Peeling pour le problème des moindres carrés avec régularisation ℓ_0

Théo GUYARD[1,2] Gilles MONNOYER[3] Clément ELVIRA[4] Cédric HERZET[1]

[1]Inria, Centre de l'Université de Rennes, France

[2]IRMAR UMR CNRS 6625, INSA Rennes, France

[3]ELEN, ICTEAM, UCLouvain, Belgique

[4]IETR UMR CNRS 6164, CentraleSupelec Rennes Campus, France

Résumé – Cet article présente une nouvelle méthode, appelée "*peeling*", visant à accélérer la résolution des problèmes de moindres carrés avec régularisation ℓ_0 via un algorithme "*Branch and Bound*". Notre procédure permet de renforcer les relaxations convexes construites à chaque nœud de l'arbre de décision exploré par l'algorithme et mène ainsi à un élagage potentiellement plus performant. Nous montrons empiriquement que le *peeling* permet des gains significatifs en termes de nombre de nœuds explorés par le *Branch and Bound* et de temps de résolution.

Abstract – We introduce a new methodology dubbed "*peeling*" to accelerate the resolution of ℓ_0-regularized least-squares problems via a Branch-and-Bound algorithm. Our procedure enables to tighten the convex relaxation constructed at each node of the decision tree and therefore potentially allows for more aggressive pruning. Numerical simulations show that our proposed methodology leads to significant gains in terms of number of nodes explored and solving time.

1 Introduction

Dans cet article, nous nous intéressons au problème

$$p^\star = \min_{\mathbf{x}\in\mathbb{R}^n} \mathrm{P}(\mathbf{x}) \triangleq \tfrac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_0 \qquad (1)$$

où $\mathbf{y}\in\mathbb{R}^m$ et $\mathbf{A}\in\mathbb{R}^{m\times n}$ sont des données, $\lambda>0$ est un hyper-paramètre et $\|\cdot\|_0$ la pseudo-norme ℓ_0 qui retourne le nombre d'entrées non-nulles de son argument.

La résolution *exacte* de ce problème est d'intérêt dans de nombreux domaines applicatifs [11]. Bien que cette tâche soit NP-difficile, certaines contributions ont récemment mis en évidence que les algorithmes de type "*Branch and Bound*" (BnB) permettent de résoudre des instances en grandes dimensions du problème (1) en un temps raisonnable [2]. Dans cet article, nous proposons une nouvelle stratégie permettant d'accélérer ce type d'algorithmes. Notre approche est basée sur la conception de tests (de faible coût calculatoire) permettant d'identifier des sous-ensembles de l'espace admissible ne contenant pas de solutions. Cette opération nous permet par la suite de renforcer la qualité des relaxations convexes considérées dans le processus d'élagage du BnB afin de (potentiellement) réduire le nombre total de nœuds traités. La technique proposée est baptisée "*peeling*" –ou "épluchage" en français– puisqu'elle permet de tronquer l'espace de recherche lors de la résolution du problème.

Structure. Dans la Sec. 2, nous introduisons les rudiments nécessaires à la compréhension des algorithmes de BnB. Nous présentons ensuite notre stratégie de *peeling* dans la Sec. 3 et testons ses performances dans la Sec. 4. Certaines simulations numériques additionnelles ainsi que les preuves de nos résultats sont disponibles dans le rapport technique [7].

Gilles Monnoyer est financé par le FNRS de Belgique. Le travail présenté dans cet article est reproductible. Notre code est disponible en ligne à l'adresse `https://github.com/TheoGuyard/BnbPeeling.jl`.

Notations. Les matrices et vecteurs sont représentés par des lettres grasses en majuscules et minuscules. La i-ème colonne d'une matrice $\mathbf{A}$ est notée $\mathbf{a}_i$ et la i-ème entrée d'un vecteur $\mathbf{x}$ est notée x_i. Nous désignons également les vecteurs ayant toutes leurs entrées égales à zéro (resp. un) par $\mathbf{0}$ (resp. $\mathbf{1}$). Les relations vectorielles sont appliquées composante par composante. Par exemple, $\mathbf{x}\in[\boldsymbol{\ell},\boldsymbol{u}]$ signifie $x_i\in[\ell_i,u_i]$ pour tout i. La fonction $\eta(\cdot)$ correspond à l'indicatrice valant 0 si la condition dans son argument est vérifiée et $+\infty$ sinon. Le cardinal d'un ensemble est désigné par $|\cdot|$. Enfin, on note $[x]_+ = \max(x,0)$ et $[\![1,n]\!] = \{1,\dots,n\}$.

