terms of number of iterations (left half), both to quickly decrease the primal objective (top left) and to quickly increase the dual objective (bottom left), though it takes more time to yield convergence than the homotopy continuation. All other algorithms are not competitive as far as increasing the dual objective is concerned (bottom part).

Looking at Figure 5.4, for higher-dimensional, uncorrelated problems, the coordinate descent algorithm is now the best for reaching optimality, both in terms of number of iterations and time. The homotopy continuation is still very efficient for quickly increasing the dual objective in terms of elapsed time (bottom right), although it is slower than coordinate descent to achieve convergence.

Consequently, in the following sections of this chapter, we will use coordinate descent and homotopy continuation algorithms. Homotopy continuation will be our choice when tackling correlated problems, and coordinate descent will be our choice for (large size) decorrelated problems.

5.2.4 Numerical experiments and results

Datasets description The performance of early pruning is benchmarked on a synthetic dataset following the protocol presented in Section 3.6. A summary of the different parameters considered is given in Table 5.1. The exploration strategy used is depth-first search (which acts here as a baseline strategy).

Size	ρ	N	Q	K
Small	$\{0.8, 0.92\}$	500	100	9
Moderate	0.7	500	1 000	9
Large	0	1 000	100 000	9

Table 5.1 – Parameters used for the different synthetic instances generated for evaluating the performance of early pruning. 10 instances are generated for each combination of parameters.

Results Table 5.2 gives an aggregated view of the performance of the early pruning for the different considered datasets. The performance metric considered is the number of inner iterations of homotopy continuation ($\rho \in \{0.7, 0.8, 0.92\}$) or coordinate descent ($\rho = 0$) saved for all the nodes of all instances for a given problem category. We can clearly see that when the problem is simpler (small ρ in the synthetic instances), the performances are better: we are able to prune nodes earlier. In particular, we save more than three quarters of the iterations of the algorithm solving Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$), for synthetic instances with $\rho = 0$. This saving is of the order of one quarter for synthetic instances

Problem category		# iterations without early pruning	# iterations with early pruning	% Saved
$\rho = 0$	$Q = 100\,000$	48 141	10 552	78.1 %
$\rho = 0.7$	$Q = 1\,000$	182 792	138 543	24.2 %
$\rho = 0.8$	$Q = 100$	338 951	304 507	10.2 %
$\rho = 0.92$	$Q = 100$	68 585 732	66 555 082	2.97 %

Table 5.2 – Duality-based early pruning: number of saved iterations without and with early pruning (sum for all the nodes of 10 instances for each problem category).

with $\rho = 0.7$. For $\rho = 0.8$, 10% of the iterations are saved. When we increase the matrix correlation up to $\rho = 0.92$, this saving drops to less than 3%.

Interesting insights come when looking more deeply at these results, particularly when separating the nodes according to their number of selected variables, which is the cardinality of S_1. Figure 5.5 shows the same performance metric (the percentage of iterations saved for the algorithm solving Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$) with a finer-grained view. For each value of ρ, all instances are aggregated together. These instances are solved by creating several nodes, and these nodes are categorized according to their number of selected variables. For each category, we draw a boxplot aggregating the performance metric value for all nodes in that category. If we first focus on the small size instances ($Q = 100$, with $\rho \in \{0.8, 0.92\}$), we can see again that increasing the correlation level decreases the performance. Additionally, nodes with a higher number of selected variables in S_1 are pruned earlier. For $\rho = 0.8$, the median line (in red) hardly exceeds 0 for $|S_1| \in \{0, .., 5\}$, while for $|S_1| = 9$ it is at 50%, meaning that we save more than 50% of the iterations for half of the nodes with 9 selected variables. For $\rho = 0.92$, we can observe the same effect, though it is way fainter. We can also see that although the overall performance is very poor, some nodes with between 9 and 13 variables in S_1 get all their iterations saved (meaning the initial dual point is good enough to prune the node).

Let us remark additionally that Figure 5.5 also shows that problems with $\rho = 0.92$ are more difficult than for $\rho = 0.8$. Indeed, in both cases, the considered instances have a true support size of 9. For $\rho = 0.8$, no node with more than 10 selected variables is created, whereas for $\rho = 0.92$, nodes with up to 16 selected variables are created, showing that the algorithm has more difficulty in identifying the correct support size. In other words: it is more difficult to prune nodes with a bigger support than the true one.

Next, if we look at the problems of moderate difficulty ($\rho = 0.7$), we can see the same trend linking performance to the size of S_1. For $|S_1| = 9$ and $|S_1| = 10$ we see an "all-or-nothing" behaviour: we either save almost all the iterations, or no iteration at all. This behaviour may be a hint that we reached a point where it is very simple to prune nodes that should be pruned.

For large size problems ($\rho = 0$), the "all-of-nothing" effect is actually present for each category, meaning nodes which should be pruned are pruned very quickly when using the

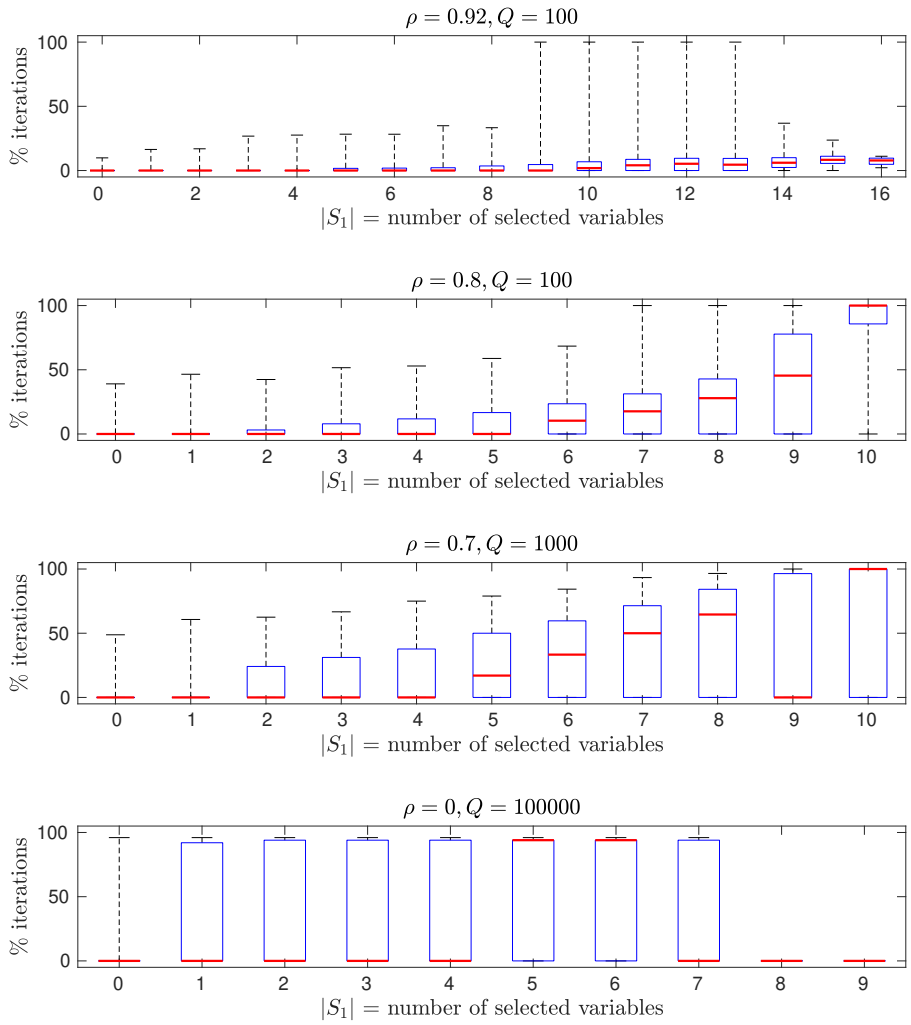


Figure 5.5 – Results for duality-based early pruning: ratio of saved sub-iterations as a function of the number of non-zero variables at the corresponding node, for four correlation levels in matrix $\mathbf{A}$ and problem sizes. The cardinality of the solution is $K = 9$. Each box shows the first and third quartiles on the fraction of saved iterations, the red dash indicates the median value, and black dashes indicate the extreme values.

dual. Moreover, there is only one node with $|S_1| = 8$ and one other with $|S_1| = 9$. This means we were able to prune all the "wrong" nodes before reaching the true cardinality of 9.

5.3 Tradeoff lower bound estimation accuracy versus estimation time

5.3.1 Principle

In this Section, we further leverage the dual objective to accelerate the lower bound algorithm in the case of a non-prunable node (that is, a node which must be divided).

Indeed, in the previous section, we use dual values to take quicker decisions to prune nodes. If a node must not be pruned (right part of Figure 5.1), we run the optimization algorithm until it reaches optimality. However, as the dual objective $D(\boldsymbol{w}^k)$ is a valid lower bound, we can also stop the algorithm before convergence, and take the value of $D(\boldsymbol{w}^k)$ as a valid lower bound. If we stop late, we won't save a huge number of iterations. If we stop too early, a quality problem happens. Interestingly, once we know that we won't prune a node, the exact value of the lower bound is only used for certain explorations strategies, so we could accept with a very crude approximation of the minimum of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$. However, the branching rule (3.17):

$$i = \arg\max_{i \in \bar{S}} |(\boldsymbol{x}^{\mathbf{N}}_{\text{lb}})_i| \tag{3.17}$$

uses the minimizer of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$.

If we stop at iteration k, we will use $D(\boldsymbol{w}^k)$ as a lower bound, and $\boldsymbol{x}^k$ as a solution. If we stop too early, $\boldsymbol{x}^k$ will be quite different from $\boldsymbol{x}^\star$ the minimizer of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, and we will take poor branching decisions. So there is a trade-off between accelerating the ℓ_1 problems and keeping good performance on the ℓ_0 problem.

5.3.2 Expression

We will use the duality gap as a stopping criterion to track how early we stop the algorithm. In other words, once the duality gap falls below a given threshold, we stop iterating and use the current $D(\boldsymbol{w}^k)$ as a lower bound.

Using an absolute threshold, like $P(\boldsymbol{x}^k) - D(\boldsymbol{w}^k) < 10^{-2}$ as a stopping criterion, is not really satisfactory. Indeed, if $P(\boldsymbol{x}^k) = 100$, reaching the stopping criterion means that $D(\boldsymbol{w}^k)$ is within 1% of the value of $P(\boldsymbol{x}^k)$, whereas if $P(\boldsymbol{x}^k) = 10\,000$, reaching the stopping criterion means that $D(\boldsymbol{w}^k)$ is within 1 ‰, or 0.01%, of $P(\boldsymbol{x}^k)$. Instead, we will favour a relative threshold, for a stopping criterion of the form of $P(\boldsymbol{x}^k) - D(\boldsymbol{w}^k) < \gamma P(\boldsymbol{x}^k)$, where γ is a relative gap to be met. In the case where $P(\boldsymbol{x}^k)$ is very close to zero, numerical issues may arise. To overcome this, we add a small absolute threshold to act as a safeguard. The definitive stopping criterion is then:

$$P(\boldsymbol{x}^k) - D(\boldsymbol{w}^k) < \gamma P(\boldsymbol{x}^k) + 10^{-8}. \tag{5.16}$$

In practice, the values of γ will typically lie in the interval $[10^{-8}, 10^{-1}]$.

5.3.3 Numerical experiments and results

In this section, we will investigate how different values of the parameter γ in (5.16) affect the performance of the branch-and-bound. We use the same datasets as in Sec-

tion 5.2.4. As a recall, the homotopy continuation algorithm is used for $\rho \in \{0.7, 0.8, 0.92\}$, while the coordinate descent algorithm is used for $\rho = 0$.

Figure 5.6 shows the performance of the technique for $\rho = 0.92$ (left) and $\rho = 0.8$ (right). Three metrics are considered: the total time required to solve the problem (top), the total number of nodes to solve the problem (middle), and the ratio between both, meaning the average computation time per node (bottom). Results for each instance (thigh colored lines) are plotted against these metrics for different values of γ, ranging from 10^{-1} (left of the axis) to 0 (right of the axis), and their average is plotted as a black dashed line. Looking at the solving time (top), we can see that while some values of γ are clearly not optimal ($\gamma \in \{10^{-1}, 10^{-2}\}$ for $\rho = 0.92$, $\gamma = 10^{-1}$ for $\rho = 0.8$), there is a range of values of γ that leads to satisfactory results. In particular, completely disabling the technique ($\gamma = 0$) is neither worse nor better than using it in the range $\gamma \in [10^{-8}, 10^{-3}]$. What is particularly interesting is that the curves are not just flat on the solving time metric, but also on the number of created nodes (middle) and the average time per node (bottom). The fact that the setting of γ has no significant impact on both the average computing time per node and the number of created nodes for the range $[0, 10^{-3}]$ means the number of iterations of the algorithm does not significantly change between two different values of γ within the range $[0, 10^{-3}]$. This means there are only few iterations of the algorithm where the duality gap lies in the interval given by $\gamma \in [10^{-8}, 10^{-3}]$.

Figure 5.7 shows the performance of the technique for $\rho = 0.7$ (left) and $\rho = 0$ (right). Looking at the solving time (top), there is a slight advantage for $\gamma = 10^{-2}$ for $\rho = 0.7$. For $\rho = 0$, there is no significant change in the range $\gamma \in [0, 10^{-2}]$, which can also be seen when looking at the total number of created nodes (middle) and the average time per node (bottom).

While the results of Figure 5.6 and Figure 5.7 show that allowing inexact convergence does not improve the overall efficiency, this technique will have more impact in the structured sparsity case (Chapter 9 on page 121).

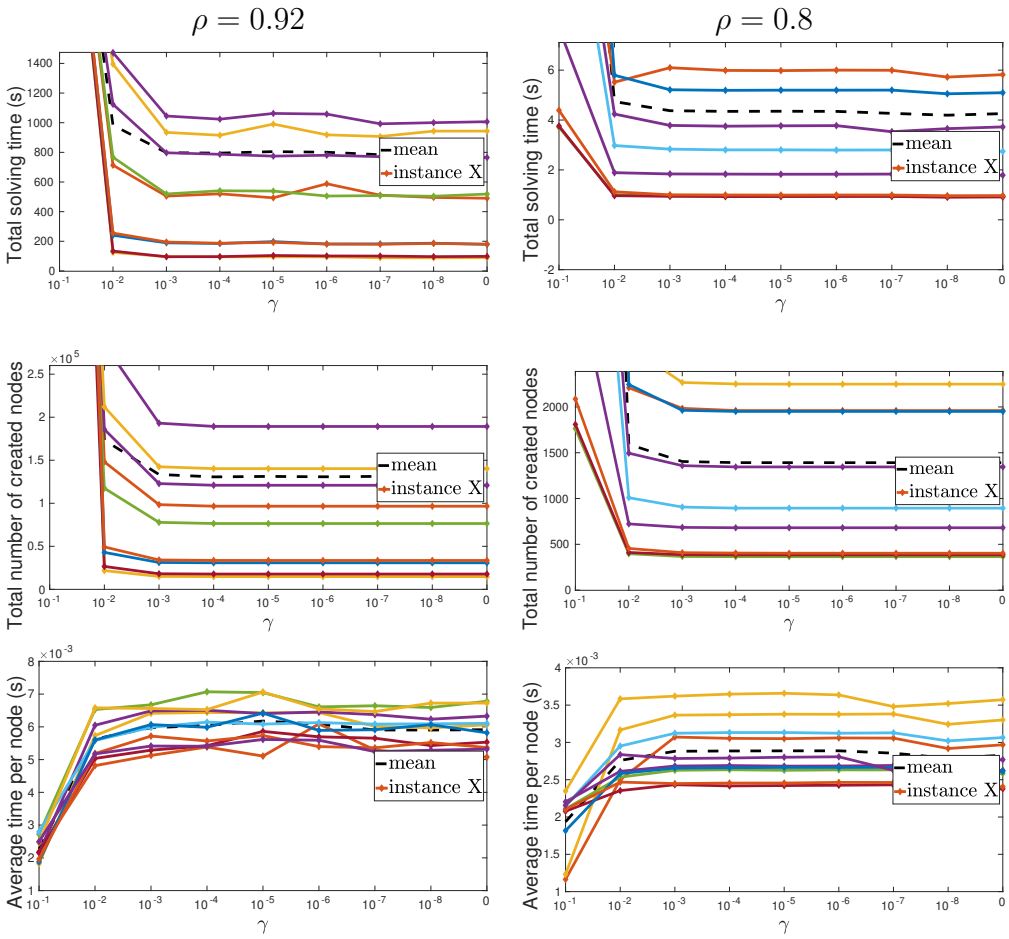


Figure 5.6 – Results of inexact convergence monitoring for small size problems ($\rho \in \{0.8, 0.92\}, Q = 100$), one color per instance, the dashed line being the mean. All curves are plotting the performance of values of γ (horizontal axis, in decreasing magnitude) against three different metric: the total time to solve the problem to optimality (top), the number of nodes created to solve the problem (middle), and the average time spent on each node (bottom). The homotopy continuation algorithm is used for both $\rho = 0.92$ and $\rho = 0.8$. The point $\gamma = 0$ corresponds to exact convergence.

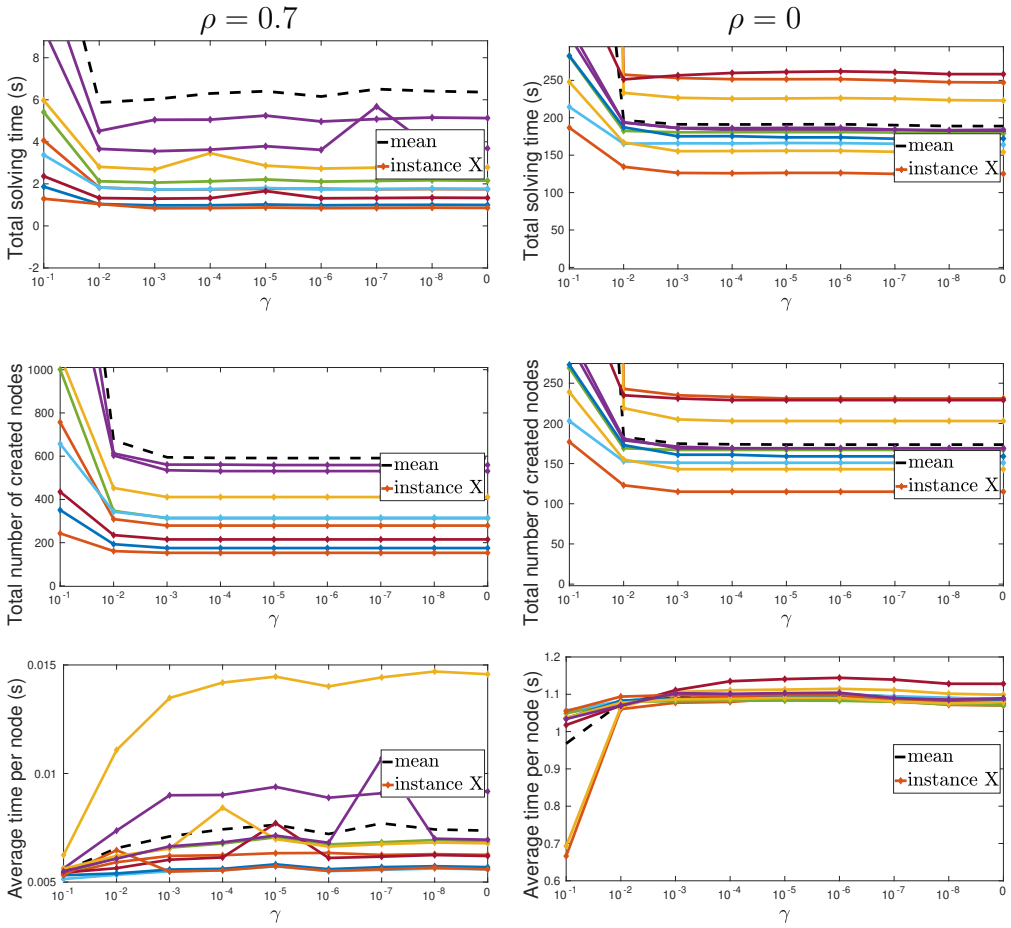


Figure 5.7 – Results of inexact convergence monitoring for moderate size ($\rho = 0.7, Q = 1\,000$) and large size ($\rho = 0, Q = 100\,000$) problems, one color per instance, the dashed line being the mean. All curves are plotting the performance of values of γ (horizontal axis, in decreasing magnitude) against three different metric: the total time to solve the problem to optimality (top), the number of nodes created to solve the problem (middle), and the average time spent on each node (bottom). The homotopy continuation algorithm is used for $\rho = 0.7$, while the coordinate descent algorithm is used for $\rho = 0$. The point $\gamma = 0$ corresponds to exact convergence.

5.4 Gap-Safe screening

5.4.1 Introduction

In the mathematical programming field, a screening method aims to find the optimal value of some variables before a given optimization algorithm ended. This way, the optimization algorithm keeps running with less variables. In the generic context of sparse optimization, a screening method will fix some variables to zero. The other variables remain to be estimated through a given optimization algorithm. In the case of Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$), a screening method will fix some variables to 0, $+M$ or $-M$.

Typology Screening methods [Bonnefoy et al. 2014; Dantas et al. 2019; El Ghaoui et al. 2010; Fercoq et al. 2015; Kuang et al. 2017; Liu et al. 2013; Ndiaye et al. 2017; Ren et al. 2017; Tibshirani et al. 2010; J. Wang et al. 2015; Y. Wang et al. 2013; Xiang and Ramadge 2012; Xiang, Y. Wang, et al. 2014; Xiang, Xu, et al. 2011; Xianli et al. 2018; Yoshida et al. 2019; Zeng et al. 2020; W. Zhang et al. 2018; Zimmert et al. 2015] are usually categorized along two dimensions: their safety guarantees, and their coupling degree with an optimization algorithm. For the safety part, we distinguish between *safe* and *strong* screening. A *safe* screening guarantees that the screened variables are actually set at their optimal value. Conversely, a *strong* screening does not provide this kind of guarantee, but can fix more variables. For the coupling degree part, we distinguish between *static*, *sequential* and *dynamic* screening. The community around the penalized LASSO ($\mathcal{P}_{2+1}$) problem contributed to popularize the screening methods, starting from the seminal paper [El Ghaoui et al. 2010]. In the context of the penalized LASSO ($\mathcal{P}_{2+1}$):

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1, \qquad (\mathcal{P}_{2+1})$$

static screening [El Ghaoui et al. 2010; Xiang and Ramadge 2012; Xiang, Y. Wang, et al. 2014; Xiang, Xu, et al. 2011] fixes some variables to 0 before optimization occurs, using just the data $\boldsymbol{y}$, the matrix $\mathbf{A}$ and the regularization parameter λ. After fixing variables, the optimization algorithm is run on the remaining variables (thus on a reduced problem). *Sequential* screening [Liu et al. 2013; J. Wang et al. 2015] was developed in the same context, where an instance is solved several times with different values of λ. In this case, sequential screening acts like static screening, in the sense that it runs before the optimization algorithm for a given value of λ. The difference lies in the quantities used to fix variables to zero. Instead of relying only on $\boldsymbol{y}, \mathbf{A}, \lambda$, sequential screening uses additionally the optimal solution got from optimizing the problem for a greater value of the regularization parameter $\lambda^0 > \lambda$. This allows sequential screening to achieve better performance, however it requires solving problems with $\lambda^0 > \lambda$ to optimality, which can be complex to do in practice [Xianli et al. 2018]. *Dynamic* screening methods [Bonnefoy

et al. 2014; Fercoq et al. 2015; Ndiaye et al. 2017; Raj et al. 2016] follow another path. They work by using a feasible dual point for the problem with the target λ. As their only requirement is a feasible dual point, we can generate several dual points and apply a dynamic screening several times, hoping to fix more variables to zero. In practice, this allows dynamic screening to be inserted *within* the optimization algorithm.

In our case, we would like to apply a screening method to Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, but we need to retrieve its minimum, otherwise we may not have a valid lower bound. Consequently, we are restricted to safe screening methods. Moreover, we need a screening which can be adapted to Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, which is not the standard penalized LASSO Problem $(\mathcal{P}_{2+1})$.

There are plenty of safe screening methods designed for the LASSO [Bonnefoy et al. 2014; Dantas et al. 2019; El Ghaoui et al. 2010; Fercoq et al. 2015; Liu et al. 2013; Ren et al. 2017; J. Wang et al. 2015; Xiang and Ramadge 2012; Xiang, Xu, et al. 2011; Xianli et al. 2018], some of which use the precise geometry of the dual problem [J. Wang et al. 2015; Xiang and Ramadge 2012; Xianli et al. 2018], making them difficult to adapt to even slight variations of the LASSO problem. Other approaches, like Gap-Safe screening [Fercoq et al. 2015; Ndiaye et al. 2017; Raj et al. 2016], use more generic analytical properties of the problem and are easier to adapt, therefore they we will used for Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$.

5.4.2 Screening rules for lower bound optimization problems

Screening rules are derived from the KKT optimality conditions, more precisely from condition (5.6b). From the expression of g in Problem (5.1), its sub-differential reads:

$$\boldsymbol{\nu} \in \partial g(\boldsymbol{x}) = \partial\left(\tfrac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_1\right) + \partial\left(I_{[-M,M]^Q}(\boldsymbol{x})\right).$$

As it is separable, its i-th component reads:

$$\begin{aligned} \forall i \in \bar{S},\ \nu_i &\in \partial\left(\tfrac{\mu}{M}|x_i|\right) + \partial\left(I_{[-M,M]}(x_i)\right) \\ \text{and } \forall i \in S_1,\ \nu_i &\in \partial\left(I_{[-M,M]}(x_i)\right), \end{aligned}$$

$$\text{with } \partial|x_i| = \begin{cases} \operatorname{sign}(x_i) & \text{if } x_i \neq 0 \\ [-1,1] & \text{if } x_i = 0 \end{cases} \text{ and } \partial I_{[-M,M]}(x_i) = \begin{cases} [0,+\infty[& \text{if } x_i = M \\ 0 & \text{if } |x_i| < M \\ ]-\infty,0] & \text{if } x_i = -M \end{cases}.$$

Therefore, we can derive the following screening rules:

Lemma 5.4.1. *Let $\boldsymbol{x}^\star$ the minimizer of the primal problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$ and $\boldsymbol{w}^\star$ the minimizer of its corresponding dual problem $(\mathcal{D}^{\mathbf{N}}_{2+1})$, we have the following screening rules for*

Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$)*:*

$$\forall i \in \bar{S}, \textit{if } |\boldsymbol{a}_i^T \boldsymbol{w}^\star| > \tfrac{\mu}{M}, \textit{ then } x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}^\star); \tag{5.17a}$$

$$\forall i \in \bar{S}, \textit{if } |\boldsymbol{a}_i^T \boldsymbol{w}^\star| < \tfrac{\mu}{M}, \textit{ then } x_i^\star = 0; \tag{5.17b}$$

$$\forall i \in S_1, \textit{if } \boldsymbol{a}_i^T \boldsymbol{w}^\star \neq 0, \textit{ then } x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}^\star). \tag{5.17c}$$

Proof. The optimality condition $-\mathbf{A}^T \boldsymbol{w}^\star \in \partial g(\boldsymbol{x}^\star)$ can be separated into the following cases:

$$\begin{cases} \forall i \in \bar{S} & \text{with} & x_i^\star = M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in [\frac{\mu}{M}, +\infty[\\ \forall i \in \bar{S} & \text{with} & x_i^\star \in]0, M[, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star = \frac{\mu}{M} \\ \forall i \in \bar{S} & \text{with} & x_i^\star = 0, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in [-\frac{\mu}{M}, \frac{\mu}{M}] \\ \forall i \in \bar{S} & \text{with} & x_i^\star \in]-M, 0[, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star = -\frac{\mu}{M} \\ \forall i \in \bar{S} & \text{with} & x_i^\star = -M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in]-\infty, -\frac{\mu}{M}] \\ \forall i \in S_1 & \text{with} & x_i^\star = M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in [0, +\infty[\\ \forall i \in S_1 & \text{with} & x_i^\star \in]-M, M[, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star = 0 \\ \forall i \in S_1 & \text{with} & x_i^\star = -M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in]-\infty, -0] \end{cases}. \tag{5.18}$$

From Equation (5.18), the rules of Lemma 5.4.1 follow. □

The rules of Lemma 5.4.1 are valid and safe for the optimal dual point $\boldsymbol{w}^\star$, meaning for example that if we have for some $i \in \bar{S}, |\boldsymbol{a}_i^T \boldsymbol{w}^\star| < \frac{\mu}{M}$, then we can set x_i to zero and this will be its optimal value ($x_i^\star$ will be zero at the end of the optimization procedure). However, to be able to do that, we must know $\boldsymbol{w}^\star$. In practice, finding $\boldsymbol{w}^\star$ may be as difficult as finding $\boldsymbol{x}^\star$, so the screening rules in Lemma 5.4.1 are not usable as-is.

Instead, in the same vein as [Bonnefoy et al. 2014; Ndiaye et al. 2017; Raj et al. 2016; J. Wang et al. 2015; Xiang, Xu, et al. 2011], we use what the screening literature calls a *safe region*, generally written $\mathcal{R}$. This safe region is built from a *feasible* dual point $\boldsymbol{w}$, and uses some regularity properties of the problem at hand to draw a region of the dual space which contains the optimal dual point $\boldsymbol{w}^\star$, in a guaranteed way for safe screening (or with good probability for strong screening). If every point in the region $\mathcal{R}$ obeys a given rule of Lemma 5.4.1, then we can fix the corresponding x_i to its optimal value. Testing over all points of $\mathcal{R}$ would be prohibitively expensive, so a simple region is considered, a sphere, and checking the validity of a given rule for each point of the sphere will be done through a single test.

Indeed, for every dual point $\boldsymbol{w}$, we have (using triangle inequality):

$$|\boldsymbol{a}_i^T \boldsymbol{w}^\star| = |\boldsymbol{a}_i^T \boldsymbol{w}^\star - \boldsymbol{a}_i^T \boldsymbol{w} + \boldsymbol{a}_i^T \boldsymbol{w}| \leq |\boldsymbol{a}_i^T \boldsymbol{w}^\star - \boldsymbol{a}_i^T \boldsymbol{w}| + |\boldsymbol{a}_i^T \boldsymbol{w}|. \tag{5.19}$$

As a recall, we assume that the columns of matrix $\mathbf{A}$ have unit ℓ_2 norm. Then, using the Cauchy-Schwarz inequality, we have:

$$|\boldsymbol{a}_i^T \boldsymbol{w}^\star - \boldsymbol{a}_i^T \boldsymbol{w}| \leq \|\boldsymbol{a}_i\|_2 \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2 = \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2,$$

so that $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| \leq |\boldsymbol{a}_i^T \boldsymbol{w}| + \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2$. This means that if we have for some $i \in \bar{S}$, $|\boldsymbol{a}_i^T \boldsymbol{w}| + \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2 < \frac{\mu}{M}$, then $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| < \frac{\mu}{M}$ and we can set $x_i^\star = 0$ from (5.17b). In practice, as we do not know $\|\boldsymbol{w}^\star - \boldsymbol{w}\|_2$, we will bound it by the radius r of a sphere centered on $\boldsymbol{w}$: $\mathcal{R} = \mathcal{B}(\boldsymbol{w}, r)$. This gives us a test usable in practice for rule (5.17b).

For rules (5.17a) and (5.17c), using reverse triangle inequality:

$$\begin{aligned} |\boldsymbol{a}_i^T \boldsymbol{w}^\star| &= |\boldsymbol{a}_i^T \boldsymbol{w} + \boldsymbol{a}_i^T \boldsymbol{w}^\star - \boldsymbol{a}_i^T \boldsymbol{w}| = |\boldsymbol{a}_i^T \boldsymbol{w} - (-\boldsymbol{a}_i^T \boldsymbol{w}^\star - \boldsymbol{a}_i^T(-\boldsymbol{w}))| \\ &\geq |\boldsymbol{a}_i^T \boldsymbol{w}| - \underbrace{|-\boldsymbol{a}_i^T(\boldsymbol{w}^\star - \boldsymbol{w})|}_{\leq \|\boldsymbol{a}_i\|_2 \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2} \\ &\geq |\boldsymbol{a}_i^T \boldsymbol{w}| - \|\boldsymbol{a}_i\|_2 \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2 = |\boldsymbol{a}_i^T \boldsymbol{w}| - \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2. \end{aligned} \tag{5.20}$$

From these properties, we can write screening tests which are usable in practice:

Theorem 5.4.2. *Given a safe sphere region $\boldsymbol{w}^\star \in \mathcal{R} = \mathcal{B}(\boldsymbol{w}, r)$, we have:*

$$\forall i \in \bar{S}, \text{ if } |\boldsymbol{a}_i^T \boldsymbol{w}| > \tfrac{\mu}{M} + r, \text{ then } x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}); \tag{5.21a}$$

$$\forall i \in \bar{S}, \text{ if } |\boldsymbol{a}_i^T \boldsymbol{w}| < \tfrac{\mu}{M} - r, \text{ then } x_i^\star = 0; \tag{5.21b}$$

$$\forall i \in S_1, \text{ if } |\boldsymbol{a}_i^T \boldsymbol{w}| > r, \text{ then } x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}). \tag{5.21c}$$

Proof. The proof follows the standard screening proofs such as the ones in [El Ghaoui et al. 2010]. As $\boldsymbol{w}^\star \in \mathcal{B}(\boldsymbol{w}, r)$, we have $\|\boldsymbol{w}^\star - \boldsymbol{w}\|_2 \leq r$. For rule (5.17a), we use Equation (5.20) stating that $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| \geq |\boldsymbol{a}_i^T \boldsymbol{w}| - \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2$. As $\|\boldsymbol{w}^\star - \boldsymbol{w}\|_2 \leq r$ holds, we have $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| \geq |\boldsymbol{a}_i^T \boldsymbol{w}| - r$. This means that if $|\boldsymbol{a}_i^T \boldsymbol{w}| - r > \frac{\mu}{M} \iff |\boldsymbol{a}_i^T \boldsymbol{w}| > \frac{\mu}{M} + r$ holds, we have $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| > \frac{\mu}{M}$, therefore $x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}^\star)$. As the rule holds for every point in $\mathcal{R}$, this implies there is no change of sign inside the sphere (otherwise we would have some points $\boldsymbol{w}^0$ with $\boldsymbol{a}_i^T \boldsymbol{w}^0$ close to zero, therefore not fulfilling the rule), meaning that $\operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}^\star) = \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w})$, so $x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w})$.

For rule (5.17b), we use Equation (5.19) which gives $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| \leq |\boldsymbol{a}_i^T \boldsymbol{w}| + \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2 \leq |\boldsymbol{a}_i^T \boldsymbol{w}| + r$. This means that if we have $|\boldsymbol{a}_i^T \boldsymbol{w}| + r < \frac{\mu}{M} \iff |\boldsymbol{a}_i^T \boldsymbol{w}| < \frac{\mu}{M} - r$, then $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| < \frac{\mu}{M}$ holds and $x_i^\star = 0$.

For rule (5.17c), we rewrite it as $|\boldsymbol{a}_i^T \boldsymbol{w}^\star| > 0$ and use again Equation (5.20). □

To get an actual screening test, we must be able to express the sphere radius r. We will resort to the Gap-Safe sphere, as detailed in the next section.

5.4.3 Gap-Safe sphere

Gap-Safe screening uses the regularity of the problem to give a safety guarantee, thus a sufficiently high radius r for a safe sphere, through the duality gap. The following proposition uses the standard argument of GapSafe screening tests [Raj et al. 2016]:

Proposition 5.4.3. *Using an arbitrary dual point $\boldsymbol{w}$ and an arbitrary primal point $\boldsymbol{x}$, we can use $r = \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}$ in the tests* (5.21) *to get safe tests, with $G(\boldsymbol{x}, \boldsymbol{w}) := P(\boldsymbol{x}) - D(\boldsymbol{w})$.*

Proof. Using an arbitrary primal-dual pair $(\boldsymbol{x}, \boldsymbol{w}) \in [-M, M]^Q \times \mathbb{R}^N$, we have:

$$\begin{aligned} G(\boldsymbol{x}, \boldsymbol{w}) &:= P(\boldsymbol{x}) - D(\boldsymbol{w}) \\ &\geq P(\boldsymbol{x}^\star) - D(\boldsymbol{w}) = D(\boldsymbol{w}^\star) - D(\boldsymbol{w}) = -f^*(\boldsymbol{w}^\star) - g^*(-\mathbf{A}^T\boldsymbol{w}^\star) + f^*(\boldsymbol{w}) + g^*(-\mathbf{A}^T\boldsymbol{w}). \end{aligned}$$

Since f^* in Problem (5.2) is 1-strongly convex, we have:

$$f^*(\boldsymbol{w}) \geq f^*(\boldsymbol{w}^\star) + \nabla f^*(\boldsymbol{w}^\star)^T(\boldsymbol{w} - \boldsymbol{w}^\star) + \tfrac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}^\star\|_2^2.$$

Since g^* in Problem (5.2) is convex, we have:

$$g^*(-\mathbf{A}^T\boldsymbol{w}) \geq g^*(-\mathbf{A}^T\boldsymbol{w}^\star) + \left(\partial g^*(-\mathbf{A}^T\boldsymbol{w}^\star)\right)^T(-\mathbf{A}^T\boldsymbol{w} + \mathbf{A}^T\boldsymbol{w}^\star).$$

Therefore, we have:

$$\begin{aligned} G(\boldsymbol{x}, \boldsymbol{w}) \geq & -f^*(\boldsymbol{w}^\star) - g^*(-\mathbf{A}^T\boldsymbol{w}^\star) \\ & + f^*(\boldsymbol{w}^\star) + \nabla f^*(\boldsymbol{w}^\star)^T(\boldsymbol{w} - \boldsymbol{w}^\star) + \tfrac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}^\star\|_2^2 \\ & + g^*(-\mathbf{A}^T\boldsymbol{w}^\star) + \left(\partial g^*(-\mathbf{A}^T\boldsymbol{w}^\star)\right)^T(-\mathbf{A}^T\boldsymbol{w} + \mathbf{A}^T\boldsymbol{w}^\star), \\ = & \nabla f^*(\boldsymbol{w}^\star)^T(\boldsymbol{w} - \boldsymbol{w}^\star) + \left(\partial g^*(-\mathbf{A}^T\boldsymbol{w}^\star)\right)^T(-\mathbf{A}^T\boldsymbol{w} + \mathbf{A}^T\boldsymbol{w}^\star) + \tfrac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}^\star\|_2^2. \end{aligned}$$

Due to the first optimality conditions, we have:

$$\nabla f^*(\boldsymbol{w}^\star)^T(\boldsymbol{w} - \boldsymbol{w}^\star) + \left(\partial g^*(-\mathbf{A}^T\boldsymbol{w}^\star)\right)^T(-\mathbf{A}^T\boldsymbol{w} + \mathbf{A}^T\boldsymbol{w}^\star) \geq 0.$$

Consequently, we have $G(\boldsymbol{x}, \boldsymbol{w}) \geq \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}^\star\|_2^2 \iff \|\boldsymbol{w} - \boldsymbol{w}^\star\|_2 \leq \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}$. Thus, we have $\boldsymbol{w}^\star \in \mathcal{B}(\boldsymbol{w}, \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})})$, and the Proposition follows. □

5.4.4 Upper bound trick

We now consider a refinement over the Gap-Safe screening detailed in Section 5.4.3. We use the Gap-Safe sphere with radius $r = \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}$, that we can also write as $r = \sqrt{2(P(\boldsymbol{x}) - D(\boldsymbol{w}))}$. While the tests (5.21) depend on the dual point $\boldsymbol{w}$, the only

dependence on $\boldsymbol{x}$ is for the value of $P(\boldsymbol{x})$. This means that as long as we have a valid primal value $P(\boldsymbol{x})$, we do not need to know the corresponding primal point $\boldsymbol{x}$ giving this value.

As a recall, $\overline{\text{ub}}$ is the best upper bound found so far in the branch-and-bound. The intent is then to use $\overline{\text{ub}}$ in the radius expression r if the current primal $P(\boldsymbol{x})$ is higher. If the current node is in the case of $\mathbf{N}^{\text{unpruned}}$ in Figure 5.1 (right), then we have for $\boldsymbol{x}$ the current iterate $P(\boldsymbol{x}) \geq \overline{\text{ub}} \geq P(\boldsymbol{x}^\star)$ and we can use $\overline{\text{ub}}$ as our primal value without losing the screening safety, because there exists an $\overline{\boldsymbol{x}}$ such that $P(\overline{\boldsymbol{x}}) = \overline{\text{ub}}$.

If the current node is in the case of $\mathbf{N}^{\text{pruned}}$ in Figure 5.1 (left), then we have $P(\boldsymbol{x}) \geq P(\boldsymbol{x}^\star) > \overline{\text{ub}}$. In this case, using $\overline{\text{ub}}$ instead of $P(\boldsymbol{x})$ can lead the screening tests to discard the optimal point $\boldsymbol{x}^\star$, and the descent algorithm operating on the remaining variables will converge to a point $\hat{\boldsymbol{x}}$ such that $P(\boldsymbol{x}) \geq P(\hat{\boldsymbol{x}}) > P(\boldsymbol{x}^\star) > \overline{\text{ub}}$: $P(\hat{\boldsymbol{x}})$, is *not* a valid lower bound for the node. However, as $P(\boldsymbol{x}^\star) > \overline{\text{ub}}$, this node should be pruned. Fortunately, as $P(\hat{\boldsymbol{x}}) > P(\boldsymbol{x}^\star) > \overline{\text{ub}}$, taking $P(\hat{\boldsymbol{x}})$ instead of $P(\boldsymbol{x}^\star)$ will lead to the same pruning decision for this node.

Summing up, when using $\overline{\text{ub}}$ instead of $P(\boldsymbol{x})$ in the radius r, for nodes such as $\mathbf{N}^{\text{unpruned}}$, the screening is safe and we will get the correct lower bound value $\text{lb}^{\mathbf{N}}$, and for nodes such as $\mathbf{N}^{\text{pruned}}$, we will take the correct pruning decision, even if the screening discards too many components. We will use that in practice to replace the radius of the sphere from $r = \sqrt{2(P(\boldsymbol{x}) - D(\boldsymbol{w}))}$ to $r = \sqrt{2(p - D(\boldsymbol{w}))}$ where $p = \min(P(\boldsymbol{x}), \overline{\text{ub}})$.

5.4.5 Numerical experiments and results

Dataset We consider the same datasets than in Section 5.2.4. In particular, we keep the same setup of synthetic datasets with $\rho \in \{0, 0.7, 0.8, 0.92\}$, 10 instances for each value of ρ. The homotopy continuation algorithm is used for $\rho \in \{0.7, 0.8, 0.92\}$, while the coordinate descent algorithm is used for $\rho = 0$.

Results Figure 5.8 shows the performance of the screening applied dynamically at each sub-iteration, the metric considered is the percentage of variables fixed to zero (no variable was screened to $\pm M$ in our experiments, probably because M was chosen large enough). The results are average percentages for all the nodes with a given number of active variables $|S_1|$, the nodes being either grouped by instance (diamond points) or aggregated across all instances (dashed line). First, a noteworthy observation lies in the evolution of the performance with $|S_1|$. While for the early pruning detailed in Section 5.2 performance improves with increasing $|S_1|$, we can see the opposite behaviour here. This may be due to the fact that as $|S_1|$ increases, we have less nodes, and as we get deeper in the tree, we are following branches containing good solutions where it becomes harder to discriminate between "good" and "bad" atoms at first glance. Then, we can see that the

performance clearly depends on the correlation level ρ: the less correlated, the better, from almost 100% screened variables for nodes with $|S_1| = 0$ for $\rho = 0$ to about 3% for $\rho = 0.92$. The performance fall is particularly visible between $\rho = 0.8$ and $\rho = 0.92$, as we lose an order of magnitude. This general behaviour was expected. Indeed, when taking the dual point $\boldsymbol{w} = \mathbf{A}\boldsymbol{x} - \boldsymbol{y}$, the screening method is trying to highlight atoms (columns) of $\mathbf{A}$ which have such a small correlation with the residual that it should be discarded from the solution. If ρ is close to 0, atoms are lowly correlated to each other, and the screening is at ease. If ρ is close to 1, then atoms tend to be highly correlated to each other. In such a context, an atom has either a high correlation with the residual, or no atom correlates with the residual the current iterate is close to the optimal solution, making the screening unable to discard variables before reaching optimality. On average, the percentage of screened variables is at 1% for $\rho = 0.92$, 8.92% for $\rho = 0.8$, 61.3% for $\rho = 0.7$ and 95.6% for $\rho = 0$.