2 Algorithmes de Branch and Bound

Nous présentons ci-dessous les principes généraux du BnB en se concentrant uniquement sur les parties nécessaires à la compréhension de cet article. Le lecteur intéressé par une description détaillée de ces méthodes pourra consulter [5, Sec. 1.2].

2.1 Élagage

L'algorithme de BnB résout un problème d'optimisation en construisant des recouvrements de l'espace admissible et en identifiant certains éléments de ces derniers ne contenant pas de solutions. Lorsque particularisé au problème (1), on peut assimiler le BnB à une recherche dans un arbre de décision dont les nœuds sont identifiés par deux sous-ensembles disjoints de $[\![1,n]\!]$, notés ν_0 and ν_1. Au nœud $\nu=(\nu_0,\nu_1)$, on s'intéresse à l'ensemble

$$\mathcal{X}^\nu \triangleq \{\mathbf{x}\in\mathbb{R}^n \colon \mathbf{x}_{\nu_0}=\mathbf{0},\ \mathbf{x}_{\nu_1}\neq\mathbf{0}\}, \qquad (2)$$

où $\mathbf{x}_{\nu_k}$ correspond à la restriction de $\mathbf{x}$ à ses indices dans ν_k. L'objectif est de déterminer si des solutions du problème (1) sont contenues dans $\mathcal{X}^\nu$. A cet effet, si l'on connaît une borne supérieure $\bar{p}$ sur $p^\star$ et que l'on définit

$$p^\nu \triangleq \inf_{\mathbf{x}\in\mathbb{R}^n} \mathrm{P}^\nu(\mathbf{x}) \triangleq \mathrm{P}(\mathbf{x}) + \eta(\mathbf{x}\in\mathcal{X}^\nu), \qquad (3)$$

on peut dériver l'implication suivante :

$$p^\nu > \bar{p} \implies \mathcal{X}^\nu \cap \mathcal{X}^\star = \emptyset, \tag{4}$$

où $\mathcal{X}^\star$ représente l'ensemble des minimiseurs de (1). En d'autres mots, si l'inégalité dans (4) est vérifiée, l'ensemble $\mathcal{X}^\nu$ ne contient aucune solution du problème (1) et peut donc être "élagué" de l'espace de recherche. Le BnB s'achève lorsque tous les ensembles ne contenant pas de minimiseur ont été identifiés et retourne les solutions du problème (1).

2.2 Relaxation

La mise en oeuvre du test d'élagage (4) nécessite la connaissance des quantités $\bar{p}$ et p^ν. Si $\bar{p}$ peut être facilement obtenue en évaluant la fonction objectif en un point admissible, l'évaluation de p^ν requiert quant à elle une complexité du même ordre de grandeur que celle nécessaire à la résolution de (1). Par conséquent, on considère le test d'élagage relâché suivant :

$$r^\nu > \bar{p} \implies \mathcal{X}^\nu \cap \mathcal{X}^\star = \emptyset, \tag{5}$$

où r^ν est une borne inférieure sur p^ν, calculable à faible coût.

Une étape classique dans la construction de r^ν consiste à rajouter une contrainte du type "$\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}]$" au problème (1). Le problème (3) est ensuite ré-exprimé comme suit :

$$p^\nu(\boldsymbol{\ell}, \boldsymbol{u}) = \inf_{\mathbf{x} \in \mathbb{R}^n} \mathrm{P}^\nu(\mathbf{x}; \boldsymbol{\ell}, \boldsymbol{u}) \tag{6}$$

où $\mathrm{P}^\nu(\mathbf{x}; \boldsymbol{\ell}, \boldsymbol{u}) \triangleq \mathrm{P}^\nu(\mathbf{x}) + \eta(\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}])$, et $p^\nu(\boldsymbol{\ell}, \boldsymbol{u})$ est considéré dans le test d'élagage (4) plutôt que p^ν.

L'ajout de la nouvelle contrainte "$\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}]$" permet de borner inférieurement la pseudo-norme ℓ_0 apparaissant dans la fonction objectif de (6) de la manière suivante :

$$\|\mathbf{x}\|_0 \geq g^\nu(\mathbf{x}) \triangleq |\nu_1| + \sum_{i \in \nu_\bullet} \tfrac{[x_i]_+}{u_i} - \tfrac{[-x_i]_+}{\ell_i}, \tag{7}$$

où $\nu_\bullet \triangleq [\![1, n]\!] \setminus (\nu_0 \cup \nu_1)$. Cette inégalité est valide $\forall \mathbf{x} \in \mathcal{X}^\nu \cap [\boldsymbol{\ell}, \boldsymbol{u}]$ avec la convention "$0/0 = 0$". En injectant cette dernière dans l'objectif du problème (6), on obtient la *relaxation* suivante :

$$r^\nu(\boldsymbol{\ell}, \boldsymbol{u}) = \min_{\mathbf{x} \in \mathbb{R}^n} \mathrm{R}^\nu(\mathbf{x}; \boldsymbol{\ell}, \boldsymbol{u}) \tag{8}$$

où $\mathrm{R}^\nu(\mathbf{x}; \boldsymbol{\ell}, \boldsymbol{u}) \triangleq \frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda g^\nu(\mathbf{x}) + \eta(\mathbf{x}_{\nu_0} = \mathbf{0}) + \eta(\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}])$. Notons que l'inégalité (7) permet de vérifier que $r^\nu(\boldsymbol{\ell}, \boldsymbol{u})$ est bien une borne inférieure sur $p^\nu(\boldsymbol{\ell}, \boldsymbol{u})$. De plus, le problème (8) est convexe et il existe de nombreuses procédures numériques permettant de calculer $r^\nu(\boldsymbol{\ell}, \boldsymbol{u})$ en temps polynomial [3].

En pratique, le choix des vecteurs $\boldsymbol{\ell}$ et $\boldsymbol{u}$ est sujet à deux exigences contradictoires. D'une part, on souhaite que la nouvelle contrainte ne disqualifie aucune des solutions du problème original (1). Plus formellement, $\boldsymbol{\ell}$ et $\boldsymbol{u}$ doivent vérifier

$$\mathcal{X}^\star \subseteq [\boldsymbol{\ell}, \boldsymbol{u}] \tag{9}$$

où $\mathcal{X}^\star$ est l'ensemble des minimiseurs de (1). Cette exigence requiert de choisir l'intervalle $[\boldsymbol{\ell}, \boldsymbol{u}]$ "suffisamment grand" puisque $\mathcal{X}^\star$ n'est pas connu *a priori*. D'autre part, la qualité de la borne inférieure $r^\nu(\boldsymbol{\ell}, \boldsymbol{u})$ se dégrade lorsque $\boldsymbol{\ell}$ et $\boldsymbol{u}$ deviennent grands en valeur absolue. Utiliser un intervalle $[\boldsymbol{\ell}, \boldsymbol{u}]$ trop large impacte alors les capacités d'élagage ainsi que le temps de résolution du BnB [4].

3 Peeling

Dans cette section, nous proposons une méthode pour remédier au problème de la calibration de l'intervalle $[\boldsymbol{\ell}, \boldsymbol{u}]$. Plus précisément, nous resserrons localement cet intervalle à chaque nœud traité par le BnB, tout en préservant la validité de l'algorithme. L'objectif est de construire un nouvel intervalle $[\boldsymbol{\ell}', \boldsymbol{u}']$ tel que

$$\forall \mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}] \setminus [\boldsymbol{\ell}', \boldsymbol{u}'] : \ \mathrm{P}^\nu(\mathbf{x}; \boldsymbol{\ell}, \boldsymbol{u}) > \bar{p} \tag{10a}$$

$$[\boldsymbol{\ell}', \boldsymbol{u}'] \subseteq [\boldsymbol{\ell}, \boldsymbol{u}]. \tag{10b}$$

Ces deux critères impliquent que la décision d'élagage (5) prise au nœud ν reste inchangée lorsqu'on remplace "$\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}]$" par "$\mathbf{x} \in [\boldsymbol{\ell}', \boldsymbol{u}']$". De plus, le raffinement de la contrainte améliore la qualité de la borne inférieure obtenue via la résolution de (8) et permet donc (potentiellement) un élagage plus aggressif.