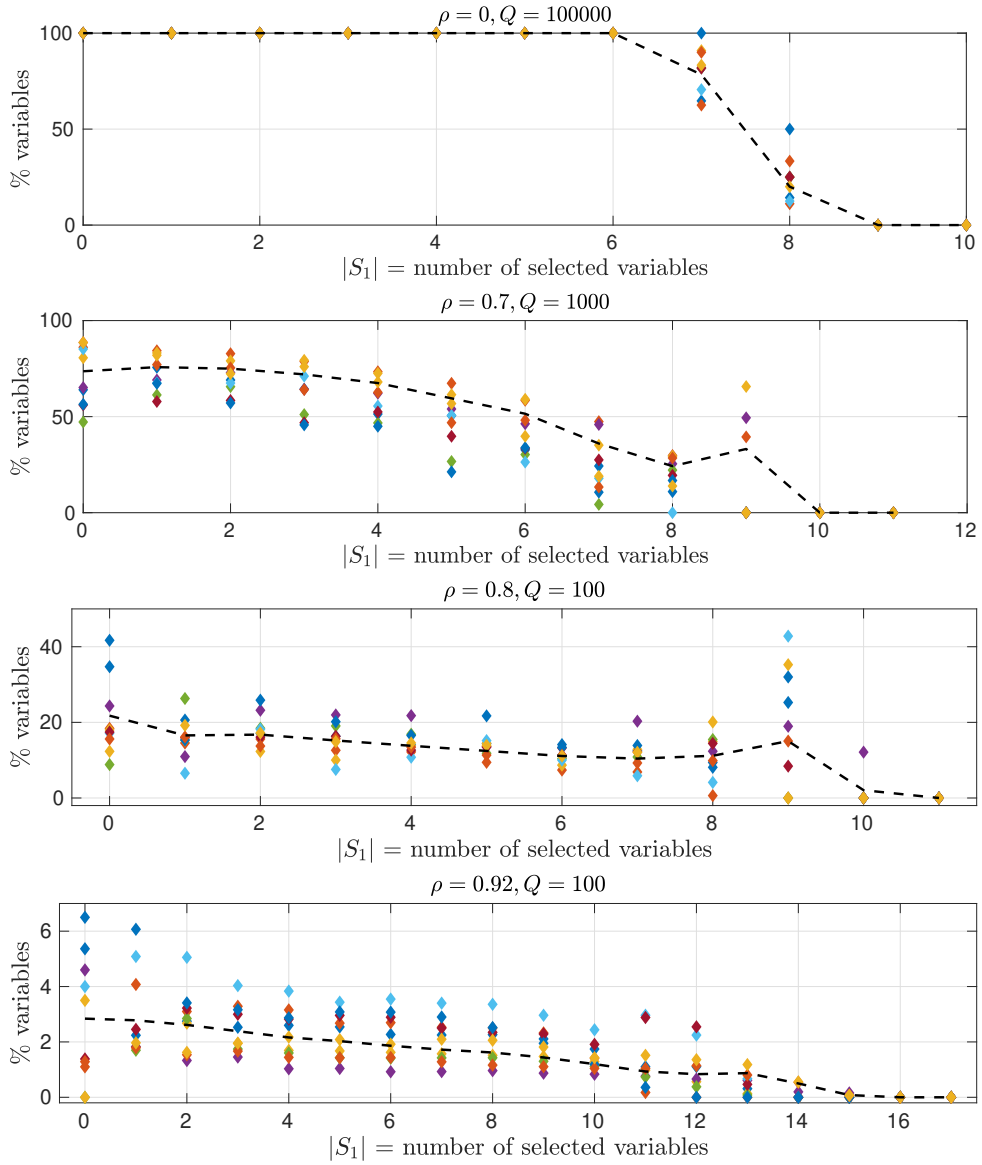


Figure 5.8 – Screening performance as a function of the number of non-zero variables at the corresponding nodes for four correlation levels in matrix $\mathbf{A}$ and problem sizes. The cardinality of the solution is $K = 9$. For each instance, one point represents the average ratio of screened variables over the nodes with cardinality $|S_1|$. The dashed line shows the average ratio of screened variables over all instances. The homotopy continuation algorithm is used for $\rho \in \{0.7, 0.8, 0.92\}$, while the coordinate descent algorithm is used for $\rho = 0$.

5.5 Discussion

Conclusion In this chapter, several contributions were developed to accelerate lower bound computations, namely:

- use the dual problem objective function of Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$) as an early lower bound to prune node as soon as possible;
- use same dual problem to stop an optimization algorithm before convergence while keeping control of the approximation quality;
- leverage the duality gap in a GapSafe screening tailored to our problem;
- show through numerical experiments the potential of each technique, which are all influenced by the correlation level ρ.

Perspectives Several aspects of this chapter dedicated to lower bound accelerations could be further investigated. On the early pruning side, we could extend the numerical study with children node pruning, called node screening in [Guyard et al. 2022], to see in which part of the branch-and-bound tree it performs better, in the hope of identifying zones where we should apply pruning and zones where it is not worth the effort.

A similar zoning strategy could be used for screening as well, given that screening tests have a computational cost. Additionally, more aggressive strategies could be used. Indeed, we need to get the optimal primal solution $\boldsymbol{x}^\star$ at the end of the optimization. A safe screening strategy guarantees that $\boldsymbol{x}^\star$ remains in the feasible space at *each* iteration, whereas we only need $\boldsymbol{x}^\star$ to be in the feasible space at the *last* iteration. We can use a two-stage method to this end. The first stage can be a strong, unsafe screening for the first iterations, and once we converged in this unsafe setup, rollback the strong screening decisions, apply a safe screening method, and continue iterating in a safe setup that constitutes our second stage. We hope that in this case, the strong screening will at the same time discard a lot of variables so that the unsafe setup is quick the converge, and that it converges to a "not so bad" $\hat{\boldsymbol{x}}$ which can reach $\boldsymbol{x}^\star$ quickly in the safe setup. Instead of a strong screening for the unsafe setup (the first stage), we can use an active set method where we solve subproblems with few variables. Some active set methods were derived from Gap-Safe screening [Massias 2017] where we use a contracted version of the Gap-Safe sphere: $\sqrt{\frac{1}{2}G(\boldsymbol{x}, \boldsymbol{w})}$ instead of $\sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}$ for example. This could be a convenient way to further leverage Gap-Safe screening in our branch-and-bound framework.

Moreover, the three techniques of this chapter, namely early pruning, dual-monitored inexact convergence and screening methods are extensively using the dual problem, for its objective function as well as for its iterates. As shown in Section 5.2.3, different primal and primal-dual optimization algorithms lead to different dual iterates, with very different objective values. Therefore, it can be interesting to further explore possibilities to refine

a given dual iterate to improve its objective function. Solving exactly the dual problem is as difficult as solving the primal problem, but heuristic search could help to quickly get a "good" dual point, enough to either prune the node or screen variables.

Part II

Structured sparsity

Chapter 6

Review of structured sparsity problems and algorithms

Contents

6.1 Introduction

In Part I we have seen several methods handling sparsity on the variables of $\boldsymbol{x}$: we want to the majority of the variables to be zero, or conversely to get a small number of variables to some non-zero value. Structured sparsity extends this approach by targeting not just individual variables but *groups* of variables. Given prescribed groups, we want a solution with a lot of these groups to zero, or conversely a small number of them to be non-zero (see Figure 6.1 for an example). As we shall see in Chapter 7, these two viewpoints (a lot of zeros or a small number of non-zeros) do not have exactly the same meaning.

In the general case, groups can be of varying size, and groups can overlap, meaning they can have a non-empty intersection (see Figure 6.2 for an example). "Structured

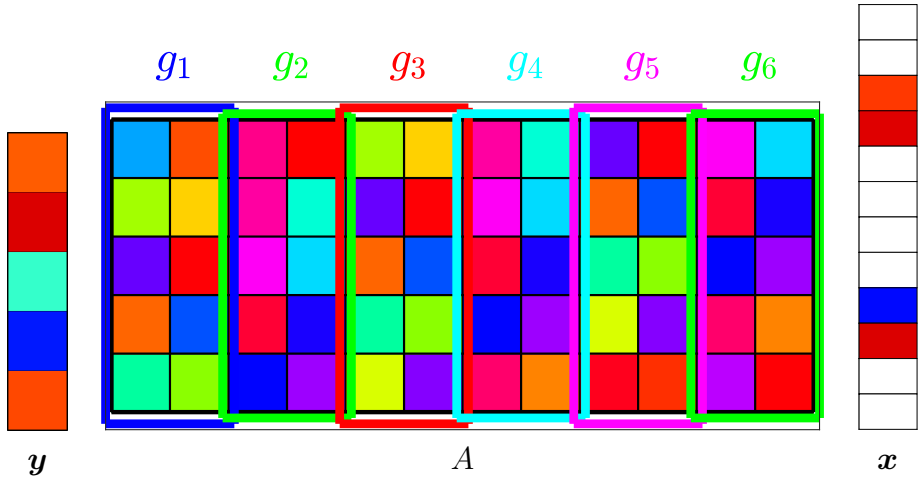


Figure 6.1 – An example of structured sparsity without overlap: all groups are disjoint. Here, g_2 and g_5 are present in the solution $\boldsymbol{x}$.

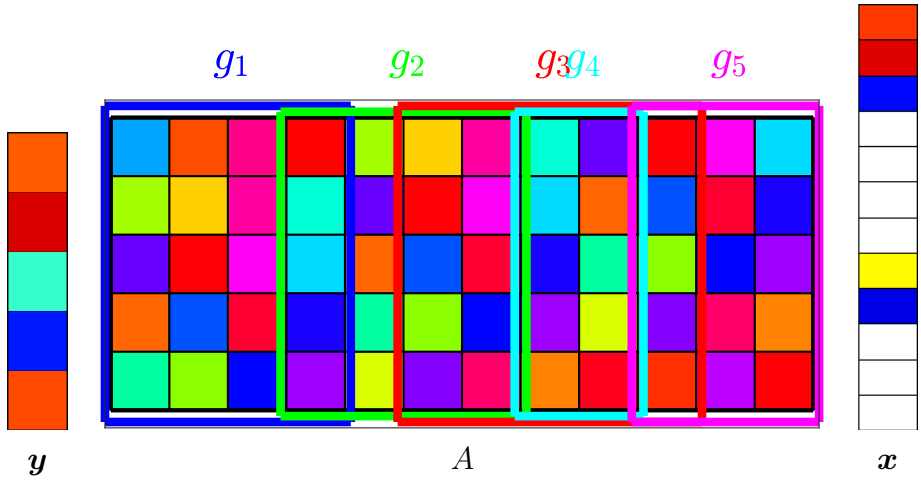


Figure 6.2 – An example of structured sparsity with overlap: some groups are intersecting with others (g_1 with g_2, g_3 with g_2, g_4 with g_5). Here, g_1, g_3 and g_4 are present in the solution $\boldsymbol{x}$.

sparsity" refers to this general case. Other terms do exist in the literature, such as "group sparsity", which refers to a non-overlapping setup [Bach et al. 2012]. Consequently, Group LASSO [Yuan et al. 2006] is to the group sparsity case what LASSO is to the scalar sparsity case. As we will see in Chapter 8, overlapping groups make the problem more difficult, and even convex relaxations are trickier to optimize.

For the mathematical notation side, g will usually denote a group, that is, a set of components. Vector $\boldsymbol{x}_g$ denotes the vector $\boldsymbol{x}$ restricted to the components in group g. The scalar sparsity term $\|\boldsymbol{x}\|_0$ can also be written as $\sum_{i\in\{1,..,Q\}} \mathbf{1}_{x_i\neq 0}$. This term quite naturally extends to structured sparsity with $\sum_{g\in G} \mathbf{1}_{\boldsymbol{x}_g\neq 0}$ where G defines the set of groups. Consequently, we write the structured sparsity counterpart of Problem $(\mathcal{P}^M_{2+0})$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \mu\|\boldsymbol{x}\|_0 \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M \qquad (\mathcal{P}^M_{2+0})$$

which as not been given in the literature to the best of our knowledge, as:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \mu \sum_{g\in G} \mathbf{1}_{\boldsymbol{x}_g\neq 0} \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M. \qquad (\mathcal{P}_{2+0s})$$

In the remainder of this chapter we will review some application problems using structured sparsity in Section 6.2 as well as existing methods to tackle these problems in Section 6.3.

6.2 Applications

This section provides some non-exhaustive list of application examples of structured sparsity found in the literature.

6.2.1 Structured hyperspectral unmixing

Section 3.2.2 (page 22) already mentioned the use-case of hyperspectral imaging, where we wish to recover the proportion of materials in a pixel, given a measured discretized spectrum with materials mixed together [Ben Mhenni, Bourguignon, Ninin, and Schmidt 2018]. Some materials do have polarity properties, meaning a single material can lead to several spectra [Iordache et al. 2011]. One way to deal with this is to stack the different spectra of a given material in one group. The optimization task is then to untangle materials from the measurements, knowing there is a small number of materials (so a small number of groups) in the scene.

6.2.2 Sparse spectral analysis

When a measured signal is known to have a sparse Fourier transform, a natural way to analyse or recover it is to look for a sparse decomposition in the Fourier domain [Bourguignon, Carfantan, et al. 2007]. For the magnitude of the Fourier transform, which is real-valued, this can be done with the branch-and-bound algorithm of Part I. For tackling the Fourier transform (including magnitude and phase), which is complex-valued, it has been shown in the ℓ_1 case [Bourguignon, Carfantan, et al. 2007] that breaking the complex numbers $c \in \mathbb{C}$ into two real numbers $(\mathfrak{Re}(c), \mathfrak{Im}(c))$, without any structure or relationship holding them together, leads to poorer results than taking into account the grouping introduced by complex numbers.

In our setting, such task would be done by transforming each complex number into a real pair, each pair corresponding to a group. Getting a solution with a small number of groups means getting a solution with few nonzero complex numbers. Structured sparsity has also been used to retrieve solutions which are sparse in the time-frequency plane [Kowalski et al. 2009].

6.2.3 Factor selection

In some applications in statistics we wish to get some variable reduction as in Section 3.2.3 (page 23), but these variables are organized into cohesive blocks. A *factor* is then one of those blocks, and we wish to retrieve a model fitting the data with few factors (see for example [Cong et al. 2017; Yuan et al. 2006]). For example, in [Yuan et al. 2006], authors the fuse different modalities of a categorical factor (for example for the factor "color", modalities can be "black", "white" or "other") into a group for each factor, the aim being to be able to discard factors which have no influence on the final outcome. The application of this model reduction is then as general as the variable selection one presented in Section 3.2.3.

6.2.4 Hierarchical selection

Hierarchical selection is a regularization structure where activating some groups imply activating some others (see [Bach et al. 2012; Jenatton et al. 2011] and references therein). As pointed out in [Bach et al. 2012], this type of regularization enjoys some use in topic modelling, wavelet decomposition, and gene networks tasks, to name a few. The general idea of hierarchical selection is that including some variable x_j in the solution requires including some other variable x_i. This example can extend to more than one implied group. If we graph the implication network, with each variable being a node, and a node x_i being linked to x_j if $x_j \implies x_i$, a hierarchical selection then corresponds to a tree, as shown in Figure 6.3 (Figure 6.3 is taken from paper [Bach et al. 2012]).

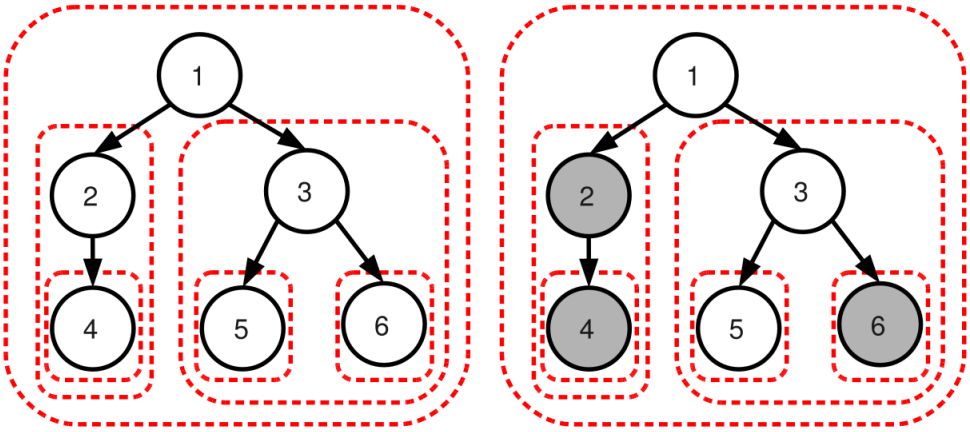


Figure 6.3 – A hierarchical sparsity example. Setting variable 2 to 0 implies setting variable 4 to 0 as well. This figure is taken from [Bach et al. 2012].

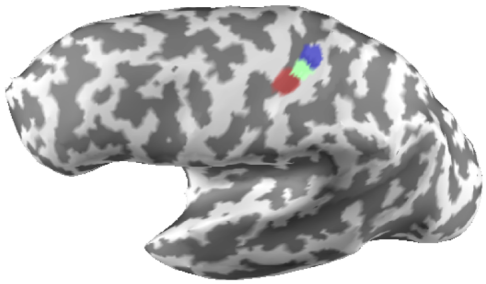


Figure 6.4 – An example of M/EEG task, with three activated neuron clusters (in red, green and blue) to reconstruct from measurements. Figure taken from [Gramfort et al. 2012]

A way to model this kind of implication constraints is to put groups enclosing each other. As shown in Figure 6.3, we create a group for each variable, this group containing the corresponding variable as well as all its descendants in the hierarchical tree.

6.2.5 Magneto and Electro-encephalography (M/EEG)

In M/EEG, we are looking for the activation of some neuron clusters in the brain (see Figure 6.4 for an example, taken from [Gramfort et al. 2012]). As there are many more sources of electric power in the brain (many neuron clusters) than sensors, the problem is underdetermined and difficult to tackle (see [Gramfort et al. 2012] and references therein). To promote sources which are both spatially concentrated and continuously activated during the time of acquisition, a regularization on both space and time is used, leading to mixed norms like we mentioned previously, called two-level mixed norms, but also three-level mixed norms [Gramfort et al. 2012], which are out of scope of the present work.

6.3 State-of-the-art methods

Like in Part I, state-of-the-art methods solving approximately the original ℓ_0 problem can be categorized into three groups: convex relaxations, ℓ_0 heuristics, and non-convex relaxations.

6.3.1 Convex relaxations: mixed norms

The original problem ($\mathcal{P}_{2+0s}$) is NP-Hard, discontinuous, non-convex. In a similar fashion to methods presented in Section 3.3.1 (page 25), convex relaxations of the original problem can be used. In this case, the term $\mathbf{1}_{\boldsymbol{x}_g \neq 0}$ is relaxed by using a ℓ_q norm: $\|\boldsymbol{x}_g\|_q$. In general, the choice of $q = 2$ is made, meaning we wish to solve the following problem:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2 + \lambda \sum_{g \in G} \|\boldsymbol{x}_g\|_2. \tag{6.1}$$

This problem is called the Group LASSO [Yuan et al. 2006]. The sum can be seen as an ℓ_1 norm of the vector:

$$\begin{pmatrix} \|\boldsymbol{x}_{g_1}\|_2 \\ \|\boldsymbol{x}_{g_2}\|_2 \\ \vdots \end{pmatrix},$$

therefore we say Problem (6.1) is a mixed norm problem, in this case with a $\ell_1 - \ell_2$ mixed norm [Gramfort et al. 2012]. When the different groups are disjoint, the proximal operator is analytical, and proximal algorithms of Section 3.3.1 can be used as-is (see Chapter 8 for more details). However, when groups are overlapping, no analytical expression of the proximal operator is available.

Let's note that when solving Problem (6.1) some groups will be set to zero, and the solution $\boldsymbol{x}$ will be composed of the variables belonging to non-zero groups only [Bach et al. 2012; Jenatton et al. 2011]. In other words, the solution zeros are formed by the *union* of the zero groups. Another approach, latent LASSO in the context of convex relaxations, consists in defining the solution zeros by the *intersection* of the zero groups (see Problem (6.2)). When groups are overlapping, this means a variable is non-zero when at least one group including this variable is active (non-zero). In Figure 6.2, this would mean the active groups are g_1 and either g_3 or g_4. In contrast, in Problem (6.1), for a variable to be non-zero we need *all* groups including it to be active. In Figure 6.2, this means the active groups are g_1, g_3 *and* g_4. Latent LASSO [Obozinski et al. 2011] has seen some fruitful applications in M/EEG [Gramfort et al. 2012], and more generally to graph sparsity applications [Huang et al. 2011], and is formulated as:

$$\min_{\boldsymbol{x} \in \mathbb{R}^Q, \mathbf{Z} \in \mathbb{R}^{Q \times |G|}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2 + \lambda \sum_{i=1}^{|G|} \|\boldsymbol{z}_i\|_2 \quad \text{s.t.} \quad \boldsymbol{x} = \sum_{i=1}^{|G|} \boldsymbol{z}_i. \tag{6.2}$$

The reader is referred to [Obozinski et al. 2011] and references therein for the optimization algorithms solving Problem (6.2).

6.3.2 Heuristics on the ℓ_0 problem

Local search algorithms have also been designed to give approximations of the solution to Problem ($\mathcal{P}_{2+0s}$). All these algorithms are tackling the constrained version of the ℓ_0 structured sparsity (or some extension of it), which write in our setting as Problem ($\mathcal{P}_{2/0q}$):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 \quad \text{s.t.} \quad \sum_{g\in G} \mathbf{1}_{\boldsymbol{x}_g\neq 0} \leq K. \tag{$\mathcal{P}_{2/0q}$}$$

Variants of the Orthogonal Matching Pursuit have been designed in [Eldar et al. 2010; Huang et al. 2011], where [Eldar et al. 2010] is dedicated to the fixed size and non-overlapping case, and [Huang et al. 2011] tackles the overlapping case, using an union of groups approach as in the Latent LASSO. In [Fujii et al. 2018], authors tackle a more generic problem of combinatorial sparsity, where the set of possible supports is not constrained by a cardinality constraint, but by a more involved definition of the feasible set (for example, allowing $\{g_1, g_2\}$ as a support but not $\{g_1, g_3\}$, even if both supports are of size 2). In [Krause et al. 2010] and [Cong et al. 2017], authors respectively developed a forward (like OMP, adding atoms iteratively) and a forward-backward (adding and deleting atoms iteratively) greedy algorithm in the (also more general) dictionary selection context, where both the features $\boldsymbol{x}$ and the dictionary $\mathbf{A}$ are learned.

Outside of the world of greedy algorithms, a variant of Iterative Hard Thresholding was designed in [Baraniuk et al. 2010] to tackle Problem ($\mathcal{P}_{2+0s}$). To the best of our knowledge, the literature tackling the Problem ($\mathcal{P}_{2+0s}$) is scarce.

6.3.3 Non-convex approaches

The field of structured sparsity through non-convex formulations is an ongoing research topic which is not yet as mature as its scalar sparsity counterpart. It appears the field emerged a decade ago, with authors in [H. Wang et al. 2013] tackling the problem of dictionary learning (meaning $\boldsymbol{x}$ and $\mathbf{A}$ are both unknowns to be recovered) by a structured sparsity prior with a non-convex surrogate, namely an mixed ℓ_q norm with $0 < q < 1$. Authors in [D. Wang et al. 2016] propose a numerical comparison of several algorithms doing the same task (dictionary learning) with the same non-convex term ($\ell_q, 0 < q < 1$ penalty).

A non-convex penalty in the form of a logarithmic based penalty for the problem of blind structured sparsity is used in [Lazzaro et al. 2015]. The blind aspect means the groups are not known in advance. In the deconvolution setting, the unknown is a vector of time-domain samples, and structured sparsity usually occurs for grouping together

several consecutive points into a time segment. For the structured sparsity looked at in this thesis, we know the position and size of the time segment. For the blind structured sparsity case, we typically do not know the segment boundaries.

More recently, authors in [X. Zhang et al. 2023] proposed a unified algorithm for solving structured sparsity problems with several types of non-convex non-separable penalties including the mixed $\ell_q, 0 < q < 1$ ones. Their target problem is more general than that of this thesis, as they formulate a problem where $\boldsymbol{x}$ is not a vector but a matrix, the (structured) sparsity prior being a (structured) low-rank prior.

6.3.4 Exact ℓ_0 optimization

As far as we know, no dedicated method was previously developed for the exact optimization of Problem $(\mathcal{P}_{2+0s})$. This work can be considered as a way to fill this gap. The structured sparsity measure considered in this work extends the scalar sparsity detailed in Part I, therefore we will use a regularizer promoting a small number of active groups. More involved forms of structured sparsity, for example by specifying explicitly the allowed supports as in [Fujii et al. 2018], are not addressed here. When developing a branch-and-bound algorithm for solving Problem $(\mathcal{P}_{2+0s})$, several issues and degrees of freedom arise compared to the scalar sparsity case in Part I. Chapter 7 presents the different issues specific to the structured sparsity case, namely the way to structure the search space, the way to bound and divide a node, as well as the choices made to tackle them. Chapter 8 is dedicated to the formulation and numerical resolution of lower bounds, where some degrees of freedom give rise to several problem formulations which are all extensions of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$:

$$\mathrm{lb}^{\mathbf{N}} = \min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|S_1| + \tfrac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_1 \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M, \boldsymbol{x}_{S_0} = 0. \quad (\mathcal{P}^{\mathbf{N}}_{2+1})$$

Chapter 9 extends the different lower bound accelerations of Chapter 5 to the structured sparsity case and studies their performance. Finally, Chapter 10 compares the proposed branch-and-bound algorithm with methods of the state-of-the-art.

Chapter 7

Branch-and-bound algorithm for structured sparsity

Contents

7.1 Introduction

As far as we know, no algorithm was designed to solve exactly Problem ($\mathcal{P}_{2+0s}$):

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2 + \mu \sum_{g \in G} \mathbf{1}_{\boldsymbol{x}_g \neq 0} \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M. \qquad (\mathcal{P}_{2+0s})$$

Building on the basis of the branch-and-bound algorithm created in [Ben Mhenni 2020] and further developed in Part I, we propose here several contributions which together form the skeleton of a sound branch-and-bound algorithm for solving optimization problems with a structured sparsity regularizer.

We start by detailing how the branch-and-bound algorithm will explore the combinatorial search space of the problem in Section 7.2. Then, we focus on the particular case of overlapping groups in Section 7.3, and how this particular case drives the way to associate groups to variables for the different subproblems. Upper and lower bounding of subproblems is formulated in Section 7.4, and branching strategies are discussed in Section 7.5. Perspectives and areas of future development are drawn in Section 7.6.

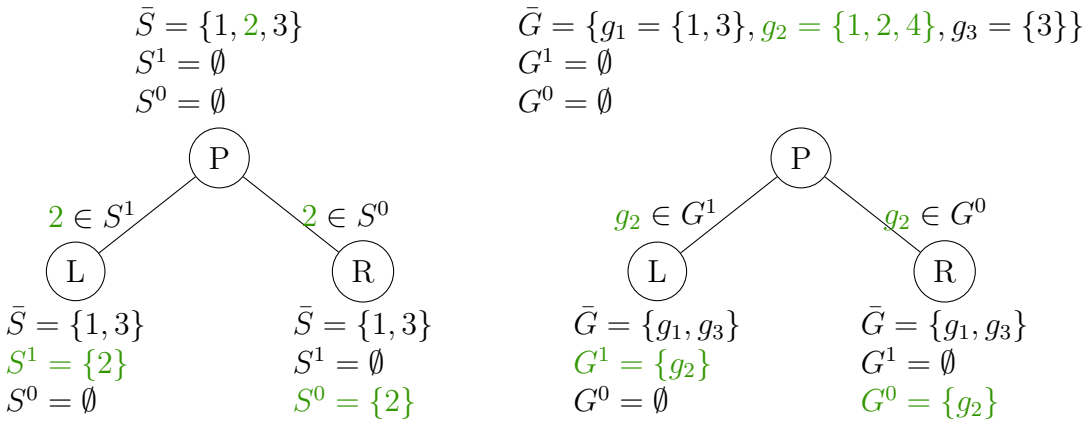
7.2 Search space

Taking a step back at what was done in Part I, what is the fundamental goal of a branch-and-bound algorithm? We may answer this question in multiple ways. One viewpoint is to say that a branch-and-bound algorithm reduces the combinatorial complexity of the original problem by making choices, which creates subproblems to be solved. As such, the choices made by a branch-and-bound algorithm should effectively reduce the complexity of the problem.

In Part I, we were dealing with problems which are NP-hard just due to the term $\|\boldsymbol{x}\|_0 = \sum_{i\in\{1,\dots,Q\}} \mathbf{1}_{x_i\neq 0}$. To reduce the complexity of the problem, we make choices on whether a given x_i is nonzero or zero. In other words, we choose the value of some $\mathbf{1}_{x_i\neq 0}$ terms in the previous sum. Here, we consider problems whose NP-hardness comes from the term $\sum_{g\in G} \mathbf{1}_{\boldsymbol{x}_g\neq 0}$. To reduce the complexity of the problem, we apply the same method as before: make choices on the value of some $\mathbf{1}_{\boldsymbol{x}_g\neq 0}$ terms of the sum. This means we make choices on whether the components $\boldsymbol{x}_g$ are part of the support of the solution or not. We can also consider groups instead of components, and make choices on whether a given group g is part of the solution or not. To highlight the role of a group in the final solution, we need a concept which links groups to the final solution, in the same way the notion of support links variables to the final solution in the scalar sparsity case. To this end, we introduce the term *group support* of the solution. The group support of a solution just designates the active groups of this solution: the groups corresponding to the variables which are in the support of the solution.

In the scalar sparsity case, we represent a support subspace by using $S_1, S_0, \bar{S}$ to define the indices of the variables which are respectively active, inactive, and undecided at a node $\mathbf{N}$ (see Section 3.4.1 on page 33). In the same vein, we use notations $G_1, G_0, \bar{G}$ for groups: G_1 is then the index set of active groups, G_0 is the index set of inactive groups, and $\bar{G}$ the index set of undecided groups.

In Part I, making a choice means dividing a node, which in turn means picking an undecided variable (a variable belonging to $\bar{S}$ in the current node) and including it in the support (putting it to S_1) for a child node or excluding from the support of the solution (putting it to S_0) for a another child node. In the context of structured sparsity, we will apply the same reasoning: we will divide a node by taking an undecided group $g \in \bar{G}$ and forcing it to be either part of the solution, putting g to G_1, or to be excluded from the solution, putting g to G_0. Figure 7.1 gives an illustration of the behaviour of the branch-and-bound algorithm in the scalar case compared to the structured case, where three groups where arbitrarily chosen, with the first group g_1 being linked to variables x_1 and x_3, g_2 being linked to variables x_1, x_2, x_4, and g_3 being linked to variable x_3.



Scalar sparsity division (Part I) Structured sparsity division (Part II)

Figure 7.1 – Node division examples, where we branch on the green variable (left) or group (right).

7.3 Overlapping groups

If the branch-and-bound algorithm makes choices about groups, a natural question is to wonder what happens to individual variables. In other words, how a change in the groups partition $\{G_1, G_0, \bar{G}\}$ influences the variables partition $\{S_1, S_0, \bar{S}\}$. The most intuitive way is to say that undecided variables are variables belonging to undecided groups, meaning that $\bar{S} = \cup_{g \in \bar{G}} \{i \in g\}$, and so on for S_1 and S_0: $S_1 = \cup_{g \in G_1} \{i \in g\}$, $S_0 = \cup_{g \in G_0} \{i \in G_0\}$. For example, in Figure 7.1 (right), we have for node P: $\bar{S} = \{1, 2, 3, 4\}$, $S_1 = \emptyset$, $S_0 = \emptyset$.

Unfortunately, it is not sufficient in the case of *overlapping groups*, which are groups with non-empty intersection. Indeed, as the example in Figure 7.2 (left) shows, a variable belonging to a group in G_1 *and* to a group in G_0 is in an ambiguous state with the aforementioned rule: should this variable be included in the solution? Excluded from the solution? Left undecided? There is a choice to be made here, which will impact the obtained solution. For example, in problems modeled by a latent LASSO approach (Problem (6.2)):

$$\min_{\boldsymbol{x} \in \mathbb{R}^Q, \mathbf{Z} \in \mathbb{R}^{Q \times |G|}} \tfrac{1}{2} \|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\|_2^2 + \lambda \sum_{i=1}^{|G|} \|\boldsymbol{z}_i\|_2 \quad \text{s.t.} \quad \boldsymbol{x} = \sum_{i=1}^{|G|} \boldsymbol{z}_i,$$

we would typically favour the groups in G_1, meaning that x_1 in Figure 7.2 (left) belongs to S_1: we favour the *activation* of variables. We will denote this choice *activation-first*. The *activation-first* strategy is well suited for problems with small groups which are aggregated together to form a bigger structure, like graph sparsity [Huang et al. 2011]. The other strategy, *exclusion-first*, favours the *exclusion* of variables, as shown in Figure 7.2 (left). This approach, which is the one of [Jenatton et al. 2011], is especially of interest for

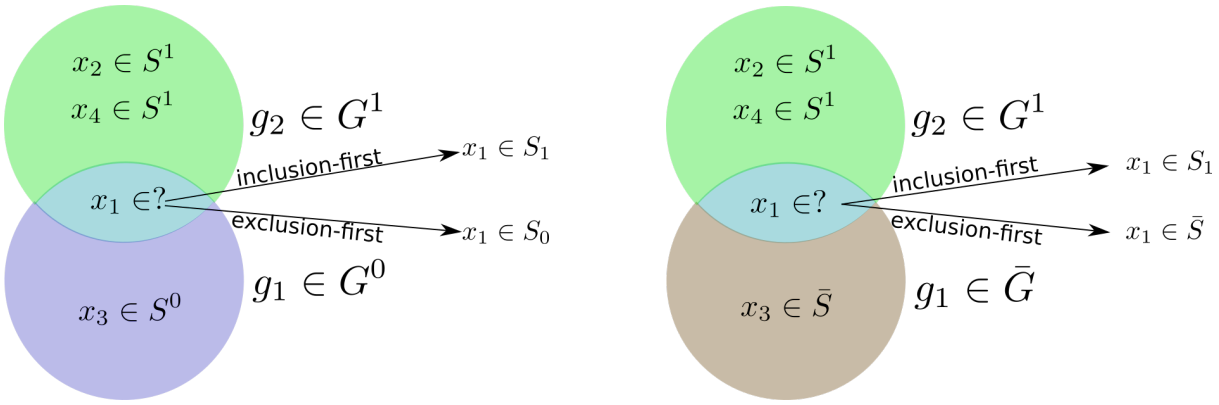


Figure 7.2 – An overlap example with G_1 and G_0 (left), and with G_1 and $\bar{G}$ (right). Taking the group structure of Figure 7.1, we place ourselves in the situation where g_2 is in G_1 and g_1 is either in G_0 (left) or $\bar{G}$ (right). Both groups have the component x_1 in common, therefore it is unclear if x_1 must be zero/non-zero (left), or non-zero/undecided (right). Two choices exist, and we will take the exclusion-first approach, where x_1 is put to 0 (left) and left undecided (right).

hierarchical sparsity [Bach et al. 2012; Jenatton et al. 2011], where some predictors must be active for other predictors to be activable (there are implication constraints), and is modelled with groups of large size and groups of small size.

Definition of the variable supports When favouring the exclusion of variables, the definition of S_0 is clear: it contains the indices of all the variables belonging to groups in G_0: $S_0 := \cup_{g \in G_0} g$. However, there is still an ambiguity in the definition of $\bar{S}$ and S_1. Indeed, as shown in Figure 7.2 (right), if a variable belongs to a group in G_1 and at the same time to a group in $\bar{G}$, should this variable go in $\bar{S}$ or in S_1?

We will make a choice based on properties of the supports $S_1, S_0, \bar{S}$ in the scalar case. First, $\{S_1, S_0, \bar{S}\}$ constitutes a partition of the variables indices. Second, whenever a variable is in S_0 or S_1 in a given node, it remains in the same set in the children nodes.

Looking back at our structured sparsity case, we would like to keep $\{S_1, S_0, \bar{S}\}$ as a partition, which means that the ambiguous variable in Figure 7.2 (right) cannot be at the same time in S_1 and $\bar{S}$. Also, we would like to keep the property that a variable put into S_1 or S_0 in given node will stay at the same place for all its descendants. This means the variable should be put in $\bar{S}$ (right part of Figure 7.2). Indeed, as this variable belongs to an undecided group, this undecided group could be put to G_0 in a child node, therefore the variable will be moved to S_0. To make variables in S_1 stay in S_1 for all descendants, we need these variables to belong *only* to groups in G_1. The mathematical definition of

the variable supports is then:

$$\begin{aligned} S_0 &:= \cup_{g \in G_0} g \\ \bar{S} &:= \cup_{g \in \bar{G}} \setminus S_0 \\ S_1 &:= \cup_{g \in G_1} \setminus \bar{S} \cup S_0 \end{aligned}$$

This way to associate the variables supports $S_1, S_0, \bar{S}$ to a given groups support configuration $G_1, G_0, \bar{G}$ is done in the *branching* step (see Algorithm 8). As such, integrating an approach like the latent-LASSO for overlapping groups (favouring inclusion of groups instead of exclusion like this work) only amounts to implementing another branching operator to get the correct search space. Now that we explicitly defined the search space at hand, we can go to the formulation of bounds.

Algorithm 8 Branching operations for structured sparsity (exclusion-first)

procedure SPLIT($\mathbf{N}(G_1, G_0, \bar{G}, S_1, S_0, \bar{S})$)
  $g \leftarrow$ getBranchingIndex($\mathbf{N}$)
  $G_1^{\text{left}} \leftarrow G_1 \cup g$, $S_1^{\text{left}} \leftarrow S_1$, $\bar{S}^{\text{left}} \leftarrow \bar{S}$
  $G_0^{\text{right}} \leftarrow G_0 \cup g$, $S_0^{\text{right}} \leftarrow S_0$, $\bar{S}^{\text{right}} \leftarrow \bar{S}$
  for $i \in g$ **do**
    if $i \in \bar{S}$ and $i \notin \bigcup_{g \in \bar{G}} g$ and $i \notin \bigcup_{g \in G_0} g$ **then** ▷ i is only in G_1 groups
      $S_1^{\text{left}} \leftarrow S_1^{\text{left}} \cup \{i\}$, $\bar{S}^{\text{left}} \leftarrow \bar{S}^{\text{left}} \setminus \{i\}$ ▷ Left node: move i from $\bar{S}$ to S_1.
    end if
    if $i \in \bar{S}$ **then**
      $S_0^{\text{right}} \leftarrow S_0^{\text{right}} \cup \{i\}$, $\bar{S}^{\text{right}} \leftarrow \bar{S}^{\text{right}} \setminus \{i\}$ ▷ Right node: move i to S_0.
    end if
  end for
  return $[\mathbf{N}^{\text{left}}(G_1^{\text{left}}, G_0, \bar{G}^{\text{left}}, S_1^{\text{left}}, S_0, \bar{S}^{\text{left}}); \mathbf{N}^{\text{right}}(G_1, G_0^{\text{right}}, \bar{G}^{\text{right}}, S_1, S_0^{\text{right}}, \bar{S}^{\text{right}})]$
end procedure

7.4 Bounding operators

With the search space defined in Section 7.2 and refined in Section 7.3 for overlapping groups, we are now ready to express the Problem $(\mathcal{P}_{2+0s})$ involved at a given node. Indeed, given a group partition $G_1, G_0, \bar{G}$ and its corresponding variable partition $S_1, S_0, \bar{S}$, Problem $(\mathcal{P}_{2+0s})$ reads:

$$\min_{\boldsymbol{x}} \ \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \mu \sum_{g \in \bar{G}} \mathbf{1}_{\boldsymbol{x}_g \neq 0} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases}. \qquad (\mathcal{P}_{2+0s}^{\mathbf{N}})$$

This is the structured sparsity equivalent of Problem $(\mathcal{P}^{\mathbf{N}}_{2+0})$:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|S_1| + \mu\|\boldsymbol{x}_{\bar{S}}\|_0 \text{ s.t. } \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = 0 \end{cases} \qquad (\mathcal{P}^{\mathbf{N}}_{2+0})$$

defined in Part I. Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$ is NP-hard due to the term $\sum_{g\in\bar{G}} \mathbf{1}_{\boldsymbol{x}_g\neq 0}$, and we will approximate its minimum using lower and upper bounds. With the variable supports choices made in Section 7.3, we can naturally extend the expressions used in Part I.

7.4.1 Formulating and solving nodes upper bound

For the upper bound, we will use the same trick as in Section 3.4.2: add the constraint $\boldsymbol{x}_{\bar{S}} = \mathbf{0}$, in other words restrict Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$ to variables in S_1 only:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1}\|_2^2 + \mu|G_1| \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \\ \boldsymbol{x}_{\bar{S}} = \mathbf{0} \end{cases}. \qquad (7.1)$$

Indeed, with the definition of $\bar{S}$ used in Section 7.3, groups in $\bar{G}$ contain variables which are either in $\bar{S}$ or in S_0 (variables in S_1 belong to groups which are all in G_1). This means that the combined constraints $\boldsymbol{x}_{S_0} = \mathbf{0}$ and $\boldsymbol{x}_{\bar{S}} = \mathbf{0}$ completely cancel the term $\sum_{g\in\bar{G}} \mathbf{1}_{\boldsymbol{x}_g\neq 0}$ in Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$, which proves that the minimum of Problem (7.1) is an upper bound of the minimum of Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$.

As in the scalar case, Problem (7.1) is a simple box-constrained least-squares problem, for which we can use again the upper bounds optimization procedure described in Section 3.4.2, namely standard active set or interior point methods.

7.4.2 Generic nodes lower bound through convex relaxation

To build lower bounds on Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$, we will use a convex relaxation of the terms $\mathbf{1}_{\boldsymbol{x}_g\neq 0}$. We will use again the link between the absolute value and the indicator function (Inequality (3.13)) to this end:

$$\|\boldsymbol{x}\|_0 = \sum_{i\in\{1..Q\}} \mathbf{1}_{x_i\neq 0} \geq \sum_{i\in\{1..Q\}} \tfrac{|x_i|}{M} = \tfrac{1}{M} \sum_{i\in\{1..Q\}} |x_i| = \tfrac{1}{M}\|\boldsymbol{x}\|_1. \qquad (3.13)$$

Proposition 7.4.1. *For any group $g \neq \emptyset$ and any vector $\boldsymbol{x}_g \in \mathbb{R}^{|g|}$, $M > 0$ and $q \in [1, +\infty]$, we have:*

$$\forall \boldsymbol{x}_g \in [-M, M]^{|g|}, \mathbf{1}_{\boldsymbol{x}_g\neq 0} \geq \frac{1}{M}\frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}}. \qquad (7.2)$$

Proof. The relation trivially holds for $\boldsymbol{x}_g = \mathbf{0}$. For $\boldsymbol{x}_g \neq \mathbf{0}$, we have:

— for $q = +\infty$:

$$\begin{aligned}\boldsymbol{x}_g \in [-M, M]^{|q|} \setminus \mathbf{0} &\iff 0 < \|\boldsymbol{x}_g\|_\infty \leq M \\ &\iff 0 < \frac{\|\boldsymbol{x}_g\|_\infty}{M} \leq 1 = \mathbf{1}_{\boldsymbol{x}_g \neq 0}\end{aligned}$$

— for $q \in \mathbb{R}_+^*$:

$$\begin{aligned}\boldsymbol{x}_g \in [-M, M]^{|g|} \setminus \mathbf{0} &\implies \forall i \in g, 0 \leq |x_i| \leq M \\ &\implies \forall i \in g, 0 \leq |x_i|^q \leq M^q \\ &\implies 0 \leq \sum_{i \in g} |x_i|^q \leq M^q |g| \\ &\iff 0 \leq \left(\sum_{i \in g} |x_i|^q\right)^{1/q} \leq M|g|^{1/q} \\ &\iff 0 \leq \frac{\|\boldsymbol{x}_g\|_q}{M|g|^{1/q}} \leq 1 = \mathbf{1}_{\boldsymbol{x}_g \neq 0}\end{aligned}$$

□

Consequently, we will compute lower bounds on Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$ instances thanks to the following proposition:

Proposition 7.4.2. *For any $q \in [1, +\infty]$, a valid lower bound on the minimum of Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$ is given by the minimum of the following problem:*

$$\min_{\boldsymbol{x} \in \mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\sum_{g \in \bar{G}} \frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases}. \quad (\mathcal{P}^{\mathbf{N}}_{2+1q})$$

Proof. Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$ is obtained from Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$ by replacing $\mu\sum_{g \in \bar{G}} \mathbf{1}_{\boldsymbol{x}_g \neq 0}$ with $\frac{\mu}{M}\sum_{g \in \bar{G}} \frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}}$. By Proposition 7.4.1, we know that $\frac{\mu}{M}\sum_{g \in \bar{G}} \frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}} \leq \mu\sum_{g \in \bar{G}} \mathbf{1}_{\boldsymbol{x}_g \neq 0}$. Consequently, the cost function of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$ is always lower than the cost function of Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$ for the same $\boldsymbol{x}$, therefore the minimum of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$ is lower than the minimum of Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$. □

Chapter 8 will further detail the influence of the choice of q, as well as how to solve Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$ in practice.

7.5 Branching rule

Dividing a node into two children in the structured case fundamentally works the same way as in the scalar case: choose a group still undecided, force it to be part of

the solution for one child, force it to be excluded from the solution for the other child. However, knowing which group should be picked for this process is a question that remains to be answered. In the scalar sparsity case, we use the maximum of amplitude branching rule defined by Problem (3.17): $i := \arg\max_{i \in \bar{S}} |\boldsymbol{x}^{\mathbf{N}}_{\mathrm{LB}}|_i$. We would like to extend this rule to the structured sparsity case. To this end, we must define what is the "amplitude of a group": how can we define the amplitude of a given subvector $\boldsymbol{x}_g$? It turns out several definitions are possible. Indeed, we can see the amplitude as an absolute value, leading quite naturally to the ℓ_1 norm: $\|\boldsymbol{x}_g\|_1$ is then our group amplitude. However, if we think about a complex number c, its amplitude is defined by $\sqrt{\mathfrak{Re}(c)^2 + \mathfrak{Im}(c)^2} = \left\| \begin{matrix} \mathfrak{Re}(c) \\ \mathfrak{Im}(c) \end{matrix} \right\|_2$. In this context, it is more natural to set the group amplitude as $\|\boldsymbol{x}_g\|_2$. Moreover, in the context of groups of varying size, we may want to favour groups with a high amplitude *density*, meaning groups which are both of high amplitude and of small size, leading to normalized versions of the two norms above: either $\frac{\|\boldsymbol{x}_g\|_1}{|g|}$ or $\frac{\|\boldsymbol{x}_g\|_2}{\sqrt{|g|}}$ (see Proposition 7.4.1 for the choice of the normalization term).

A way to define the branching rule is then to reuse the penalty term of the lower bound problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$). As shown in Section 7.4.2, an appropriate normalization is required to get a valid lower approximation of the ℓ_0 term. Our branching rule reads:

$$g = \arg\max_{g \in \bar{G}} \frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}}, \tag{7.3}$$

where $q \in \mathbb{N}^* \cup \{+\infty\}$ defines the convex relaxation in Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$). We will call this branching rule the *maximum normalized* ℓ_q branching rule, it can be seen as an extension of the maximum amplitude branching rule of the scalar case given in Problem (3.17).

7.6 Discussion

Conclusion To propose a working branch-and-bound method in the structured sparsity case, several design choices were made in this chapter, namely:

- explore the space of group supports (thanks to the use of the partition $G_1, G_0, \bar{G}$);
- give priority to the exclusion of a group over its inclusion in the solution, leading to a specific link between the groups partition $\{G_1, G_0, \bar{G}\}$ and the variables partition $\{S_1, S_0, \bar{S}\}$;
- define the upper bound of a node using a problem restricted to the set of active variables, S_1;
- define the lower bound of a node using a $\ell_1 - \ell_q$ mixed-norm optimization problem;
- extend the scalar sparsity branching strategy to the structured case: the maximum normalized ℓ_q rule.

Perspectives A branch-and-bound algorithm has multiple degrees of freedom. On top of this, for the structured sparsity studied here, some choices must be made between several alternatives to avoid ambiguous situations. Consequently, the design choices introduced in this chapter, driven by simplicity and consistency with the scalar sparsity case, could be discussed as well as extended to handle more use cases. To begin with, the current choice for the search space is to give priority to the exclusion of groups: it is exclusion-first. Inclusion-first corresponds to what is done in the Latent-LASSO problem. As this way to structure the support is completely contained in the branching strategy, it is possible to allow both exclusion-first and inclusion-first in the branch-and-bound algorithm, letting the user choose the structuration of the search space. Also, tying together the branching strategy with the lower bound formulation is an intuitive but questionable choice. Carrying numerical studies to see which are the best combinations between branching and lower bounding is an open perspective of this thesis. Finally, the upper bound formulation is a straightforward extension from the scalar sparsity case. However, even in the scalar sparsity case, upper bounds could be refined thanks to some local search algorithm (for example greedy algorithms such as OMP). The idea holds in the structured sparsity case too.

Chapter 8

Formulating and solving structured sparsity lower bound problems in practice

Contents

8.1 Introduction

Building on the branch-and-bound framework of Chapter 7 (page 89), this chapter presents in depth the formulation and resolution of the lower bounds given by Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}_{S_1}\boldsymbol{x}_{S_1}-\mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2+\mu|G_1|+\tfrac{\mu}{M}\sum_{g\in\bar{G}}\frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}} \quad \text{s.t.} \quad \begin{cases}\|\boldsymbol{x}\|_\infty \leq M\\ \boldsymbol{x}_{S_0}=\mathbf{0}\end{cases}, \quad (\mathcal{P}^{\mathbf{N}}_{2+1q})$$

with $q \in \{1, 2, +\infty\}$. The link between Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$) and problems tackled in the literature is studied in Section 8.2. Then, different choices of q in Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$) are considered in Section 8.2. Practical aspects exposed in Section 8.3 will lead to a separation between the case $q = 1$, detailed in Section 8.4, and the other values of q, detailed in Section 8.5. A hybrid formulation dedicated to the case of overlapping groups is then introduced in Section 8.6. Numerical experiments assess the performance of the different formulations in Section 8.7, and the contributions of this chapter are discussed in Section 8.8.

8.2 ℓ_q formulations and links with other works

The lower bound problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$) is a variant of well-known mixed norm problems in the literature [Bourguignon, Carfantan, et al. 2007; Jenatton et al. 2011; Yuan et al. 2006]. In a standard mixed norm, we apply a given norm to groups of unknowns, and then take another norm on the results to get our final value. These two-stage norms are not the only example of mixed norms, in particular 3-stage mixed norms were used in [Gramfort et al. 2012] in the context of MEG imaging. Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$) uses a $\ell_1 - \ell_q$ mixed norm: summing ℓ_q norm terms applied on groups.

The choice of $q = 2$, meaning applying an ℓ_2 norm to each block, has been used and investigated in the literature (see for example [Bourguignon, Carfantan, et al. 2007; Jenatton et al. 2011; Yuan et al. 2006]). If the LASSO is seen as a standard way to promote sparse solutions through a convex problem, $\ell_1 - \ell_2$ is seen as its standard counterpart to promote structured sparse solutions. Group LASSO is related to Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$) where $q = 2$ and the group support partition $G_1, G_0, \bar{G}$ is taken so that $G_1 = G_0 = \emptyset, \bar{G} = \{1, ..., |G|\}$ where $|G|$ is the number of groups (see Section 8.5 for more details about this link).

However, the Group LASSO formulation, with its $\ell_1 - \ell_2$ mixed norm, is not always the best suited formulation. Indeed, in the case of overlapping groups, Group LASSO will retrieve a solution whose zeros will be the union of the inactive groups (groups whose variables are set to zero). This means the non-zero components are the complement of the

union of inactive groups. This corresponds to the example of Figure 7.2 (on page 92) of Chapter 7. In other words, the optimization algorithm chooses groups to be excluded from the support of the solution. In some application contexts [Huang et al. 2011; Obozinski et al. 2011], we are more interested in an algorithm which chooses groups to be *included* in the support of the solution, leading to the Latent-LASSO problem (6.2):

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q, \mathbf{Z}\in\mathbb{R}^{Q\times|G|}} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \lambda\sum_{i=1}^{|G|}\|\boldsymbol{z}_i\|_2 \quad \text{s.t.} \quad \boldsymbol{x}=\sum_{i=1}^{|G|}\boldsymbol{z}_i. \tag{4.2}$$

We can see this problem as solving a Group LASSO on latent vectors $\boldsymbol{z}_i$, and then reconstructing $\boldsymbol{x}$ with the union of groups in $\mathbf{Z}$. Additionally, on the algorithmic side, solving Group LASSO with overlapping groups is computationally demanding (see Section 8.3 for more details), whereas Latent-LASSO is computationally tractable. While Problem (6.2) allows one to model a different context than the one of Group LASSO, its immediate drawback is that the latent formulation doubles the number of variables. In high-dimensional contexts, this augmentation can be expensive.

In Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$), we can of course choose other values of q than $q=2$, since it provides a valid relaxation for all $q\geq 1$. In particular, the $q=\infty$ case has seen some investigation, although this penalty tends to give minimizer with many components of the same magnitude, which could be undesired [Bach et al. 2011].

8.3 Overview of the different formulations

In the remaining of this thesis, we will use $q\in\{1,2,\infty\}$ (which give rise to quite simple optimization problems), which means we will deal with the following mixed norms: $\sum_{g\in\bar{G}}\frac{\|\boldsymbol{x}_g\|_1}{|g|}$, $\sum_{g\in\bar{G}}\frac{\|\boldsymbol{x}_g\|_2}{\sqrt{|g|}}$ and $\sum_{g\in\bar{G}}\|\boldsymbol{x}_g\|_\infty$. These three mixed norms will be compared on a theoretical side as well as on a computational, algorithmic side.

From a theoretical perspective, we can order the quality of the different ℓ_q relaxations with respect to the ℓ_0 term:

Proposition 8.3.1. *For any vector $\boldsymbol{x}_g\in[-M,M]^{|g|}, M>0$, we have:*

$$\frac{\|\boldsymbol{x}_g\|_1}{M|g|}\leq\frac{\|\boldsymbol{x}_g\|_2}{M\sqrt{|g|}}\leq\frac{\|\boldsymbol{x}_g\|_\infty}{M}\leq\mathbf{1}_{\boldsymbol{x}_g\neq 0}. \tag{8.1}$$

Proof. Proposition 7.4.1 already establishes that $\frac{\|\boldsymbol{x}_g\|_\infty}{M}\leq\mathbf{1}_{\boldsymbol{x}_g\neq 0}$. What remains to be proved is that:

$$\frac{\|\boldsymbol{x}_g\|_1}{|g|}\leq\frac{\|\boldsymbol{x}_g\|_2}{\sqrt{|g|}}\leq\|\boldsymbol{x}_g\|_\infty,$$

which comes from the equivalence relations between norms. Indeed, for the first inequality,

we have using Cauchy-Schwarz inequality:

$$\frac{\|\boldsymbol{x}_g\|_1}{|g|} = \sqrt{\left(\sum_{i\in g}|x_i|\frac{1}{|g|}\right)^2} \leq \sqrt{\sum_{i\in g}|x_i|^2 \times \sum_{i\in g}\frac{1}{|g|^2}} = \|\boldsymbol{x}_g\|_2 \times \sqrt{\frac{|g|}{|g|^2}} = \frac{\|\boldsymbol{x}_g\|_2}{\sqrt{|g|}}.$$

For the second inequality, we have:

$$\frac{\|\boldsymbol{x}_g\|_2}{\sqrt{|g|}} = \sqrt{\frac{\sum_{i\in g}|x_i|^2}{|g|}} \leq \sqrt{\frac{\sum_{i\in g}(\max_{i\in g}|x_i|)^2}{|g|}} = \sqrt{\frac{\sum_{i\in g}\|\boldsymbol{x}_g\|_\infty^2}{|g|}} = \sqrt{\frac{|g|\ \|\boldsymbol{x}_g\|_\infty^2}{|g|}} = \|\boldsymbol{x}_g\|_\infty.$$

□

Proposition 8.3.1 tells us that the $\ell_1-\ell_\infty$ relaxation is tighter than the $\ell_1-\ell_2$ relaxation, and the $\ell_1-\ell_1$ relaxation is the worst among all three. An illustration of this ordering is given in Figure 8.1.

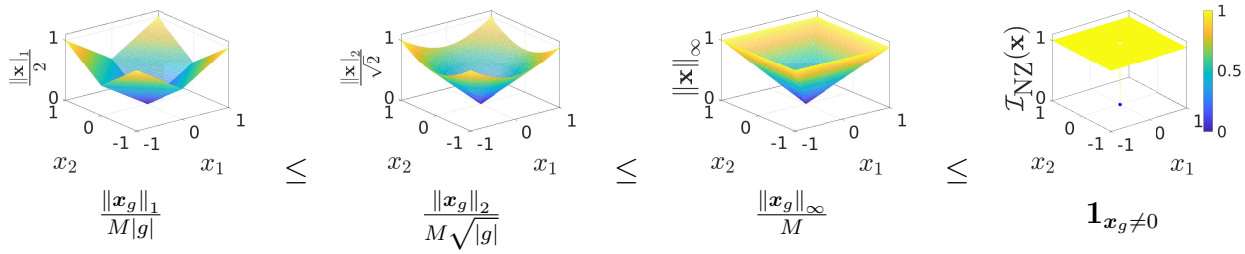


Figure 8.1 – Illustration of the ordering of the different penalties used. The $\ell_1-\ell_1$ (left) penalty is the loosest relaxation of the original ℓ_0 penalty (right).

Consequently, on a theoretical side, we would like to always use the $\ell_1-\ell_\infty$ relaxation to ensure good lower bounds. On top of that, even though the $\ell_1-\ell_2$ is less tight, it still preserves the non-separability of a given group, whereas the $\ell_1-\ell_1$ relaxation destroys the group-wise penalty to get back to a component-wise penalty, leading to poorer quality with respect to the desired structured sparsity prior.

State-of-the-art methods [İ. Bayram 2018; I. Bayram 2011; Mosci et al. 2010] solving mixed norm problems involve a proximal splitting scheme, taking the proximal operator of the penalty, which reads:

$$\boldsymbol{\nu} = \underset{\tau\frac{\mu}{M}\left(I_{[-M,M]^Q}(\cdot)+\sum_{g\in\bar{G}}\frac{\|\cdot_g\|_q}{|g|^{1/q}}\right)}{\text{prox}}(\boldsymbol{x}^k) = \underset{\boldsymbol{x}\in[-M,M]^Q}{\arg\min}\ \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{x}^k\|_2^2 + \tau\tfrac{\mu}{M}\sum_{g\in\bar{G}}\frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}}. \tag{8.2}$$

When groups in $\bar{G}$ are disjoint, this operator is group-separable and can decomposed for all $g\in\bar{G}$ as:

$$\forall g\in\bar{G}, \boldsymbol{\nu}_g = \underset{\boldsymbol{x}\in[-M,M]^Q}{\arg\min}\ \tfrac{1}{2}\|\boldsymbol{x}_g-\boldsymbol{x}_g^k\|_2^2 + \tau\tfrac{\mu}{M}\frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}}. \tag{8.3}$$

This minimization problem is similar to the subproblems solved by a block coordinate descent algorithm, where the original problem is solved by optimizing a sequence of smaller subproblems (see Section 5.2.3 on page 57 and 8.5.2 on page 104 for more details). Algorithms can be designed to solve this proximal operator. However, if groups are overlapping, we cannot decompose the proximal operator (8.2) this way, and we must stick with the sum of groups. If $q > 1$, the ℓ_q norm is not separable, and the proximal operator is not computationally tractable. If $q = 1$, the proximal operator rewrites as a separable weighted ℓ_1-norm problem whose analytical solution is known. This means that in the case of overlapping groups, we do not have a straightforward descent algorithm to compute the minimum of the $\ell_1 - \ell_2$ and $\ell_1 - \ell_\infty$ relaxations, this minimum being needed to give a valid lower bound on the ℓ_0 Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$:

$$\min_{\boldsymbol{x}} \ \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \mu \sum_{g\in\bar{G}} \mathbf{1}_{\boldsymbol{x}_g \neq 0} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases}. \qquad (\mathcal{P}^{\mathbf{N}}_{2+0s})$$

Consequently, we will use these formulations in the case of disjoint groups only. In the case of overlapping groups, we will resort to the $\ell_1 - \ell_1$ relaxation, detailed in Section 8.4. In the case of disjoint groups, we will use the algorithm designed to solve the $\ell_1 - \ell_2$ and $\ell_1 - \ell_\infty$ formulations which is detailed in Section 8.5.

8.4 Separable formulation: $\ell_1 - \ell_1$ relaxation for overlapping and disjoint groups cases

8.4.1 Motivation

The $\ell_1 - \ell_1$ relaxation somehow destroys the structure of the problem: we no longer deal with groups of variables, we deal with individual variables. This means the minimizer can be expected to be of poor quality with respect to the structured sparsity prior. As Proposition 8.3.1 shows, this is the loosest relaxation compared to $\ell_1 - \ell_2$ and $\ell_1 - \ell_\infty$. However, the $\ell_1 - \ell_1$ relaxation can be solved with overlapping groups quickly, and we can reuse almost directly the algorithms described for the scalar sparsity case in Section 3.3 (on page 24).

8.4.2 Multiple weights and multiple bounds variants

Let us recall Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$ in the $q = 1$ case:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \ \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M} \sum_{g\in\bar{G}} \frac{\|\boldsymbol{x}_g\|_1}{|g|} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases}. \qquad (\mathcal{P}^{\mathbf{N}}_{2+11})$$

We know that each group in $\bar{G}$ contains variables which are either in S_0, in that case they value 0, or in $\bar{S}$. This means we can reorder the expression of the mixed norm in the following way:

$$\sum_{g\in\bar{G}} \frac{\|\boldsymbol{x}_g\|_1}{|g|} = \sum_{g\in\bar{G}} \sum_{i\in\bar{S}\cap g} \frac{|x_i|}{|g|} = \sum_{i\in\bar{S}} \sum_{g\in\bar{G}:i\in g} \frac{|x_i|}{|g|} = \sum_{i\in\bar{S}} |x_i| \left(\sum_{g\in\bar{G}:i\in g} \frac{1}{|g|} \right).$$

We denote $\alpha_i := \sum_{g\in\bar{G}:i\in g} \frac{1}{|g|}$. We reach a multiple-weight ℓ_1-norm problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\sum_{i\in\bar{S}} \alpha_i|x_i| \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases}. \quad (\mathcal{P}^{\mathbf{N}\text{weight}}_{2+11})$$

The quantities $\alpha_i = \sum_{g\in\bar{G}:i\in g} \frac{1}{|g|}$ depend on the configuration of $\bar{G}$ and $\bar{S}$, so they must be updated at each node. They are counting the number of free groups a given variable is part of. That is, with groups of size 4, $\alpha_i = \frac{1}{4}$ means a variable which is part of only one undecided group, whereas $\alpha_i = 1$ means it is part of 4 such groups, which are therefore overlapping.

Problem $(\mathcal{P}^{\mathbf{N}\text{weight}}_{2+11})$ can be solved by several standard algorithms through little variations from the scalar sparsity case. Indeed, Problem $(\mathcal{P}^{\mathbf{N}\text{weight}}_{2+11})$ differs from Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$ (on page 37) only on the variable weights before the absolute value terms. Proximal, coordinate descent and active-set algorithms can be adapted in a straightforward way to solve this problem. However, the homotopy continuation algorithm, which enjoys particularly good performance in the scalar case (see Chapter 5 and [Bach et al. 2011]), is based on the property that the LASSO problem is piecewise linear in the regularization parameter. To adapt this algorithm, the regularization parameter must be a scalar, not a vector. We can make a variable change to get back to a single-weight ℓ_1 norm problem. Let us define vector $\boldsymbol{z} \in \mathbb{R}^Q$ such that:

$$\begin{aligned} \forall i \in S_1,\ & z_i = x_i, \\ \forall i \in \bar{S},\ & z_i = x_i\alpha_i, \\ \forall i \in S_0,\ & z_i = x_i = 0. \end{aligned}$$

We can now replace $\alpha_i|x_i|$ by $|z_i|$. We also need to replace the different products $\boldsymbol{a}_i x_i$ appearing in the data-fitting term. We define $\mathcal{A}$ as:

$$\begin{aligned} \forall i \in S_1,\ & \mathcal{A}_i = \boldsymbol{a}_i, \\ \forall i \in \bar{S},\ & \mathcal{A}_i = \boldsymbol{a}_i/\alpha_i, \\ \forall i \in S_0,\ & \mathcal{A}_i = \boldsymbol{a}_i, \end{aligned}$$

such that $\mathcal{A}_i z_i = \boldsymbol{a}_i x_i$. Problem $(\mathcal{P}^{\mathbf{N}\text{weight}}_{2+11})$ rewrites as:

$$\min_{\boldsymbol{z}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathcal{A}_{S_1}\boldsymbol{z}_{S_1} - \mathcal{A}_{\bar{S}}\boldsymbol{z}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\|\boldsymbol{z}_{\bar{S}}\|_1 \text{ s.t. } \begin{cases} \boldsymbol{z}_{S_0} = \mathbf{0} \\ \|\boldsymbol{z}_{S_1}\|_\infty \leq M \\ \forall i \in \bar{S}, |z_i| \leq M\alpha_i \end{cases} . \quad (\mathcal{P}^{\mathbf{N}\text{bound}}_{2+11})$$

Problem $(\mathcal{P}^{\mathbf{N}\text{bound}}_{2+11})$ can be tackled by the homotopy continuation algorithm adapted from Chapter 5.

8.5 Disjoint groups formulations: $\ell_1 - \ell_2$ and $\ell_1 - \ell_\infty$ relaxations

8.5.1 Applicability

As mentioned in Section 8.3, Problem $(\mathcal{P}^{\mathbf{N}}_{2+12})$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\sum_{g\in\bar{G}} \frac{\|\boldsymbol{x}_g\|_2}{\sqrt{|g|}} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases} \quad (\mathcal{P}^{\mathbf{N}}_{2+12})$$

and Problem $(\mathcal{P}^{\mathbf{N}}_{2+1\infty})$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\sum_{g\in\bar{G}} \|\boldsymbol{x}_g\|_\infty \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases} \quad (\mathcal{P}^{\mathbf{N}}_{2+1\infty})$$

can only be used in the case of disjoint groups, otherwise we do not have any simple and efficient optimization algorithm available. In both cases, we need an algorithm able to tackle the mixed norm term, which is non-differentiable. This is the topic of the next section.

8.5.2 Resolution: Iterative Group Descent algorithm

Problems $(\mathcal{P}^{\mathbf{N}}_{2+12})$ and $(\mathcal{P}^{\mathbf{N}}_{2+1\infty})$ cannot be solved by a homotopy continuation method. In the scalar case, the homotopy continuation method and the coordinate descent algorithm are the best performing algorithms (see Chapter 5). Consequently, we resort to a block coordinate descent algorithm called *Iterative Group Descent*, abbreviated in the following as IGD, which is a block-coordinate descent algorithm. The general optimization scheme looks as Algorithm 9. Algorithm 9, which is a particularization of [Tseng 2001] for Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$, sequentially solves subproblems where only some components are optimized. For variables in S_1, we solve scalar subproblems where only one variable is updated (step 6). For variables in $\bar{S}$, we solve subproblems where only variables belong-

Algorithm 9 Iterative Group Descent algorithm for $(\mathcal{P}^{\mathbf{N}}_{2+1q})$

1: **procedure** IGD($\boldsymbol{y}$, $\mathbf{A}$, μ, $\boldsymbol{x}^0$, G_1, $\bar{G}$, S_1, $\bar{S}$)
2: $\quad\boldsymbol{x} \leftarrow \boldsymbol{x}^0$
3: $\quad$**while** not convergence **do**
4: $\quad\quad$**for** $i \in S_1$ **do**
5: $\quad\quad\quad\boldsymbol{r}_i \leftarrow \boldsymbol{y} - \mathbf{A}_{S_1\setminus\{i\}}\boldsymbol{x}_{S_1\setminus\{i\}} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}$
6: $\quad\quad\quad x_i \leftarrow \arg\min_{v\in[-M,M]} \frac{1}{2}\|\boldsymbol{r}_i - \boldsymbol{a}_i v\|_2^2$
7: $\quad\quad$**end for**
8: $\quad\quad$**for** $g \in \bar{G}$ **do**
9: $\quad\quad\quad\boldsymbol{r}_g \leftarrow \boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}\setminus g}\boldsymbol{x}_{\bar{S}\setminus g}$
10: $\quad\quad\quad\boldsymbol{x}_g \leftarrow \arg\min_{\boldsymbol{\nu}\in[-M,M]^{|g|}} \frac{1}{2}\|\boldsymbol{r}_g - \mathbf{A}_g\boldsymbol{\nu}\|_2^2 + \frac{\mu}{M}\frac{\|\boldsymbol{\nu}\|_q}{|g|^{1/q}}$
11: $\quad\quad$**end for**
12: $\quad$**end while**
13: $\quad$**return** $\boldsymbol{x}$
14: **end procedure**

ing to one group of $\bar{G}$ are updated (step 10). In both cases, subproblems are done using a residual expression (step 5 and 9), where the contribution of the currently optimized variables is removed. One question is then if this algorithm doable in practice or not. In other words, are the steps 6 and 10 doable in practice ? It turns out that the S_1-specific step 6 is easy to implement:

Proposition 8.5.1. *Step 6 of Algorithm 9 is computed as:*

$$x_i \leftarrow \Pi_{[-M,M]}\left(\boldsymbol{a}_i^T\boldsymbol{r}_i\right), \quad \textit{with } \Pi_{[-M,M]} \textit{ the projection operator on } [-M,M].$$

The proof of Proposition 8.5.1 is given in Appendix A.3 (page 166). An illustration giving a sense of the proof is given in Figure 8.2.

However, in Algorithm 9, Step 10:

$$\boldsymbol{x}_g \leftarrow \underset{\boldsymbol{\nu}\in[-M,M]^{|g|}}{\arg\min} \frac{1}{2}\|\boldsymbol{r}_g - \mathbf{A}_g\boldsymbol{\nu}\|_2^2 + \frac{\mu}{M}\frac{\|\boldsymbol{\nu}\|_q}{|g|^{1/q}}$$

has no analytical solution, due to the linear mixing operator $\mathbf{A}_g$. This means that step 10 will be computed approximately, and at the global scale the convergence of Algorithm 9 will be difficult to ensure. This convergence is however needed to get a valid lower bound on the ℓ_0-penalized Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$.

Indeed, the convergence proof of a Block-Coordinate Descent algorithm, available in [Tseng 2001], relies on the fact that both steps 6 and 10 are performed exactly. In the literature, the spacer step argument of [Bertsekas 1982] is a classical tool for handling approximated steps. However, this does not solve our issue. Indeed, the spacer step

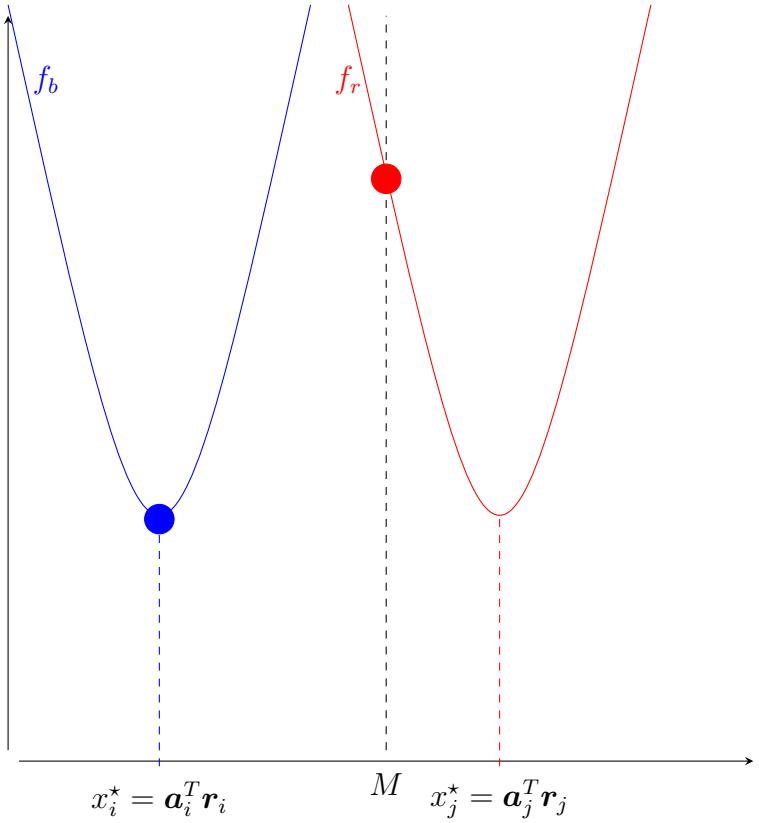


Figure 8.2 – An illustration giving a sketch of a proof for Proposition 8.5.1. Given two functions $f_b(x_i) := \frac{1}{2}\|\boldsymbol{r}_i - \boldsymbol{a}_i x_i\|_2^2$, in blue, and $f_r(x_j) := \frac{1}{2}\|\boldsymbol{r}_j - \boldsymbol{a}_j x_j\|_2^2$, in red, such that the minimizer $x_i^\star$ of f_b lies inside the feasible domain $x_i \in [-M, M]$, whereas the minimizer $x_j^\star$ of f_r violates the constraint $x_j \in [-M, M]$. Graphically, the minimizer of f_b inside the domain $[-M, M]$ is $x_i^\star$, while the minimizer of f_r inside the domain $[-M, M]$ is M.

argument tells us that we can insert into a convergent algorithm, using proved convergent steps, any number of arbitrary approximate steps and still have a convergent algorithm, as long as these approximate steps do not increase the objective function. While this is useful to accelerate a convergent algorithm by inserting fast approximate steps, our problem here is that we cannot construct the proved convergent steps: we only know how to make approximate ones. This means we do not have any theoretical convergence guarantee as long as step 10 is performed approximately, which looks rather mandatory as no analytical solution is known. In our case, we will solve approximately step 10 using a Davis-Yin proximal algorithm [D. Davis et al. 2015] (see Algorithm 10), handling separately the data-fitting term, the ℓ_q penalty and the box constraint. We will construct our approximate steps by using only one iteration of the Davis-Yin scheme: we apply only one iteration of the proximal algorithm, and then update variables in the next group in $\bar{G}$. This algorithm is called in Step 10 of Algorithm 9 successively for each group of

Algorithm 10 Davis-Yin proximal algorithm applied to Step 10 of Algorithm 9.

1: **procedure** DavisYin($\boldsymbol{y}$, $\mathbf{A}$, $\lambda = \mu/M$, $\boldsymbol{x}^0$, $g \in \bar{G}$, θ, ρ, q)
2: $\quad\boldsymbol{x} \leftarrow \boldsymbol{x}^0$
3: $\quad\boldsymbol{s} \leftarrow \boldsymbol{x}^0$
4: $\quad$**while** not convergence **do**
5: $\quad\quad\boldsymbol{e} \leftarrow \boldsymbol{y} - \mathbf{A}\boldsymbol{x}$
6: $\quad\quad\boldsymbol{\gamma} \leftarrow -\mathbf{A}_g^T\boldsymbol{e}$
7: $\quad\quad\boldsymbol{s} \leftarrow \boldsymbol{s} + \rho\left(\mathrm{prox}_{\theta\lambda\|.\|_q}(2\boldsymbol{x}_g - \boldsymbol{s} - \theta\boldsymbol{\gamma}) - \boldsymbol{x}_g\right)$
8: $\quad\quad\boldsymbol{x}_g \leftarrow \Pi_{[-M,M]^{|g|}}(\boldsymbol{s})$
9: $\quad$**end while**
10: $\quad$**return** $\boldsymbol{x}$
11: **end procedure**

variables $\boldsymbol{x}_g$, $g \in \bar{G}$. In practice, the initial point $\boldsymbol{x}^0$ is set to the primal point of the current iteration of Algorithm 9, which is most probably different from a trivial setting such as $\mathbf{0}$.

In order to make an efficient algorithm in practice while still having some hint about the solution quality, a different approach is used according to the relaxation at hand: either $\ell_1 - \ell_2$ or $\ell_1 - \ell_\infty$, that we detail hereafter.

$\ell_1 - \ell_\infty$ relaxation: dual at the rescue Fortunately, in the disjoint group case, the convergence of Algorithm 9 when solving Problem ($\mathcal{P}^{\mathbf{N}}_{2+1\infty}$) can be monitored by using the duality gap. Indeed, as Proposition 8.5.2 shows, the dual problem is available using a similar method than in Section 5.2.1 (page 54).

Proposition 8.5.2. *For the disjoint groups case, the dual problem of Problem* ($\mathcal{P}^{\mathbf{N}}_{2+1\infty}$)

reads:

$$\begin{aligned}\max_{\boldsymbol{w}\in\mathbb{R}^N} D(\boldsymbol{w}) := &-\tfrac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2)\\ &-M\left(\|\mathbf{A}_{S_1}^T\boldsymbol{w}\|_1 + \sum_{g\in\bar{G}}[\|\mathbf{A}_g^T\boldsymbol{w}\|_1 - \tfrac{\mu}{M}]_+\right) \qquad (\mathcal{D}^{\mathbf{N}}_{2+1\infty})\\ &+\mu|G_1|.\end{aligned}$$

Proof. The primal problem $(\mathcal{P}^{\mathbf{N}}_{2+1\infty})$ reads:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} f(\mathbf{A}\boldsymbol{x}) + h(\boldsymbol{x}) \tag{8.4}$$

with $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{x}\|_2^2$ and $h(\boldsymbol{x}) = \mu|G_1| + \frac{\mu}{M}\|\boldsymbol{x}\|_\infty + I_{[-M,M]^Q}(\boldsymbol{x}) + I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0})$. Using the Fenchel-Rockafellar theorem ([Rockafellar 1970], theorem 31.2), the dual problem reads:

$$\max_{\boldsymbol{w}\in\mathbb{R}^N} -f^*(\boldsymbol{w}) - h^*(-\mathbf{A}^T\boldsymbol{w}) \tag{8.5}$$

with f^* and g^* the Fenchel conjugates of f and g respectively. Function f^* reads simply $f^*(\boldsymbol{w}) = \frac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2)$. In the case of disjoint groups, we have $S_1 = \bigcup_{g\in G_1} g$, $\bar{S} = \bigcup_{g\in\bar{G}} g$, $S_0 = \bigcup_{g\in G_0} g$, therefore $h^*(\boldsymbol{u}) = \sup_{\boldsymbol{x}\in\mathbb{R}^Q} \boldsymbol{u}^T\boldsymbol{x} - h(\boldsymbol{x}) = h^*_{S_1}(\boldsymbol{u}) + h^*_{S_0}(\boldsymbol{u}) + h^*_{\bar{G}}(\boldsymbol{u})$ is separable into three blocks:

— the S_1 block: $h^*_{S_1}(\boldsymbol{u}) := \sup_{\boldsymbol{x}_{S_1}\in[-M,M]^{|S_1|}} \boldsymbol{u}_{S_1}^T\boldsymbol{x}_{S_1} - \mu|G_1|$,

— the S_0 block: $h^*_{S_0}(\boldsymbol{u}) := 0$,

— and the $\bar{G}$ block: $h^*_{\bar{G}}(\boldsymbol{u}) := \sum_{g\in\bar{G}} \sup_{\boldsymbol{x}_g\in\mathbb{R}^{|g|}} s_g(\boldsymbol{x}_g)$, with $s_g(\boldsymbol{x}_g) := \boldsymbol{u}_g^T\boldsymbol{x}_g - \frac{\mu}{M}\|\boldsymbol{x}_g\|_\infty - I_{[-M,M]^{|g|}}(\boldsymbol{x}_g)$.

The term for the S_1 block has already been studied in Section 5.2.1, and gives rise to the term $h^*_{S_1}(\boldsymbol{u}) = M\sum_{i\in S_1}|u_i| - \mu|G_1| = M\|\boldsymbol{u}_{S_1}\|_1 - \mu|G_1|$. For the $\bar{G}$ block, using the subdifferential of the ℓ_∞ norm (see Appendix B.1 on page 168), the first-order optimality conditions give us for each supremum operator, using $\boldsymbol{x}_g$ as the optimal point:

Case 1: $\boldsymbol{x}_g = \mathbf{0}$

$$\begin{aligned}\boldsymbol{x}_g = \operatorname*{arg\,max}_{\boldsymbol{x}_g\in\mathbb{R}^{|g|}} s_g(\boldsymbol{x}_g) &\iff \mathbf{0}\in\{\boldsymbol{u}_g\} - \tfrac{\mu}{M}\{\boldsymbol{v}\in\mathbb{R}^{|g|} \mid \|\boldsymbol{v}\|_1 \le 1\} - \{\mathbf{0}\}\\ &\implies \|\boldsymbol{u}_g\|_1 \le \tfrac{\mu}{M}\end{aligned}$$

and we have $s_g(\boldsymbol{x}_g) = \mathbf{0}$.

Case 2: $\boldsymbol{x}_g \neq \mathbf{0}$

$$
\begin{aligned}
&\boldsymbol{x}_g = \underset{\boldsymbol{x}\in\mathbb{R}^{|g|}}{\arg\max}\, s_g(\boldsymbol{x}) \\
\iff\ & 0 \in \{\boldsymbol{u}_g\} - \tfrac{\mu}{M}\{\boldsymbol{v}\in\mathbb{R}^{|g|} \mid \|\boldsymbol{v}\|_1 = 1, \boldsymbol{v}^T\boldsymbol{x}_g = \|\boldsymbol{x}_g\|_\infty\} \\
&\qquad - \left\{\boldsymbol{z}\in\mathbb{R}^{|g|} \mid \forall i\in\{1,..,|g|\}, z_i \in \begin{cases} [0,+\infty[& \text{if } (\boldsymbol{x}_g)_i = M \\ \{0\} & \text{if } (\boldsymbol{x}_g)_i \in]-M, M[\\ ]-\infty, 0] & \text{if } (\boldsymbol{x}_g)_i = -M \end{cases}\right\} \\
\implies\ & \|\boldsymbol{u}_g\|_1 \in \{\tfrac{\mu}{M}\} + \begin{cases} [0,+\infty[& \text{if } \|\boldsymbol{x}_g\|_\infty = M \\ 0 & \text{if } \|\boldsymbol{x}_g\|_\infty < M \end{cases}.
\end{aligned}
$$

This means that $\boldsymbol{x}_g \neq \mathbf{0} \implies \|\boldsymbol{u}_g\|_1 \geq \frac{\mu}{M}$. Now, if we look at the supremum value of $s_g(\boldsymbol{x}_g)$, we have:

$$
\sup_{\boldsymbol{x}_g\in[-M,M]^{|g|}} \boldsymbol{u}_g^T\boldsymbol{x}_g - \tfrac{\mu}{M}\|\boldsymbol{x}\|_\infty = \sup_{\boldsymbol{x}_g\in[-M,M]^{|g|}} \|\boldsymbol{x}_g\|_\infty \left(\boldsymbol{u}_g^T \tfrac{\boldsymbol{x}_g}{\|\boldsymbol{x}_g\|_\infty} - \tfrac{\mu}{M}\right).
$$

$\|\boldsymbol{x}_g\|_\infty$ acts as a scaling here. The quantity to maximize is then $\boldsymbol{u}_g^T \frac{\boldsymbol{x}_g}{\|\boldsymbol{x}_g\|_\infty}$. The maximum of this quantity is reached when $\frac{\boldsymbol{x}_g}{\|\boldsymbol{x}_g\|_\infty} = \text{sign}(\boldsymbol{u}_g)$, meaning $\boldsymbol{u}_g^T \frac{\boldsymbol{x}_g}{\|\boldsymbol{x}_g\|_\infty} = \|\boldsymbol{u}_g\|_1 \geq \frac{\mu}{M}$, implying $\boldsymbol{u}_g^T \frac{\boldsymbol{x}_g}{\|\boldsymbol{x}_g\|_\infty} - \frac{\mu}{M} \geq 0$. If $\|\boldsymbol{u}_g\|_1 = \frac{\mu}{M}$, the supremum of s_g is 0, if $\|\boldsymbol{u}_g\|_1 > \frac{\mu}{M}$, then the supremum of s_g will be reached when $\|\boldsymbol{x}_g\|_\infty = M$ (higher values of the ℓ_∞-norm are forbidden by the constraint), and this value is:

$$
\sup_{\boldsymbol{x}_g\in[-M,M]^{|g|}} \|\boldsymbol{x}_g\|_\infty \left(\boldsymbol{u}_g^T \tfrac{\boldsymbol{x}_g}{\|\boldsymbol{x}_g\|_\infty} - \tfrac{\mu}{M}\right) = M\left(\|\boldsymbol{u}_g\|_1 - \tfrac{\mu}{M}\right).
$$

Summing up, we have:

— if $\boldsymbol{x}_g = \mathbf{0}$, then $\|\boldsymbol{u}_g\|_1 \leq \frac{\mu}{M}$ and $s_g(\boldsymbol{x}_g) = 0$,

— if $\boldsymbol{x}_g \neq \mathbf{0}$, then $\|\boldsymbol{u}_g\|_1 \geq \frac{\mu}{M}$ and $s_g(\boldsymbol{x}_g) = M(\|\boldsymbol{u}_g\|_1 - \frac{\mu}{M})$.

When we reverse the implications between $\boldsymbol{x}_g$ and $\boldsymbol{u}_g$, we have:

— $\|\boldsymbol{u}_g\|_1 > \frac{\mu}{M} \implies \neg(\boldsymbol{x}_g = \mathbf{0}) \iff \boldsymbol{x}_g \neq \mathbf{0} \implies s_g(\boldsymbol{x}_g) = M(\|\boldsymbol{u}_g\|_1 - \frac{\mu}{M})$.

— $\|\boldsymbol{u}_g\|_1 < \frac{\mu}{M} \implies \neg(\boldsymbol{x}_g \neq \mathbf{0}) \iff \boldsymbol{x}_g = \mathbf{0} \implies s_g(\boldsymbol{x}_g) = 0$.

— $\|\boldsymbol{u}_g\|_1 = \frac{\mu}{M} \implies s_g(\boldsymbol{x}_g) = 0$.

Aggregating all cases gives:

$$
\sup_{\boldsymbol{x}} s_g(\boldsymbol{x}) = M[\|\boldsymbol{u}_g\|_1 - \tfrac{\mu}{M}]_+. \tag{8.6}
$$

Finally, function h^* reads:

$$\begin{aligned}
h^*(\boldsymbol{u}) &= \sup_{\boldsymbol{x}\in\mathbb{R}^Q} \boldsymbol{u}^T\boldsymbol{x} - \mu|G_1| - \tfrac{\mu}{M}\|\boldsymbol{x}\|_\infty - I_{[-M,M]^Q}(\boldsymbol{x}) - I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0}) \\
&= \underbrace{M\|\boldsymbol{u}_{S_1}\|_1 - \mu|G_1|}_{h^*_{S_1}(\boldsymbol{u})} + \underbrace{\sum_{g\in\bar{G}} \sup_{\boldsymbol{x}\in\mathbb{R}^{|g|}} \boldsymbol{u}_g^T\boldsymbol{x}_g - \tfrac{\mu}{M}\|\boldsymbol{x}_g\|_\infty - I_{[-M,M]^{|g|}}(\boldsymbol{x})}_{h^*_{\bar{G}}(\boldsymbol{u})} \\
&= M\|\boldsymbol{u}_{S_1}\|_1 - \mu|G_1| + \sum_{g\in\bar{G}} \underbrace{M[\|\boldsymbol{u}_g\|_1 - \tfrac{\mu}{M}]_+}_{\text{from (8.6)}} \\
&= M\left(\|\boldsymbol{u}_{S_1}\|_1 + \sum_{g\in\bar{G}}[\|\boldsymbol{u}_g\|_1 - \tfrac{\mu}{M}]_+\right) - \mu|G_1|, \\
\implies h^*(-\mathbf{A}^T\boldsymbol{w}) &= M\left(\|\mathbf{A}_{S_1}^T\boldsymbol{w}\|_1 + \sum_{g\in\bar{G}}[\|\mathbf{A}_g^T\boldsymbol{w}\|_1 - \tfrac{\mu}{M}]_+\right) - \mu|G_1|.
\end{aligned}$$

□

The objective function of Problem $(\mathcal{D}^{\mathbf{N}}_{2+1\infty})$ will be used to get valid lower bounds on Problem $(\mathcal{P}^{\mathbf{N}}_{2+0s})$. Let's denote by $P(\boldsymbol{x})$ the objective function of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1\infty})$ applied at the point $\boldsymbol{x}$, and by $D(\boldsymbol{w})$ the objective function Problem $(\mathcal{D}^{\mathbf{N}}_{2+1\infty})$ applied at point $\boldsymbol{w}$. We know that $D(\boldsymbol{w}) \leq P(\boldsymbol{x})\ \forall(\boldsymbol{x},\boldsymbol{w}) \in [-M,M]^Q \times \mathbb{R}^N$, which means that each dual value $D(\boldsymbol{w})$ is a valid lower bound for our problem: $D(\boldsymbol{w}) \leq P(\boldsymbol{x}^\star) = \text{lb}^{\mathbf{N}}$ holds. Strong duality also holds: at optimality, we have $D(\boldsymbol{w}^\star) = P(\boldsymbol{x}^\star)$. This allows us to ensure an empirical convergence. More precisely, even though we cannot ensure that Algorithm 9 converges to the minimum value of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1\infty})$, we can track the duality gap $P(\boldsymbol{x}) - D(\boldsymbol{w})$ and stop the algorithm by monitoring this gap (assuming this duality gap vanishes to zero in practice).

$\ell_1 - \ell_2$ relaxation: it is all about approximation In the $\ell_1 - \ell_2$ case, we do not have access to an analytical formulation of the solution of the dual problem. Indeed, computing the Fenchel conjugate of the regularization term corresponds to solving Problem (8.7):

$$g^\star(\boldsymbol{u}) = \sup_{\boldsymbol{x}\in[-M,M]^Q} \boldsymbol{u}^T\boldsymbol{x} - \tfrac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_2 - I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0}), \tag{8.7}$$

for which no analytical solution could be found. This means we do not have any way to check the convergence of Algorithm 9 in the $\ell_1 - \ell_2$ case, neither by a convergence proof nor by monitoring the duality gap. Consequently, we resort to some heuristic tuning to make the algorithm converge in practice. We will emulate a "convergent" step by using more iterations of the Davis-Yin proximal algorithm [D. Davis et al. 2015]. The intent is to get sufficiently close to the minimizer of the subproblem in step 10 of Algorithm 9 so that the algorithm does converge properly, even though we have no way to monitor

its convergence in practice. Through empirical simulations, using 10 iterations of the Davis-Yin proximal algorithm is good enough to get bounds which are in-between the ones provided by the $\ell_1 - \ell_1$ and the $\ell_1 - \ell_\infty$ relaxations.