3.1 Construction du nouvel intervalle $[\boldsymbol{\ell}', \boldsymbol{u}']$

Dans cette partie, nous montrons comment construire un intervalle $[\boldsymbol{\ell}', \boldsymbol{u}']$ vérifiant (10a)-(10b). Par contrainte de place, nous traitons uniquement le cas du vecteur $\boldsymbol{u}'$. La symétrie du problème permet de traiter $\boldsymbol{\ell}'$ de façon similaire.

Nous considérons le problème suivant :

$$p_\alpha^\nu(\boldsymbol{\ell}, \boldsymbol{u}) \triangleq \inf_{\mathbf{x} \in \mathbb{R}^n} \mathrm{P}^\nu(\mathbf{x}; \boldsymbol{\ell}, \boldsymbol{u}) + \eta(x_j > \alpha) \tag{11}$$

pour un indice $j \in \nu_\bullet$ et un scalaire $\alpha > 0$. Ce problème correspond à une version perturbée de (6) où x_j est contraint à être strictement plus grand que α. Par construction de $p_\alpha^\nu(\boldsymbol{\ell}, \boldsymbol{u})$, nous pouvons déduire le résultat suivant :

Lemma 1. *S'il existe un scalaire $\alpha \in [0, u_j[$ vérifiant*

$$p_\alpha^\nu(\boldsymbol{\ell}, \boldsymbol{u}) > \bar{p}, \tag{12}$$

alors (10a)-(10b) *sont vérifiés pour $\mathbf{u}'$ défini comme suit :*

$$u_i' = \begin{cases} \alpha & \textit{si } i = j \\ u_i & \textit{sinon.} \end{cases} \tag{13}$$

Le lemme ci-dessus fournit un principe de base pour la construction d'un vecteur $\boldsymbol{u}'$ satisfaisant (10a)-(10b) : il suffit d'identifier un réel $\alpha \in [0, u_j[$ vérifiant (12). Malheureusement, calculer $p_\alpha^\nu(\boldsymbol{\ell}, \boldsymbol{u})$ est une tâche NP-difficile et ce principe ne peut donc pas être appliqué en l'état. Dans la section suivante, nous proposons une méthode de faible coût calculatoire et se basant sur une version relâchée de (12) pour identifier α.

3.2 Implémentation efficace

En appliquant la dualité de Fenchel-Rockafellar au problème (8), on peut montrer (voir [7, App. A]) que l'inégalité suivante est valide pour tout $\mathbf{w} \in \mathbb{R}^m$:

$$p_\alpha^\nu(\boldsymbol{\ell}, \boldsymbol{u}) \geq \mathrm{D}^\nu(\mathbf{w}; \boldsymbol{\ell}, \boldsymbol{u}) + \psi_j(\mathbf{w}; \boldsymbol{\ell}, \boldsymbol{u}) + \alpha[-\mathbf{a}_j^{\mathrm{T}}\mathbf{w}]_+$$

où

$$\mathrm{D}^\nu(\mathbf{w}; \boldsymbol{\ell}, \boldsymbol{u}) \triangleq \tfrac{1}{2}\|\mathbf{y}\|_2^2 - \tfrac{1}{2}\|\mathbf{y} - \mathbf{w}\|_2^2 + \lambda|\nu_1| - \sum_{i \in \nu_1} \mu_{0,i}(\mathbf{a}_i^{\mathrm{T}}\mathbf{w}) - \sum_{i \in \nu_\bullet} \mu_{\lambda,i}(\mathbf{a}_i^{\mathrm{T}}\mathbf{w})$$

$$\psi_j(\mathbf{w}; \boldsymbol{\ell}, \boldsymbol{u}) \triangleq \mu_{\lambda,j}(\mathbf{a}_j^{\mathrm{T}}\mathbf{w}) - u_j[\mathbf{a}_j^{\mathrm{T}}\mathbf{w}]_+ + \lambda$$

et $\mu_{\rho,i}(v) \triangleq [u_i v - \rho]_+ + [\ell_i v - \rho]_+$. Par conséquent, l'inégalité (12) du Lemme 1 peut être relâchée comme suit :

$$\mathrm{D}^{\nu}(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u}) + \psi_j(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u}) + \alpha[-\mathbf{a}_j^{\mathrm{T}}\mathbf{w}]_+ > \bar{p} \tag{14}$$

et tout réel $\alpha \in [0, u_j[$ vérifiant (14) permet donc de construire (via (13)) un nouvel intervalle $[\boldsymbol{\ell}', \boldsymbol{u}']$ vérifiant (10a)-(10b).