8.6 Hybrid $\ell_1 - (\ell_1 + \ell_\infty)$ formulation for the overlapping group case

8.6.1 Motivation

Section 8.3 highlighted the fact that for overlapping groups, we do not have any satisfactory way to solve the $\ell_1 - \ell_2$ and $\ell_1 - \ell_\infty$ formulations. This issue arises with overlapping groups in $\bar{G}$, which is the set of undecided groups. Overlapping groups in G_1 and G_0 are not problematic, and groups in $\bar{G}$ which are overlapping only with groups in G_1 and G_0 are also fine.

Moreover, to divide a node, a group from $\bar{G}$ is taken and placed in G_1 (left child) or G_0 (right child). Consequently, children nodes have less groups in $\bar{G}$, so potentially less overlapping groups in $\bar{G}$. Therefore, we propose a hybrid formulation, which considers a separable $(\ell_1 - \ell_1)$ penalty on the overlapping groups of $\bar{G}$, and a non-separable penalty $(\ell_1 - \ell_\infty)$ on the disjoint groups of $\bar{G}$ only [1].

8.6.2 Formulation

The proposed hybrid formulation, trying to get the best of both worlds between the ability of the $\ell_1 - \ell_1$ formulation to deal with overlapping groups and the relaxation quality of the $\ell_1 - \ell_\infty$ formulation, reads:

$$\begin{aligned} \mathrm{LB}^{\mathrm{hyb}} = \min_{\boldsymbol{x} \in \mathbb{R}^Q} \; & \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M} \sum_{g \in \bar{G}^{\mathrm{over}}} \frac{\|\boldsymbol{x}_g\|_1}{|g|} + \tfrac{\mu}{M} \sum_{g \in \bar{G}^{\mathrm{dis}}} \|\boldsymbol{x}_g\|_\infty \\ \text{s.t.} \; & \begin{cases} \boldsymbol{x}_{S_0} = 0 \\ \|\boldsymbol{x}\|_\infty \leq M \end{cases} . \end{aligned} \tag{8.8}$$

1. the non-separable penalty is chosen to be the $\ell_1 - \ell_\infty$ penalty and not the $\ell_1 - \ell_2$ penalty, for the $\ell_1 - \ell_\infty$ relaxation quality (see Section 8.3) and the ability to monitor the convergence of the algorithm solving it (see Section 8.5.2)

where $\bar{G}^{\text{over}}$ is the set of overlapping groups in $\bar{G}$, and $\bar{G}^{\text{dis}}$ the set of disjoint groups in $\bar{G}$. Formally, we have:

$$\begin{aligned}\bar{G} = \bar{G}^{\text{over}} \cup \bar{G}^{\text{dis}}, \bar{G}^{\text{over}} \cap \bar{G}^{\text{dis}} = \emptyset,\\ \bar{G}^{\text{over}} = \{g \in \bar{G} \mid \exists g' \in \bar{G} \setminus \{g\}, g \cap g' \neq \emptyset\},\\ \bar{G}^{\text{dis}} = \{g \in \bar{G} \mid \forall g' \in \bar{G} \setminus \{g\}, g \cap g' = \emptyset\}.\end{aligned}$$

This hybrid formulation mixes an $\ell_1 - \ell_1$ term for overlapping groups (those in $\bar{G}^{\text{over}}$), and an $\ell_1 - \ell_\infty$ term for disjoint groups (those in $\bar{G}^{\text{dis}}$). We will call this formulation $\ell_1 - (\ell_1 + \ell_\infty)$. For the $\ell_1 - \ell_1$ term, we can use the same reordering as in Section 8.4.2. Using a multiple-weight formulation for the $\ell_1 - \ell_1$ term, we get:

$$\begin{aligned}\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\sum_{i\in\bar{S}^{\text{over}}}\alpha_i|x_i| + \tfrac{\mu}{M}\sum_{g\in\bar{G}^{\text{dis}}}\|\boldsymbol{x}_g\|_\infty\\ \text{s.t. } \begin{cases}\boldsymbol{x}_{S_0} = 0\\ \|\boldsymbol{x}\|_\infty \leq M\end{cases},\end{aligned} \qquad (\mathcal{P}^{\mathbf{N}}_{2+1(1+\infty)})$$

with $\bar{S}^{\text{over}}$ the set of all variables contained in overlapping groups of $\bar{G}$, and the α_i defined as in Section 8.4.2. Formally, we have:

$$\begin{aligned}\bar{S}^{\text{over}} := \{i \in \{1,..,Q\} \mid \exists g \in \bar{G}^{\text{over}}, i \in g\} = \bigcup_{g\in\bar{G}^{\text{over}}} g,\\ \forall i \in \bar{S}^{\text{over}}, \alpha_i = \sum_{g\in\bar{G}^{\text{over}}:i\in g}\frac{1}{|g|}.\end{aligned}$$

8.6.3 Resolution: hybrid IGD

We propose a block coordinate descent algorithm to solve Problem $(\mathcal{P}^{\mathbf{N}}_{2+1(1+\infty)})$, in a similar vein than Algorithm 9. The different subproblems of this algorithm are:

— one subproblem for each variable in S_1, solved using Proposition 8.5.1,

— one subproblem for each variable in $\bar{S}^{\text{over}}$, solved using (5.13),

— one subproblem for each group in $\bar{G}^{\text{dis}}$, *approximated* using a Davis-Yin scheme (Algorithm 10).

The overall convergence of the algorithm is empirically ensured by monitoring the duality gap, in a similar way to the IGD algorithm of Section 8.5.2 when applied to the $\ell_1 - \ell_\infty$ relaxation. The dual problem of Problem $(\mathcal{P}^{\mathbf{N}}_{2+1(1+\infty)})$ is detailed in Proposition 8.6.1 below. The resulting algorithm is called *Hybrid Iterative Group Descent* and detailed in Algorithm 11, using $\mathbf{A}_{-g}$ (respectively $\boldsymbol{x}_{-g}$), with g a set of indices, as a notation for $\mathbf{A}_{\{1,..,Q\}\setminus g}$ (respectively $\boldsymbol{x}_{\{1,..,Q\}\setminus g}$).

Proposition 8.6.1. *The dual problem of Problem* $(\mathcal{P}^{\mathbf{N}}_{2+1(1+\infty)})$ *reads:*

$$\begin{aligned}\max_{\boldsymbol{w}\in\mathbb{R}^N} D(\boldsymbol{w}) := &-\tfrac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2)\\ &-M\left(\|\mathbf{A}_{S_1}^T\boldsymbol{w}\|_1 + \sum_{i\in\bar{S}^{over}}\left[|\boldsymbol{a}_i^T\boldsymbol{w}| - \alpha_i\tfrac{\mu}{M}\right]_+\right.\\ &\qquad\left.+\sum_{g\in\bar{G}^{dis}}\left[\|\mathbf{A}_g^T\boldsymbol{w}\|_1 - \tfrac{\mu}{M}\right]_+\right)\\ &+\mu|G_1|\end{aligned} \qquad (\mathcal{D}^{\mathbf{N}}_{2+1(1+\infty)})$$

Proof. Problem $(\mathcal{P}^{\mathbf{N}}_{2+1(1+\infty)})$ can be recast as $\min_{\boldsymbol{x}} f(\mathbf{A}\boldsymbol{x})+g(\boldsymbol{x})$ with $f(\mathbf{A}\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2$ and:

$$g(\boldsymbol{x}) = I_{[-M,M]^Q}(\boldsymbol{x}) + \mu|G_1| + I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0}) + \tfrac{\mu}{M}\sum_{i\in\bar{S}^{\text{over}}}\alpha_i|x_i| + \tfrac{\mu}{M}\sum_{g\in\bar{G}^{\text{dis}}}\|\boldsymbol{x}_g\|_\infty.$$

The Fenchel conjugate of f is $f^*(u) = \frac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2)$, while the Fenchel conjugate of g reads by separability $g^*(u) = \nu_{S_1}(\boldsymbol{u}_{S_1}) + \nu_{S_0}(\boldsymbol{u}_{S_0}) + \nu_{\bar{S}^{\text{over}}}(\boldsymbol{u}_{\bar{S}^{\text{over}}}) + \nu_{\bar{G}^{\text{dis}}}(\boldsymbol{u}_{\bar{S}^{\text{dis}}})$ with:

— $\bar{S}^{\text{dis}}$ the set of variable in disjoint groups: $\bar{S}^{\text{dis}} = \bigcup_{g\in\bar{G}^{\text{dis}}} g$;

— $\nu_{S_1}(\boldsymbol{u}_{S_1}) := \sup_{\boldsymbol{x}_{S_1}\in[-M,M]^{|S_1|}} \boldsymbol{u}_{S_1}^T\boldsymbol{x}_{S_1} - \mu|G_1| = M\|\boldsymbol{u}_{S_1}\|_1 - \mu|G_1|$, this term being shared with the $\ell_1-\ell_1$ $(\mathcal{D}^{\mathbf{N}\text{weight}}_{2+11})$ and $\ell_1-\ell_\infty$ $(\mathcal{D}^{\mathbf{N}}_{2+1\infty})$ formulations dual problems;

— $\nu_{S_0}(\boldsymbol{u}_{S_0}) := \sup_{\boldsymbol{x}_{S_0}\in\{0\}^{|S_0|}} \boldsymbol{u}_{S_0}^T\boldsymbol{x}_{S_0} = 0$, this term being shared with the $\ell_1-\ell_1$ $(\mathcal{D}^{\mathbf{N}\text{weight}}_{2+11})$ and $\ell_1-\ell_\infty$ $(\mathcal{D}^{\mathbf{N}}_{2+1\infty})$ formulations dual problems;

— $\nu_{\bar{S}^{\text{over}}}(\boldsymbol{u}_{\bar{S}^{\text{over}}}) := \sup_{\boldsymbol{x}_{\bar{S}^{\text{over}}}\in[-M,M]^{|\bar{S}^{\text{over}}|}} \boldsymbol{u}_{\bar{S}^{\text{over}}}^T\boldsymbol{x}_{\bar{S}^{\text{over}}} - \frac{\mu}{M}\sum_{i\in\bar{S}^{\text{over}}}\alpha_i|x_i| = M\sum_{i\in\bar{S}^{\text{over}}}\left[|\boldsymbol{u}_i| - \alpha_i\frac{\mu}{M}\right]_+$, this term being similar to the $\bar{S}$ term of the $\ell_1-\ell_1$ formulation dual problem $(\mathcal{D}^{\mathbf{N}\text{weight}}_{2+11})$;

— $\nu_{\bar{G}^{\text{dis}}}(\boldsymbol{u}_{\bar{S}^{\text{dis}}}) := \sup_{\boldsymbol{x}_{\bar{S}^{\text{dis}}}\in[-M,M]^{|\bar{S}^{\text{dis}}|}} \boldsymbol{u}_{\bar{S}^{\text{dis}}}^T\boldsymbol{x}_{\bar{S}^{\text{dis}}} - \frac{\mu}{M}\sum_{g\in\bar{G}^{\text{dis}}}\|\boldsymbol{x}_g\|_\infty = M\sum_{g\in\bar{G}^{\text{dis}}}\left[\|\boldsymbol{u}_g\|_1 - \frac{\mu}{M}\right]_+$, this term being similar to the $\bar{G}$ term of the $\ell_1-\ell_\infty$ formulation dual problem $(\mathcal{D}^{\mathbf{N}}_{2+1\infty})$.

The dual problem follows by injecting the definitions of f^* and g^* into the dual objective function $D(\boldsymbol{w}) := -f^*(\boldsymbol{w}) - g^*(-\mathbf{A}^T\boldsymbol{w})$. □

Algorithm 11 Hybrid IGD algorithm for problem ($\mathcal{P}^{\mathbf{N}}_{2+1(1+\infty)}$).

procedure IGDHYB($\boldsymbol{y}$, $\mathbf{A}$, μ, $\boldsymbol{x}^0$, G_1, $\bar{G}$, S_1, $\bar{S}$)
  while not convergence **do**
    for $i \in S_1$ **do**
      $\boldsymbol{r}_i \leftarrow \boldsymbol{y} - \mathbf{A}_{-\{i\}}\boldsymbol{x}_{-\{i\}}$
      $x_i^{k+1} \leftarrow \arg\min_{x_i \in [-M,M]} \frac{1}{2}\|\boldsymbol{r}_i - \boldsymbol{a}_i x_i\|_2^2 = \Pi_{[-M,M]}((\boldsymbol{a}_i^T \boldsymbol{a}_i)^{-1}(\boldsymbol{a}_i^T \boldsymbol{r}_i))$
    end for
    for $i \in \bar{S}^{\text{over}}$ **do**
      $\boldsymbol{r}_i \leftarrow \boldsymbol{y} - \mathbf{A}_{-\{i\}}\boldsymbol{x}_{-\{i\}}$
      $x_i^{k+1} \leftarrow \arg\min_{x_i \in [-M,M]} \frac{1}{2}\|\boldsymbol{r}_i - \boldsymbol{a}_i x_i\|_2^2 + \alpha \frac{\mu}{M}|x_i|$
    end for
    for $g \in \bar{G}^{\text{dis}}$ **do**
      $\boldsymbol{r}_g \leftarrow \boldsymbol{y} - \mathbf{A}_{-g}\boldsymbol{x}_{-g}$
      $\boldsymbol{x}_g^{k+1} \leftarrow \text{DavisYin}(\boldsymbol{y}, \mathbf{A}, \frac{\mu}{M}, \boldsymbol{x}^{k+1}, g, \theta, \rho, q = \infty)$
    end for$k \leftarrow k + 1$
  end while
  return $\boldsymbol{x}^k$
end procedure

8.7 Numerical experiments

8.7.1 Dataset

The goal here is to benchmark the performance of the different formulations of the lower bounds in both the separable case and the overlapping case. We consider synthetic datasets generated by the protocol described in Section 3.6 (page 40). The datasets in this section were generated with a common group structure for ease of comparison. All groups are of size 4 and non-overlapping, so that with $Q = 100$ (respectively $Q = 1\,000$) variables, there are 25 (respectively 250) groups. A summary of the different parameters considered is given in Table 8.1. Given a group structure, the ground truth solution $\boldsymbol{x}_{\text{truth}}$

Size	ρ	N	Q	K
Small	$\{0.8, 0.92\}$	500	100	6
Moderate	0.7	500	1 000	6

Table 8.1 – Parameters used for the different synthetic instances generated for evaluating the performance of the branch-and-bound algorithm for structured sparsity problems. 10 instances are generated for each combination of parameters, with groups of 4 variables.

is generated by setting variables in K groups to 1. Solving the instances generated from the parameters of Table 8.1 without any group structure given to the solver would be prohibitively expensive, because it would mean looking for a solution with 24 nonzero variables (6 groups with 4 variables). Two benchmarks are proposed in the following: one in a separable case, one in an overlapping case. For the separable case, the group structure given to the solver is the one used for generating the instances. For the overlapping case, the group structure given to the solver is composed of groups of size 4, with two consecutive groups overlapping over 2 variables (see Figure 8.3 for an illustration of both cases). The exploration strategy used is depth-first search (which acts as a baseline strategy).

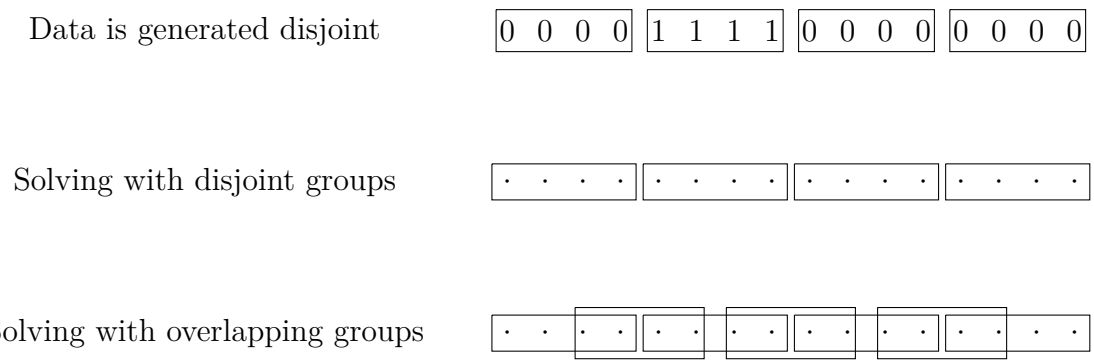


Figure 8.3 – An illustration of the data generation and modelisation setup. The data is generated by putting groups of 4 variables to 0 or 1. After that, two different modelisations of the problem are used. The disjoint case reuses the same group structure, while the overlapping case adds groups so that two consecutive groups overlap by 2 variables.

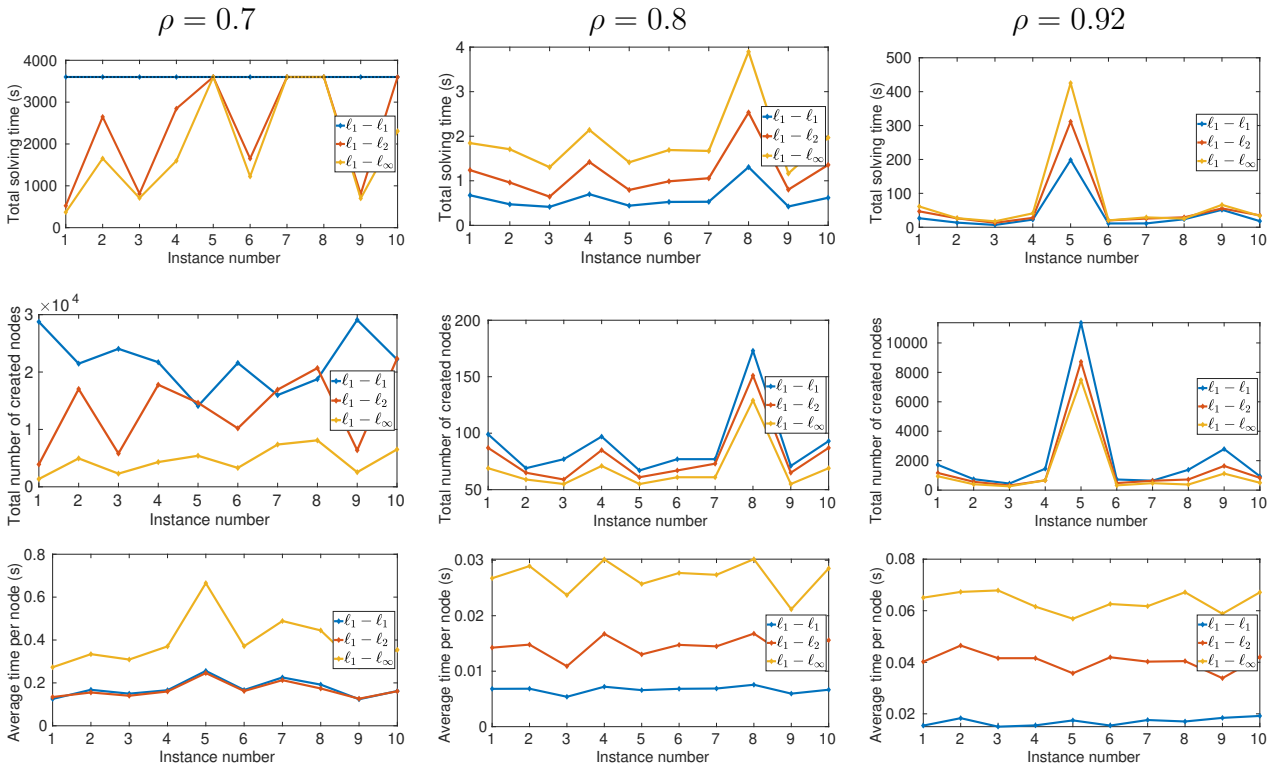


Figure 8.4 – Comparison of the different choices for the lower bound formulation on the separable group structure with $\rho \in \{0.7, 0.8, 0.92\}$. The different metrics considered are the time to solve the problem to optimality in seconds (top), the number of nodes created to solve to optimality (middle) and the ratio between both, in other words the average time spent to bound a node in seconds (bottom). The time limit is set to one hour.

8.7.2 Results

Separable case For each instance where the branch-and-bound converged, all the formulations gave the same minimizer. Figure 8.4 shows the performance of the different formulations ($\ell_1 - \ell_1$, $\ell_1 - \ell_2$, $\ell_1 - \ell_\infty$) for solving the instances in the separable setting. For each value of ρ, three metrics are plotted, one point per instance, for each formulation: the total time to solve a given instance in seconds, the number of nodes created to solve this instance, and the average time in seconds spent on solving a node of this instance. For $\rho = 0.7$ (containing the larger size instances, with $Q = 1\,000$), the hierarchy between the formulation is quite clear when looking at the solving time, with the $\ell_1 - \ell_1$ formulation yielding the slowest algorithm (no instance did terminate in 3 600s) and the $\ell_1 - \ell_\infty$ formulation the fastest (1 937s per instance on average), with the $\ell_1 - \ell_2$ relaxation in-between (2 369s per instance on average). The branch-and-bound algorithm computing lower bounds with the $\ell_1 - \ell_2$ formulation reaches the time limit for 4 instances, and with the $\ell_1 - \ell_\infty$ one for 3 instances. This difference comes from the number of explored nodes, which is the lowest for the branch-and-bound algorithm using $\ell_1 - \ell_\infty$ relaxation (4 615 nodes per instance on average), and the largest when using the $\ell_1 - \ell_1$ relaxation (21 766 nodes per instance on average), with the $\ell_1 - \ell_2$ relaxation in-between (13 571

nodes per instance on average), which is consistent with Inequality (8.1). Interestingly, when looking at the average bounding time of each node, solving the $\ell_1 - \ell_\infty$ relaxation takes substantially more time *per node* (0.389s per node on average) than solving the $\ell_1 - \ell_1$ and $\ell_1 - \ell_2$ formulations (respectively 0.17s and 0.17s per node on average). That is, in the case of $\rho = 0.7, Q = 1\,000$, increasing the bound quality by using the $\ell_1 - \ell_\infty$ relaxation instead of the $\ell_1 - \ell_1$ relaxation decreases enough the number of created nodes (divided by 4.7 on average) so that the higher solving time per node (multiplied by 2.2 on average) is more than compensated. For $\rho = 0.8$ and $\rho = 0.92$ (smaller size instances, with $Q = 100$), the opposite hierarchy appears: using the $\ell_1 - \ell_1$ relaxation gives the fastest branch-and-bound algorithm (38s on average for $\rho = 0.92$, 0.61s on average for $\rho = 0.8$), while using the $\ell_1 - \ell_\infty$ relaxation gives the slowest branch-and-bound algorithm (75s on average for $\rho = 0.92$, 1.9s on average for $\rho = 0.8$), with the $\ell_1 - \ell_2$ relaxation giving in-between results (59s on average for $\rho = 0.92$, 1.2s on average for $\rho = 0.8$). This is because the small gain in the number of created nodes when using the $\ell_1 - \ell_\infty$ relaxation instead of the $\ell_1 - \ell_1$ relaxation (44% less nodes created on average for $\rho = 0.92$, 34% less nodes created on average for $\rho = 0.8$) cannot compensate the gap in the bounding time per node (multiplied by 1.9 on average for $\rho = 0.92$, multiplied by 3.1 on average for $\rho = 0.8$). This was expected, because when ρ increases, the convex relaxations get looser, and improving the quality of the relaxation by using the $\ell_1 - \ell_\infty$ relaxation instead of the $\ell_1 - \ell_1$ relaxation is not sufficient to get significantly tighter bounds, meaning the algorithm does not prune significantly more nodes. Here, this cap on the lower bound quality makes the $\ell_1 - \ell_\infty$ formulation not worth the effort, and the quickest formulation to compute, namely the $\ell_1 - \ell_1$ formulation here, is to be preferred.

Overlapping case For each instance where the branch-and-bound converged, all the formulations gave the same minimizer. Figure 8.5 shows the performance of the branch-and-bound algorithm when using the $\ell_1 - \ell_1$ relaxation and the $\ell_1 - (\ell_1 + \ell_\infty)$ relaxation, in the overlapping group structure setting (as a recall, these relaxations are the only valid ones developed in this thesis in the overlapping groups case). For $\rho = 0.7, Q = 1\,000$, when looking at the solving time, both formulations hit the time limit for every instance: these formulations cannot scale to this kind of setting because of its size. As these instances did not converge, the number of created nodes cannot be compared fairly. For the average solving time per node, the results tend to favour the $\ell_1 - (\ell_1 + \ell_\infty)$ relaxation (0.039s per node on average) over the $\ell_1 - \ell_1$ relaxation (0.042s per node on average). For $\rho = 0.8, Q = 100$, using the $\ell_1 - \ell_1$ relaxation gives a slightly faster branch-and-bound algorithm than choosing the $\ell_1 - (\ell_1 + \ell_\infty)$ relaxation, with a solving time of 25.6s on average compared to 26.2s on average. Here, using the branch-and-bound algorithm with the $\ell_1 - (\ell_1 + \ell_\infty)$ relaxation creates twice less nodes than with the $\ell_1 - \ell_1$ relaxation, but it doubles the time to solve each node on average. For $\rho = 0.92, Q = 100$, when looking

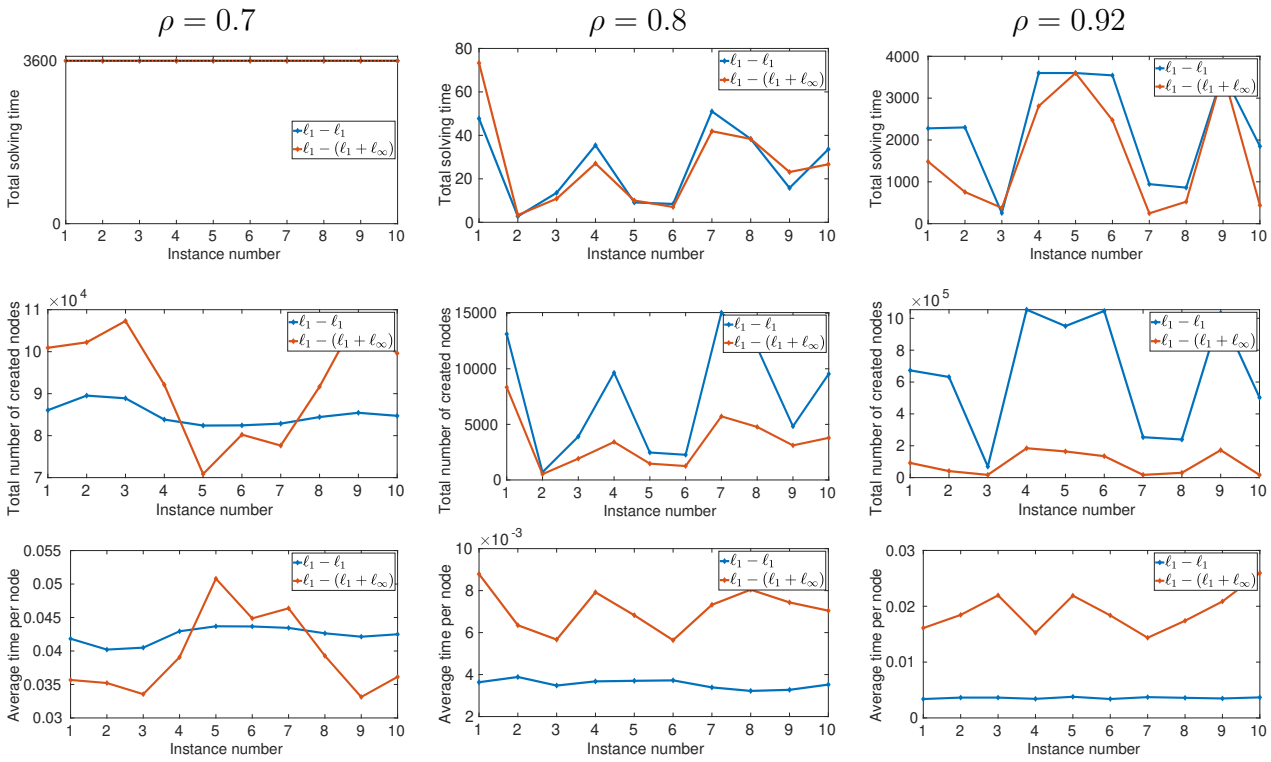


Figure 8.5 – Comparison of the different choices for the lower bound formulation on the overlapping group structure with $\rho \in \{0.7, 0.8, 0.92\}$. The different metrics considered are the time to solve the problem to optimality ins (top), the number of nodes created to solve to optimality (middle) and the ratio between both, in other words the average time spent to bound a node ins (bottom). The time limit is set to one hour.

at the solving time, the branch-and-bound using the $\ell_1 - (\ell_1 + \ell_\infty)$ formulation is faster (1 631s on average) than its counterpart using the $\ell_1 - \ell_1$ relaxation (2 283s on average). Although the $\ell_1 - (\ell_1 + \ell_\infty)$ formulation takes 5.3 more time per node, it creates 7.4 less nodes, making it the best choice overall.

8.8 Discussion

Conclusion This chapter explored the way to formulate lower bounds in the branch-and-bound framework proposed in Chapter 7. To this end:

- three mixed norms formulations were proposed, namely $\ell_1 - \ell_1$, $\ell_1 - \ell_2$ and $\ell_1 - \ell_\infty$, to build convex relaxations of the original ℓ_0 problems,
- algorithms solving these relaxation problems were proposed,
- the tightness of each relaxation and their ability to tackle the specific case of overlapping groups leads to design a hybrid relaxation tailored for the overlapping case, with its own solving algorithm,
- the quality of these relaxations was assessed theoretically and empirically, the best performing relaxation depending on the problem at hand.

Perspectives Building good lower bounds at each node of the branch-and-bound algorithm described in Chapter 7 amounts to finding a relaxation of the original problem, which should be as tight as possible, and finding a fast algorithm for solving it: we are looking for the optimal relaxation-algorithm pair. Here, the choice was to use a convex mixed-norm-based relaxation, with several choices of mixed norms, solved by a tailored block coordinate descent algorithm. In this setup, there was no universally best relaxation, as the tightest relaxations are also the most computationally demanding, and the best trade-off depends on the context. It would be interesting to compare this choice with other ones. Indeed, on the relaxation side, non-convex approaches described in Section 6.3.3 (page 87) provide tighter relaxation at the cost of a less tractable problem. Such an approach would be interesting to compare with the present contributions. On the algorithmic side, three questions arise when using Algorithm 9, dedicated to solving non-separable mixed norms ($\ell_1 - \ell_2$ and $\ell_1 - \ell_\infty$) formulations. When using this algorithm, we approximate the analytically unknown minimization step 10 by resorting to the Davis-Yin scheme. We used one iteration of the scheme to approximate the minimization step. The first question is then if using more iterations of the Davis-Yin scheme could lead to greater overall performance, spending more time on inner steps to get better approximations. The second question is related to the performance of this scheme, compared with other optimization schemes. Indeed, other proximal algorithms involving three functions could be used (see [Condat et al. 2019] for a review), and a proximal algorithm involving

two functions with augmented data (merging two functions in one at the cost of doubling the size of the dual vector) is also possible. Among all these methods, which one is the best performing one in the context of exact structured sparsity optimization is an open question. The third question is a theoretical one. Approximating the minimization step 10 in Algorithm 9 instead of doing it exactly breaks the assumptions of the convergence proofs in [Tseng 2001], and makes us resort to an empirical monitoring of the duality gap to check convergence. In particular, it makes solving the $\ell_1 - \ell_2$ formulation very fragile algorithmically speaking, with no convergence proof nor practical way to monitor convergence, even though in our experiments the branch-and-bound algorithm retrieved the same minimizer than with the other formulations. Therefore, the possibility to derive convergence proofs using an approximated step such as the Davis-Yin scheme is worth being studied.

For the overlapping case, the choice made here is to either consider all the groups together ($\ell_1 - \ell_1$ formulation), or to separate between disjoint groups and overlapping groups ($\ell_1 - (\ell_1 + \ell_\infty)$ formulation). This separation could be further refined. Indeed, a number of overlapping group structures use groups which are overlapping with a *few* number of other groups. This means that for a given group, it is possible to find a number of other groups that does not intersect with it. In [I. Bayram 2011], the author propose an optimization algorithm in an overlapping group case by identifying, inside the overlapping groups, subsets of groups disjoint to each other. The proposed optimization algorithm then solves subproblems, where only one such subset is allowed to move. What is the empirical performance of such an approach for lower bounding nodes inside a branch-and-bound algorithm is still an open question.

Chapter 9

Structured sparsity lower bound accelerations

Contents

9.1 Introduction

As pointed out in Section 5.1 (on page 52) considering scalar sparsity, a huge part of the running time of the branch-and-bound algorithm is dedicated to the lower bounds computations. In this part, we wish to implement and evaluate the same kind of accelerations as in Chapter 5 (page 52), adapted to the structured sparsity case. In other words, while Chapter 8 was dedicated to formulating the lower bound problems and the descent algorithms solving them, this chapter is dedicated to accelerating the computation of these lower bounds.

Three different accelerations have been studied in Chapter 5:

— Using the dual objective to prune nodes before convergence.

— Using the dual objective to get a valid lower bound before convergence for non-pruned nodes.

— Using the GapSafe screening method to reduce dimension and accelerate the optimization algorithm.

All three accelerations require the knowledge of an analytical expression for the dual objective. The first two use the dual objective value directly (see Chapter 5), while the screening method requires a more involved development. Both the dual objective expression and the screening method are detailed for the $\ell_1 - \ell_1$ relaxation (Section 9.2) and the $\ell_1 - \ell_\infty$ relaxation (Section 9.3) before going to some numerical experiments. As no analytical expression of the dual objective is known for the $\ell_1 - \ell_2$ relaxation, it is ignored in the following.

9.2 Accelerations for the $\ell_1 - \ell_1$ relaxation

As a recall, the $\ell_1 - \ell_1$ relaxation reads:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\sum_{i\in\bar{S}}\alpha_i|x_i| \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases} \quad (\mathcal{P}_{2+11}^{\mathbf{N}\text{weight}})$$

for the multiple-weights variant, and:

$$\min_{\boldsymbol{z}} \tfrac{1}{2}\|\boldsymbol{y} - \mathcal{A}_{S_1}\boldsymbol{z}_{S_1} - \mathcal{A}_{\bar{S}}\boldsymbol{z}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\|\boldsymbol{z}_{\bar{S}}\|_1 \quad \text{s.t.} \quad \begin{cases} \boldsymbol{z}_{S_0} = \mathbf{0} \\ \|\boldsymbol{z}_{S_1}\|_\infty \leq M \\ \forall i \in \bar{S}, |z_i| \leq M\alpha_i \end{cases} \quad (\mathcal{P}_{2+11}^{\mathbf{N}\text{bound}})$$

for the multiple-bounds variant, with $\mathcal{A}$ such that $\forall i \in \bar{S}, \mathcal{A}_i = \boldsymbol{a}_i/\alpha_i$, and $\forall i \in S_1 \cup S_0, \mathcal{A}_i = \boldsymbol{a}_i$. As a recall, both formulations can be solved by state-of-the-art algorithms (Section 3.3 on page 24) with minor modifications. The goal here is to accelerate the computation of lower bounds thanks to these formulations.

Adapting the previously mentioned accelerations means getting the expression of the dual objective, which is the subject of Section 9.2.1, and getting the correct screening tests, which is the subject of Section 9.2.2. In the $\ell_1 - \ell_1$ case, a lot of concepts from scalar sparsity are easily transferable.

9.2.1 Dual objective

We derive here the dual problem for both variants of the $\ell_1 - \ell_1$ formulation: the multiple-weight variant ($\mathcal{P}_{2+11}^{\mathbf{N}\text{weight}}$) and the multiple-bound variant ($\mathcal{P}_{2+11}^{\mathbf{N}\text{bound}}$). As we will see in Proposition 9.2.2, both objectives are strictly equivalent.

Proposition 9.2.1. *The dual problem of the multiple-weight variant* ($\mathcal{P}_{2+11}^{\mathbf{N}\text{weight}}$) *reads:*

$$\begin{aligned}\max_{\boldsymbol{w}\in\mathbb{R}^N} \; & -\tfrac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2-\|\boldsymbol{y}\|_2^2)+\mu|G_1| \\ & -M\left(\sum_{i\in\bar{S}}\left[|\boldsymbol{a}_i^T\boldsymbol{w}|-\alpha_i\tfrac{\mu}{M}\right]_+ + \|\mathbf{A}_{S_1}^T\boldsymbol{w}\|_1\right)\end{aligned} \tag{$\mathcal{D}_{2+11}^{\mathbf{N}\text{weight}}$}$$

with $[\cdot]_+ = \max(0,\cdot)$.

Proof. Like Problem (5.1) (page 54), we are in the setting of the Fenchel-Rockafellar theorem ([Rockafellar 1970], theorem 31.2) with $f(\mathbf{A}\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2$ and $g(\boldsymbol{x}) = \mu|G_1| + \frac{\mu}{M}\sum_{i\in\bar{S}}\alpha_i|x_i| + I_{[-M,M]^Q}(\boldsymbol{x}) + I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0})$. As a recall, the dual problem reads $\max_{\boldsymbol{w}} -f^*(\boldsymbol{w}) - g^*(-\mathbf{A}^T\boldsymbol{w})$, with f^* and g^* the Fenchel conjugates of f and g respectively. The Fenchel conjugate of f reads $f^*(\boldsymbol{w}) = \frac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2-\|\boldsymbol{y}\|_2^2)$ (see Section 5.2.1 on page 54). The expression of g^* is developed hereafter, following the same technique as in Section 5.2.1:

$$\begin{aligned}g^*(\boldsymbol{u}) &= \sup_{\boldsymbol{x}\in[-M,M]^Q} \underbrace{\left(\boldsymbol{u}^T\boldsymbol{x}-\mu|G_1|-\tfrac{\mu}{M}\sum_{i\in\bar{S}}\alpha_i|x_i|-I_{[-M,M]^Q}(\boldsymbol{x})-I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0})\right)}_{\text{separable in each variable } x_i}, \\ &= \sum_{i\in\bar{S}}\sup_{|x_i|\le M}(u_ix_i-\tfrac{\mu}{M}\alpha_i|x_i|)+\sum_{i\in S_1}\sup_{|x_i|\le M}u_ix_i \\ &\quad +\sum_{i\in S_0}\sup_{|x_i|\le M}\underbrace{(u_ix_i-I_{\{0\}}(x_i))}_{=0}-\mu|G_1|, \\ &= \sum_{i\in\bar{S}}\sup_{|x_i|\le M}(|u_i||x_i|-\alpha_i\tfrac{\mu}{M}|x_i|)+\sum_{i\in S_1}\sup_{|x_i|\le M}|u_i||x_i|-\mu|G_1|, \\ &= \sum_{i\in\bar{S}}\sup_{|x_i|\le M}\left(|x_i|(|u_i|-\alpha_i\tfrac{\mu}{M})\right)+\sum_{i\in S_1}|u_i|M-\mu|G_1|, \\ &= M\left(\sum_{i\in\bar{S}}\left[|u_i|-\alpha_i\tfrac{\mu}{M}\right]_+ + \|\boldsymbol{u}_{S_1}\|_1\right)-\mu|G_1|. \\ \implies g^*(-\mathbf{A}^T\boldsymbol{w}) &= M\left(\sum_{i\in\bar{S}}\left[|\boldsymbol{a}_i^T\boldsymbol{w}|-\alpha_i\tfrac{\mu}{M}\right]_+ + \sum_{i\in S_1}|\boldsymbol{a}_i^T\boldsymbol{w}|\right)-\mu|G_1|.\end{aligned} \tag{9.1}$$

The dual expression in Problem ($\mathcal{D}_{2+11}^{\mathbf{N}\text{weight}}$) follows straightforwardly. □

Proposition 9.2.2. *The dual problem of the multiple-bound variant* ($\mathcal{P}_{2+11}^{\mathbf{N}\text{bound}}$) *reads:*

$$\begin{aligned}\max_{\boldsymbol{w}\in\mathbb{R}^N} \; & -\tfrac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2-\|\boldsymbol{y}\|_2^2)+\mu|G_1| \\ & -\left(\sum_{i\in\bar{S}}M\alpha_i\left[|\mathcal{A}_i^T\boldsymbol{w}|-\tfrac{\mu}{M}\right]_+ + M\|\mathcal{A}_{S_1}^T\boldsymbol{w}\|_1\right)\end{aligned} \tag{$\mathcal{D}_{2+11}^{\mathbf{N}\text{bound}}$}$$

Moreover, this problem is the same than Problem ($\mathcal{D}_{2+11}^{\mathbf{N}\text{weight}}$).

The proof of Proposition 9.2.2 is given in Appendix C.1 (page 176).

9.2.2 GapSafe screening

Like in the scalar sparsity case, the goal here is to set individual variables to 0 and $\pm M$. We assume here without loss of generality that $\mathbf{A}$ is normalized to have unit ℓ_2-norm columns. We derive here the screening tests for both variants of the $\ell_1 - \ell_1$ formulation: the multiple-weight variant ($\mathcal{P}_{2+11}^{\mathbf{N}\text{weight}}$) and the multiple-bound variant ($\mathcal{P}_{2+11}^{\mathbf{N}\text{bound}}$). As we will see in Proposition 9.2.4, both tests are equivalent.

Theorem 9.2.3. *The GapSafe screening test for Problem ($\mathcal{P}_{2+11}^{\mathbf{N}\text{weight}}$) reads:*

$$\forall i \in \bar{S}, \text{ if } |\boldsymbol{a}_i^T \boldsymbol{w}| > \alpha_i \tfrac{\mu}{M} + \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}, \text{ then } x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}); \tag{9.2a}$$

$$\forall i \in \bar{S}, \text{ if } |\boldsymbol{a}_i^T \boldsymbol{w}| < \alpha_i \tfrac{\mu}{M} - \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}, \text{ then } x_i^\star = 0; \tag{9.2b}$$

$$\forall i \in S_1, \text{ if } |\boldsymbol{a}_i^T \boldsymbol{w}| > \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}, \text{ then } x_i^\star = -M \operatorname{sign}(\boldsymbol{a}_i^T \boldsymbol{w}). \tag{9.2c}$$

Proof. The screening rules come from the regularization term, which in the $\ell_1 - \ell_1$ multiple-weight formulation reads:

$$g(\boldsymbol{x}) := \mu |G_1| + \tfrac{\mu}{M} \sum_{i \in \bar{S}} \alpha_i |x_i| + I_{[-M,M]^Q}(\boldsymbol{x}) + I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0}). \tag{9.3}$$

From the Karush-Kuhn-Tucker optimality conditions (5.6):

$$\begin{aligned} \boldsymbol{w}^\star &\in \partial f(\mathbf{A}\boldsymbol{x}^\star) \\ -\mathbf{A}^T \boldsymbol{w}^\star &\in \partial g(\boldsymbol{x}^\star) \\ \mathbf{A}\boldsymbol{x}^\star &\in \partial f^*(\boldsymbol{w}^\star) \\ \boldsymbol{x}^\star &\in \partial g^*(-\mathbf{A}^T \boldsymbol{w}^\star) \end{aligned}$$

we use the condition (5.6b): $-\mathbf{A}^T \boldsymbol{w}^\star \in \partial g(\boldsymbol{x}^\star)$, where ∂ denotes the subdifferential operator. This condition is separable variable per variable, and gives the following cases:

$$\begin{cases} \forall i \in \bar{S} & \text{with} & x_i^\star = M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in [\alpha_i \frac{\mu}{M}, +\infty[\\ \forall i \in \bar{S} & \text{with} & x_i^\star \in]0, M[, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star = \alpha_i \frac{\mu}{M} \\ \forall i \in \bar{S} & \text{with} & x_i^\star = 0, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in [-\alpha_i \frac{\mu}{M}, \alpha_i \frac{\mu}{M}] \\ \forall i \in \bar{S} & \text{with} & x_i^\star \in]-M, 0[, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star = -\alpha_i \frac{\mu}{M} \\ \forall i \in \bar{S} & \text{with} & x_i^\star = -M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in]-\infty, -\alpha_i \frac{\mu}{M}] \\ \forall i \in S_1 & \text{with} & x_i^\star = M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in [0, +\infty[\\ \forall i \in S_1 & \text{with} & x_i^\star \in]-M, M[, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star = 0 \\ \forall i \in S_1 & \text{with} & x_i^\star = -M, & -\boldsymbol{a}_i^T \boldsymbol{w}^\star \in]-\infty, -0] \end{cases}. \tag{9.4}$$

This gives us screening rules which are the equivalent of Lemma 5.4.1 (page 72). In the scalar sparsity case, we used Lemma 5.4.1 coupled with Inequality (5.19):

$$|\boldsymbol{a}_i^T \boldsymbol{w}^\star| \leq |\boldsymbol{a}_i^T \boldsymbol{w}| + \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2$$

and Inequality (5.20):

$$|\boldsymbol{a}_i^T \boldsymbol{w}^\star| \geq |\boldsymbol{a}_i^T \boldsymbol{w}| - \|\boldsymbol{w}^\star - \boldsymbol{w}\|_2$$

to get the screening tests of Theorem 5.4.2. In a similar manner here, using 9.4 along with equations (5.19) and (5.20), we get the screening tests (9.2).

□

Proposition 9.2.4. *The GapSafe screening test for Problem* ($\mathcal{P}_{2+11}^{\mathbf{N}\text{bound}}$) *reads:*

$$\forall i \in \bar{S}, \textit{if } |\mathcal{A}_i^T \boldsymbol{w}| > \tfrac{\mu}{M} + \|\mathcal{A}_i\|_2 \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}, \textit{ then } z_i^\star = -M\alpha_i \operatorname{sign}(\mathcal{A}_i^T \boldsymbol{w}); \tag{9.5a}$$

$$\forall i \in \bar{S}, \textit{if } |\mathcal{A}_i^T \boldsymbol{w}| < \tfrac{\mu}{M} - \|\mathcal{A}_i\|_2 \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}, \textit{ then } z_i^\star = 0; \tag{9.5b}$$

$$\forall i \in S_1, \textit{if } |\mathcal{A}_i^T \boldsymbol{w}| > \|\mathcal{A}_i\|_2 \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}, \textit{ then } z_i^\star = -M \operatorname{sign}(\mathcal{A}_i^T \boldsymbol{w}). \tag{9.5c}$$

with $\boldsymbol{z}$ *defined as:*

— $\forall i \in \bar{S}, z_i = \alpha_i x_i$,

— $\forall i \in S_1 \cup S_0, z_i = x_i$.

Moreover, tests (9.5) *are equivalent to tests* (9.2).

The proof of Proposition 9.2.4 is given in Appendix C.2 (page 177).

In the same vein than in Section 5.4.4 (page 74), $\overline{\text{ub}}$ will be leveraged to further improve the screening performance when $\overline{\text{ub}}$ is lower than the current primal value $P(x)$.

9.3 Accelerations for the $\ell_1 - \ell_\infty$ relaxation (disjoint groups case)

In a similar manner to the previous section, we will give the dual expression in Section 9.3.1, and then we develop screening tests in Section 9.3.2.

9.3.1 Dual objective

The dual objective function has already been obtained through Problem ($\mathcal{D}_{2+1\infty}^{\mathbf{N}}$) in Section 8.5.2, as it was needed to monitor the convergence of the IGD algorithm. As a

recall, the dual problem of the $\ell_1 - \ell_\infty$ formulation reads:

$$\max_{\boldsymbol{w}\in\mathbb{R}^N} D(\boldsymbol{w}) := -\tfrac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2) - M\left(\|\mathbf{A}_{S_1}^T\boldsymbol{w}\|_1 + \sum_{g\in\bar{G}}[\|\mathbf{A}_g^T\boldsymbol{w}\|_1 - \tfrac{\mu}{M}]_+\right) + \mu|G_1|$$

9.3.2 GapSafe screening

As explained in Section 5.4 (page 70) of Chapter 5, the GapSafe screening depends on screening rules, which depend on the formulation, and the screening sphere, which is always the same throughout all this thesis, as the data-fitting term is always $\frac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2$. The screening rules come from the regularization term, which in the $\ell_1 - \ell_\infty$ formulation reads:

$$g(\boldsymbol{x}) := \mu|G_1| + \tfrac{\mu}{M}\sum_{g\in\bar{G}}\|\boldsymbol{x}_g\|_\infty + I_{[-M,M]^Q}(\boldsymbol{x}) + I_{\{0\}^{|S_0|}}(\boldsymbol{x}_{S_0}). \tag{9.6}$$

More precisely, the screening rules come from the KKT optimality condition (5.6b), which reads $-\mathbf{A}^T\boldsymbol{w}^\star \in \partial g(\boldsymbol{x}^\star)$, where ∂ denotes the subdifferential operator.

In function g, terms involving variables in S_1 and S_0 are the same than for the $\ell_1 - \ell_1$ formulation, giving the same rules, therefore they are omitted in the following. The $\ell_1 - \ell_\infty$ relaxation is computationally tractable for disjoint groups only, therefore this is the only case considered in the following. This means separability of the different groups in $\bar{G}$ holds, which means the conditions of interest rewrite:

$$\forall g\in\bar{G}, -\mathbf{A}_g^T\boldsymbol{w}^\star \in \tfrac{\mu}{M}\partial\|\boldsymbol{x}_g^\star\|_\infty + \partial I_{[-M,M]^{|g|}}(\boldsymbol{x}_g^\star). \tag{9.7}$$

Using the expression of the subdifferential of the ℓ_∞ norm (see Appendix B.1 on page 168 for the proof), we rewrite the previous expression as:

$$\begin{aligned}
\forall g\in\bar{G}, -\mathbf{A}_g^T\boldsymbol{w}^\star \in \tfrac{\mu}{M}&\left\{\begin{matrix}\boldsymbol{u}\in\mathbb{R}^{|g|} \mid & \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{x}_g = \|\boldsymbol{x}_g\|_\infty & \text{if } \boldsymbol{x}_g\neq\mathbf{0}\\ \boldsymbol{u}\in\mathbb{R}^{|g|} \mid & \|\boldsymbol{u}\|_1 \leq 1 & \text{if } \boldsymbol{x}_g = \mathbf{0}\end{matrix}\right\}\\
&+\left\{\boldsymbol{v}\in\mathbb{R}^{|g|} \mid \forall i\in\{1,..,|g|\}, v_i\in\left\{\begin{matrix}[0,+\infty[& \text{if } x_i = M\\ \{0\} & \text{if } x_i\in\,]-M,M[\\ ]-\infty,0] & \text{if } x_i = -M\end{matrix}\right\}\right\}\\
\implies \forall g\in\bar{G}, \|-\mathbf{A}_g^T\boldsymbol{w}^\star\|_1 &\in \tfrac{\mu}{M}\left\{\begin{matrix}\{1\} & \text{if } \boldsymbol{x}_g\neq\mathbf{0}\\ [0,1] & \text{if } \boldsymbol{x}_g = \mathbf{0}\end{matrix}\right\} + \left\{\begin{matrix}[0,+\infty[& \text{if } \|\boldsymbol{x}_g\|_\infty = M\\ \{0\} & \text{if } \|\boldsymbol{x}_g\|_\infty < M\end{matrix}\right\}\\
\iff \forall g\in\bar{G}, \|\mathbf{A}_g^T\boldsymbol{w}^\star\|_1 &\in \left\{\begin{matrix}[\frac{\mu}{M},+\infty[& \text{if } \|\boldsymbol{x}_g\|_\infty = M\\ \{\frac{\mu}{M}\} & \text{if } \|\boldsymbol{x}_g\|_\infty\in\,]0,M[\\ [0,\frac{\mu}{M}] & \text{if } \|\boldsymbol{x}_g\|_\infty = 0\end{matrix}\right\}.
\end{aligned}$$

This means we have the following screening rules:

$$\forall g \in \bar{G}, \text{if } \left\|\mathbf{A}_g^T \boldsymbol{w}^\star\right\|_1 > \tfrac{\mu}{M} \text{ then } \left\|\boldsymbol{x}_g^\star\right\|_\infty = M; \tag{9.8a}$$

$$\forall g \in \bar{G}, \text{if } \left\|\mathbf{A}_g^T \boldsymbol{w}^\star\right\|_1 < \tfrac{\mu}{M} \text{ then } \left\|\boldsymbol{x}_g^\star\right\|_\infty = 0. \tag{9.8b}$$

In practice, the rule leading to $\left\|\boldsymbol{x}_g^\star\right\|_\infty = M$ is seldom informative, because while it does give the value of the objective function, it does not give the solution reaching this value. In the following, only the rule leading to $\left\|\boldsymbol{x}_g^\star\right\|_\infty = 0 \iff \boldsymbol{x}_g^\star = \mathbf{0}$ will be used. Now, these rules use the optimal dual point $\boldsymbol{w}^\star$, which is not known in practice. From Section 5.4 (page 70), we already know that $\forall \boldsymbol{w} \in \mathbb{R}^N, \forall \boldsymbol{x} \in [-M, M]^Q, \left\|\boldsymbol{w} - \boldsymbol{w}^\star\right\|_2 \leq \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}$. Indeed, our function g in (9.6) is still convex like in the scalar sparsity case, and the function f did not change, so the same result holds here.

Now, the question is how we can plug this into the screening rules defined earlier. Using a similar technique as in Section 5.4:

$$\begin{aligned}
\forall \boldsymbol{w} \in \mathbb{R}^N \left\|\mathbf{A}_g^T \boldsymbol{w}^\star\right\|_1 &= \left\|\mathbf{A}_g^T(\boldsymbol{w}^\star - \boldsymbol{w}) + \mathbf{A}_g^T \boldsymbol{w}\right\|_1 \leq \left\|\mathbf{A}_g^T(\boldsymbol{w}^\star - \boldsymbol{w})\right\|_1 + \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 \\
&= \sum_{i \in g} \left|\boldsymbol{a}_i^T(\boldsymbol{w}^\star - \boldsymbol{w})\right| + \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 \\
&= \sum_{i \in g} \left\|\boldsymbol{a}_i^T(\boldsymbol{w}^\star - \boldsymbol{w})\right\|_2 + \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 \\
&\leq \sum_{i \in g} \left\|\boldsymbol{a}_i\right\|_2 \left\|\boldsymbol{w}^\star - \boldsymbol{w}\right\|_2 + \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 \\
&= \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 + \left\|\boldsymbol{w}^\star - \boldsymbol{w}\right\|_2 \sum_{i \in g} \left\|\boldsymbol{a}_i\right\|_2.
\end{aligned} \tag{9.9}$$

We assume a normalized matrix $\mathbf{A}$, meaning that $\sum_{i \in g} \left\|\boldsymbol{a}_i\right\|_2 = |g|$. Inequality (9.9) can be used to construct the following test:

Proposition 9.3.1. *$\forall (\boldsymbol{x}, \boldsymbol{w}) \in [-M, M]^Q \times \mathbb{R}^N$, we have:*

$$\forall g \in \bar{G}, \text{if } \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 < \tfrac{\mu}{M} - |g|\sqrt{2G(\boldsymbol{x}, \boldsymbol{w})} \text{ then } \boldsymbol{x}_g^\star = \mathbf{0}. \tag{9.10}$$

Proof. Using Inequality (9.9), we have:

$$\left\|\mathbf{A}_g^T \boldsymbol{w}^\star\right\|_1 \leq \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 + \left\|\boldsymbol{w}^\star - \boldsymbol{w}\right\|_2 |g| \leq \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 + |g|\sqrt{2G(\boldsymbol{x}, \boldsymbol{w})}.$$

This means that we have:

$$\begin{aligned}
&\left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 < \tfrac{\mu}{M} - |g|\sqrt{2G(\boldsymbol{x}, \boldsymbol{w})} \iff \left\|\mathbf{A}_g^T \boldsymbol{w}\right\|_1 + |g|\sqrt{2G(\boldsymbol{x}, \boldsymbol{w})} < \tfrac{\mu}{M} \\
\implies &\left\|\mathbf{A}_g^T \boldsymbol{w}^\star\right\|_1 < \tfrac{\mu}{M} \implies \boldsymbol{x}_g^\star = \mathbf{0}.
\end{aligned}$$

□

Note Even though both are setting groups of variables to zero, this screening test is different from the screening test of the Group LASSO which typically reads [Raj et al. 2016]:

$$\text{If } \|\mathbf{A}_g^T \boldsymbol{w}\|_2 < \tfrac{\mu}{M} - \|\mathbf{A}_g\|_{\text{fro}} \sqrt{2G(\boldsymbol{x}, \boldsymbol{w})} \text{ then } \boldsymbol{x}_g^\star = \mathbf{0}, \tag{9.11}$$

with $\|\mathbf{A}_g\|_{\text{fro}}$ the Frobenius norm of submatrix $\mathbf{A}_g$. This is because the Group LASSO uses an ℓ_2 norm as a regularizer, leading to a rule depending on $\|\mathbf{A}_g^T \boldsymbol{w}^\star\|_2$ instead of a rule depending on $\|\mathbf{A}_g^T \boldsymbol{w}^\star\|_1$.

9.4 Numerical experiments and results

In the following, the performance of the different acceleration strategies is benchmarked on synthetic datasets for the branch-and-bound algorithm using the $\ell_1 - \ell_1$ as well as the $\ell_1 - \ell_\infty$ relaxations. The metrics considered are the ones already used in Chapter 5. The datasets are the ones introduced in Chapter 8. In this section, as we benchmark non-separable formulations, we restrict ourselves to the non-overlapping setup of Chapter 8. As a recall, there are 6 groups of 4 components in the ground truth solution, with the correlation level of matrix $\mathbf{A}$ set to $\rho \in \{0.7, 0.8, 0.92\}$. Ten instances are generated for each value of ρ. Here again computations for solving each instance are limited to one hour.

9.4.1 Early pruning

Figure 9.1 shows the performance of early pruning for the branch-and-bound algorithm using the $\ell_1 - \ell_1$ relaxation (left) and the $\ell_1 - \ell_\infty$ relaxation (right), with the different datasets considered. As a recall, early pruning aims to take the decision to prune a given node before the optimization algorithm solving Problem ($\mathcal{P}^{\mathbf{N}}_{2+1q}$):

$$\min_{\boldsymbol{x}} \ \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \mu \sum_{g \in \bar{G}} \mathbf{1}_{\boldsymbol{x}_g \neq 0} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases} \qquad (\mathcal{P}^{\mathbf{N}}_{2+0s})$$

converged to the final lower bound value. The iterative coordinate descent detailed in Section 5.2.3 (page 57)is used for solving the $\ell_1 - \ell_1$ relaxation, while the iterative group descent algorithm detailed in Section 8.5.2 (page 105) is used for solving the $\ell_1 - \ell_\infty$ relaxation. The nodes are categorized on $|G_1|$ the number of groups included in the solution. We can see here an "all-or-nothing" behaviour, where there is no real gain for small values of $|G_1|$, and as $|G_1|$ increases a large number of iterations are saved thanks to early pruning. For $\rho = 0.92$, the dual pruning is really effective for nodes with more

than 7 selected groups. For $\rho = 0.8$, this effect is seen for nodes with more than 6 selected groups, and for $\rho = 0.7$ this is the case for nodes with more than 8 selected groups in the case of the $\ell_1 - \ell_1$ relaxation, and for nodes with more than 7 selected groups for the $\ell_1 - \ell_\infty$ relaxation. Interestingly, both relaxations behave in a similar way, although the $\ell_1 - \ell_1$ formulation enjoys a stronger gain from early pruning than the $\ell_1 - \ell_\infty$ formulation. Table 9.1 shows the overall performance of the branch-and-bound algorithm using the early pruning method when aggregating all nodes of all instances together for a fixed value of ρ. The performance of early pruning is very promising, with at least 40% of saved iterations ($\rho = 0.92$ dataset, solved with a branch-and-bound using the $\ell_1 - \ell_\infty$ relaxation), up to 85.5% saved sub-iterations ($\rho = 0.8$ dataset, solved with a branch-and-bound using the $\ell_1 - \ell_1$ relaxation). Here also, results show that the branch-and-bound using the $\ell_1 - \ell_1$ relaxation enjoys more saved iterations than the one using the $\ell_1 - \ell_\infty$ relaxation, with more than 20 points of difference in favour of the $\ell_1 - \ell_1$ relaxation for $\rho = 0.8$ and $\rho = 0.92$, and less than 3 points in favour of the $\ell_1 - \ell_\infty$ for $\rho = 0.7$.

$\ell_1 - \ell_1$ relaxation	$\ell_1 - \ell_\infty$ relaxation

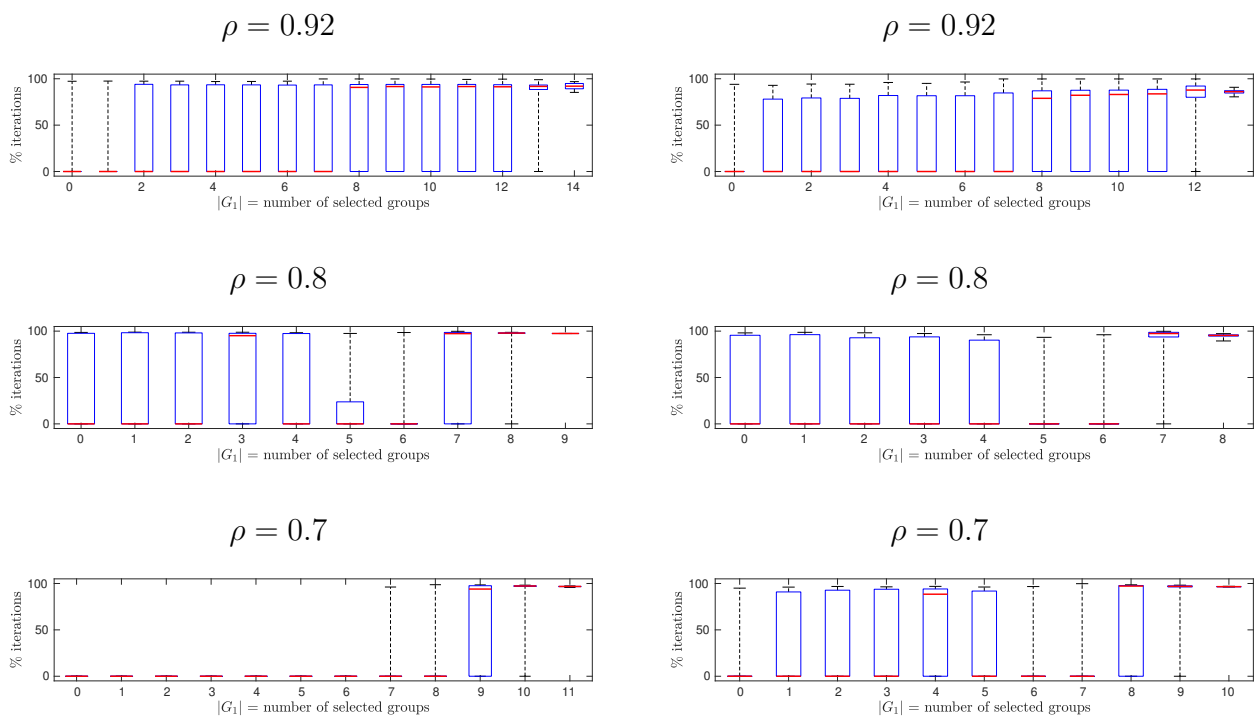


Figure 9.1 – Early pruning performance, for the branch-and-bound algorithm using the $\ell_1 - \ell_1$ and the $\ell_1 - \ell_\infty$ relaxations, on the different datasets considered ($\rho \in \{0.7, 0.8, 0.92\}$). The boxplots show the ratio of saved iterations by the use of early pruning, given the number of selected groups in the underlying nodes.

Problem category	# iterations without early pruning	# iterations with early pruning	% Saved
$\ell_1 - \ell_1$ relaxation, $\rho = 0.7$	49 182 748	15 043 858	69.4 %
$\ell_1 - \ell_1$ relaxation, $\rho = 0.8$	471 261	68 148	85.5 %
$\ell_1 - \ell_1$ relaxation, $\rho = 0.92$	14 981 676	4 712 299	68.5 %
$\ell_1 - \ell_\infty$ relaxation, $\rho = 0.7$	26 919 997	8 915 651	66.9 %
$\ell_1 - \ell_\infty$ relaxation, $\rho = 0.8$	443 306	159 198	64.1 %
$\ell_1 - \ell_\infty$ relaxation, $\rho = 0.92$	12 374 791	7 194 778	41.9 %

Table 9.1 – Results for duality-based early pruning: number of saved iterations without and with early pruning, for the $\ell_1 - \ell_1$ and $\ell_1 - \ell_\infty$ relaxations.

9.4.2 Inexact convergence inside relaxed problems

We are here interested in trading off the quality of the lower bound with the computational effort needed to obtain it. Similar to the scalar sparsity case in Section 5.3 (page 65), we do this by stopping before convergence the optimization algorithm solving Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \tfrac{\mu}{M}\sum_{g\in\bar{G}} \frac{\|\boldsymbol{x}_g\|_q}{|g|^{1/q}} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases}. \quad (\mathcal{P}^{\mathbf{N}}_{2+1q})$$

The quality of the approximation is monitored by the quantity γ, which is linked to the duality gap through the stopping criterion (5.16), which is recalled below for a given primal iterate $\boldsymbol{x}^k$ and a given dual iterate $\boldsymbol{w}^k$:

$$P(\boldsymbol{x}^k) - D(\boldsymbol{w}^k) < \gamma|P(\boldsymbol{x}^k)| + 10^{-8}.$$

Figure 9.2 shows the performance of the inexact convergence technique described in Section 5.3 (page 65) for the branch-and-bound algorithm using the $\ell_1 - \ell_1$ formulation. When looking at the time required to solve the instances (top), we can see that the best parameter value is around $\gamma = 10^{-3}$ for $\rho = 0.92$ (left) and $\rho = 0.8$ (middle). On average, using $\gamma = 10^{-3}$ instead of $\gamma = 0$ (meaning exact convergence) is 1.97 times faster for $\rho = 0.92$, and 1.95 times faster for $\rho = 0.8$. For $\rho = 0.7$, there is no gain in using $\gamma \neq 0$, as a lot of instances are reaching the 1h limit for every value of γ (recall that in Chapter 8 the $\ell_1 - \ell_1$ formulation was always hitting this limit for $\rho = 0.7$). For $\rho = 0.92$ and $\rho = 0.8$, the previously optimal choice of $\gamma = 10^{-3}$ corresponds to the last (reading the plots from left to right) value providing a significant decrease in the number of created nodes (middle), while the time spent on each node always increases as γ decreases (bottom).

Figure 9.3 shows the corresponding results using the $\ell_1 - \ell_\infty$ formulation. For the three values of ρ, when looking at the solving time (top), we clearly see an optimal value around $\gamma = 10^{-2}$. On average, using $\gamma = 10^{-2}$ instead of $\gamma = 0$ (exact convergence) is 2.88 times faster for $\rho = 0.92$, 4.25 times faster for $\rho = 0.8$, and 2.42 times faster for $\rho = 0.7$. Interestingly, this optimal value $\gamma = 10^{-2}$ is a trade-off between creating less nodes and spending less time on each node. Indeed, setting the parameter to $\gamma = 10^{-3}$ leads to a branch-and-bound algorithm creating less nodes but taking more time for each node. Setting the parameter to $\gamma = 10^{-1}$ yields the opposite behaviour: more nodes created by the branch-and-bound algorithm, but less time spent on each node.

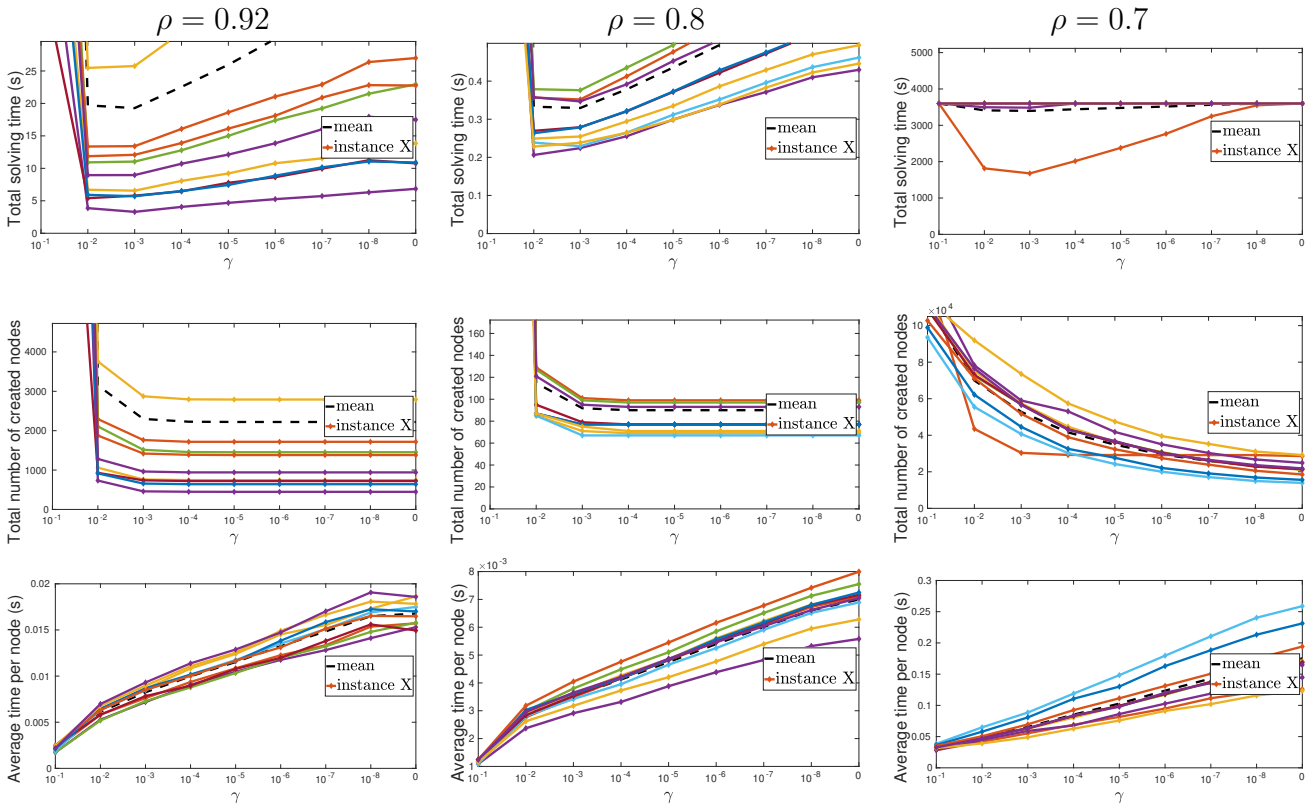


Figure 9.2 – Results of inexact convergence monitoring when using the $\ell_1 - \ell_1$ relaxation for structured sparsity problems ($\rho \in \{0.7, 0.8, 0.92\}$), one color per instance, the dashed line being the mean. All curves plot the performance as a function of γ against three different metrics: the total time to solve the problem to optimality (top), the number of nodes created to solve the problem (middle), and the average time spent on each node (bottom). The point $\gamma = 0$ corresponds to exact convergence.

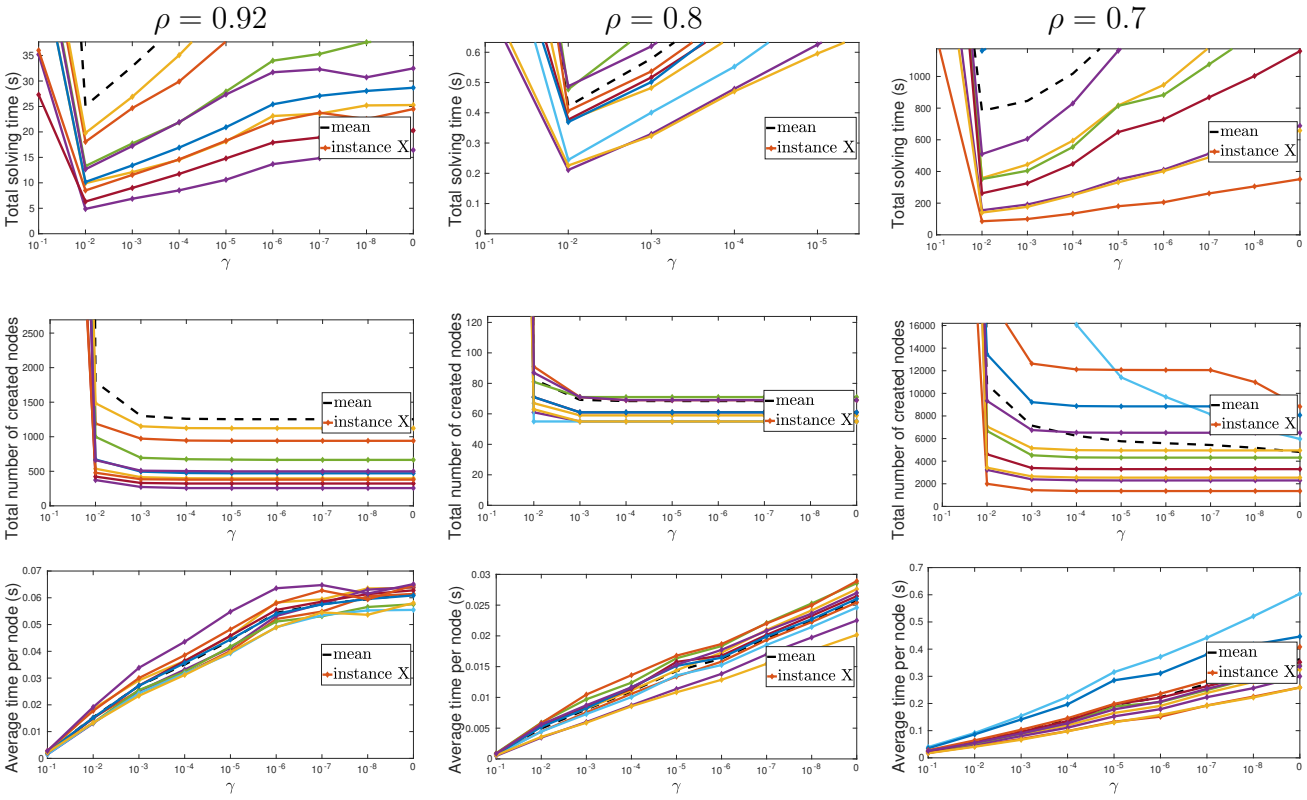


Figure 9.3 – Results of inexact convergence monitoring when using the $\ell_1 - \ell_\infty$ relaxation for structured sparsity problems ($\rho \in \{0.7, 0.8, 0.92\}$), one color per instance, the dashed line being the mean. All curves plot the performance as a function of γ against three different metrics: the total time to solve the problem to optimality (top), the number of nodes created to solve the problem (middle), and the average time spent on each node (bottom). The point $\gamma = 0$ corresponds to exact convergence.

9.4.3 Screening

In this section, we evaluate the performance of the screening tests derived previously in Sections 9.2.2 and 9.3.2, for the $\ell_1 - \ell_1$ and the $\ell_1 - \ell_\infty$ relaxations respectively.

Figure 9.4 shows the performance of the Gap-Safe screening for both relaxations. We draw some general observations similar to the ones of Chapter 5, namely that the performance of screening decreases as ρ increases, and that it can fix variables to 0 even in the case of nodes with a small amount of selected groups. Interestingly, this performance drop is much more visible for the $\ell_1 - \ell_\infty$ formulation than for the $\ell_1 - \ell_1$ formulation, with the mean percentage of screened variable for $\rho = 0.8$ being at 21.6% for the $\ell_1 - \ell_1$ formulation while it is at 3.17% for the $\ell_1 - \ell_\infty$ formulation. This is due to the fact that in the $\ell_1 - \ell_\infty$ relaxation, the screening method aims to set groups (containing 4 variables) to zero, whereas in the $\ell_1 - \ell_1$ relaxation, the screening method aims to set single variables to zero. Setting a group of 4 variables to zero is a stronger decision than setting a single variable to zero, therefore the test used for the $\ell_1 - \ell_\infty$ relaxation is more difficult to pass than the one used for the $\ell_1 - \ell_1$ relaxation, particularly when ρ is high. For $\rho = 0.92$, the average percentage of screening variable is at 19% for the $\ell_1 - \ell_1$ formulation, and at 2.05% for the $\ell_1 - \ell_\infty$ formulation. Consequently, the screening on the $\ell_1 - \ell_\infty$ relaxation is practically useless for $\rho = 0.8$ and $\rho = 0.92$. For $\rho = 0.7$, the average percentage of screening variable is at 62.8% for the $\ell_1 - \ell_1$ formulation, and at 35.5% for the $\ell_1 - \ell_\infty$ formulation, therefore for both formulations, the screening gives promising results. The difference between $\rho = 0.7$ and $\rho \in \{0.8, 0.92\}$ was already observed in the scalar case in Section 5.4.5 (page 75), and comes from the fact that as ρ decreases, the relaxations get easier and the screening is able to set more variables to 0.

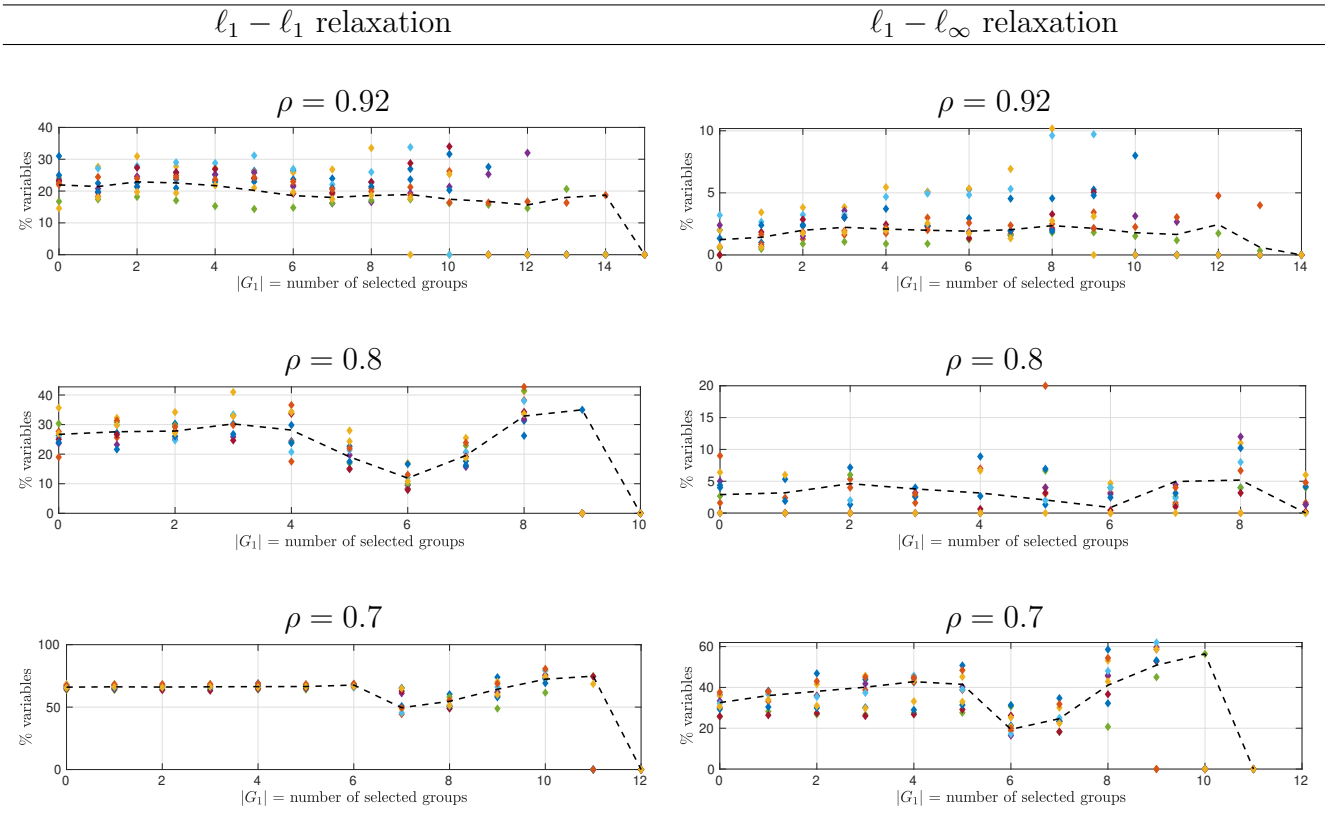


Figure 9.4 – Performance of the screening rule for discarding variables, expressed as a percentage of variables set to 0, given the number of selected groups of the underlying nodes, for the $\ell_1 - \ell_1$ and $\ell_1 - \ell_\infty$ formulations, with $\rho \in \{0.7, 0.8, 0.92\}$.

9.5 Discussion

Conclusion This chapter took the accelerations detailed in Chapter 5 and adapted them to the structured sparsity case with disjoint groups. To this end:

- the dual problem of the $\ell_1 - \ell_1$ formulation was derived, to get early pruning and inexact convergence techniques running;
- screening rules for both the $\ell_1 - \ell_1$ and $\ell_1 - \ell_\infty$ formulations were derived ;
- the performance of the three techniques detailed previously (early pruning, inexact convergence, and screening) was assessed through numerical simulations, showing the $\ell_1 - \ell_1$ formulation tends to get a greater benefit from these accelerations than the $\ell_1 - \ell_\infty$ formulation.

Perspectives On top of the perspectives for the accelerations themselves detailed in Chapter 5, several aspects can be developed to enhance these accelerations in the structured sparsity world. First, it could be possible to combine the dual expressions and the screening rules of the $\ell_1 - \ell_1$ and $\ell_1 - \ell_\infty$ formulations to build the dual problem and screening rules for the hybrid $\ell_1 - (\ell_1 + \ell_\infty)$ formulation (which is a promising formulation, as Chapter 8 showed). Assessing the performance of the different techniques on this hybrid formulation opens the path for benchmarking these techniques on datasets with overlapping groups. Will the different techniques suffer or benefit from this additional flexibility? Also, some overlapping group structure in the literature models a hierarchy between groups, or logical constraints [Bach et al. 2012; Jenatton et al. 2011]. It could be interesting to see if an optimization algorithm could be designed to explicitly use these constraints to reduce the search space and accelerate the computation of lower bounds. Finally, as this section showed that the acceleration techniques were more powerful on the $\ell_1 - \ell_1$ formulation than on the $\ell_1 - \ell_\infty$ formulation, most notably the screening method, it could be interesting to see if we can pass information from the $\ell_1 - \ell_1$ screening to help to solve the $\ell_1 - \ell_\infty$ formulation. A possible way is to apply the decisions of the $\ell_1 - \ell_1$ screening for the first iterations of the optimization algorithm solving the $\ell_1 - \ell_\infty$ formulation. Those decisions act as heuristics: the global minimum may be lost in the process. Then, after some iterations of the algorithm, the decisions of the $\ell_1 - \ell_1$ screening are cancelled, and the remaining iterations of the optimization algorithm are done using only the decisions of the $\ell_1 - \ell_\infty$ screening.

Chapter 10

Branch-and-bound algorithm compared to alternative methods

Contents

10.1 Introduction

In this chapter, the branch-and-bound method built throughout Chapter 7 to 9 is compared against state-of-the-art approaches. Two baseline approaches unrelated to this thesis will be used here for comparisons. The first one relies on a convex relaxation of the ℓ_0 problem. This approach leads to different solutions, but is obviously much more efficient computationally. The second approach uses CPLEX to solve the ℓ_0 problems, using a Mixed-Integer Problem formulation. This approach leads to the same solutions, and the computation time will be compared. These competing methods are detailed in Section 10.2, then numerical experiments are carried in Section 10.3 before concluding and drawing some perspectives in Section 10.4.

10.2 The competing methods

10.2.1 Convex relaxation

A standard way to solve structured sparsity problems in a separable case is to use the Group LASSO Problem (6.1), whose definition is recalled below:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \lambda \sum_{g\in G} \|\boldsymbol{x}_g\|_2. \tag{6.1}$$

In our case, to be closer to the setting of Problem $(\mathcal{P}_{2+0s})$, where a box constraint is present, the convex relaxation used will be:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \lambda \sum_{g\in G} \|\boldsymbol{x}_g\|_2 \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M. \tag{$\mathcal{P}^M_{2+12}$}$$

This corresponds to solving Problem $(\mathcal{P}^{\mathbf{N}}_{2+1q})$ (on page 95)at the root node of the branch-and-bound algorithm search tree, choosing $q=2$. Problem $(\mathcal{P}^M_{2+12})$ will be solved thanks to the IGD algorithm detailed in Section 8.5.2 (page 104).

10.2.2 Mixed-Integer problem

In this Part II, a branch-and-bound algorithm dedicated to solving Problem $(\mathcal{P}_{2+0s})$:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \mu \sum_{g\in G} \mathbf{1}_{\boldsymbol{x}_g\neq 0} \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M \tag{$\mathcal{P}_{2+0s}$}$$

is detailed. However, this is not the only method available for solving this problem. Indeed, one could also use a generic Mixed-Integer Programming (MIP) solver, such as CPLEX, GUROBI or CBC, to minimize Problem $(\mathcal{P}_{2+0s})$. To do that, Problem $(\mathcal{P}_{2+0s})$ has to be reformulated in a MIP form:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q, \boldsymbol{b}\in\{0,1\}^{|G|}} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \mu \sum_{i=1}^{|G|} b_i \text{ s.t. } \forall i \in \{1,..,|G|\}, \forall j \in G_i, -Mb_i \leq x_j \leq Mb_i \tag{10.1}$$

where G defines the set of groups, and G_i the i-th group in this set according to some ordering. This problem is similar to the scalar sparsity MIP formulation given by Problem (3.10), the only difference being that the binary variables are tied to groups instead of individual variables x_i. In other words, $b_i = 0 \implies \forall j \in G_i, x_j = 0$, and $b_i = 1$ is just a boundedness assumption on the different x_j.

10.3 Numerical experiments

The branch-and-bound method for structured sparsity problems developed in this thesis is compared on the solution quality side against the convex relaxation formulated by Problem ($\mathcal{P}^M_{2+12}$) in Section 10.3.2, and on the computing time side against CPLEX solving Problem (10.1) in Section 10.3.3.

10.3.1 Dataset

The datasets used are the same than the ones in Chapter 9. As a recall, the different parameters considered are in Table 10.1. For each instance, the ground truth solution has 6 groups of 4 non-zero variables, which means there are 24 non-zero variables.

Size	ρ	N	Q	K
Small	$\{0.8, 0.92\}$	500	100	6
Moderate	0.7	500	1 000	6

Table 10.1 – Parameters used for the different synthetic instances generated for evaluating the performance of the accelerations in the structured sparsity case. 10 instances are generated for each combination of parameters.

10.3.2 Comparison with convex relaxation

In this section, we will compare the branch-and-bound algorithm proposed in this thesis with the convex method detailed in Section 10.2.1.

Comparison protocol The goal here is to compare the quality of the solutions given by both methods. The metric used is the number of variables which are not matching the ground truth solution. In other words, given an estimate $\hat{\boldsymbol{x}}$ and a ground truth solution $\boldsymbol{x}_{\text{truth}}$, both with the same number of non-zero variables, the metric considered reads:

$$\mathcal{D}(\hat{\boldsymbol{x}}, \boldsymbol{x}_{\text{truth}}) = \|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\text{truth}}\|_0/2.$$

The lower the metric, the better.

Results As the tailored branch-and-bound algorithm always matches the ground truth, meaning the method of this thesis always recover the best solution possible, only the results concerning the convex method will be shown here. Table 10.2 shows the results of the convex method used when considering solution quality. For $\rho = 0.7$, the convex method is able to retrieve the correct support for each instance, owing to the easiness of the problems (compared to $\rho \in \{0.8, 0.92\}$). In this case, the benefit of the tailored

branch-and-bound algorithm, or any other algorithm exactly solving the ℓ_0 problem, is not on the solution quality, but on the *guarantee* of this quality. Indeed, the branch-and-bound algorithm has a guarantee of optimality, whereas the convex method does not. For $\rho = 0.8$, the convex method suffers from recovery errors on two instances, and for $\rho = 0.92$, no instance is correctly recovered, with a support error of one third (8.4) of the 24 components (6 groups of 4 variables) on average. As expected, the quality of the solution retrieved by the convex method degrades when ρ increases. Indeed, for the scalar case, it is known that under certain conditions such as RIP [Candès et al. 2006] or ERC [Tropp 2004], the ℓ_1-norm method is proven to retrieve the correct support. In our data generation setting, an extension of these conditions to the structured sparsity case, like Structured-RIP [Huang et al. 2011], would be fulfilled with small values of ρ.

	# instances with perfect recovery	mean value of $\mathcal{D}(\hat{\boldsymbol{x}}, \boldsymbol{x}_{\text{truth}})$
$\rho = 0.7$	10	0
$\rho = 0.8$	8	0.8
$\rho = 0.92$	0	8.4

Table 10.2 – Results for Problem ($\mathcal{P}^M_{2+12}$) regarding the solution quality on different values of ρ (left column). The metrics considered are the number of instances with perfect support recovery (middle column) and the mean value of support errors (right). The tailored branch-and-bound algorithm always matches the ground truth

10.3.3 Comparison with CPLEX

In this section, we will compare the branch-and-bound algorithm proposed in this thesis with CPLEX. As both methods are solving exactly the ℓ_0 problem, the solution is the same. The comparison concerns computation time.

Comparison protocol The experiments were carried with CPLEX v12.8.0. For the branch-and-bound algorithm of this thesis, given that each acceleration technique has an underlying cost, which means it is not always worth using it, several parameters for the accelerations were considered:

— for early pruning: disabling completely or enabling it at each iteration,

— for inexact convergence: $\gamma \in \{0, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$,

— for screening: disabling it completely, enabling it at each iteration, enabling it every 10 iterations.

Among all these configurations, the best one was kept to be compared against CPLEX. Both methods were run on one processor core, disabling parallelization.