On remarquera que le membre de gauche de (14) dépend *linéairement* de α. Pour un $\mathbf{w} \in \mathbb{R}^m$ fixé, on peut donc caractériser précisément les valeurs de α vérifiant cette condition. Cette observation nous amène au résultat suivant :

Proposition 1. *Soit* $\mathbf{w} \in \mathbb{R}^m$. *Si* $\mathbf{a}_j^{\mathrm{T}}\mathbf{w} \geq 0$ *et*

$$\mathrm{D}^{\nu}(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u}) + \psi_j(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u}) > \bar{p}, \tag{15}$$

alors le vecteur $\boldsymbol{u}'$ *défini par* (13) *avec* $\alpha = 0$ *vérifie* (10a)-(10b). *Si au contraire* $\mathbf{a}_j^{\mathrm{T}}\mathbf{w} < 0$, *alors le vecteur* $\boldsymbol{u}'$ *défini par* (13) *avec n'importe quel* $\alpha \in [0, u_j[$ *tel que*

$$\alpha > \bar{\alpha} \triangleq \frac{\bar{p} - \mathrm{D}^{\nu}(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u}) - \psi_j(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u})}{[-\mathbf{a}_j^{\mathrm{T}}\mathbf{w}]_+} \tag{16}$$

vérifie (10a)-(10b).

En d'autres termes, la Proposition 1 donne une procédure pour construire un vecteur $\boldsymbol{u}'$ permettant de raffiner la contrainte $\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}]$ à partir d'un vecteur $\mathbf{w} \in \mathbb{R}^m$ quelconque.

On montre maintenant que le résultat de la Proposition 1 peut être appliqué *en parallèle* à tous les indices $j \in [\![1, n]\!]$.

Lemma 2. *Soit* $[\boldsymbol{\ell}', \boldsymbol{u}']$ *et* $[\boldsymbol{\ell}'', \boldsymbol{u}'']$ *deux intervalles vérifiant* (10a)-(10b). *Alors, l'intervalle* $[\boldsymbol{\ell}', \boldsymbol{u}'] \cap [\boldsymbol{\ell}'', \boldsymbol{u}'']$ *satisfait également* (10a)-(10b).

En terme de complexité, appliquer la Proposition 1 en parallèle à tout les indices $j \in [\![1, n]\!]$ nécessite de calculer les produits scalaires $\{\mathbf{a}_i^{\mathrm{T}}\mathbf{w}\}_{i=1}^n$ ainsi que d'évaluer de la fonction $\mathrm{D}^{\nu}(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u})$. D'une part, les produits scalaires sont généralement calculés (pour un $\mathbf{w}$ donné) lors de la résolution de (8) car ils correspondent à l'opposé du gradient du terme des moindres carrés. On peut donc les obtenir sans aucune charge calculatoire supplémentaire. D'autre part, l'évaluation de la fonction $\mathrm{D}^{\nu}(\mathbf{w};\boldsymbol{\ell},\boldsymbol{u})$ a un coût calculatoire $\mathcal{O}(n+m)$, ce qui est négligeable par rapport aux autres opérations effectuées au sein de l'algorithm de BnB.

3.3 Propagation des nouvelles contraintes

Pour conclure cette section, nous montrons que n'importe quel intervalle $[\boldsymbol{\ell}', \boldsymbol{u}']$ vérifiant les critères (10a)-(10b) à un nœud ν peut être utilisé comme point de départ pour appliquer la méthode de *peeling* de la section précédente aux successeurs de ν dans l'arbre de décision du BnB.