Results Figure 10.1 shows the computing time of CPLEX and our branch-and-bound algorithm. For $\rho = 0.7$ (left), we can see that our algorithm is clearly better than CPLEX. The choice of the formulation ($\ell_1 - \ell_\infty$ relaxation) and the accelerations (early pruning enabled, $\gamma = 10^{-2}$, screening tests used every 10 iterations) is important here. Indeed, as seen in Chapter 8, solving the same dataset with the non-accelerated $\ell_1 - \ell_1$ formulation hits the timeout for each instance, whereas the non-accelerated $\ell_1 - \ell_\infty$ formulation hits the timeout for 3 instances, and the accelerated one for one instance only. By comparison, CPLEX hits the timeout for 7 instances. On average, the tailored branch-and-bound algorithm is 4.03 times faster than CPLEX for $\rho = 0.7, Q = 1\,000$. The picture is reversed for $\rho = 0.8$ (middle) and $\rho = 0.92$ (right), where CPLEX is a clear winner when compared to our algorithm. On average, CPLEX is 2.86 times faster for $\rho = 0.8$, and 45.3 times faster for $\rho = 0.92$. This can be explained by two factors. The first one is the correlation level ρ. When ρ increases, the lower bounds worsen, and the performance of the branch-and-bound depends more on the exploration strategy and the branching rule used. On these two aspects, CPLEX enjoys several decades of development and fine-tuning that could not possibly be matched during this thesis. The second factor is the dimension. Indeed, CPLEX evaluates the lower bounds through a generic quadratic programming algorithm. Generic quadratic programming algorithms are very efficient as long as the number of optimization variable is limited ($Q = 100$ for $\rho \in \{0.8, 0.92\}$). Conversely, as dimension grows, such algorithms do not scale well, and tailored ℓ_1-norm algorithms are more suited.

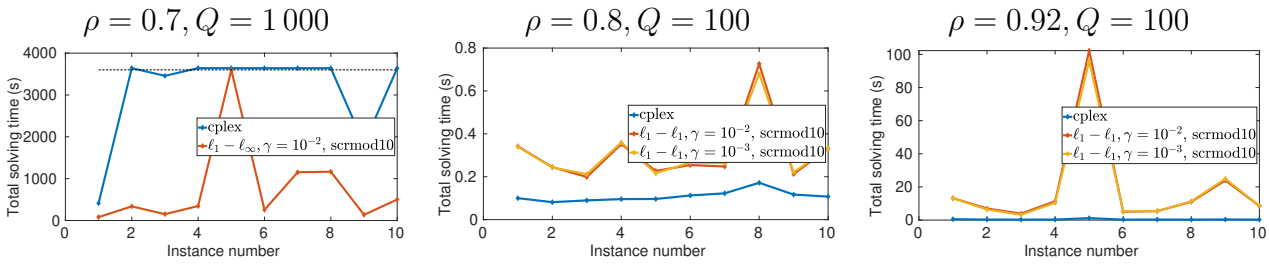


Figure 10.1 – Total computing time for instances with $\rho \in \{0.7, 0.8, 0.92\}$. CPLEX (blue) is compared against the branch-and-bound algorithm of this thesis (red and yellow), each time with early pruning enabled at each iteration and screening enabled every 10 iterations.

10.4 Discussion

Conclusion This chapter compared the performance of the branch-and-bound algorithm detailed in this thesis against state-of-the-art methods. To this end:

- two competing methods were chosen and detailed, namely a convex relaxation close to the $\ell_1 - \ell_2$ relaxation, and a Mixed Integer Programming reformulation of the ℓ_0 problem solved by CPLEX;
- the quality of the solution retrieved by the convex relaxation method was compared against the tailored branch-and-bound algorithm solutions, the latter one providing solutions of better quality for highly correlated problems;
- the computation time required to solve exactly the ℓ_0 problem was compared for CPLEX and the tailored branch-and-bound algorithm, the latter one providing quicker solving time for weakly correlated problems.

Perspectives The numerical results of this chapter draw two areas of development. The considered datasets were successfully solved by the tailored branch-and-bound, however they were either solved correctly by the convex method ($\rho = 0.7$) or solved faster with CPLEX ($\rho = 0.8$ and $\rho = 0.92$). A natural question is then to wonder what is the class of problems were using our tailored branch-and-bound is a net gain compared to state-of-the-art methods. In the same vein, identifying the cases were the convex method gives the same solution quality *in practice* would be a valuable contribution for the research community, helping it to decide when it is sufficient to use a convex method, and when an exact ℓ_0 method is required. A second question is about the reasons of CPLEX outperforming our branch-and-bound algorithm for $\rho = 0.8$ and $\rho = 0.92$. This is certainly due to the branching rules and exploration strategies used in CPLEX. What is the best choice for these two blocks of a branch-and-bound algorithm in the case of structured sparsity, and how to efficiently implement them, is an open question. Finally, one could wonder how the current code competes with other methods than the two presented here. Most notably, comparing the dedicated branch-and-bound against non-convex approaches would also be a valuable contribution.

Part III

Open-source software

Chapter 11

Software engineering concerns

This chapter describes some software engineering aspects of the branch-and-bound code used throughout this thesis, starting with the software architecture used in Section 11.1. Then, the way to take into account the group structure for the structured sparsity case is explained in Section 11.2 before concluding and drawing some perspectives in Section 11.3.

11.1 Modular branch-and-bound

Contents

11.1.1 Introduction

To test the different contributions of this work, which are generally "orthogonal" to each other (exploration strategies, branching strategies, lower bounds, accelerations inside these lower bounds), a modularization of the code developed during the former PhD thesis work by Ramzi Ben Mhenni [Ben Mhenni 2020] was done to ease the development of new contributions as well as their integration in the branch-and-bound code. The contribution here is a standard software development work: identify components which are building blocks of the method, define interfaces for the remainder of the code to interact with these components, and implement that in the actual code, which generally means make a redesign of the way code paths are calling each other. This way, we can go from a tightly coupled code to a more modular architecture with lightly coupled components. This code is available at `https://gitlab.univ-nantes.fr/samain-g/mimosa-solver`.

As this branch-and-bound is a research code, a particular attention has been put into the definition of correct and feature-complete interfaces between the components. Indeed, we wish to have several concurrent implementations of the same component living at the same time in our code base. Before going to the detailed UML (Unified Modeling Language) drawings in Section 11.1.3, the main components are described more informally in Section 11.1.2.

11.1.2 General components

What is a branch-and-bound algorithm? Basically, it is a main loop creating nodes and bounding these nodes in a specific order. This means we have these very fundamental components:

— the node division component,

— the node scheduling component,

— the node lower bounding component,

— the node upper bounding component.

This corresponds to the synthetic view of Algorithm 4 (page 34): the main loop takes a node from the node scheduler, divides it, and computes the bounds on the children node created this way. Each child which is not pruned is then given back to the node scheduler.

These components are drawn as UML interfaces in Figure 11.1. The simplest interfaces are `SplitInterface`, `LBInterface`, `UBInterface`, , corresponding respectively to the node division component, the node lower bounding component and the node upper bounding component. These interfaces are actually what C++ calls abstract object function classes. From an object oriented programming perspective, these are just abstract classes redefining the function call operator `operator()`, meaning the concrete objects from (concrete) derived classes will be callable as functions. From a procedural paradigm (Matlab, C, Fortran), abstract object functions are actually quite simple: they are function pointers (also called function handle in Matlab) with some inner state kept between different calls. As function pointers, they are just declaring that somewhere in the code will exist a function callable with the corresponding arguments. In a sense, this is a function specification. We can define several functions following this specification, and choose at run time which one will be used. The main interfaces are described below.

`SplitInterface` `SplitInterface` is declaring a specification for a function taking a node as an argument and returning a list of (usually two) children nodes. Different branching strategies can be coded, the final choice being done at run time, which allows for user-side configuration but also dynamic change of branching strategies while iterating in the branch-and-bound main loop. The last feature was not developed (but is doable quite quickly), while the user configuration is an implemented feature of the modular

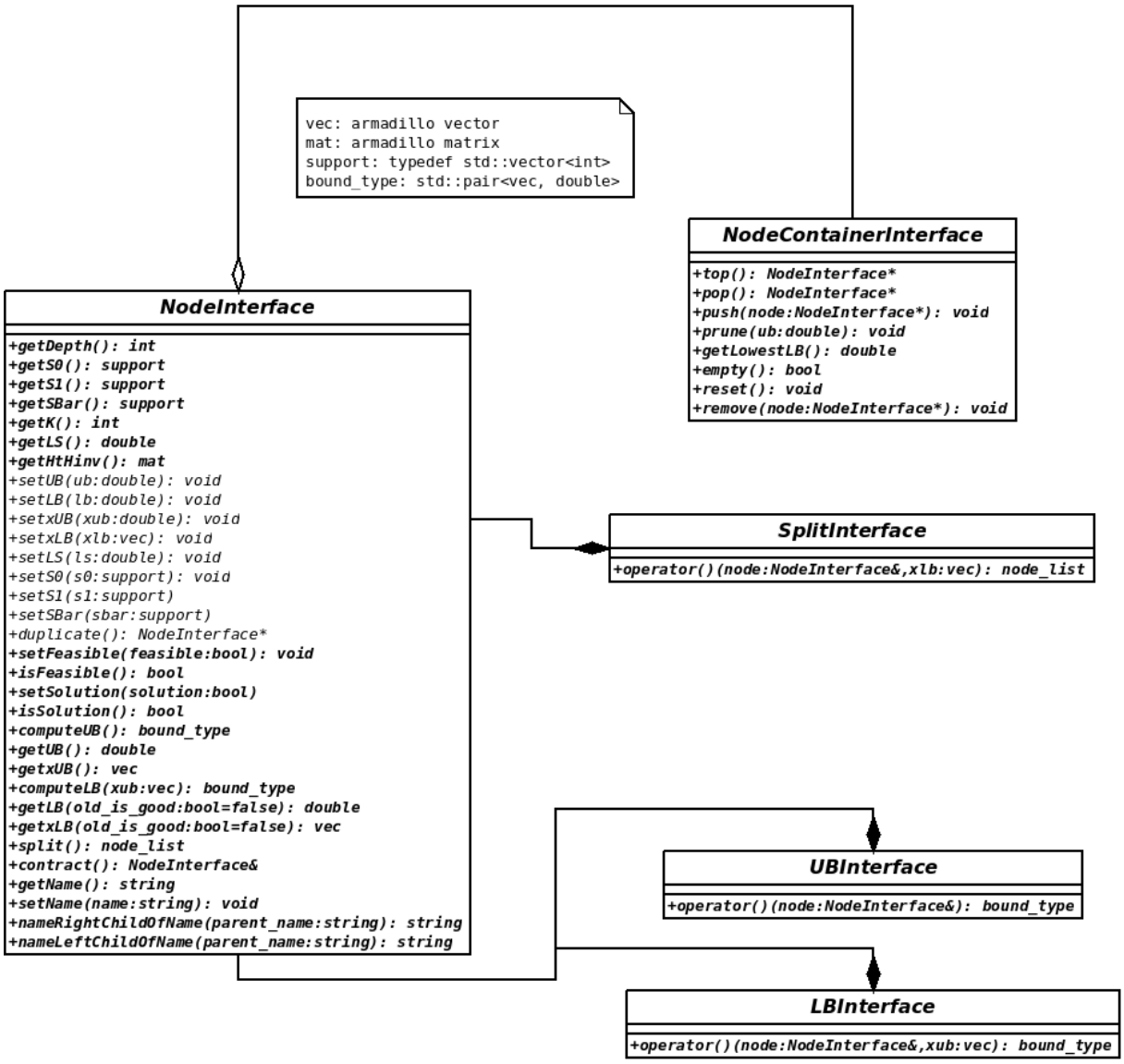


Figure 11.1 – The core interface providing the components of the branch-and-bound algorithm. The links displayed between interfaces are simplifications.

branch-and-bound. With the previous, tightly coupled code, that choice was done through a global variable set at compile time, meaning one had to modify the correct variable in the code and then recompile to test different branching strategies.

`UBInterface` `UBInterface` specifies a function taking a node and returning an upper bound on the problem at hand as well as its antecedent (the $\boldsymbol{x}^{\mathrm{UB}}$ point at which the upper bound value is reached). Several implementations of the upper bounding interface coexist, because the branch-and-bound code is actually solving more than just one problem. Indeed, the penalized Problem ($\mathcal{P}^M_{2+0}$) is considered, as well as constrained variants for scalar sparsity (these constrained variants are contributions of [Ben Mhenni 2020]), and structured sparsity problems (this is a contribution of this thesis, coming from Part II).

`LBInterface` For `LBInterface`, several implementations do exist to compute a lower bound of a given problem, for example Problem ($\mathcal{P}^{\mathbf{N}}_{2+0}$):

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|S_1| + \mu\|\boldsymbol{x}_{\bar{S}}\|_0 \text{ s.t. } \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = 0 \end{cases}. \qquad (\mathcal{P}^{\mathbf{N}}_{2+0})$$

Currently, for Problem ($\mathcal{P}^{\mathbf{N}}_{2+0}$), each implementation is a different algorithm finding the minimum of the same function, namely solving Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$):

$$\text{lb}^{\mathbf{N}} = \min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|S_1| + \tfrac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_1 \text{ s.t. } \|\boldsymbol{x}\|_\infty \leq M, \boldsymbol{x}_{S_0} = 0. \quad (\mathcal{P}^{\mathbf{N}}_{2+1})$$

If we used other problem formulations than Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$) to find a valid lower bound of Problem ($\mathcal{P}^{\mathbf{N}}_{2+0}$), the algorithms solving this new formulation would also be instances of `LBInterface`, which is specifying a function taking a node and the upper bound point given by `UBInterface`, and returns the lower bound value and its antecedent. This is the case for the structured sparsity Problem ($\mathcal{P}^{\mathbf{N}}_{2+0s}$):

$$\min_{\boldsymbol{x}} \ \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2 + \mu|G_1| + \mu\sum_{g\in\bar{G}} \mathbf{1}_{\boldsymbol{x}_g \neq 0} \quad \text{s.t.} \quad \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = \mathbf{0} \end{cases}, \qquad (\mathcal{P}^{\mathbf{N}}_{2+0s})$$

where different problems lower bounding the ℓ_0 one are used. Like `UBInterface`, more implementations exist in the code than presented in this report: they are tackling other problems described in [Ben Mhenni 2020]. The choice of adding the upper bound antecedent in the arguments is a design decision which could be removed in the future. In more pragmatic terms, it was put there because every algorithm implemented so far use it. If we find a case were this antecedent is not used, this could lead to a redesign of this interface.

More involved interfaces are also used, which are not reducible to object functions: we now define several methods operating on them.

`NodeContainerInterface` `NodeContainerInterface` corresponds to the node scheduler component, also named the exploration strategy. As highlighted in Chapter 4, an exploration strategy can be implemented as a data structure, therefore `NodeContainerInterface` specifies the interface of a data structure. The most important methods are `push` and `pop`, which are respectively storing a node in the node container (using some underlying data structure) and getting a node out of the node container. In the Rust programming language way of phrasing it, it owns nodes, it does not borrow them. This means that when we call `push` the node is absorbed in the node container and cannot be used elsewhere in the code as long as a call to `pop` did not eject that node from the container. Methods `push` and `pop` are encoding together a given exploration strategy. Methods `empty`, `reset`

and `remove` are utility methods, respectively checking if the container is empty, forcing the container to be empty, and removing a specific node. Note that in an usual workflow, `remove` is never used: node removal happens through `pop` and also `prune`. Indeed, `prune` removes nodes with a lower bound higher than the current value of $\overline{\text{ub}}$. While this check happens before inserting new nodes to the container as seen in Algorithm 4 (page 34), it also needs to be done on older nodes once the value of $\overline{\text{ub}}$ is updated. `top` returns the top-priority node according to the exploration strategy. As `remove`, it is currently more for debugging purpose than for a production workflow. `pop` can be seen as the combination of `top` and then `remove`. Finally, `getLowestLB` gives the lowest lower bound value among all nodes stored in the container. This can be used for a certificate of optimality when stopping the branch-and-bound before convergence. As the main goal of this thesis is to propose a method solving exactly ℓ_0 problems, this possibility of using a truncated branch-and-bound as a good ℓ_0 heuristic has not been investigated, therefore `getLowestLB` is not currently used (but is implemented for every data structure).

`NodeInterface` Finally, the most complex of the core interfaces encodes a so far unmentioned aspect of a branch-and-bound, namely the role of a node. We apply splitting operators, bounding operators on nodes, we push and pop them from containers. No algorithmic intelligence lies inside a node. However, a node is a storage vessel, and as such it has a separated interface. Currently there is one node implementation in use in the code, but it can change according to the storage needs. Indeed, the current implementation assumes we have enough memory for pre-computed quantities, even with millions of nodes stored at a given time. In a more restricted setup in terms of working memory, some quantities which are currently stored can be recomputed on-the-fly at the price of some computing time, leading to other node implementations. In a node, we typically store support information: the $S_1, S_0, \bar{S}$ supports, leading to the `getS1, setS1, getS0, setS0, getSBar, setSBar` methods, for getting and setting these supports (for structured sparsity detailed in Part II, we have the corresponding methods for handling group supports). The same thing goes for the bounds, with `getUB, setUB, getxUB, setxUB, getLB, setLB, getxLB, setxLB` manipulating either a bound or the antecedent (the minimizer) of a given bound. Nodes are attributed names to be able to log node-specific information to the user, this is done through the `getName, setName, nameRightChildOfName, nameLeftChildOfName` methods, the last two encoding the convention for naming children of a given node. Some utility functions were added, `getDepth` and `getK`, respectively giving the depth and the number of selected variables ($|S_1|$ quantity) of a node. Some quantities corresponds to partial terms, most notably used for the exploration strategy study detailed in Chapter 4. Namely, `getUBLS, setUBLS` handle the upper bound data fitting term $\frac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1}\|_2^2$ of Problem (3.14), `getLBLS, setLBLS` the corresponding term $\frac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1} - \mathbf{A}_{\bar{S}}\boldsymbol{x}_{\bar{S}}\|_2^2$ for the lower bound Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, and `getLBL1, setLBL1`

handle the sparsity term $\frac{\mu}{M}\|\boldsymbol{x}_{\bar{S}}\|_1$ of the same lower bound Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$). `getLPS, setLPS` are getting and setting the ℓ_1 path selection score, which is a score able to guide branching (node division, the splitting component obeying `SplitInterface`), see [Ben Mhenni 2020] for more details about this score. Some flags are here to drive the actions of the branch-and-bound main loop. Namely `isFeasible, setFeasible` respectively check and set whether the current node is feasible or not (this is guiding node division or pruning for constrained variant of Problem ($\mathcal{P}^{M}_{2+0}$)), while `isSolution, setSolution` check and set whether the current node's lower bound solution is already a solution of the original ℓ_0 problem being solved, meaning we actually found the minimum of Problem ($\mathcal{P}^{\mathbf{N}}_{2+0}$) for this node. In a Mixed-Integer Programming (MIP) framework, `isSolution, setSolution` correspond to checking if the relaxed variables $b_i \in [0, 1]$ are integral in the lower bound solution for a given node. `duplicate`, as its name suggests, duplicates a node. This is a raw operation exclusively used inside splitting component implementations. This method is meant to create children nodes from the current one, and as such some quantities are copied automatically, such as the bounds values. `getHtHinv` gives the quantity $(\mathbf{A}^T_{S_1}\mathbf{A}_{S_1})^{-1}$, the exact way to obtain this inverse (for example through some matrix factorization) being left to the node component implementation. Finally, there are also methods calling the splitting, bounding and contracting components accordingly, with the noteworthy presence of a boolean argument for `getLB` and `getxLB` telling the node if the lower bound of the parent node is good enough or if we should compute the exact lower bound of this node. Relying on the parent's lower bound may look rather strange, but it is actually the way we handle warm start in lower bound algorithms: at the beginning of the algorithm we ask the current node its by-default lower bound, which is the lower bound of its parent, and at the end of the algorithm we will erase these by-default values with the correct ones.

11.1.3 Detailed view on components

The goal of this section is to uncover more details about the current software architecture, both in the areas where modularization was needed as well as in areas where modularization is expected in the future, as displayed in Figure 11.2, as well as in a 3 pages long format in Appendix D. On top of the core components described in Section 11.1.2, several classes were added.

Class `ContractInterface` acts as a placeholder for methods able to reduce the dimension of the problem in a given node, called a contractor. The node screening of [Guyard et al. 2022] could be implemented through a contractor component. A technique such as the ℓ_0 screening of [Atamtürk et al. 2020] would also be implemented through subclassing `ContractInterface`.

Moreover, a `ContextData` class was added, to pass some global context to compo-

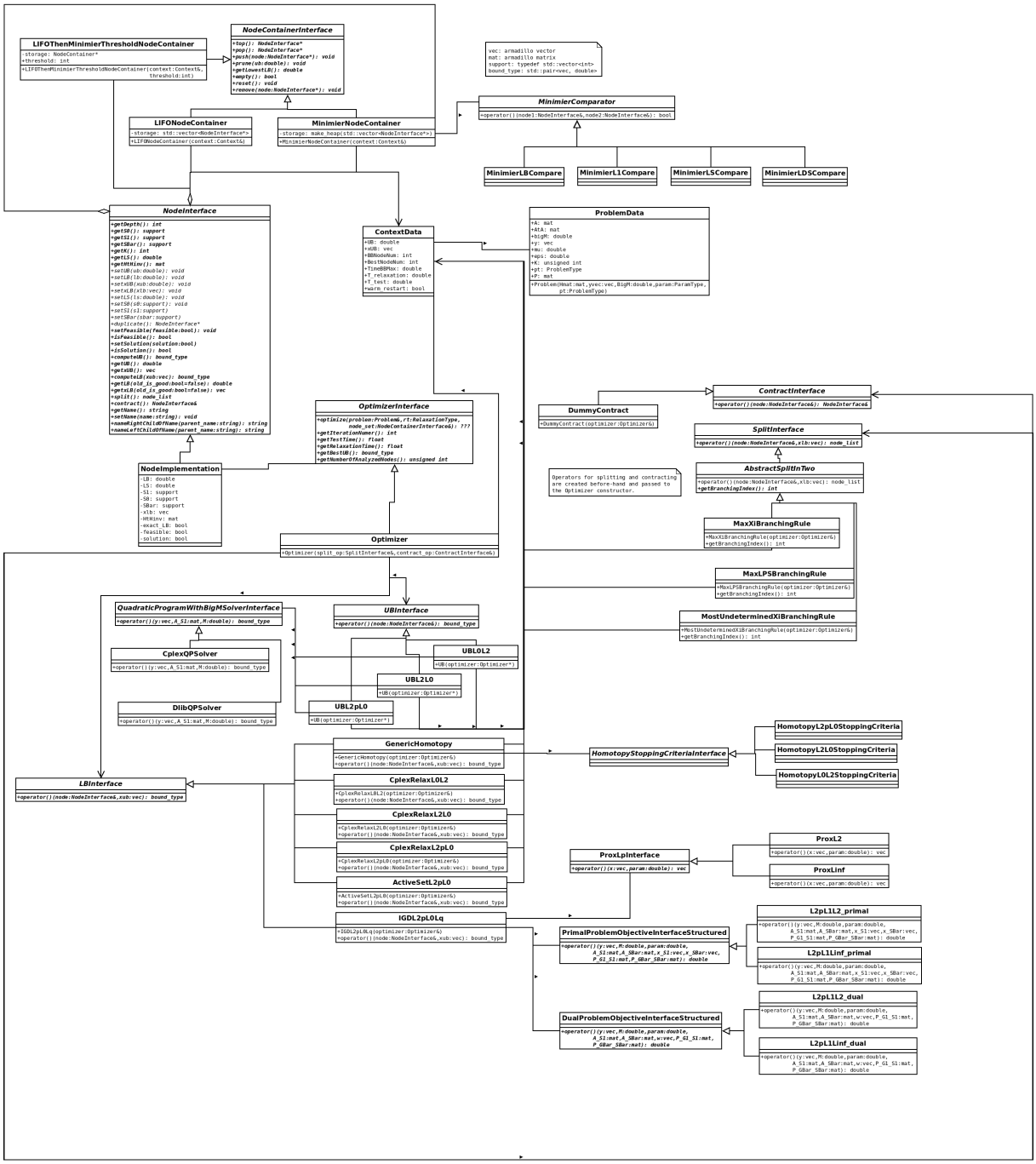


Figure 11.2 – Overview of all the classes of the code with methods details. Some classes were skipped for brevity.

nents. Originally created to blindly pack all the global variables that were present in the tightly coupled code, it progressively expanded to contain also information such as the current upper bound value and its antecedent, as well as a link to a `ProblemData` instance. `ProblemData` contains all the instance information, meaning it holds $\boldsymbol{y}$, $\mathbf{A}$, μ, M and the group structure in the case of structured sparsity. It also computes once, according to some flags, different derived quantities which are used in every node, such as $\mathbf{A}^T\mathbf{A}$, which is used every time the gradient of a least-squares term is needed, whose formula is $\mathbf{A}^T(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}) = \mathbf{A}^T\boldsymbol{y} - \mathbf{A}^T\mathbf{A}\boldsymbol{x}$. This avoids recomputing these quantities inside each node. This construction makes it easy to provide some information about global options and/or problem setup to components: we just have to pass the `ContextData` instance as a reference at the construction of the given component. For example, one branching strategy (one node splitting component), implemented as a derived class of `SplitInterface`, named `MaxNormalizedL1GiBranchingRule`, chooses in the structured sparsity case the undecided group to branch on by looking at the lower bound antecedent $\boldsymbol{x}^{\mathrm{LB}}$ of Problem ($\mathcal{P}_{2+11}^{\mathbf{N}^{\mathrm{weight}}}$) and taking the undecided group with the maximum ℓ_1-norm, normalizing by the group size. The group size is given by the group structure, which is not stored in each node: that would take an unnecessary amount of memory. Instead, a reference to `ContextData` is passed at the construction of the `MaxNormalizedL1GiBranchingRule` object to retrieve the group structure. We could have passed a reference to `ProblemData` instead. The choice of `ContextData` was driven by genericity: even if the implementation of `MaxNormalizedL1GiBranchingRule` changes and uses options which are not in `ProblemData`, it will most likely finds every information it needs in the node and the `ContextData` object. Also, this eases usage of components: if a component needs some global information, it always uses a `ContextData` object, regardless of the precise information it retrieves from this context object.

Let us now look briefly at the different subclasses of the core components described in Section 11.1.2.

`NodeContainerInterface` The derived classes of `NodeContainerInterface` are following the strategies described in Chapter 4. `LIFONodeContainer` implements a stack data structure, meaning a Depth-first exploration strategy. This is nothing more than the standard stack data structure in the `NodeContainerInterface` framework. In particular, no reordering of nodes occurs. This means that depending on the order in which someone `push`es the right and left child of a given node, the Depth-first exploration will favour in priority the left branch or the right branch. `MinimierNodeContainer` implements a generic heap, independent of a given comparison operator. This means it implements at the same time heaps sorted on the lower bound, on the ℓ_1 value of $\boldsymbol{x}^{\mathrm{LB}}$, etc. Internally, `MinimierNodeContainer` is a C++ template with a parameter, a comparator, which is an object function. `MinimierLBCompare` implements the lower bound (partial)

sorting, used for Best-first search. `MinimierL1Compare` is used for ℓ_1-first exploration, `MinimierLSCompare` for Least-square-first exploration, `MinimierLDSCompare` for Limited Discrepancy Search exploration (see Chapter 4 for more details on all these exploration strategies). `LIFOThenMinimierThresholdNodeContainer` implements the mixed strategies described in Chapter 4 using the same template mechanism than `MinimierNodeContainer`.

`UBInterface` For the upper bound, we have one subclass for each problem formulation. `UBL2pL0` is the one used for solving Problem (3.14):

$$\text{ub}^{\mathbf{N}} = \min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{y} - \mathbf{A}_{S_1}\boldsymbol{x}_{S_1}\|_2^2 + \mu|S_1| \text{ s.t. } \begin{cases} \|\boldsymbol{x}\|_\infty \leq M \\ \boldsymbol{x}_{S_0} = 0 \\ \boldsymbol{x}_{\bar{S}} = 0 \end{cases}. \tag{3.14}$$

As all the variants are really close to each other, internally all the subclasses, during their operation, call the same object function `QuadraticProgramWithBigMSolverInterface` whose goal is to solve a least-square problem under box constraints. The different upper bound subclasses are just recombining the outputs of that solver to get the precise objective function of the problem formulation at hand. Two implementations of `QuadraticProgramWithBigMSolverInterface` exists, one using the CPLEX quadratic solver, called `CplexQPSolver`, and one using a quadratic solver of an open-source library called dlib [1], called `DlibQPSolver`, in order to compare the performance of both solvers.

`SplitInterface` Splitting implementations work like upper bound implementation, and the `MinimierNodeContainer` class: there is one entry point doing the heavy lifting, and then several small variants. `AbstractSplitInTwo` does the main job of splitting a node into two children, and delegates to subclasses the work of picking the variable to branch on to make the two children. The subclasses `MaxXiBranchingRule`, `MostUndeterminedXiBranchingRule`, `MaxLPSBranchingRule` are just overloading the hook method `getBranchingIndex` which is left pure virtual (therefore undefined) in `AbstractSplitInTwo`.

`OptimizerInterface` The current layer of abstraction (and separation of concerns) with `OptimizerInterface` on one hand and `Optimizer` on the other hand is unnecessarily complex in the current implementation: we could do the same thing by getting rid of `OptimizerInterface`. However, this was left here to ease future developments of other main branch-and-bound loops. If parallel or distributed implementations of the branch-and-bound are developed, this will most likely lead to other subclasses of `OptimizerInterface`.

1. Accessible on `http://dlib.net`

`NodeInterface` This is approximately the same case than for `OptimizerInterface`: we have currently something too complex for our current needs, the actual objects in use being instances of `OptimizerNode` which derives from `CommonNodeImplementation` which itself derives from `NodeInterface`. In particular, while the intermediate level `CommonNodeImplementation` eased the implementation transition which happened during this thesis (from the tightly coupled code to the modular one), its presence in the final code is highly questionable, and actually depends on whether future implementation on node classes using different trade-offs (in terms of memory consumed vs cpu time) will share common points with the current `OptimizerNode`.

`LBInterface` As a great part of the work of [Ben Mhenni 2020] and of this work is dedicated to computations of lower bounds, it is not a surprise if it is one of the richest class hierarchy in the code. Similarly to the upper bounds, we must be able to compute lower bounds for different problem variants. On top of this, several algorithms were implemented and compared. During this thesis, the work of [Ben Mhenni 2020] was repacked into components. This concerns the `CplexRelaxXXX` subclasses, as well as `GenericHomotopy` and `ActiveSetL2pL0`. The `CplexRelaxXXX` subclasses are using a MIP formulation given to the CPLEX generic solver to compute the lower bounds problems such as Problem ($\mathcal{P}^{\mathbf{N}}_{2+0}$). No work was done on them during this thesis. Only repackaging happened. For `ActiveSetL2pL0` and `GenericHomotopy`, accelerations of Chapter 5 were implemented. `GenericHomotopy` uses different stopping criteria according to the problem variant considered, the stopping criterion being specified by the interface `HomotopyStoppingCriteriaInterface`. `HomotopyL2pL0StoppingCriteria` is the criterion used to solve Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$). `ActiveSetL2pL0` corresponds to an active set algorithm (see Section 3.3.1 starting on page 25 for more details), while `GenericHomotopy` corresponds to a homotopy continuation algorithm (see Section 3.3.1 for more details). During this thesis, several coordinate descent algorithms were also implemented, among which the one solving structured sparsity for non-separable formulation, IGD (see Algorithm 9 on page 105), which corresponds to the class `IGDL2pL0Lq`. Contrary to an active set method or an homotopy continuation method, where the main loop stops by design in a fixed number of steps, the definition of a coordinate descent algorithm requests a stopping criterion. Coordinate descent algorithm is an iterative procedure, where the convergence is asymptotic, and stopping criteria often used are a maximum number of iterations, a small change between consecutive iterates ($\|\boldsymbol{x}^k - \boldsymbol{x}^{k-1}\|_2 \leq \epsilon_x$), a small change of objective function between consecutive iterates ($\|f(\boldsymbol{x}^k) - f(\boldsymbol{x}^{k-1})\|_2 \leq \epsilon_f$). Here, we will use the fact that we know the dual of our problem of interest and will prefer a small duality gap $G(\boldsymbol{x}, \boldsymbol{w}) \leq \epsilon_g$ as our stopping criterion, knowing from the theory of convex analysis that $G(\boldsymbol{x}, \boldsymbol{w}) = 0$ is a sufficient condition for convergence. This, paired with the need to be problem-agnostic at the higher level (so that we can for ex-

ample use early pruning and screening with the generic code inside `GenericHomotopy`) leads to the creation of primal objective and dual objective interfaces, respectively named `PrimalProblemObjectiveInterface` and `DualProblemObjectiveInterface`. These are specifying interfaces for object functions. For example, the pair used for the $\ell_1 - \ell_\infty$ Problem ($\mathcal{P}^{\mathbf{N}}_{2+1\infty}$) is `L2pL1Linf_primal` and `L2pL1Linf_dual`. Also, as the IGD algorithm requires computing the proximal operator of the non-separable formulation at hand, and as it can solve several different non-separable formulations, the way to compute the proximal operator was also abstracted through `ProxLpInterface`, with `ProxL2` giving the proximal operator for the $\ell_1 - \ell_2$ formulation, while `ProxLinf` gives the proximal operator of the $\ell_1 - \ell_\infty$ operator.

11.2 Structured sparsity: group indicator matrix P

In the structured sparsity case, another issue arises due to the grouping. To be able to handle the general case of groups, with varying size and overlaps, while still keeping a unified code as simple as possible, we internally use a matrix called $\mathbf{P}$ ("partitioning" matrix), whose general term is $\forall j \in \{1, .., |G|\}, \forall i \in \{1, .., Q\}, P_{ji} = \mathbf{1}_{g_j}(i)$. The goal here is to leverage the performance of linear algebra libraries, relying on several decades of hardware optimization for vector calculus, (with for example development of the Single Instruction Multiple Data (SIMD) instructions of processors), to be computationally more efficient than with sequential loops on a list of groups.

In other words, $\mathbf{P} \in \mathbb{R}^{|G| \times Q}$ has this kind of shape:

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Each row of $\mathbf{P}$ corresponds to a group. In each row j, we have a 1 at P_{ji} if x_i belongs to the j-th group. In the previous example of $\mathbf{P}$, there are 7 variables and 4 groups, these groups being:

— $g_1 = \{x_1, x_2\}$,

— $g_2 = \{x_2, x_3, x_5\}$,

— $g_3 = \{x_4\}$,

— $g_4 = \{x_2, x_3, x_5, x_6, x_7\}$.

With this matrix, several computations about inclusion of variables into groups are solved using vector (or matrix-vector) calculus. A first example is how to compute the

cardinality of each group: $|g_j|$. For this, we just sum all the columns together giving us:

$$\boldsymbol{p}^{\text{colsum}} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 5 \end{pmatrix}.$$

Moreover, summing all the rows together, we get:

$$\boldsymbol{p}^{\text{rowsum}} = \begin{pmatrix} 1 & 3 & 2 & 1 & 2 & 1 & 1 \end{pmatrix}.$$

This gives us the number of groups containing a given variable (x_1 belongs to one group only, x_2 to 3 groups, and so on). After that, identifying overlapping and disjoint groups is easy: we just have to select the indices of the variables with a score greater than 1. Using something such as the Matlab expression `find(`$\boldsymbol{p}^{\texttt{rowsum}}$ `> 1)` would give us the following indicator vector:

$$\boldsymbol{p}^{\text{overlapindex}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

This index can be used to restrict the $\mathbf{P}$ matrix to the overlapping variables:

$$\mathbf{P}^{\text{overlap}} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

Summing all the columns gives us the number of overlapping variables each group contains:

$$\boldsymbol{p}^{\text{overlapcolsum}} = \begin{pmatrix} 1 \\ 3 \\ 0 \\ 3 \end{pmatrix}.$$

Overlapping groups are all the groups where $\boldsymbol{p}_j^{\text{overlapcolsum}} > 0$. At a given node, looking for the overlapping groups of $\bar{G}$ amounts to applying the steps above on a submatrix of $\mathbf{P}$ containing only the $\bar{G}$ lines. Also, note that if we do $\mathbf{P}|\boldsymbol{x}|$, the result we get is the ℓ_1 norm of all the groups:

$$\mathbf{P}|\boldsymbol{x}| = \begin{pmatrix} \|\boldsymbol{x}_{g_1}\|_1 \\ \|\boldsymbol{x}_{g_2}\|_1 \\ \|\boldsymbol{x}_{g_3}\|_1 \\ \|\boldsymbol{x}_{g_4}\|_1 \end{pmatrix}.$$

$\mathbf{P}|\boldsymbol{x}|$ can also be written as $\sum_{i=1}^{Q} \boldsymbol{p}_i |x_i| = \sum_{i=1}^{Q} (\mathbf{P} \circ |X|)_i$ where $X \in \mathbb{R}^{|G| \times Q}$ duplicates $\boldsymbol{x}^T$

on each row and $\circ$ denotes the Hadamard-Schur (term-by-term) product. This allows us to do with matrix-vector products the ℓ_q norm of the different groups, given any value of $q \in \mathbb{N}^* \cup \{\infty\}$. For $q \in \mathbb{N}^*$, the general formula is:

$$\begin{pmatrix} \|\boldsymbol{x}_{g_1}\|_q \\ \|\boldsymbol{x}_{g_2}\|_q \\ \|\boldsymbol{x}_{g_3}\|_q \\ \|\boldsymbol{x}_{g_4}\|_q \end{pmatrix} = \left(\sum_{i=1}^{Q} (\mathbf{P} \circ |X|^q)_i \right)^{1/q}$$

where $.^q$ stands for the term-by-term power operator. In particular, for the ℓ_2 norm:

$$\begin{pmatrix} \|\boldsymbol{x}_{g_1}\|_2 \\ \|\boldsymbol{x}_{g_2}\|_2 \\ \|\boldsymbol{x}_{g_3}\|_2 \\ \|\boldsymbol{x}_{g_4}\|_2 \end{pmatrix} = \sqrt{\sum_{i=1}^{Q} (\mathbf{P} \circ |X|^2)_i}.$$

where $\sqrt{...}$ applies term-by-term. For $q = \infty$, the formula is:

$$\begin{pmatrix} \|\boldsymbol{x}_{g_1}\|_\infty \\ \|\boldsymbol{x}_{g_2}\|_\infty \\ \|\boldsymbol{x}_{g_3}\|_\infty \\ \|\boldsymbol{x}_{g_4}\|_\infty \end{pmatrix} = \max_{i \in \{1,..,Q\}} (\mathbf{P} \circ |X|)_i.$$

where max applies term-by-term.

While this matrix is quite useful and allows parallelization of some computations, it is costly in memory. Indeed, $\mathbf{P} \in \mathbb{R}^{|G| \times Q}$. If we have a number of groups which is proportional to the number of components, which happens for example in the case of fixed size disjoint groups (like Group LASSO), then $\mathbf{P}$ grows quadratically as Q increases, which is the same rate as $\mathbf{A}^T\mathbf{A}$. So, in practice, using $\mathbf{P}$ limits us to no more than $Q = 10^4$. For now, no alternative code path (like encoding $\mathbf{P}$ as a sparse matrix) were developed to overcome this scaling issue.

11.3 Discussion

Conclusion This chapter presented the software architecture of the branch-and-bound algorithm used in this thesis, as well as the way to handle structured sparsity. To this end:

- the need for modularity in the code base was introduced;
- the core components were given abstract interfaces with precisely defined operations;
- several different implementations of the same component were described;
- the matrix P used for handling the structured sparsity case was detailed.

On a software engineering perspective, the current implementation draws several areas of future work. First, on a fine-grained level, some data types could be refined, especially when variables of these types arise very frequently. This is the case for the supports (the different $G_1, G_0, \bar{G}, S_1, S_0, \bar{S}$), which are currently defined as vectors of integers, whereas a bit-array implementation would be much more memory efficient. On a higher level, several places of the code uses a generic function which is further refined. This is the case for `AbstractSplitInTwo` and `IGDL2pL0Lq` for example. However, different ways to implement this kind of behaviour were used: subclassing with overloading of hook methods in one hand (case of `AbstractSplitInTwo`), and passing object function references to generic functions on the other hand (case of `IGDL2pL0Lq`). Standardizing the practice throughout the code would ease future collaborative work on this code base.

Several leads can be followed to enhance the scalability of the branch-and-bound. Apart from tiny refactoring and optimization of the implementation here and there, four classes of levers could be used in this code. The first one relates to saving computations. Indeed, as things are currently standing, all instances are treated as completely different problems. While this is appealing to get a simple generic code, in real applications we can expect the model at hand to be reused for several instances. In other words, $\boldsymbol{x}$, $\boldsymbol{y}$ and potentially μ are specific to each instance, but the matrix $\mathbf{A}$ and potentially the group structure are common to several instances. This means that some computations on the matrix $\mathbf{A}$ (in particular when computing column norms, or eigenvalues of submatrices) could be done once for a batch of instances, instead of being done for each instance.

A second lever is to be able to perform computations on a matrix regardless of its ability to fit in memory. Out-of-memory computations require an abstraction layer between the object manipulated and the exact location of the data corresponding to this object. Once that layer is set up, the data could be in memory if it fits, on disk if it does not, distributed on several computers, or even stored on a remote storage service. This would allow us to use the current code for instances which do not fit in memory. However, care must be taken, as out-of-memory computations is no silver bullet: if the data are not in

memory, this will most probably mean a latency increase to retrieve the needed content, which means the wall-clock computation time will explode because the code will often wait for data transfers to happen. To overcome that, two mitigation techniques can be used. The first technique is to reduce the size/frequency of the issue. Basically, it amounts to try to find other ways of computing our numbers so that we use more compact data. The second technique, caching, consists in seeing if the huge object we are manipulating has some "hot" and some "cold" parts. Typically, if a matrix is too huge to fit in memory, we would wonder if some columns of this matrix are frequently used, the "hot" parts, the rest being used quite infrequently, the "cold" parts. The "hot" parts are then stored in memory, drastically reducing latency, while the cold parts are accessed through standard out-of-memory transfers, keeping the memory consumption at an acceptable level.

A third lever is to be smarter in the way to handle matrices. Indeed, for now all the matrices are considered dense and stored that way. Detecting sparse matrices, storing them efficiently is a first step. A second step is to approximate dense matrices by sparse ones. In that case, if we're still interested in the global optimality property of the solution retrieved by the branch-and-bound, we need to monitor the approximation quality.

Finally, extending the implementation to have, in a similar way to CPLEX, a multi-threaded, or even distributed (on multiple machines) branch-and-bound is manageable yet non trivial. Indeed, data synchronisation issues must be taken into account, which will limit the ability to benefit from parallelization. Most probably, tackling these issues will imply having another implementation of `OptimizerInterface`. Moreover, as the best upper bound $\overline{\text{ub}}$ is shared across all nodes, it means it should be shared across all threads of the branch-and-bound. This means that `ContextData` should be modified, perhaps by separating into two different classes the quantities which are read-only (and poses no data synchronization issues) and the ones which are updated during the branch-and-bound (where some data synchronization must be provided). Another option would be to keep a local $\overline{\text{ub}}_{\text{thread}}$, which is the lowest upper bound among all the nodes explored by the current thread. As we have by construction $\overline{\text{ub}}_{\text{thread}} \geq \overline{\text{ub}}$, less nodes will be pruned. Depending on the amount of nodes whose lower bounds are in between $\overline{\text{ub}}$ and $\overline{\text{ub}}_{\text{thread}}$, the wall-clock time of the parallel branch-and-bound may be lower or greater than it's single-core, sequential counterpart (which is the current implementation). Of course, between always-on synchronization and no synchronization, there is a middle ground which can be explored too. To balance the work between the different threads, a work-stealing approach could be used [Wikipedia contributors 2023b], where each thread has its own tasks, and thread with no remaining tasks steals some to busy threads. In our case, a task is a node to evaluate, and work-stealing would be implemented by having each thread hold its own instance of an implementation of `NodeContainerInterface`, implementation which must be thread-safe to allow a thread to steal some nodes from a busy thread without synchronization issues. This need to keep threads "busy" will drive a particular way to

assign nodes to threads, resulting in some kind of meta exploration strategy. How this meta exploration strategy affects the overall performance is an open question. The other components of the branch-and-bound take place on one node at a given time, and should pose no problem for parallelizing the branch-and-bound code.

Chapter 12

General conclusion and perspectives

Based on the branch-and-bound algorithm first developed in [Ben Mhenni 2020; Ben Mhenni, Bourguignon, and Ninin 2021], this thesis proposed three types of contributions. First, acceleration techniques, located in the choice of the exploration strategy, as well as in leveraging convex duality for the ℓ_1-norm optimization problems used to compute lower bounds, were proposed in Part I. The potential of a given technique is heavily influenced by the problem at hand. The choice of an exploration strategy is of particular importance when the matrix of the problem is very correlated, represented in this thesis by datasets with ρ close to 1. When the matrix is less correlated, which corresponds to datasets with ρ close to 0, all exploration strategies tend to give similar performance. Early pruning and screening techniques are influenced in the other way: these techniques are less effective when the matrix of the problem is more correlated. Moreover, these techniques have a computational cost, so to use them as efficiently as possible, one must apply them only when they are worth it. In this thesis, the efficiency of early pruning and screening was assessed according to $|S_1|$, the number of selected variables, inside each node created during the branch-and-bound algorithm iterations. It was shown that early pruning performs better for nodes with a good amount of selected variables with respect to the sparsity of the ground truth solution, while the screening method performs better for nodes with few selected variables. Other criteria for choosing when to apply a given technique may be worth being explored. Then, an extension of the original algorithm to tackle more general structured sparsity optimization problems, where the sparsity prior is put on groups of variables, was proposed and discussed in Part II. Using the structure of the problem makes the branch-and-bound method of this thesis, which is the first work towards a dedicated algorithm for exactly solving ℓ_0 structured sparsity problems, able to tackle problems with a higher number of active variables. However if this dedicated method outperforms generic solvers such as CPLEX in high-dimensional, lowly correlated settings, it is not competitive in the case of low-dimensional, highly correlated datasets. This is most probably due to the decades of research in combinatorial optimization put into such generic solvers, which could not be fully implemented in our approach in a reasonable time. Finally, in the hope of making the current code a usable tool for the research

community, software engineering contributions were detailed in Part III. In particular, the current software architecture strived for modularity, and is a first step towards more parallelization of the code.

This work opens the path for future works in different areas. On the software engineering side, integrating the current code, as well as the datasets used, into a standardized benchmarking solution such as BenchOpt [T. Moreau et al. 2022] [1], which is a tool for comparing optimization algorithm in a reproducible way, would be a valuable contribution for making the software more accessible to new users. This would also allow an easier comparison with competing methods, as well as using other datasets in a more streamlined way. On the modeling side, taking a step back, this thesis focused on Problem $(\mathcal{P}_{2+0})$, which reads:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 + \mu\|\boldsymbol{x}\|_0, \tag{$\mathcal{P}_{2+0}$}$$

as well as its structured sparsity extension. Three perspectives arise. The first one relates to the sparsity term $\mu\|\boldsymbol{x}\|_0$. Tuning μ is not straightforward given an application problem, in part because we may not know in advance what is the preferred trade-off between sparsity and data fidelity. To overcome this, a bi-objective formulation could be used, such as:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \left\{\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2, \|\boldsymbol{x}\|_0\right\}. \tag{$\mathcal{P}_{\{2,0\}}$}$$

Solving Problem $(\mathcal{P}_{\{2,0\}})$ allows the user to choose among a series of possible trade-offs between data fidelity and solution sparsity. Since the ℓ_0 term can only take integer values, it roughly amounts to solving a sequence of the cardinality constrained Problem $(\mathcal{P}_{2/0})$ which reads:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2 \text{ subject to (s.t.) } \|\boldsymbol{x}\|_0 \leq K, \tag{$\mathcal{P}_{2/0}$}$$

for $K \in \{1,..,K_{\max}\}$. This could be done by allowing the branch-and-bound algorithm to compute several bounds for the same node, one for each value of $K \in \{1,..,K_{\max}\}$. As explained in [Ben Mhenni 2020], this could be done with the same cost as for solving Problem $(\mathcal{P}_{2/0})$ for $K = K_{\max}$ only, using the homotopy algorithm, which is an efficient strategy in low-dimensional settings, to compute the lower bounds. In higher dimension, it was shown (see Section 5.2.3 on page 57 and [Bach et al. 2011]) that other algorithmic designs (e.g. coordinate descent algorithms) are better suited for the computation of lower bounds. How to achieve an efficient bi-objective resolution in such settings or for structured sparsity cases remains to be explored.

A second modeling perspective is related to the data fidelity term $\|\boldsymbol{y}-\mathbf{A}\boldsymbol{x}\|_2^2$. While quite standard, this choice of cost function is restrictive, and may not be appropriate for every application case. In the general case, the branch-and-bound algorithm of this thesis can be straightforwardly adapted to problems using different data fidelity terms,

1. Available at `https://benchopt.github.io`.

as long as an efficient algorithm for computing the ℓ_1-norm-relaxed problem is available. Less standard data fidelity terms may be worth being considered. For example, when estimating the reproduction coefficient of Covid-19 [Abry et al. 2020], the state-of-the-art epidemiology model leads to a data fidelity term which is the Kullback-Leibler divergence. Also, sparsity prior reformulations can be used in classification tasks, most notably for allowing a small number of outlier samples to be misclassified, as Support Vector Machine [Wikipedia contributors 2023a] does when using a soft-margin. Classification tasks also use different data fidelity terms.

A third modeling perspective is related to the structured sparsity problem used. It is a direct extension of its scalar sparsity equivalent, and methods to tackle overlapping groups, and groups of varying size, were shown. However, it does not handle all possible cases of structured sparsity. For example, our approach is not suited to model constraints of mutual exclusion, where selecting a variable implies excluding another one. Extending the method to integrate other structured sparsity constraints is worth being studied.

Finally, on a practitioner side, a branch-and-bound algorithm which converges to the optimal solution in an unknown amount of time may not be adequate. In practice, it could be preferable to have a branch-and-bound algorithm running using a fixed time budget, giving the best solution found in this time. To this end, anytime exploration strategies [Luo et al. 2021] is part of the answer. In this case, what is the best upper bound algorithm (for example an ℓ_0 heuristic algorithm such as OMP, OLS, IHT, ...) for giving the best estimate in a given time budget is an open question.

Appendix A

Standard results

This appendix is dedicated to the proofs of some standard results, which are given for the sake of completeness to readers who are new to the field.

A.1 Sketch of the proof for the solution of Problem (3.2)

We wish to recover the minimum of the following problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^Q} \tfrac{1}{2}\|\mathbf{A}^T\boldsymbol{y}-\boldsymbol{x}\|_2^2 \text{ s.t. } \|\boldsymbol{x}\|_0 \leq K.$$

This problem can be formulated as a bi-level optimization procedure: first find the correct location of the non-zeros, satisfying $\|\boldsymbol{x}\|_0 \leq K$, and then optimize the amplitudes of these non-zero components:

$$\min_{S\subset\{1,..,Q\} \mid |S|\leq K} \tfrac{1}{2}\left(\sum_{i\notin S}(\boldsymbol{a}_i^T\boldsymbol{y})^2 + \sum_{i\in S}\min_{x_i}(\boldsymbol{a}_i^T\boldsymbol{y}-x_i)^2\right).$$

The inner minimization problems are not constrained, so the solution is trivially $x_i^\star = \boldsymbol{a}_i^T\boldsymbol{y}$, which means the problem reads:

$$\begin{aligned}&\min_{S\subset\{1,..,Q\} \mid |S|\leq K} \tfrac{1}{2}\sum_{i\notin S}(\boldsymbol{a}_i^T\boldsymbol{y})^2,\\ =&\min_{S\subset\{1,..,Q\} \mid |S|\leq K} \tfrac{1}{2}\left(\underbrace{\sum_{i=1}^{Q}(\boldsymbol{a}_i^T\boldsymbol{y})^2}_{\text{constant}} - \sum_{i\in S}(\boldsymbol{a}_i^T y)^2\right)\end{aligned}$$

As $(\boldsymbol{a}_i^T y)^2 \geq 0$, we want the set S to be as large as possible, meaning that S is such that $|S| = K$. Then, inside these K components, we want to select those which maximizes

the term $\sum_{i\in S}(\boldsymbol{a}_i^T y)^2$. This leads to selecting the components with the largest $|\boldsymbol{a}_i^T \boldsymbol{y}|$ value, meaning the solution of Problem (3.2) is:

$$S = \arg\max^K |\mathbf{A}^T \boldsymbol{y}|, \boldsymbol{x}_S^\star = \mathbf{A}_S^T \boldsymbol{y}, \forall i \notin S, x_i^\star = 0.$$

where $\arg\max^K$ returns a set containing the K largest components of a given vector.

A.2 Analytical solution of the scalar ℓ_1 problem

Many other sources giving this proof can be found, although it is generally done in the case of a normalized matrix $\mathbf{A}$ (i.e. for all columns $\boldsymbol{a}_i$ of the matrix, we have $\|\boldsymbol{a}_i\|_2 = 1$), whereas here we do not use a normalization assumption. The original problem states as:

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{e}_i - \boldsymbol{a}_i x_i\|_2^2 + \lambda |x_i|$$

where $\boldsymbol{e}_i$ is a constant for the problem. We express the first order optimality condition using the subdifferential:

$$\begin{aligned}
0 \in \{-\boldsymbol{a}_i^T(\boldsymbol{e}_i - \boldsymbol{a}_i x_i)\} + \begin{cases} \lambda & \text{if } x_i > 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ -\lambda & \text{if } x_i < 0 \end{cases} \\
\iff 0 \in \{-\boldsymbol{a}_i^T \boldsymbol{e}_i\} + \{\boldsymbol{a}_i^T \boldsymbol{a}_i x_i\} + \begin{cases} \lambda & \text{if } x_i > 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ -\lambda & \text{if } x_i < 0 \end{cases} \\
\iff -\{\boldsymbol{a}_i^T \boldsymbol{a}_i x_i\} \in \{-\boldsymbol{a}_i^T \boldsymbol{e}_i\} + \begin{cases} \lambda & \text{if } x_i > 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ -\lambda & \text{if } x_i < 0 \end{cases} \\
\iff \{\boldsymbol{a}_i^T \boldsymbol{a}_i x_i\} \in -\{-\boldsymbol{a}_i^T \boldsymbol{e}_i\} - \begin{cases} \lambda & \text{if } x_i > 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ -\lambda & \text{if } x_i < 0 \end{cases} \\
\iff \{\|\boldsymbol{a}_i\|_2^2 x_i\} \in \{\boldsymbol{a}_i^T \boldsymbol{e}_i\} + \begin{cases} -\lambda & \text{if } x_i > 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ \lambda & \text{if } x_i < 0 \end{cases} \\
\iff x_i \in \begin{cases} \frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i - \lambda) & \text{if } x_i > 0 \\ [\frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i - \lambda), \frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i + \lambda)] & \text{if } x_i = 0 \\ \frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i + \lambda) & \text{if } x_i < 0 \end{cases}.
\end{aligned}$$

So once we know the sign of the x_i, we know its value. Actually, we can use the previous relation to get the sign of x_i. Indeed, we have the following cases:

Case 1: $x_i > 0$

We have:

$$x_i = \frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i - \lambda) > 0 \iff \boldsymbol{a}_i^T \boldsymbol{e}_i > \lambda$$

Case 2: $x_i < 0$

We have:

$$x_i = \frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i + \lambda) < 0 \iff \boldsymbol{a}_i^T \boldsymbol{e}_i < -\lambda$$

Case 3: $x_i = 0$

We have:

$$\begin{aligned} & 0 \in [\frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i - \lambda), \frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i + \lambda)] \\ \iff & -\boldsymbol{a}_i^T \boldsymbol{e}_i \in [-\lambda, \lambda] \\ \iff & \boldsymbol{a}_i^T \boldsymbol{e}_i \in [-\lambda, \lambda] \end{aligned}$$

(the last line coming from $-[-\lambda, \lambda] = [-\lambda, -(-\lambda)] = [-\lambda, \lambda]$)

Summing up these three cases, we have:

— $x_i > 0 \implies \boldsymbol{a}_i^T \boldsymbol{e}_i > \lambda$,

— $x_i = 0 \implies \boldsymbol{a}_i^T \boldsymbol{e}_i \in [-\lambda, \lambda]$,

— $x_i < 0 \implies \boldsymbol{a}_i^T \boldsymbol{e}_i < -\lambda$.

These three cases are disjoint in the values of $\boldsymbol{a}_i^T \boldsymbol{e}_i$ and x_i, and they cover the full range of $\boldsymbol{a}_i^T \boldsymbol{e}_i$, so we can write (using contraposition of implications):

$$\begin{aligned} \boldsymbol{a}_i^T \boldsymbol{e}_i > \lambda & \iff \neg(\boldsymbol{a}_i^T \boldsymbol{e}_i \leq \lambda) \\ & \iff \neg(\boldsymbol{a}_i^T \boldsymbol{e}_i \in [-\lambda, \lambda]) \text{ and } \neg(\boldsymbol{a}_i^T \boldsymbol{e}_i < -\lambda) \\ & \implies \neg(x_i = 0) \text{ and } \neg(x_i < 0) \\ & \iff x_i > 0. \end{aligned}$$

Thus, we have $\boldsymbol{a}_i^T \boldsymbol{e}_i > \lambda \iff x_i > 0$. The same reasoning can be applied to all three cases to get:

— $x_i > 0 \iff \boldsymbol{a}_i^T \boldsymbol{e}_i > \lambda$,

— $x_i = 0 \iff \boldsymbol{a}_i^T \boldsymbol{e}_i \in [-\lambda, \lambda]$,

— $x_i < 0 \iff \boldsymbol{a}_i^T \boldsymbol{e}_i < -\lambda$.

So the final analytical solution reads:

— if $\boldsymbol{a}_i^T \boldsymbol{e}_i > \lambda$ then $x_i = \frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i - \lambda)$,

— if $\boldsymbol{a}_i^T \boldsymbol{e}_i \in [-\lambda, \lambda]$ then $x_i = 0$,

— if $\boldsymbol{a}_i^T \boldsymbol{e}_i < -\lambda$ then $\frac{1}{\|\boldsymbol{a}_i\|_2^2}(\boldsymbol{a}_i^T \boldsymbol{e}_i + \lambda)$,

which can be restated as $x_i = \frac{1}{\|\boldsymbol{a}_i\|_2^2}\mathrm{ST}_\lambda(\boldsymbol{a}_i^T\boldsymbol{e}_i)$, with ST_λ the soft-thresholding operator defined as:

$$\mathrm{ST}_\lambda(u) = \begin{cases} u-\lambda & u > \lambda \\ 0 & u \in [-\lambda, \lambda] \\ u+\lambda & u < -\lambda \end{cases}.$$

A.3 Proof of Proposition 8.5.1

The goal here is to show the proof of the expression allowing us to compute the S_1 steps in the IGD algorithm:

$$x_i \leftarrow \underset{v\in[-M,M]}{\arg\min}\ \tfrac{1}{2}\|\boldsymbol{r}_i - \boldsymbol{a}_i v\|_2^2.$$

Proof. The first-order optimality condition for problem in Algorithm 9 step 6 reads (using sub-differentials):

$$0 \in \{-\boldsymbol{a}_i^T(\boldsymbol{r}_i - \boldsymbol{a}_i x_i)\} + \begin{cases} [0, +\infty[& x_i = M \\ \{0\} & |x_i| < M \\ ]-\infty, 0] & x_i = -M \end{cases} \tag{A.1}$$

Let us separate the three different cases:

Case 1: $|x_i| < M$

Condition (A.1) then reads:

$$\begin{aligned} & 0 \in \{-\boldsymbol{a}_i^T(\boldsymbol{r}_i - \boldsymbol{a}_i x_i)\} \\ \iff & 0 = -\boldsymbol{a}_i^T(\boldsymbol{r}_i - \boldsymbol{a}_i x_i) \\ \iff & x_i = (\boldsymbol{a}_i^T\boldsymbol{a}_i)^{-1}\boldsymbol{a}_i^T\boldsymbol{r}_i \in\ ]-M, M[. \end{aligned}$$

Case 2: $x_i = M$

Condition (A.1) then reads:

$$\begin{aligned} & 0 \in \{-\boldsymbol{a}_i^T(\boldsymbol{r}_i - \boldsymbol{a}_i M)\} + [0, +\infty[\\ \iff & 0 \in \{-\boldsymbol{a}_i^T\boldsymbol{r}_i\} + \{\boldsymbol{a}_i^T\boldsymbol{a}_i M\} + [0, +\infty[\\ \iff & \boldsymbol{a}_i^T\boldsymbol{r}_i \in [\boldsymbol{a}_i^T\boldsymbol{a}_i M, +\infty[\\ \iff & (\boldsymbol{a}_i^T\boldsymbol{a}_i)^{-1}\boldsymbol{a}_i^T\boldsymbol{r}_i \geq M. \end{aligned}$$

(as a recall $(\boldsymbol{a}_i^T\boldsymbol{a}_i)^{-1} \geq 0$ holds)

Case 3: $x_i = -M$

Condition (A.1) then reads:

$$\begin{aligned} & 0 \in \{-\boldsymbol{a}_i^T(\boldsymbol{r}_i - \boldsymbol{a}_i(-M))\} + [-\infty, 0[\\ \iff & \boldsymbol{a}_i^T \boldsymbol{r}_i \in]-\infty, \boldsymbol{a}_i^T \boldsymbol{a}_i(-M)] \\ \iff & (\boldsymbol{a}_i^T \boldsymbol{a}_i)^{-1} \boldsymbol{a}_i^T \boldsymbol{r}_i \leq -M. \end{aligned}$$

Summing up, we have (as all cases are disjoint):

$$\begin{aligned} x_i = M & \iff (\boldsymbol{a}_i^T \boldsymbol{a}_i)^{-1} \boldsymbol{a}_i^T \boldsymbol{r}_i \geq M, \\ x_i \in]-M, M[& \iff (\boldsymbol{a}_i^T \boldsymbol{a}_i)^{-1} \boldsymbol{a}_i^T \boldsymbol{r}_i \in]-M, M[, \\ x_i = -M & \iff (\boldsymbol{a}_i^T \boldsymbol{a}_i)^{-1} \boldsymbol{a}_i^T \boldsymbol{r}_i \leq -M. \end{aligned}$$

This gives then the expression $x_i = \Pi_{[-M,M]}((\boldsymbol{a}_i^T \boldsymbol{a}_i)^{-1} \boldsymbol{a}_i^T \boldsymbol{r}_i)$. □

Appendix B

Properties of the ℓ_∞ norm

This chapter is dedicated to the subdifferential and the proximity operator of the ℓ_∞ norm, both of which are used in Part II of this thesis. The results of this appendix are not novel. However, as the author had some difficulty finding clear and detailed resources about these operators applied on the ℓ_∞ norm, this is given as a pedagogical resource.

B.1 Subdifferential of the ℓ_∞ norm

We are looking for the subdifferential (also called subgradient) of the ℓ_∞ norm of some $\boldsymbol{x} \in \mathbb{R}^d$. The basic expression reads:

$$\begin{aligned}\partial\|\boldsymbol{x}\|_\infty &= \{\boldsymbol{u} \in \mathbb{R}^d \mid \forall \boldsymbol{z} \in \mathbb{R}^d, \boldsymbol{u}^T(\boldsymbol{z}-\boldsymbol{x}) + \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{z}\|_\infty\} \\ &= \{\boldsymbol{u} \in \mathbb{R}^d \mid \forall \boldsymbol{z} \in \mathbb{R}^d, \boldsymbol{u}^T\boldsymbol{z} - \boldsymbol{u}^T\boldsymbol{x} + \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{z}\|_\infty\}.\end{aligned}$$

As a recall, in this expression $\boldsymbol{x}$ is given, and we wish to retrieve the set of $\boldsymbol{u}$ such that the relationship holds for *every* $\boldsymbol{z}$. The result is:

$$\begin{aligned}\partial\|\boldsymbol{0}\|_\infty &= \{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 \leq 1\} \\ \forall \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{x} \neq \boldsymbol{0} \implies \partial\|\boldsymbol{x}\|_\infty &= \{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty\}.\end{aligned}$$

The proof is separated into two cases, depending on the nullity of $\boldsymbol{x}$.

Case 1: $\boldsymbol{x} = \boldsymbol{0}$.

In this case, we have:

$$\partial\|\boldsymbol{0}\|_\infty = \{\boldsymbol{u} \in \mathbb{R}^d \mid \forall \boldsymbol{z} \in \mathbb{R}^d, \boldsymbol{u}^T\boldsymbol{z} \leq \|\boldsymbol{z}\|_\infty\}.$$

Letting $\boldsymbol{u}$ such that $\|\boldsymbol{u}\|_1 > 1$, we can construct some $\boldsymbol{z}$ such that the inequality does not hold. Indeed, with $\boldsymbol{z} = M \operatorname{sign}(\boldsymbol{u}), M \in \mathbb{R}_+^*$, we have:

$$\boldsymbol{u}^T\boldsymbol{z} = M\sum_{i=1}^{d} u_i \operatorname{sign}(u_i) = M\sum_{i=1}^{d}|u_i| = M\|\boldsymbol{u}\|_1 > M$$
$$\|\boldsymbol{z}\|_\infty = \max_{i\in\{1,..,d\}}(\operatorname{sign}(u_i)M) = M < \boldsymbol{u}^T\boldsymbol{z}$$

This means that we have $\partial\|\mathbf{0}\|_\infty \subseteq \{\boldsymbol{u} \mid \|\boldsymbol{u}\|_1 \leq 1\}$. We are now proving the equality of these two sets by showing that $\boldsymbol{u} \in \{\boldsymbol{u} \mid \|\boldsymbol{u}\|_1 \leq 1\} \implies \boldsymbol{u} \in \partial\|\mathbf{0}\|_\infty$. Let's denote by i_∞ one index of $\boldsymbol{z}$ attaining its ℓ_∞ norm: $|z_{i_\infty}| = \|\boldsymbol{z}\|_\infty$. Then:

$$\begin{aligned}\forall \boldsymbol{z} \in \mathbb{R}^d, \|\boldsymbol{u}\|_1 \leq 1 &\iff \sum_{i=1}^{d}|u_i| \leq 1 \iff (\sum_{i=1}^{d}|u_i|)|z_{i_\infty}| \leq \|\boldsymbol{z}\|_\infty \\ &\implies \sum_{i=1}^{d} u_i z_i \leq \sum_{i=1}^{d}|u_i||z_i| \leq \sum_{i=1}^{d}|u_i||z_{i_\infty}| \leq \|\boldsymbol{z}\|_\infty \\ &\implies \boldsymbol{u}^T\boldsymbol{z} \leq \|\boldsymbol{z}\|_\infty.\end{aligned} \tag{B.1}$$

This means that we have $\partial\|\mathbf{0}\|_\infty = \{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 \leq 1\}$.

Case 2: $\boldsymbol{x} \neq \mathbf{0}$

The result here is that:

$$\boldsymbol{x} \neq \mathbf{0} \implies \partial\|\boldsymbol{x}\|_\infty = \{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty\}.$$

The proof is laid out in these steps:

— $\partial\|\boldsymbol{x}\|_\infty \implies \|\boldsymbol{u}\|_1 \leq 1$,

— we can find a counter-example in this set, prompting the need to have additionally $\boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty$,

— $\boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty \implies \|\boldsymbol{u}\|_1 = 1$.

Let's begin with $\partial\|\boldsymbol{x}\|_\infty \implies \|\boldsymbol{u}\|_1 \leq 1$ (first step). This is similar to what has been done in case 1. The inequality which must be fulfilled for all $\boldsymbol{z}$ is $\boldsymbol{u}^T\boldsymbol{z} - \boldsymbol{u}^T\boldsymbol{x} + \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{z}\|_\infty$. Let's use a $\boldsymbol{u}$ such that $\|\boldsymbol{u}\|_1 > 1$. Then, with $\boldsymbol{z} = M\operatorname{sign}(\boldsymbol{u}), M \in \mathbb{R}_+^*$, we have $\boldsymbol{u}^T\boldsymbol{z} > \|\boldsymbol{z}\|_\infty$. More precisely, we have $\boldsymbol{u}^T\boldsymbol{z} - \|\boldsymbol{z}\|_\infty = (\|\boldsymbol{u}\|_1 - 1)M$. As we can choose M as large as we want, we can pick one such that $\boldsymbol{u}^T\boldsymbol{z} - \|\boldsymbol{z}\|_\infty > \boldsymbol{u}^T\boldsymbol{x} - \|\boldsymbol{x}\|_\infty \iff \boldsymbol{u}^T\boldsymbol{z} - \boldsymbol{u}^T\boldsymbol{x} + \|\boldsymbol{x}\|_\infty > \|\boldsymbol{z}\|_\infty$. As the inequality must hold for every $\boldsymbol{z}$, so we have $\partial\|\boldsymbol{x}\|_\infty \subseteq \{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 \leq 1\}$.

Then (second step), this set for $\boldsymbol{u}$ is actually too large. Indeed, reusing Equation (B.1) for $\boldsymbol{x}$, we have:

$$\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 \leq 1 \implies \boldsymbol{u}^T\boldsymbol{x} \leq \|\boldsymbol{x}\|_\infty \iff -\boldsymbol{u}^T\boldsymbol{x} + \|\boldsymbol{x}\|_\infty \geq 0.$$

However, if we take $\boldsymbol{z} = \mathbf{0}$, the definition of the subdifferential yields:

$$-\boldsymbol{u}^T\boldsymbol{x} + \|\boldsymbol{x}\|_\infty \leq 0,$$

which means that $-\boldsymbol{u}^T\boldsymbol{x} + \|\boldsymbol{x}\|_\infty = 0 \iff \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty$ must hold.

Now (third step), $\boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty$ implies that $\|\boldsymbol{u}\|_1 = 1$. Indeed, let's take again Equation B.1 with strict inequalities this time, and applied to $\boldsymbol{x}$:

$$\begin{aligned}\forall \boldsymbol{x} \in \mathbb{R}^d, \|\boldsymbol{u}\|_1 < 1 &\iff \sum_{i=1}^{d} |u_i| < 1 \iff (\sum_{i=1}^{d} |u_i|)|x_{i_\infty}| < \|\boldsymbol{x}\|_\infty \\ &\implies \sum_{i=1}^{d} u_i x_i \leq \sum_{i=1}^{d} |u_i||x_i| \leq \sum_{i=1}^{d} |u_i||x_{i_\infty}| < \|\boldsymbol{x}\|_\infty \\ &\implies \boldsymbol{u}^T\boldsymbol{x} < \|\boldsymbol{x}\|_\infty.\end{aligned} \tag{B.2}$$

Step 2 showed that $\boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty$, which implies by Equation (B.2) that $\|\boldsymbol{u}\|_1 = 1$. So we are now at $\partial\|\boldsymbol{x}\|_\infty \subseteq \{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty\} = \{\boldsymbol{u} \mid \|\boldsymbol{u}\|_1 = 1\} \cap \{\boldsymbol{u} \mid \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty\}$. Let's prove the reverse inclusion.

By definition of the set and by Equation (B.1), we have:

$$\begin{aligned}\forall \boldsymbol{z} \in \mathbb{R}^d, \forall \boldsymbol{u} \in \{\boldsymbol{u} \mid \|\boldsymbol{u}\|_1 = 1\} \cap \{\boldsymbol{u} \in \mathbb{R}^d \mid \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty\}, \\ \boldsymbol{u}^T\boldsymbol{z} - \boldsymbol{u}^T\boldsymbol{x} + \|\boldsymbol{x}\|_\infty = \boldsymbol{u}^T\boldsymbol{z} \leq \|\boldsymbol{z}\|_\infty \\ \iff \boldsymbol{u}^T(\boldsymbol{z} - \boldsymbol{x}) + \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{z}\|_\infty \\ \iff \boldsymbol{u} \in \partial\|\boldsymbol{x}\|_\infty.\end{aligned}$$

This proves $\{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty\} \subseteq \partial\|\boldsymbol{x}\|_\infty$, which completes the previous proof to get:

$$\forall \boldsymbol{x} \neq \mathbf{0}, \partial\|\boldsymbol{x}\|_\infty = \{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{x} = \|\boldsymbol{x}\|_\infty\}.$$

B.2 Proximal operator of the ℓ_∞ norm

This section is given as a reference for understanding how to compute the proximal operator of the ℓ_∞ norm. This is given as a pedagogical reference for the proof of its expression, as well as a (hopefully) clear explanation about the way to compute that expression efficiently. We are looking to get the expression of the following optimization problem:

$$\hat{\boldsymbol{x}} = \underset{\tau\|\cdot\|_\infty}{\text{prox}}(\boldsymbol{z}) = \arg\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{z} - \boldsymbol{x}\|_2^2 + \tau\|\boldsymbol{x}\|_\infty$$

The result is: if $\exists s \in \mathbb{R}_*^+ \mid \sum_{i=1}^Q \max(0, |z_i| - s) = \tau$, then $\hat{\boldsymbol{x}} = \text{sign}(\boldsymbol{z}) \odot \min(|\boldsymbol{z}|, s)$, otherwise $\hat{\boldsymbol{x}} = 0$. For proving this, let's define two sets of indices: S_∞ contains the indices of variables which are concerned by the term $\tau\|\boldsymbol{x}\|_\infty$. In other words:

$$S_\infty = \{i, |x_i| = \|\boldsymbol{x}\|_\infty\}.$$

Its counterpart, $S_<$, will hold the remaining variables:

$$S_< = \{i, |x_i| < \|\boldsymbol{x}\|_\infty\}.$$

With $\boldsymbol{x} \in \mathbb{R}^Q$, we have $\{1, .., Q\} = S_\infty \cup S_<$. Decomposing the proximal operator using those two sets, we have:

$$\underset{\tau\|.\|_\infty}{\text{prox}}(\boldsymbol{z}) = \underset{\boldsymbol{x}}{\arg\min}\, \tfrac{1}{2}\|\boldsymbol{z}_{S_<} - \boldsymbol{x}_{S_<}\|_2^2 + \tfrac{1}{2}\|\boldsymbol{z}_{S_\infty} - \boldsymbol{x}_{S_\infty}\|_2^2 + \tau\|\boldsymbol{x}_{S_\infty}\|_\infty.$$

The proximal operator is completely separable for the variables in $S_<$, and the quadratic term naturally lead to $\boldsymbol{x}_{S_<} = \boldsymbol{z}_{S_<}$. Let's now focus on the variables in S_∞. If $\|\boldsymbol{x}_{S_\infty}\|_\infty = 0$, then $\boldsymbol{x} = 0$. Otherwise, we have $\forall (i,j) \in S_\infty, |x_i| = |x_j| \neq 0$. Note that the signs may be different. Let's write the sub-differential of this very specific ℓ_∞ term, using $\boldsymbol{t} = \boldsymbol{x}_{S_\infty}$:

$$\partial\|\boldsymbol{t}\|_\infty = \{\boldsymbol{u}, \forall \boldsymbol{y} \in \mathbb{R}^{|S_\infty|}, \boldsymbol{u}^T(\boldsymbol{y} - \boldsymbol{t}) + \|\boldsymbol{t}\|_\infty \leq \|\boldsymbol{y}\|_\infty\}.$$

Using Appendix B.1, this subdifferential is:

$$\partial\|\boldsymbol{t}\|_\infty = \{\boldsymbol{u} \in \mathbb{R}^{S_\infty} \mid \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{t} = \|\boldsymbol{t}\|_\infty\}.$$

Looking back at the proximal operator reduced to S_∞, we have:

$$\begin{aligned}
\boldsymbol{x}_{S_\infty} = \underset{\tau\|.\|_\infty}{\text{prox}}(\boldsymbol{z}_{S_\infty}) &= \underset{\boldsymbol{t}}{\arg\min}\, \underbrace{\tfrac{1}{2}\|\boldsymbol{z}_{S_\infty} - \boldsymbol{t}\|_2^2 + \tau\|\boldsymbol{t}\|_\infty}_{p(\boldsymbol{t})} \\
&\iff 0 \in \partial p(\boldsymbol{x}_{S_\infty}) \iff 0 \in \{-(\boldsymbol{z}_{S_\infty} - \boldsymbol{x}_{S_\infty})\} + \tau\{\boldsymbol{u} \in \mathbb{R}^{|S_\infty|} \mid \|\boldsymbol{u}\|_1 = 1, \boldsymbol{u}^T\boldsymbol{x}_{S_\infty} = \|\boldsymbol{x}_{S_\infty}\|_\infty\} \\
&\iff \exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \boldsymbol{z}_{S_\infty} - \tau\boldsymbol{u} = \boldsymbol{x}_{S_\infty} \\
&\iff \exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \forall i \in \{1, .., |S_\infty|\}\ (z_{S_\infty})_i - \tau u_i = (\boldsymbol{x}_{S_\infty})_i = \pm\|\boldsymbol{x}\|_\infty \neq 0.
\end{aligned}$$

Let's prove two sign equalities that will come handy after:

Proposition B.2.1. *Given the previously defined $\boldsymbol{x}_{S_\infty}$, we have:*

$$\textit{if } \boldsymbol{x} \neq \boldsymbol{0}; \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \implies \forall i \in \{1, .., |S_\infty|\}, \text{sign}(u_i) = \text{sign}[(x_{S_\infty})_i]. \tag{B.3}$$

Proof.

$$\begin{aligned}
&\boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \implies \boldsymbol{u}^T\boldsymbol{x}_{S_\infty} = \|\boldsymbol{x}_{S_\infty}\|_\infty > 0 \text{ and } \|\boldsymbol{u}\|_1 = 1 \\
&\|\boldsymbol{u}\|_1 = 1 \iff \sum_{i=1}^{|S_\infty|} |u_i| = 1 \\
&\iff \boldsymbol{u}^T\boldsymbol{x}_{S_\infty} = \sum_{i=1}^{|S_\infty|} u_i(x_{S_\infty})_i \leq \sum_{i=1}^{|S_\infty|} |u_i||x_{S_\infty}|_i = \|\boldsymbol{x}_{S_\infty}\|_\infty > 0 \\
&\qquad \text{as } \boldsymbol{u}^T\boldsymbol{x}_{S_\infty} = \|\boldsymbol{x}_{S_\infty}\|_\infty \text{ also holds,} \\
&\qquad \text{we must have equality between both sums:} \\
&\implies \forall i \in \{1,..,|S_\infty|\}, u_i(x_{S_\infty})_i = |u_i||x_{S_\infty}|_i \iff \operatorname{sign}(u_i) = \operatorname{sign}[(x_{S_\infty})_i].
\end{aligned}$$

□

Proposition B.2.2. *Given the previously defined $\boldsymbol{z}_{S_\infty}$ and $\boldsymbol{x}_{S_\infty}$, we have:*

$$\text{if } \boldsymbol{x} \neq 0; \forall i \in \{1,..,|S_\infty|\}, \operatorname{sign}\left[(z_\infty)_i\right] = \operatorname{sign}\left[(x_\infty)_i\right]. \tag{B.4}$$

Proof. The proximal operator gives rise to:

$$\begin{aligned}
&\exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \forall i \in \{1,..,|S_\infty|\}\ (z_{S_\infty})_i - \tau u_i = (x_{S_\infty})_i = \pm\|\boldsymbol{x}\|_\infty \neq 0 \\
\iff\ &\exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \forall i \in \{1,..,|S_\infty|\}\ (z_{S_\infty})_i \operatorname{sign}[(x_{S_\infty})_i] - \tau u_i \operatorname{sign}[(x_{S_\infty})_i] = \|\boldsymbol{x}\|_\infty > 0 \\
&\qquad \text{using Proposition B.2.1:} \\
\iff\ &\exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \forall i \in \{1,..,|S_\infty|\}\ (z_{S_\infty})_i \operatorname{sign}[(x_{S_\infty})_i] = \tau|u_i| + \|\boldsymbol{x}\|_\infty > 0 \\
&\qquad\qquad \implies \forall i \in \{1,..,|S_\infty|\} \operatorname{sign}[(z_{S_\infty})_i] = \operatorname{sign}[(x_{S_\infty})_i].
\end{aligned}$$

□

Now, following the proximal operator implications:

$$\begin{aligned}
&\exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \forall i \in \{1,..,|S_\infty|\}\ (z_{S_\infty})_i - \tau u_i = (x_{S_\infty})_i = \pm\|\boldsymbol{x}\|_\infty \neq 0 \\
&\iff \exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \forall i \in \{1,..,|S_\infty|\}\ (z_{S_\infty})_i \operatorname{sign}[(x_{S_\infty})_i] - \tau u_i \operatorname{sign}[(x_{S_\infty})_i] = \|\boldsymbol{x}\|_\infty \\
&\text{using proposition B.2.1 and B.2.2:} \\
&\iff \exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \forall i \in \{1,..,|S_\infty|\}\ |(z_{S_\infty})_i| - \tau|u_i| = \|\boldsymbol{x}\|_\infty \\
&\implies \exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \sum_{i\in\{1,..,|S_\infty|\}} (|(z_{S_\infty})_i| - \|\boldsymbol{x}\|_\infty) = \tau \sum_{i\in\{1,..,|S_\infty|\}} |u_i| = \tau \\
&\implies \exists \boldsymbol{u} \in \partial\|\boldsymbol{x}_{S_\infty}\|_\infty \mid \sum_{i\in\{1,..,|S_\infty|\}} |(z_{S_\infty})_i| > \tau.
\end{aligned}$$

Let's recall that all this reasoning is in the case $\|\boldsymbol{x}\|_\infty > 0$. Now, let's try to transform these implications into rules to get values from $\boldsymbol{z}$ to $\boldsymbol{x}$, keeping the oracle separation

S_∞ and $S_<$ unknown in practice as these sets depend on the solution of the proximal problem which is unknown. If we have $\sum_{i\in\{1,..,|S_\infty|\}} |(z_{S_\infty})_i| \leq \tau$, then this means that $\|\boldsymbol{x}\|_\infty = 0$, which means that $\boldsymbol{x} = 0$. Otherwise, we have some $s \in \mathbb{R}_*^+$ such that $\sum_{i\in\{1,..,|S_\infty|\}}(|(z_{S_\infty})_i| - s) = \tau$. This s is our value for $\|\boldsymbol{x}\|_\infty$, which means we will do an ℓ_∞ ball projection with this radius. In other words, we have $\boldsymbol{x}_{S_\infty} = s \odot \text{sign}(\boldsymbol{z}_{S_\infty})$, where $\odot$ is the term-by-term product (Hadamar product). In that case, we should also have $\boldsymbol{x}_{S_<} = \boldsymbol{z}_{S_<}$.

This separation of the solution into two sets, S_∞ and $S_<$, is unknown in practice, so we must have a way to infer them from $\boldsymbol{z}$ if $s > 0$, **before the computation of** s which requires the knowledge of S_∞. It turns out that when $s > 0$ exists, we have:

$$\forall i \in S_<, |x_i| < s \iff |z_i| < s \iff |z_i| - s < 0.$$

This means that we can compute the previous sum on the whole $\boldsymbol{z}$ just by cancelling the negative values. This leads to the following final expression: If $\exists s \in \mathbb{R}_*^+ \mid \sum_{i=1}^Q \max(0, |z_i| - s) = \tau$, then $\boldsymbol{x} = \text{sign}(\boldsymbol{z}) \odot \min(|\boldsymbol{z}|, s)$, otherwise $\boldsymbol{x} = 0$.

How to compute that Finding the s value inside the summed max terms seems rather intricate at first glance. However, with some manipulation, it is tractable. Let's denote by $\boldsymbol{\gamma}$ the magnitude sorted version of $\boldsymbol{z}$:

$$\begin{aligned}\forall i \in \{1,..,Q\}, \exists! j \in \{1,..,Q\}, z_i = \gamma_j \\ \forall i \in \{1,..,Q-1\}, |\gamma_i| \geq |\gamma_{i+1}|.\end{aligned}$$

Then, we have:

$$\begin{aligned}&\sum_{i=1}^{Q} \max(0, |z_i| - s) = \tau \\ \iff &\sum_{i=1}^{Q} \max(0, |\gamma_i| - s) = \tau \\ \iff &\sum_{i=1}^{K} (|\gamma_i| - s) = \tau.\end{aligned}$$

with K separating the non-zero and zero terms: $\forall i \leq K, \gamma_i > s, \forall i > K, \gamma_i \leq s$. Rearranging the equation, we have:

$$\begin{aligned}
&\sum_{i=1}^{K}(|\gamma_i| - s) = \tau \\
&\iff (\sum_{i=1}^{K}|\gamma_i|) - sK = \tau \\
&\iff \frac{(\sum_{i=1}^{K}|\gamma_i|) - \tau}{K} = s.
\end{aligned}$$

Let's denote by $s(K)$ this quantity: $s(K) = \frac{(\sum_{i=1}^{K}|\gamma_i|)-\tau}{K}$. These $s(K)$ would be the correct s if there was only the K largest coordinates of $\boldsymbol{z}$ in the sum. In other words, the $Q-K$ remaining coordinates should be cancelled by the value of $s(K)$. This means we look for the value of K such that:

$$\forall j > K, |\gamma_K| > s(K) \geq |\gamma_j| \iff |\gamma_K| > s(K) \geq |\gamma_{K+1}|.$$

Looking back at the definition of $s(K)$, we have the following equality:

$$s(K) = \tfrac{K+1}{K}s(K+1) - \tfrac{|\gamma_{K+1}|}{K}.$$

This means that:

$$\begin{aligned}
s(K) &\geq |\gamma_{K+1}| \\
\iff \tfrac{K+1}{K}s(K+1) - \tfrac{|\gamma_{K+1}|}{K} &\geq |\gamma_{K+1}| \\
\iff \tfrac{K+1}{K}s(K+1) &\geq |\gamma_{K+1}|(1+\tfrac{1}{K}) \\
\iff s(K+1) &\geq |\gamma_{K+1}|.
\end{aligned}$$

Consequently, our condition rewrites:

$$|\gamma_K| > s(K) \geq |\gamma_{K+1}| \iff |\gamma_K| > s(K) \text{ and } |\gamma_{K+1}| \leq s(K+1).$$

Therefore, we are looking for the last γ_K such that $|\gamma_K| > s(K)$ holds. In terms of practical implementation, the different $s(K)$ can be computed in one row, using cumulative summing functions (such as `cumsum` in Matlab) and element-wise subtraction and division. After that, taking the max of a vector condition such as $|\boldsymbol{\gamma}| > \boldsymbol{s}$ has a complexity of $O(n)$. To get the $\boldsymbol{\gamma}$ vector, we must use a sorting operator, which generally has a complexity of $O(n\log(n))$, therefore this is the driving term for the algorithmic complexity of the overall procedure. A semi-formal writing of this procedure is given in Algorithm 12.

Algorithm 12 Proximal operator of the ℓ_∞ function

procedure PROX-$\ell_\infty(\boldsymbol{z}, \tau)$
    $\boldsymbol{\gamma} \leftarrow$ sort($\boldsymbol{z}$, 'descend')
    $\boldsymbol{c} \leftarrow$ (cumsum($|\boldsymbol{\gamma}|$) - τ) ./ cumsum(ones(Q, 1)) ▷ Our candidates for s, using element-wise division and subtraction.
    good $\leftarrow (|\boldsymbol{\gamma}| > \boldsymbol{c})$ ▷ This is to be read as a vector boolean expression.
    $K \leftarrow \arg\max(\textbf{good})$ ▷ Get the latest $|\gamma_i|$ matching the condition.
    $s \leftarrow c_K$
    $\boldsymbol{x} \leftarrow \text{sign}(\boldsymbol{z}) \odot \min(|\boldsymbol{z}|, s)$ ▷ Note that we use $\boldsymbol{z}$ and not $\boldsymbol{\gamma}$ to keep the original order.
    return $\boldsymbol{x}$
end procedure

Appendix C

Remaining proofs of the thesis contributions

This appendix is dedicated to proofs omitted in the main text. They are part of the contributions of this thesis.

C.1 Proof of Proposition 9.2.2

Proof. We will just show the equivalence with Problem ($\mathcal{D}_{2+11}^{\mathbf{N}\text{weight}}$):

$$\begin{aligned}\max_{\boldsymbol{w}\in\mathbb{R}^N} \; & -\tfrac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2) + \mu|G_1| \\ & - M\left(\sum_{i\in\bar{S}}\left[|\boldsymbol{a}_i^T\boldsymbol{w}| - \alpha_i\tfrac{\mu}{M}\right]_+ + \|\mathbf{A}_{S_1}^T\boldsymbol{w}\|_1\right)\end{aligned} \qquad (\mathcal{D}_{2+11}^{\mathbf{N}\text{weight}})$$

Injecting the definition of $\mathcal{A}$, namely:

— $\forall i \in \bar{S}, \mathcal{A}_i = \boldsymbol{a}_i/\alpha_i$,

— $\forall i \in S_1 \cup S_0, \mathcal{A}_i = \boldsymbol{a}_i$,

into the dual definition, we get:

$$\begin{aligned}\max_{\boldsymbol{w}\in\mathbb{R}^N} \; & -\tfrac{1}{2}(\|\boldsymbol{w}+\boldsymbol{y}\|_2^2 - \|\boldsymbol{y}\|_2^2) + \mu|G_1| \\ & - \left(\sum_{i\in\bar{S}} M\alpha_i\left[\tfrac{1}{\alpha_i}(|\boldsymbol{a}_i^T\boldsymbol{w}| - \alpha_i\tfrac{\mu}{M})\right]_+ + M\|\mathbf{A}_{S_1}^T\boldsymbol{w}\|_1\right)\end{aligned}$$

We have:

$$\begin{aligned}\sum_{i\in\bar{S}} M\alpha_i\left[\tfrac{1}{\alpha_i}(|\boldsymbol{a}_i^T\boldsymbol{w}| - \alpha_i\tfrac{\mu}{M})\right]_+ &= \sum_{i\in\bar{S}} M\alpha_i\tfrac{1}{\alpha_i}\left[|\boldsymbol{a}_i^T\boldsymbol{w}| - \alpha_i\tfrac{\mu}{M}\right]_+ \\ &= M\sum_{i\in\bar{S}}\left[|\boldsymbol{a}_i^T\boldsymbol{w}| - \alpha_i\tfrac{\mu}{M}\right]_+ .\end{aligned}$$

Injecting this last form into the dual objective, we get the multiple-weight variant.

□

C.2 Proof of Proposition 9.2.4

Proof. The tests can be proved in the same way than for Theorem (9.2.3) (on page 125), starting from $-\mathcal{A}^T\boldsymbol{w}^\star \in \partial g(\boldsymbol{z}^\star)$, separating the different cases, and then plugging Inequality (5.19):

$$|\boldsymbol{a}_i^T\boldsymbol{w}^\star| \leq |\boldsymbol{a}_i^T\boldsymbol{w}| + \|\boldsymbol{a}_i\|_2\|\boldsymbol{w}^\star - \boldsymbol{w}\|_2$$

and Inequality (5.20):

$$|\boldsymbol{a}_i^T\boldsymbol{w}^\star| \geq |\boldsymbol{a}_i^T\boldsymbol{w}| - \|\boldsymbol{a}_i\|_2\|\boldsymbol{w}^\star - \boldsymbol{w}\|_2.$$

In this case the different columns $\mathcal{A}_i, i \in \bar{S}$ are not unit norm, so the different $\|\mathcal{A}_i\|_2$ must be kept.