Lemma 3. *Soit* $[\boldsymbol{\ell}', \boldsymbol{u}']$ *un intervalle vérifiant* (10a)-(10b) *au nœud* ν *et soit* ν' *un successeur de* ν *dans l'arbre de décision du BnB. Considérons également l'intervalle* $[\boldsymbol{\ell}'', \boldsymbol{u}'']$ *obtenu en appliquant la procédure de peeling définie dans la Proposition 1 au nœud* ν' *en remplaçant* $[\boldsymbol{\ell}, \boldsymbol{u}]$ *par* $[\boldsymbol{\ell}', \boldsymbol{u}']$. *Alors,* $[\boldsymbol{\ell}'', \boldsymbol{u}'']$ *vérifie*

$$\forall \mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}] \setminus [\boldsymbol{\ell}'', \boldsymbol{u}''] : \ \mathrm{P}^{\nu'}(\mathbf{x};\boldsymbol{\ell},\boldsymbol{u}) > \bar{p} \tag{17a}$$

$$[\boldsymbol{\ell}'', \boldsymbol{u}''] \subseteq [\boldsymbol{\ell}, \boldsymbol{u}]. \tag{17b}$$

Autrement dit, le Lemme 3 montre que n'importe quelle contrainte $\mathbf{x} \in [\boldsymbol{\ell}', \boldsymbol{u}']$ construite via notre procédure de *peeling* au nœud ν peut être propagée à ses successeurs. On peut donc espérer raffiner la contrainte initiale $\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}]$ le long des branches de l'arbre de décision construit par le BnB et, par conséquent, améliorer progressivement la qualité des bornes obtenues via la résolution du problème (8).

4 Simulations numériques

Dans cette dernière partie, nous évaluons notre stratégie de *peeling* et démontrons son efficacité pour accélérer la résolution du problème (1).

4.1 Données expérimentales

Nous considérons des instances synthétiques du problème (1) où $(m, n) = (100, 150)$. Les données $\mathbf{A}$ et $\mathbf{y}$ sont générées comme suit. Chaque ligne de la matrice $\mathbf{A}$ correspond à une réalisation d'une loi normale multivariée centrée avec une matrice de covariance $\mathbf{K} \in \mathbb{R}^{n \times n}$ dont l'entrée (i, j) est égale à $K_{ij} = 10^{-|i-j|}$. On fixe $\mathbf{y} = \mathbf{A}\mathbf{x}^{\dagger} + \mathbf{n}$ où $\mathbf{n}$ est un bruit blanc Gaussien et $\mathbf{x}^{\dagger} \in \mathbb{R}^n$ un vecteur avec 5 entrées non-nulles et équi-espacées dont l'amplitude est égale à $x_i^{\dagger} = \mathrm{sign}(r_i) + r_i$, avec $r_i \sim \mathcal{N}(0, \sigma^2)$ et $\sigma > 0$. La variance de $\mathbf{n}$ est ajustée pour que le $\mathrm{SNR} \triangleq 10 \log_{10}(\|\mathbf{A}\mathbf{x}^{\dagger}\|_2^2 / \|\mathbf{n}\|_2^2)$ soit égal à 15dB. Le paramètre λ est calibré avec les procédures du package L0LEARN [8]. Des informations supplémentaires sur la calibration de λ se trouvent dans notre rapport technique [7].

4.2 Procédures de résolution

Nous comparons les méthodes de résolution suivantes : *i)* CPLEX [10], un solveur générique ; *ii)* L0BNB [9][1] et *iii)* SBNB [1], deux algorithmes de BnB spécialisés dans la résolution du problème (1) ; *iv)* SBNB-N, correspondant à une amélioration de SBNB avec la méthode de "*node-screening*" présentée dans [6] ; *v)* SBNB-P, correspondant à une amélioration de SBNB avec la méthode de "*peeling*" présentée dans cet article. CPLEX résout la relaxation (8) avec un algorithme de simplexe. De leur côté, L0BNB, SBNB, SBNB-N et SBNB-P utilisent une procédure de *coordinate descent*. CPLEX[2] est implémenté en C++, L0BNB[3] est implémenté en Python et SBNB, SBNB-N et SBNB-P[4] sont implémentés en Julia. Toutes les procédures de résolution sont initialisées avec $\boldsymbol{\ell} = -M\mathbf{1}$ et $\boldsymbol{u} = M\mathbf{1}$, pour un certain $M > 0$. Cela correspond à l'approche "*Big-M*", standard dans la littérature de recherche opérationnelle [1, 2]. Dans nos simulations, nous fixons $M = \gamma\|\mathbf{x}^{\star}\|_{\infty}$ où $\gamma \geq 1$ et $\mathbf{x}^{\star}$ est la solution du problème (1). Cette dernière est unique avec probabilité 1 au vu de notre modèle de génération de $\mathbf{A}$ et $\mathbf{y}$. Fixer M de cette manière nécessite de résoudre (1) au préalable. Cette procédure n'est effectuée ici que dans le but de comparer la sensibilité des méthodes vis-à-vis du paramètre γ. En pratique, M est fixé heuristiquement.

[1] Dans cet article, les auteurs considèrent le problème (1) avec une régularisation Ridge supplémentaire. Ils permettent cependant de pondérer cette dernière par zéro, ce qui ramène au problème que nous étudions.

[2] https://github.com/jump-dev/CPLEX.jl

[3] https://github.com/hazimehh/L0Learn

[4] https://github.com/TheoGuyard/BnbPeeling.jl

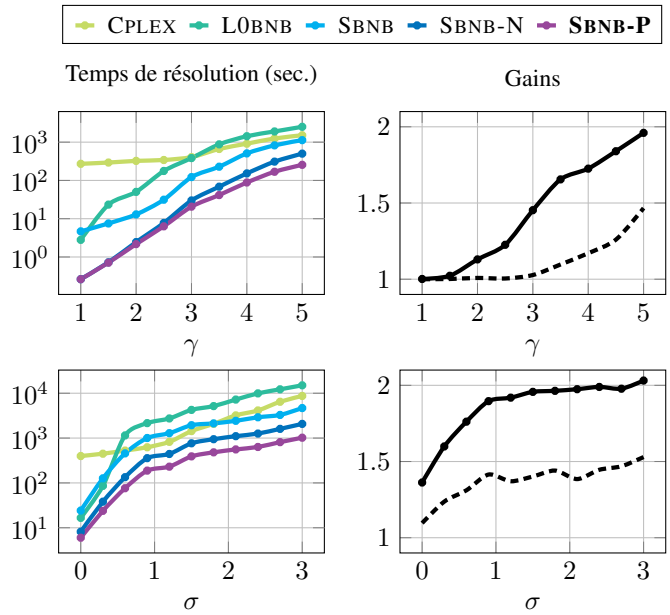

FIGURE 1 : Temps de résolution en fonction de γ (haut, $\sigma = 1$) et de σ (bas, $\gamma = 5$) et gains de SBNB-P en temps (ligne) et en nombre de nœuds explorés (pointillés) par rapport à SBNB-N.

Dans la méthode SBNB-P, la stratégie de *peeling* proposée est appliquée à chaque itération lors de la résolution de (8). On utilise l'itéré courant $\mathbf{x}^{(k)}$ pour définir le vecteur $\mathbf{w} \triangleq \mathbf{y} - \mathbf{A}\mathbf{x}^{(k)}$ utilisé dans la Proposition 1. Cette dernière est appliquée en parallèle, *i.e.,* simultanément pour toutes les entrées de $\mathbf{x}$. La valeur α vérifiant (16), si elle existe, est fixée à $\alpha = \bar{\alpha} + 10^{-16}$. Le raffinement de la contrainte $\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}]$ est propagée au travers des branches de l'arbre de BnB, comme expliqué dans la Sec. 3.3. Le vecteur $\mathbf{w}$ utilisé n'est pas nécéssairement celui permettant de raffiner au mieux l'interval $[\boldsymbol{\ell}, \boldsymbol{u}]$, mais il permet d'implémenter la Proposition 1 sans aucune surcharge calculatoire au sein de l'algorithme de BnB.

4.3 Performances

La Fig. 1 montre les performances des différentes méthodes de résolution considérées. Les résultats sont moyennés sur la résolution de 50 instances du problème (1). Les simulations sont effectuées sur un CPU Intel Xeon E5-2660 v3 cadencé à 2.60 GHz avec 16 GB de mémoire RAM. La colonne de gauche représente le temps moyen de résolution en fonction de γ et de σ ; celle de droite illustre les gains de notre méthode (SBNB-P) en termes de temps et de nombre de nœuds explorés comparé à son compétiteur le plus performant (SBNB-S).