Here, we will instead directly show that tests (9.5) (on page 125) are equivalent to tests (9.2) (on page 124). Injecting the definitions of $\mathcal{A}, \boldsymbol{z}$, namely:

— $\forall i \in \bar{S}, \mathcal{A}_i = \boldsymbol{a}_i/\alpha_i$,

— $\forall i \in S_1 \cup S_0, \mathcal{A}_i = \boldsymbol{a}_i$,

— $\forall i \in \bar{S}, z_i = \alpha_i x_i$,

— $\forall i \in S_1 \cup S_0, z_i = x_i$,

into tests (9.5), we get:

$$\forall i \in \bar{S}, \text{if } \tfrac{1}{\alpha_i}|\boldsymbol{a}_i^T\boldsymbol{w}| > \tfrac{\mu}{M} + \tfrac{1}{\alpha_i}\underbrace{\|\boldsymbol{a}_i\|_2}_{=1}\sqrt{2G(\boldsymbol{x},\boldsymbol{w})}, \text{ then } \alpha_i x_i^\star = -\alpha_i M \operatorname{sign}(\underbrace{\tfrac{1}{\alpha_i}}_{>0}\boldsymbol{a}_i^T\boldsymbol{w});$$

$$\forall i \in \bar{S}, \text{if } \tfrac{1}{\alpha_i}|\boldsymbol{a}_i^T\boldsymbol{w}| < \tfrac{\mu}{M} - \tfrac{1}{\alpha_i}\underbrace{\|\boldsymbol{a}_i\|_2}_{=1}\sqrt{2G(\boldsymbol{x},\boldsymbol{w})}, \text{ then } \alpha_i x_i^\star = 0;$$

$$\forall i \in S_1, \text{if } |\boldsymbol{a}_i^T\boldsymbol{w}| > \underbrace{\|\boldsymbol{a}_i\|_2}_{=1}\sqrt{2G(\boldsymbol{x},\boldsymbol{w})}, \text{ then } x_i^\star = -M_i \operatorname{sign}(\boldsymbol{a}_i^T\boldsymbol{w}).$$

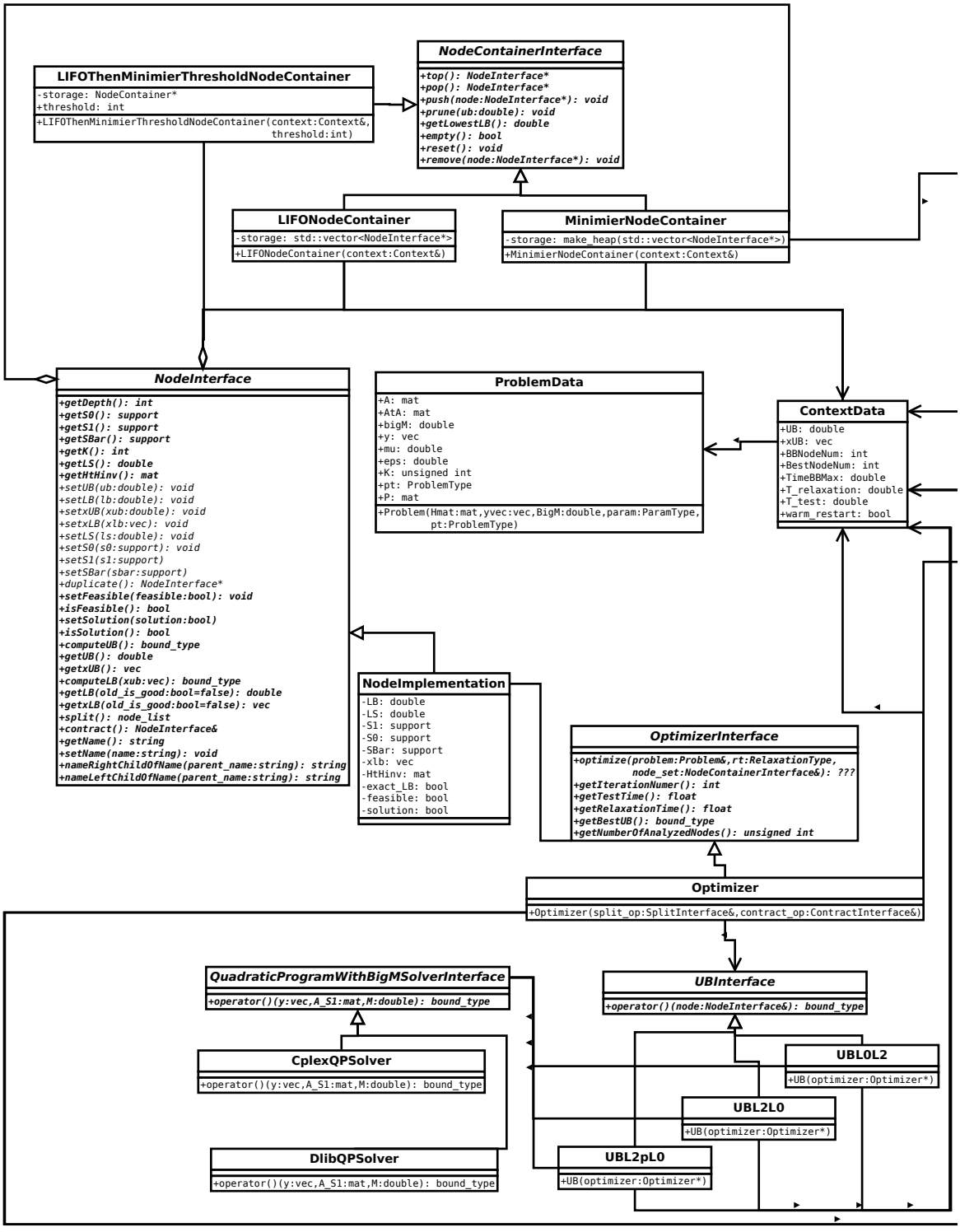
The last test is already exactly the same as (9.2c), and the two previous ones are just the first two tests of (9.2) where we multiplied by $\frac{1}{\alpha_i}$ (which is strictly positive) on both sides of the inequalities, an multiplied by α_i (also strictly positive) on the tests consequences.

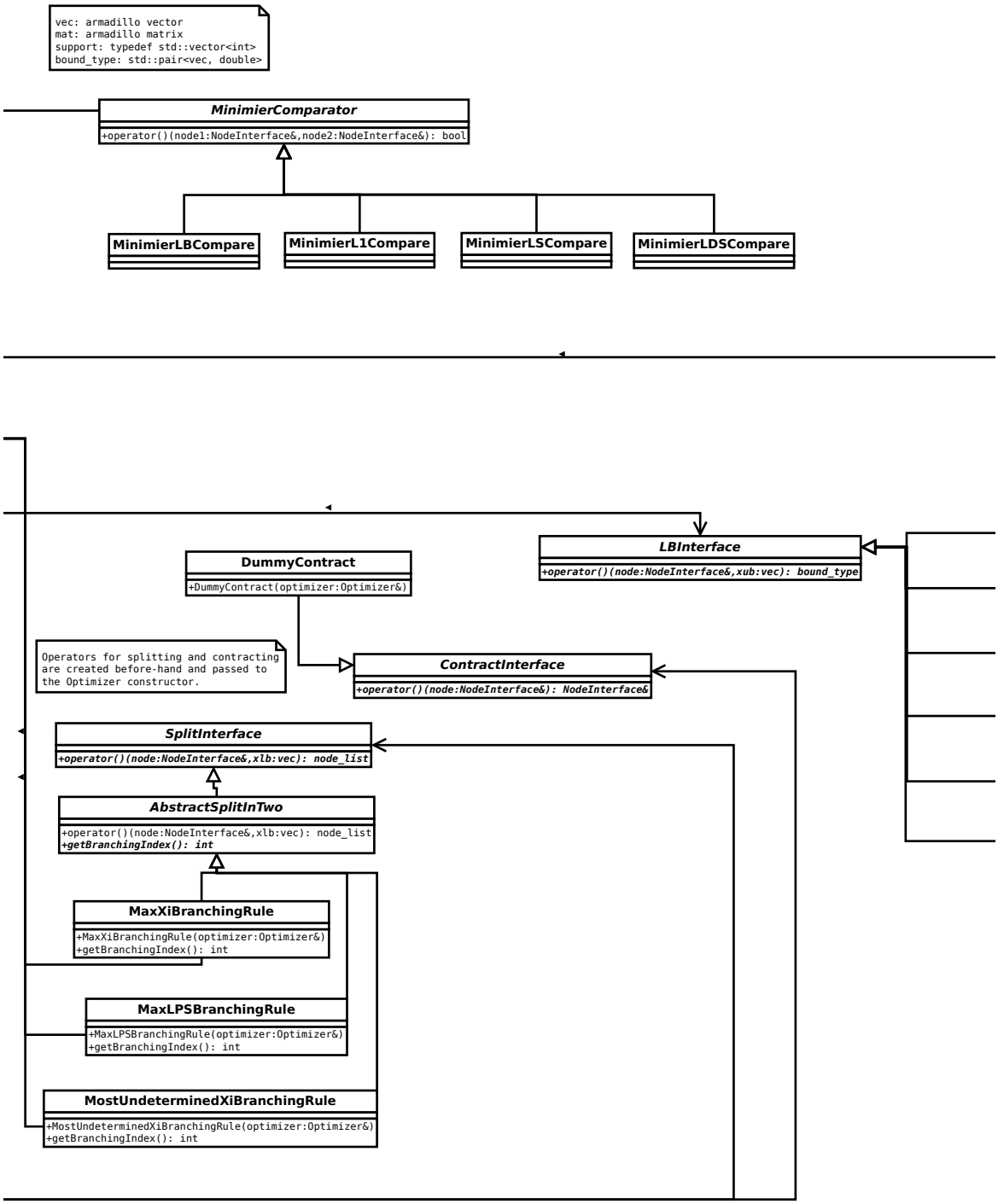
□

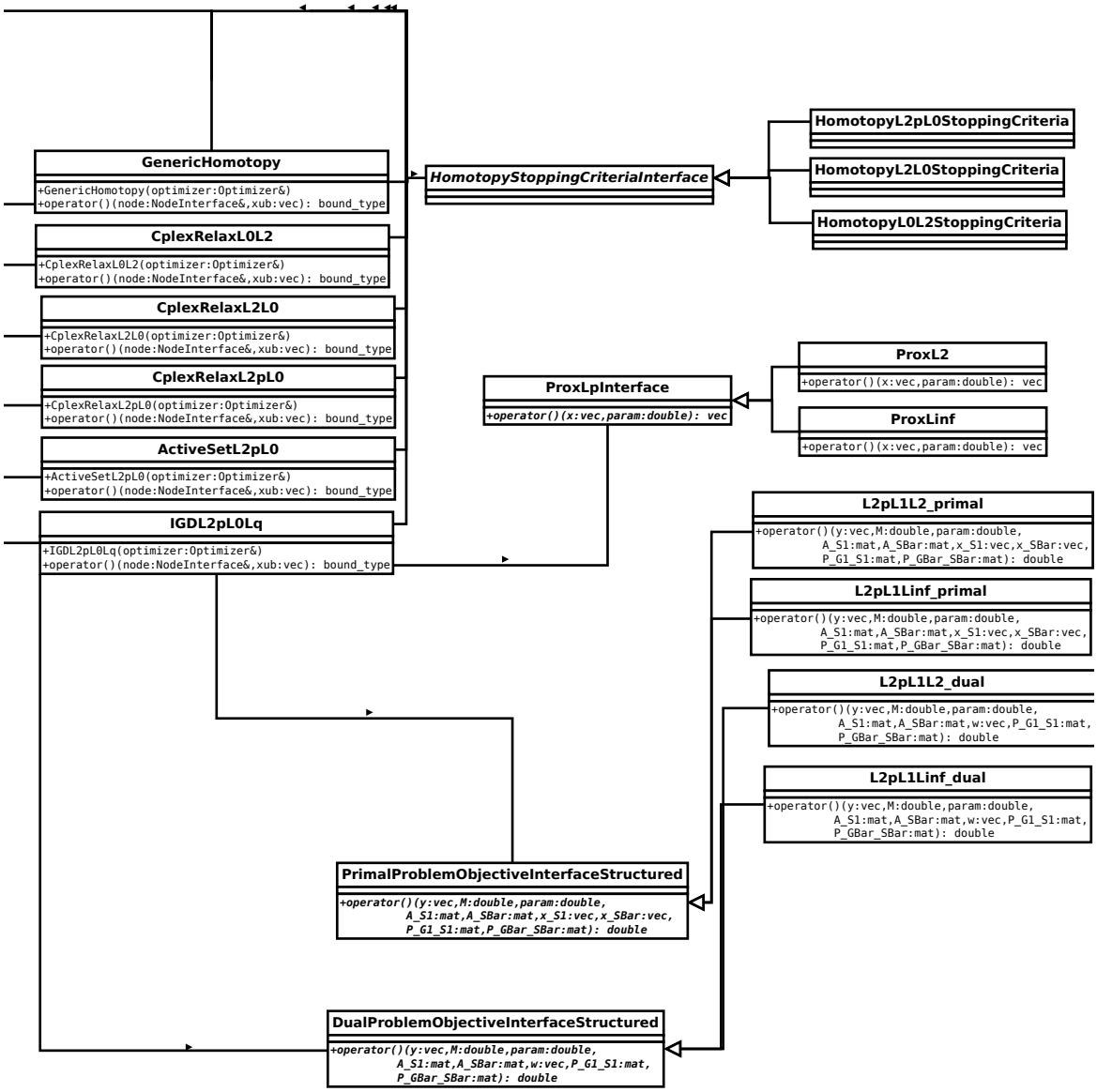
Appendix D

Overview of all the classes of the code

This covers the same software architecture as in Figure 11.2 (on page 150), displayed on 3 pages instead of a single figure.


LIFOThenMinimierThresholdNodeContainer
-storage: NodeContainer*
+threshold: int
+LIFOThenMinimierThresholdNodeContainer(context:Context&, threshold:int)
NodeContainerInterface
+top(): NodeInterface*
+pop(): NodeInterface*
+push(node:NodeInterface*): void
+prune(ub:double): void
+getLowestLB(): double
+empty(): bool
+reset(): void
+remove(node:NodeInterface*): void
LIFONodeContainer
-storage: std::vector<NodeInterface*>
+LIFONodeContainer(context:Context&)
MinimierNodeContainer
-storage: make_heap(std::vector<NodeInterface*>)
+MinimierNodeContainer(context:Context&)
NodeInterface
+getDepth(): int
+getS0(): support
+getS1(): support
+getSBar(): support
+getK(): int
+getLS(): double
+getHtHinv(): mat
+setUB(ub:double): void
+setLB(lb:double): void
+setxUB(xub:double): void
+setxLB(xlb:vec): void
+setLS(ls:double): void
+setS0(s0:support): void
+setS1(s1:support)
+setSBar(sbar:support)
+duplicate(): NodeInterface*
+setFeasible(feasible:bool): void
+isFeasible(): bool
+setSolution(solution:bool)
+isSolution(): bool
+computeUB(): bound_type
+getUB(): double
+getxUB(): vec
+computeLB(xub:vec): bound_type
+getLB(old_is_good:bool=false): double
+getxLB(old_is_good:bool=false): vec
+split(): node_list
+contract(): NodeInterface&
+getName(): string
+setName(name:string): void
+nameRightChildOfName(parent_name:string): string
+nameLeftChildOfName(parent_name:string): string
ProblemData
+A: mat
+AtA: mat
+bigM: double
+y: vec
+mu: double
+eps: double
+K: unsigned int
+pt: ProblemType
+P: mat
+Problem(Hmat:mat,yvec:vec,BigM:double,param:ParamType, pt:ProblemType)
ContextData
+UB: double
+xUB: vec
+BBNodeNum: int
+BestNodeNum: int
+TimeBBMax: double
+T_relaxation: double
+T_test: double
+warm_restart: bool
NodeImplementation
-LB: double
-LS: double
-S1: support
-S0: support
-SBar: support
-xlb: vec
-HtHinv: mat
-exact_LB: bool
-feasible: bool
-solution: bool
OptimizerInterface
+optimize(problem:Problem&,rt:RelaxationType, node_set:NodeContainerInterface&): ???
+getIterationNumer(): int
+getTestTime(): float
+getRelaxationTime(): float
+getBestUB(): bound_type
+getNumberOfAnalyzedNodes(): unsigned int
Optimizer
+Optimizer(split_op:SplitInterface&,contract_op:ContractInterface&)
QuadraticProgramWithBigMSolverInterface
+operator()(y:vec,A_S1:mat,M:double): bound_type
UBInterface
+operator()(node:NodeInterface&): bound_type
CplexQPSolver
+operator()(y:vec,A_S1:mat,M:double): bound_type
UBL0L2
+UB(optimizer:Optimizer*)
UBL2L0
+UB(optimizer:Optimizer*)
DlibQPSolver
+operator()(y:vec,A_S1:mat,M:double): bound_type
UBL2pL0
+UB(optimizer:Optimizer*)


vec: armadillo vector
mat: armadillo matrix
support: typedef std::vector<int>
bound_type: std::pair<vec, double>
MinimierComparator
+operator()(node1:NodeInterface&,node2:NodeInterface&): bool
MinimierLBCompare
MinimierL1Compare
MinimierLSCompare
MinimierLDSCompare
LBInterface
+operator()(node:NodeInterface&,xub:vec): bound_type
DummyContract
+DummyContract(optimizer:Optimizer&)
Operators for splitting and contracting are created before-hand and passed to the Optimizer constructor.
ContractInterface
+operator()(node:NodeInterface&): NodeInterface&
SplitInterface
+operator()(node:NodeInterface&,xlb:vec): node_list
AbstractSplitInTwo
+operator()(node:NodeInterface&,xlb:vec): node_list
+getBranchingIndex(): int
MaxXiBranchingRule
+MaxXiBranchingRule(optimizer:Optimizer&)
+getBranchingIndex(): int
MaxLPSBranchingRule
+MaxLPSBranchingRule(optimizer:Optimizer&)
+getBranchingIndex(): int
MostUndeterminedXiBranchingRule
+MostUndeterminedXiBranchingRule(optimizer:Optimizer&)
+getBranchingIndex(): int


GenericHomotopy
+GenericHomotopy(optimizer:Optimizer&)
+operator()(node:NodeInterface&,xub:vec): bound_type
HomotopyStoppingCriteriaInterface
HomotopyL2pL0StoppingCriteria
HomotopyL2L0StoppingCriteria
HomotopyL0L2StoppingCriteria
CplexRelaxL0L2
+CplexRelaxL0L2(optimizer:Optimizer&)
+operator()(node:NodeInterface&,xub:vec): bound_type
CplexRelaxL2L0
+CplexRelaxL2L0(optimizer:Optimizer&)
+operator()(node:NodeInterface&,xub:vec): bound_type
CplexRelaxL2pL0
+CplexRelaxL2pL0(optimizer:Optimizer&)
+operator()(node:NodeInterface&,xub:vec): bound_type
ActiveSetL2pL0
+ActiveSetL2pL0(optimizer:Optimizer&)
+operator()(node:NodeInterface&,xub:vec): bound_type
IGDL2pL0Lq
+IGDL2pL0Lq(optimizer:Optimizer&)
+operator()(node:NodeInterface&,xub:vec): bound_type
ProxLpInterface
+operator()(x:vec,param:double): vec
ProxL2
+operator()(x:vec,param:double): vec
ProxLinf
+operator()(x:vec,param:double): vec
L2pL1L2_primal
+operator()(y:vec,M:double,param:double,
A_S1:mat,A_SBar:mat,x_S1:vec,x_SBar:vec,
P_G1_S1:mat,P_GBar_SBar:mat): double
L2pL1Linf_primal
+operator()(y:vec,M:double,param:double,
A_S1:mat,A_SBar:mat,x_S1:vec,x_SBar:vec,
P_G1_S1:mat,P_GBar_SBar:mat): double
L2pL1L2_dual
+operator()(y:vec,M:double,param:double,
A_S1:mat,A_SBar:mat,w:vec,P_G1_S1:mat,
P_GBar_SBar:mat): double
L2pL1Linf_dual
+operator()(y:vec,M:double,param:double,
A_S1:mat,A_SBar:mat,w:vec,P_G1_S1:mat,
P_GBar_SBar:mat): double
PrimalProblemObjectiveInterfaceStructured
+operator()(y:vec,M:double,param:double,
A_S1:mat,A_SBar:mat,x_S1:vec,x_SBar:vec,
P_G1_S1:mat,P_GBar_SBar:mat): double
DualProblemObjectiveInterfaceStructured
+operator()(y:vec,M:double,param:double,
A_S1:mat,A_SBar:mat,w:vec,P_G1_S1:mat,
P_GBar_SBar:mat): double

Bibliography

Abry, P., N. Pustelnik, S. G. Roux, P. Jensen, P. Flandrin, R. Gribonval, C.-G. Lucas, É. Guichard, P. Borgnat, and N. B. Garnier (2020), « Spatial and temporal regularization to estimate COVID-19 reproduction number R(t): Promoting piecewise smoothness via convex optimization », *in*: *PLoS ONE*.

Achterberg, T., T. Koch, and A. Martin (2005), « Branching rules revisited », *in*: *Operations Research Letters*.

Alliney, S. and S. A. Ruzinsky (1994), « An algorithm for the minimization of mixed ℓ_1 and ℓ_2 norms with application to Bayesian estimation », *in*: *IEEE Transactions on Signal Processing*.

Atamtürk, A. and A. Gómez (2020), « Safe screening rules for L0-regression from Perspective Relaxations », *in*: *International Conference on Machine Learning*.

Bach, F., R. Jenatton, J. Mairal, and G. Obozinski (2011), *Optimization with Sparsity-Inducing Penalties*, now publishers Inc.

— (2012), « Structured sparsity through convex optimization », *in*: *Statistical Science*.

Baraniuk, R. G., V. Cevher, M. F. Duarte, and C. Hegde (2010), « Model-Based Compressive Sensing », *in*: *IEEE Trans. Inf. Theor.*

Bauschke, H. and P. Combettes (2011), *Convex Analysis and Monotone Operator Theory in Hilbert Space*, Springer.

Bayram, İ. (2018), « Sparsity Within and Across Overlapping Groups », *in*: *IEEE Signal Processing Letters*.

Bayram, I. (2011), « Mixed norms with overlapping groups as signal priors », *in*: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Beck, A. and M. Teboulle (2009), « A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems », *in*: *SIAM J. Imaging Sci.*

Ben Mhenni, R. (2020), « Méthodes de programmation en nombres mixtes pour l'optimisation parcimonieuse en traitement du signal », PhD thesis, École Centrale de Nantes.

Ben Mhenni, R., S. Bourguignon, and J. Idier (2020), « A Greedy Sparse Approximation Algorithm Based On L1-Norm Selection Rules », *in*: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Ben Mhenni, R., S. Bourguignon, M. Mongeau, J. Ninin, and H. Carfantan (2020), « Sparse Branch and Bound for Exact Optimization of ℓ_0-Norm Penalized Least Squares »,

in: *ICASSP 2020, IEEE International Conference on Acoustics, Speech and Signal Processing*, Barcelona, Spain: IEEE.

Ben Mhenni, R., S. Bourguignon, and J. Ninin (2021), « Global optimization for sparse solution of least squares problems », *in*: *Optimization Methods and Software*.

Ben Mhenni, R., S. Bourguignon, J. Ninin, and F. Schmidt (2018), « Spectral Unmixing with Sparsity and Structuring Constraints », *in*: *2018 9th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*.

Bertsekas, D. P. (1982), *Constrained Optimization and Lagrange Multiplier Methods*, Athena Scientific.

Bertsimas, D., A. King, and R. Mazumder (2016), « Best subset selection via a modern optimization lens », *in*: *The Annals of Statistics*.

Bertsimas, D. and R. Shioda (2009), « Algorithm for cardinality-constrained quadratic optimization », *in*: *Computational Optimization and Applications*.

Bienstock, D. (1996), « Computational Study of a Family of Mixed-Integer Quadratic Programming Problems », *in*: *Mathematical Programming*.

Blumensath, T. and M. Davies (2008), « Iterative Thresholding for Sparse Approximations », *in*: *Journal of Fourier Analysis and Applications*.

Bonnefoy, A., V. Emiya, L. Ralaivola, and R. Gribonval (2014), « A Dynamic Screening Principle for the Lasso », *in*: *European Signal Processing Conference EUSIPCO*, Lisbon, Portugal.

Bourguignon, S., H. Carfantan, and J. Idier (2007), « A sparsity-based method for the estimation of spectral lines from irregularly sampled data », *in*: *IEEE J. Selected Topics Sig. Proc., Issue: Convex Optimization Methods for Signal Processing*.

Bourguignon, S., D. Mary, and É. Slezak (2011), « Restoration of Astrophysical Spectra With Sparsity Constraints: Models and Algorithms », *in*: *IEEE Journal of Selected Topics in Signal Processing*.

Bourguignon, S., J. Ninin, H. Carfantan, and M. Mongeau (2016), « Exact sparse approximation problems via mixed-integer programming: Formulations and computational performance », *in*: *IEEE Transactions on Signal Processing*.

Briceño-Arias, L. M. and N. Pustelnik (2023), « Theoretical and numerical comparison of first-order algorithms for cocoercive equations and smooth convex optimization », *in*: *Signal Processing*.

Candès, E. J., Y. C. Eldar, and D. Needell (2011), « Compressed Sensing with Coherent and Redundant Dictionaries », *in*: *Applied and Computational Harmonic Analysis*.

Candès, E. J., J. K. Romberg, and T. Tao (2006), « Stable signal recovery from incomplete and inaccurate measurements », *in*: *Communications on Pure and Applied Mathematics*.

Chambolle, A., R. A. DeVore, N. Lee, and B. J. Lucier (1998), « Nonlinear wavelet image processing: variational problems, compression, and noise removal through wavelet

shrinkage », *in*: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*.

Chambolle, A. and T. Pock (2011), « A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging », *in*: *Journal of Mathematical Imaging and Vision*.

Chaux, C., J.-C. Pesquet, and N. Pustelnik (2009), « Nested Iterative Algorithms for Convex Constrained Image Recovery Problems », *in*: *SIAM Journal on Imaging Sciences*.

Chen, S., S. Billings, and W. Luo (1988), *Orthogonal Least Squares Methods and Their Application to Nonlinear System Identification*, Research Report, URL: `http://eprints.whiterose.ac.uk/78100/`.

Chouzenoux, E., J.-C. Pesquet, and A. Repetti (2016), « A block coordinate variable metric forward–backward algorithm », *in*: *Journal of Global Optimization*.

Combettes, P. L. and L. E. Glaudin (2019), « Proximal Activation of Smooth Functions in Splitting Algorithms for Convex Image Recovery », *in*: *SIAM Journal on Imaging Sciences*.

Condat, L., D. Kitahara, A. Contreras, and A. Hirabayashi (2019), « Proximal Splitting Algorithms for Convex Optimization: A Tour of Recent Advances, with New Twists », *in*: *SIAM Rev.*

Cong, Y., J. Liu, G. Sun, Q. You, Y. Li, and J. Luo (2017), « Adaptive Greedy Dictionary Selection for Web Media Summarization », *in*: *IEEE Transactions on Image Processing*.

Cui, X., X. Zheng, S. S. Zhu, and X. Sun (2013), « Convex relaxations and MIQCQP reformulations for a class of cardinality-constrained portfolio selection problems », *in*: *Journal of Global Optimization*.

Dantas, C. and R. Gribonval (2019), « Stable safe screening and structured dictionaries for faster L1 regularization », *in*: *IEEE Transactions on Signal Processing*.

Daubechies, I., M. Defrise, and C. De Mol (2004), « An iterative thresholding algorithm for linear inverse problems with a sparsity constraint », *in*: *Communications on Pure and Applied Mathematics*.

Davis, D. and W. Yin (2015), « A Three-Operator Splitting Scheme and its Optimization Applications », *in*: *Set-Valued and Variational Analysis*.

Davis, G. M., S. Mallat, and M. Avellaneda (1997), « Adaptive greedy approximations », *in*: *Constructive Approximation*.

Dolan, E. and J. J. Moré (2002), « Benchmarking optimization software with performance profiles », *in*: *Mathematical Programming*.

Efron, B., T. J. Hastie, I. M. Johnstone, and R. Tibshirani (2004), « Least angle regression », *in*: *Annals of Statistics*.

Efroymson, M. A. (1960), « Multiple regression analysis », *in*: *Mathematical methods for digital computers*.

El Ghaoui, L., V. Viallon, and T. Rabbani (2010), *Safe Feature Elimination for the LASSO and Sparse Supervised Learning Problems*, tech. rep., EECS Department, University of California, Berkeley.

Eldar, Y. C., P. Kuppinger, and H. Bolcskei (2010), « Block-Sparse Signals: Uncertainty Relations and Efficient Recovery », *in*: *IEEE Transactions on Signal Processing*.

Fan, J. and R. Li (2001), « Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties », *in*: *Journal of the American Statistical Association*.

Fercoq, O., A. Gramfort, and J. Salmon (2015), « Mind the duality gap: safer rules for the LASSO », *in*: *Proceedings of the 32nd International Conference on Machine Learning*, ed. by F. Bach and D. Blei, Lille, France: PMLR.

Friedman, J. H., T. Hastie, and R. Tibshirani (2010), « Regularization Paths for Generalized Linear Models via Coordinate Descent », *in*: *Journal of Statistical Software*.

Fujii, K. and T. Soma (2018), « Fast Greedy Algorithms for Dictionary Selection with Generalized Sparsity Constraints », *in*: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Montréal, Canada: Curran Associates Inc.

Gilbert, A. C., S. Muthukrishnan, and M. Strauss (2003), « Approximation of functions over redundant dictionaries using coherence », *in*: *ACM-SIAM Symposium on Discrete Algorithms*.

Gramfort, A., M. Kowalski, and M. Hämäläinen (2012), « Mixed-norm estimates for the M/EEG inverse problem using accelerated gradient methods. », *in*: *Physics in Medicine and Biology*.

Guyard, T., C. Herzet, and C. Elvira (2022), « Node-Screening Tests For The L0-Penalized Least-Squares Problem », *in*: *ICASSP 2022 - IEEE International Conference on Acoustics, Speech and Signal Processing*, Singapore.

Harvey, W. D. and M. L. Ginsberg (1995), « Limited Discrepancy Search », *in*: *International Joint Conference on Artificial Intelligence*.

Hazimeh, H., R. Mazumder, and A. Saab (2021), « Sparse Regression at Scale: Branch-and-Bound rooted in First-Order Optimization », *in*: *Mathematical Programming*.

Huang, J., T. Zhang, and D. Metaxas (2011), « Learning with Structured Sparsity », *in*: *Journal of Machine Learning Research*.

Iordache, M.-D., J. M. Bioucas-Dias, and A. Plaza (2011), « Sparse Unmixing of Hyperspectral Data », *in*: *IEEE Transactions on Geoscience and Remote Sensing*.

Jenatton, R., J.-Y. Audibert, and F. Bach (2011), « Structured Variable Selection with Sparsity-Inducing Norms », *in*: *Journal of Machine Learning Research*.

Johnson, T. B. and C. Guestrin (2017), « StingyCD: Safely Avoiding Wasteful Updates in Coordinate Descent », *in*: *Proceedings of the 34th International Conference on Machine Learning*, ed. by D. Precup and Y. W. Teh, PMLR.

Joint Photographic Experts Group (1992), *JPEG Standard (JPEG ISO/IEC 10918-1 ITU-T Recommendation T.81)*, [Online; accessed July 04, 2023], URL: https://www.w3.org/Graphics/JPEG/itu-t81.pdf.

— (2000), *JPEG 2000 Standard Draft (final version is not publicly available)*, [Online; accessed July 04, 2023], URL: https://web.archive.org/web/20060702065150/http://www.jpeg.org/jpeg2000/CDs15444.html.

Komodakis, N. and J.-C. Pesquet (2014), « Playing with Duality: An Overview of Recent Primal-Dual Approaches for Solving Large-Scale Optimization Problems », *in*: *Signal Processing Magazine, IEEE*.

Kormylo, J. J. and J. M. Mendel (1982), « Maximum likelihood detection and estimation of Bernoulli - Gaussian processes », *in*: *IEEE Transactions on Information Theory*.

Kowalski, M. and B. Torrésani (2009), « Sparsity and persistence: mixed norms provide simple signal models with dependent coefficients », *in*: *Signal, Image and Video Processing*.

Krause, A. and V. Cevher (2010), « Submodular Dictionary Selection for Sparse Representation », *in*: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Haifa, Israel: Omnipress.

Kuang, Z., S. Geng, and D. Page (2017), « A Screening Rule for l1-Regularized Ising Model Estimation », *in*: *Advances in neural information processing systems*.

Lazzaro, D., L. B. Montefusco, and S. Papi (2015), « Blind cluster structured sparse signal recovery: A nonconvex approach », *in*: *Signal Process.*

Lee, H., A. Battle, R. Raina, and A. Ng (2006), « Efficient sparse coding algorithms », *in*: *Advances in Neural Information Processing Systems*, ed. by B. Schölkopf, J. Platt, and T. Hoffman, Vancouver, B.C., Canada: MIT Press.

Liu, J., Z. Zhao, J. Wang, and J. Ye (2013), « Safe Screening With Variational Inequalities and Its Applicaiton to LASSO », *in*: *31st International Conference on Machine Learning, ICML 2014*.

Loth, M. (2011), « Active Set Algorithms for the LASSO », PhD thesis, Université des Sciences et Technologie de Lille - Lille I.

Luo, J., M. Zhou, and J.-Q. Wang (2021), « AB&B: An Anytime Branch and Bound Algorithm for Scheduling of Deadlock-Prone Flexible Manufacturing Systems », *in*: *IEEE Transactions on Automation Science and Engineering*.

Malitsky, Y. and T. Pock (2018), « A First-Order Primal-Dual Algorithm with Linesearch », English, *in*: *SIAM Journal on Optimization*.

Massias, M. (2017), « From safe screening rules to working sets for faster lasso-type solvers », *in*: *10th NIPS Workshop on Optimization for Machine Learning*, Long Beach, California: Curran Associates, Inc.

Massias, M., S. Vaiter, A. Gramfort, and J. Salmon (2020), « Dual Extrapolation for Sparse Generalized Linear Models », *in*: *Journal of Machine Learning Research*.

Miller, A. (1990), *Subset Selection in Regression*, New York: Chapman and Hall/CRC.

Moreau, J.-J. (1962), « Fonctions convexes duales et points proximaux dans un espace hilbertien », *in*: *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences, Paris*.

Moreau, T., M. Massias, A. Gramfort, P. Ablin, P.-A. Bannier, B. Charlier, M. Dagréou, T. Dupré la Tour, G. Durif, C. F. Dantas, Q. Klopfenstein, J. Larsson, E. Lai, T. Lefort, B. Malézieux, B. Moufad, B. T. Nguyen, A. Rakotomamonjy, Z. Ramzi, J. Salmon, and S. Vaiter (2022), « Benchopt: Reproducible, efficient and collaborative optimization benchmarks », *in*: *NeurIPS*, New Orleans, Louisiana: Curran Associates, Inc.

Mosci, S., S. Villa, A. Verri, and L. Rosasco (2010), « A Primal-Dual Algorithm for Group Sparse Regularization with Overlapping Groups », *in*: *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, Vancouver, British Columbia, Canada: Curran Associates Inc.

Natarajan, B. K. (1995), « Sparse Approximate Solutions to Linear Systems », *in*: *SIAM Journal on Computing*.

Ndiaye, E., O. Fercoq, A. Gramfort, and J. Salmon (2017), « Gap Safe screening rules for sparsity enforcing penalties », *in*: *Journal of Machine Learning Research*.

Nemhauser, G. and L. Wolsey (1988), « General Algorithms », *in*: *Integer and Combinatorial Optimization*, John Wiley & Sons, Ltd, chap. II.4.

Neveu, B., G. Trombettoni, and I. Araya (2016), « Node selection strategies in interval Branch and Bound algorithms. », *in*: *Journal of Global Optimization*.

Nocedal, J. and S. J. Wright (2006), *Numerical Optimization*, 2nd ed., New York: Springer.

Obozinski, G., L. Jacob, and J.-P. Vert (2011), *Group Lasso with Overlaps: the Latent Group Lasso approach*, Research Report, p. 60, URL: `https://inria.hal.science/inria-00628498`.

Osborne, M., B. Presnell, and B. Turlach (2000), « A new approach to variable selection in least squares problems », *in*: *IMA Journal of Numerical Analysis*.

Pati, Y., R. Rezaiifar, and P. Krishnaprasad (1993), « Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition », *in*: *Conference Record of the Asilomar Conference on Signals, Systems & Computers*.

Pearson, K. (1901), « LIII. On lines and planes of closest fit to systems of points in space », *in*: *Philosophical Magazine Series 1*.

Qian, J., W. Du, Y. Tanigawa, M. Aguirre, R. Tibshirani, M. Rivas, and T. Hastie (May 2019), « A Fast and Flexible Algorithm for Solving the Lasso in Large-scale and Ultrahigh-dimensional Problems », working paper or preprint.

Raj, A., J. Olbrich, B. Gärtner, B. Schölkopf, and M. Jaggi (2016), « Screening Rules for Convex Problems », *in*: *9th NIPS Workshop on Optimization for Machine Learning*, Barcelona, Spain: Curran Associates, Inc.

Ren, S., S. Huang, J. Ye, and X. Qian (2017), « Safe Feature Screening for Generalized LASSO », *in*: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Rockafellar, R. T. (1970), *Convex Analysis*, Princeton: Princeton University Press.

Singer, R. B. and T. B. McCord (1979), « Mars - Large scale mixing of bright and dark surface materials and implications for analysis of spectral reflectance », *in*: *Lunar and Planetary Science Conference Proceedings*.

Soubies, E., L. Blanc-Féraud, and G. Aubert (2015), « A Continuous Exact ℓ_0 Penalty (CEL0) for Least Squares Regularized Problem », *in*: *SIAM Journal on Imaging Sciences*.

— (2017), « A Unified View of Exact Continuous Penalties for ℓ_2-ℓ_0 Minimization », *in*: *SIAM Journal on Optimization*.

Soussen, C., J. Idier, D. Brie, and J. Duan (2011), « From Bernoulli-Gaussian Deconvolution to Sparse Signal Restoration », *in*: *IEEE Transactions on Signal Processing*.

Stéphane, M. (2009), « CHAPTER 13 - Inverse Problems », *in*: *A Wavelet Tour of Signal Processing (Third Edition)*, ed. by M. Stéphane, Third Edition, Boston: Academic Press, pp. 699–752, ISBN: 978-0-12-374370-1, DOI: https://doi.org/10.1016/B978-0-12-374370-1.00017-3, URL: https://www.sciencedirect.com/science/article/pii/B9780123743701000173.

Tibshirani, R., J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. Tibshirani (2010), « Strong rules for discarding predictors in Lasso-type problems », *in*: *Journal of the Royal Statistical Society Series B (Statistical Methodology)*.

Tropp, J. A. (2004), « Greed Is Good: Algorithmic Results for Sparse Approximation », *in*: *IEEE Transactions on Information Theory*.

Tseng, P. (2001), « Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization », *in*: *Journal of Optimization Theory and Applications*.

Wainwright, M. J. (2009), « Sharp Thresholds for High-Dimensional and Noisy Sparsity Recovery Using ℓ_1-Constrained Quadratic Programming (Lasso) », *in*: *IEEE Transactions on Information Theory*.

Wang, D., X. Zhang, M. Fan, and X. Ye (2016), « Semi-Supervised Dictionary Learning via Structural Sparse Preserving », *in*: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, Arizona: AAAI Press.

Wang, H., F. Nie, W. Cai, and H. Huang (2013), « Semi-supervised Robust Dictionary Learning via Efficient l-Norms Minimization », *in*: *2013 IEEE International Conference on Computer Vision*.

Wang, J., P. Wonka, and J. Ye (2015), « Lasso Screening Rules via Dual Polytope Projection », *in*: *Journal of Machine Learning Research*.

Wang, Y., Z. J. Xiang, and P. J. Ramadge (2013), « Lasso screening with a small regularization parameter », *in*: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*.

Wikipedia contributors (2023a), *Support vector machine — Wikipedia, The Free Encyclopedia*, [Online; accessed 14-November-2023], URL: https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=1183475870.

— (2023b), *Work stealing — Wikipedia, The Free Encyclopedia*, [Online; accessed 13-September-2023], URL: https://en.wikipedia.org/w/index.php?title=Work_stealing&oldid=1167049202.

Xiang, Z. and P. Ramadge (2012), « Fast lasso screening tests based on correlations », *in*: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Xiang, Z., Y. Wang, and P. Ramadge (2014), « Screening Tests for Lasso Problems », *in*: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Xiang, Z., H. Xu, and P. J. Ramadge (2011), « Learning Sparse Representations of High Dimensional Data on Large Scale Dictionaries », *in*: *Advances in Neural Information Processing Systems*, ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Granada, Spain: Curran Associates, Inc.

Xianli, P. and Y. Xu (2018), « A safe reinforced feature screening strategy for lasso based on feasible solutions », *in*: *Information Sciences*.

Yoshida, T., I. Takeuchi, and M. Karasuyama (2019), « Safe Triplet Screening for Distance Metric Learning », *in*: *Neural Computation*.

Yuan, M. and Y. Lin (2006), « Model selection and estimation in regression with grouped variables », *in*: *Journal of the Royal Statistical Society Series B: Statistical Methodology*.

Zala, C. (1992), « High-resolution inversion of ultrasonic traces », *in*: *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency control*.

Zeng, Y., T. Yang, and P. Breheny (2020), « Hybrid safe-strong rules for efficient optimization in lasso-type problems », *in*: *Computational Statistics & Data Analysis*.

Zhang, C.-H. (2010), « Nearly Unbiased Variable Selection Under Minimax Concave Penalty », *in*: *The Annals of Statistics*, (visited on 07/05/2023).

Zhang, W., B. Hong, L. Ma, W. Liu, and T. Zhang (May 2018), *Safe Element Screening for Submodular Function Minimization*.

Zhang, X., J. Zheng, D. Wang, G. Tang, Z. Zhou, and Z. Lin (2023), « Structured Sparsity Optimization With Non-Convex Surrogates of $\ell_{2,0}$-Norm: A Unified Algorithmic Framework », *in*: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zimmert, J., C. S. de Witt, G. Kerg, and M. Kloft (2015), « Safe screening for support vector machines », *in*: *NIPS Workshop on Optimization for Machine Learning*.

COLLEGE SCIENCES
DOCTORAL DE L'INGENIERIE
PAYS DE LA LOIRE ET DES SYSTEMES

CENTRALE NANTES

Titre : Algorithme de branch-and-bound pour la résolution efficace de problèmes d'optimisation parcimonieuse.

Mot clés : Branch-and-bound, ℓ_0, Optimisation combinatoire, Optimisation continue

Résumé : De nombreux problèmes inverses en traitement du signal, statistique, imagerie biomédicale, astronomie et apprentissage machine peuvent se formuler comme la recherche de la meilleure combinaison de motifs expliquant les données, ces motifs étant choisis dans un catalogue connu. L'aspect parcimonieux du problème réside dans le faible nombre de motifs que l'on cherche à sélectionner via l'utilisation d'un terme ℓ_0. Plusieurs méthodes standards, telles que des algorithmes gloutons (OMP, OLS) et des reformulations convexes du problème (notamment en norme ℓ_1), permettent d'obtenir des solutions approchées de ce problème ℓ_0. Plus récemment, des méthodes permettant de résoudre exactement le problème ℓ_0 ont été développées, reposant sur des algorithmes de branch-and-bound. L'objectif de cette thèse est double. D'une part, explorer les possibilités d'accélérations des algorithmes branch-and-bound ℓ_0. D'autre part, étendre ces méthodes à des cas de parcimonie structurée, où l'on ne cherche plus simplement un faible nombre de motifs, mais un faible nombre de groupes de motifs. Ces contributions font l'objet d'un code open-source proposé au plus grand nombre.

Title: Branch-and-bound algorithm for efficient resolution of sparse optimization problems

Keywords: Branch-and-bound, ℓ_0, Combinatorial optimization, Continuous optimization

Abstract: Numerous inverse problems in signal processing, statistics, biomedical imaging, astronomy and machine learning can be cast as the search for the best pattern combination fitting measurements, these patterns being taken from a known dictionary. The sparsity of the problem comes from the small number of patterns desired, using a ℓ_0 term. Several standard methods, among which greedy algorithms (OMP, OLS) and convex reformulations (most notably the ℓ_1-norm reformulation), provide approximate solutions to this ℓ_0 problem. More recently, methods able to solve exactly the ℓ_0 problem were designed, using branch-and-bound algorithms. This PhD thesis has two goals. First, explore possible accelerations of current branch-and-bound algorithms dedicated to the ℓ_0 problem. Then, extend these branch-and-bound algorithms to the case of structured sparsity, where we are not looking for a small number of patterns fitting data, but a small number of *groups* of patterns fitting data. These contributions lead to the development of an open-source code publicly released.