La méthode SBNB-P donne les meilleures performances en terme de temps de résolution. Comme cette dernière correspond à une amélioration de SBNB avec notre stratégie de *peeling*, l'espace entre la courbe violette et la courbe bleu clair matérialise le gain apporté par notre méthode. Celui-ci est d'environ un ordre de grandeur. On note également que cette accélération se produit même si $\gamma = 1$, c'est-à-dire lorsque la contrainte *Big-M* est parfaitement calibrée. Cela s'explique par le fait que le *peeling* permet de raffiner *individuellement* chaque composante de la contrainte initiale $\mathbf{x} \in [\boldsymbol{\ell}, \boldsymbol{u}]$, et ce à *chaque nœud* de l'arbre de BnB.

Enfin, on remarque que SBNB-P permet une réduction du temps de résolution par rapport à son meilleur concurrent SBNB-N. En particulier, SBNB-P surclasse toujours SBNB-N, comme le montre la partie droite de la Fig. 1. Ce graphique reflète les améliorations apportées par le *peeling* en terme de nombre de nœuds traités. Comme prévu, le *peeling* permet un élagage plus agressif et diminue donc le temps de résolution.

Comme remarque finale, nous notons que le *peeling* peut s'appliquer quelque soit la valeur initiale de $M > 0$. Cependant, le resserrement de l'interval $[\boldsymbol{\ell}, \boldsymbol{u}]$ effectué via la Proposition 1 dépend de la qualité de la relaxation (8), qui se dégrade lorsque M augmente. En pratique, s'attend donc à devoir explorer un grand nombre de noeuds du BnB avant de réussir à raffiner une valeur de M initialement très élevée.

5 Conclusion

Dans cet article, nous avons présenté une méthode de "*peeling*" permettant d'accélérer une procédure de BnB dédiée à la résolution de problèmes des moindres carrés avec régularisation ℓ_0. Nous raffinons localement une contrainte du problème, ce qui permet de renforcer les relaxations construites à chaque nœud de l'arbre du BnB. Ce processus mène à un élagage plus agressif et améliore significativement le temps de résolution.

Références

[1] R. BEN MHENNI, S. BOURGUIGNON, M. MONGEAU, J. NININ et H. CARFANTAN : Sparse branch and bound for exact optimization of l0-norm penalized least squares. *In International Conference on Acoustics, Speech and Signal Processing*, pages 5735–5739. IEEE, 2020.

[2] D. BERTSIMAS, A. KING et R. MAZUMDER : Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2), 2016.

[3] S. BOYD et L. VANDENBERGHE : *Convex optimization*. Cambridge university press, 2004.

[4] J. D. CAMM, A. S. RATURI et S. TSUBAKITANI : Cutting big-m down to size. *Interfaces*, 20(5):61–66, 1990.

[5] M. CONFORTI, G. CORNUÉJOLS, G. ZAMBELLI *et al.* : *Integer programming*. Springer, 2014.

[6] T. GUYARD, C. HERZET et C. ELVIRA : Node-screening tests for the l0-penalized least-squares problem. *In International Conference on Acoustics, Speech and Signal Processing*, pages 5448–5452. IEEE, 2022.

[7] T. GUYARD, G. MONNOYER, C. ELVIRA et C. HERZET : Safe peeling for l0-regularized least-squares with supplementary material. *arXiv preprint*, 2023.

[8] H. HAZIMEH, R. MAZUMDER et T. NONET : L0learn : A scalable package for sparse learning using l0 regularization. *arXiv preprint*, 2022.

[9] H. HAZIMEH, R. MAZUMDER et A. SAAB : Sparse regression at scale : Branch-and-bound rooted in first-order optimization. *Mathematical Programming*, 196(1-2):347–388, 2022.

[10] IBM ILOG : User's manual for cplex v12.1. *International Business Machines Corporation*, 46(53):157, 2009.

[11] A. M. TILLMANN, D. BIENSTOCK, A. LODI et A. SCHWARTZ : Cardinality minimization, constraints, and regularization : a survey. *arXiv preprint*, 2021.