

Un solveur efficace pour la résolution de problèmes parcimonieux avec pénalité L0

Theo Guyard

▶ **To cite this version:**

Theo Guyard. Un solveur efficace pour la résolution de problèmes parcimonieux avec pénalité L0. ROADEF 2023 - 24ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF, Feb 2023, Rennes, France. pp.1-6. hal-03960204

HAL Id: hal-03960204
https://hal.science/hal-03960204

Submitted on 27 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Un solveur efficace pour la résolution de problèmes parcimonieux avec pénalité L0

Théo Guyard

Inria, Centre de l'Université de Rennes, 35000 Rennes, France
INSA Rennes, IRMAR, 35000 Rennes, France
theo.guyard@insa-rennes.fr

Mots-clés : *PMNL, Problèmes Parcimonieux, Pénalité L0, Branch-and-Bound.*

1 Introduction

Trouver une représentation parcimonieuse est un problème fondamental dans le domaine des statistiques, de l'apprentissage automatique ou des problèmes inverses [1]. Cela consiste à décomposer un vecteur $\mathbf{y} \in \mathbb{R}^m$ comme une combinaison d'un petit nombre de colonnes d'une matrice $\mathbf{A} \in \mathbb{R}^{m \times n}$ au travers d'un certain modèle. Cette tâche peut être effectuée en résolvant

$$p^\star = \min_{\mathbf{x} \in \mathbb{R}^n} \Big\{ \mathrm{P}(\mathbf{x}) = f(\mathbf{A}\mathbf{x}) + \lambda \|\mathbf{x}\|_0 + \eta(\|\mathbf{x}\|_\infty \leq M) \Big\} \tag{$\mathcal{P}$}$$

où l'on définit la pseudo-norme ℓ_0 comme $\|\mathbf{x}\|_0 = \mathrm{card}(\{i, x_i \neq 0\})$ et la fonction $\eta(\mathcal{C})$ comme une indicatrice convexe valant 0 si la contrainte $\mathcal{C}$ est vérifiée et valant $+\infty$ sinon. Les entrées de $\mathbf{x}$ pondèrent chaque colonne de $\mathbf{A}$. D'une part, la fonction $f(\cdot)$ assure la qualité de la décomposition de $\mathbf{y}$. D'autre part, la pseudo-norme ℓ_0 force la parcimonie. Le paramètre $\lambda > 0$ permet de contrôler le compromis fait entre ces deux paradigmes. La contrainte imposée pour un $M > 0$ n'est pas intrinsèquement liée à la décomposition parcimonieuse mais est indispensable pour construire des relaxations bornées du problème.

Le problème $(\mathcal{P})$ est NP-difficile dans son expression générale. Ceci a initialement conduit au développement de procédures sous-optimales pour approcher ses solutions afin de traiter des instances en grande dimension. Celles-ci ne sont toutefois garanties de résoudre $(\mathcal{P})$ que sous des conditions restrictives et rarement satisfaites en pratique. C'est pourquoi il y a eu un récent regain d'intérêt pour les méthodes résolvant $(\mathcal{P})$ de manière exacte. En particulier, ce problème peut être reformulé comme un Programme Mathématique Non Linéaire (PMNL) avec des variables binaires codant la nullité des entrées de $\mathbf{x}$ [2].

Contributions. Avec des données en grande dimension, les solveurs génériques de PMNL échouent généralement à résoudre $(\mathcal{P})$ en un temps raisonnable. Il y a donc un besoin de méthodes efficaces spécifiquement dédiées à la résolution de ce problème. Dans cet article, nous montrons comment concevoir un algorithme de Branch-and-Bound (BnB) qui exploite la structure parcimonieuse de $(\mathcal{P})$. Nous présentons plusieurs stratégies d'accélération qui peuvent être mises en œuvre pour considérablement améliorer son temps de résolution. Notre implémentation est disponible en `Julia` via le package `El0ps.jl`.

Notations et hypothèses. Les lettres calligraphiques (*e.g.,* $\mathcal{S}$) désignent des ensembles. Les lettres majuscules et minuscules en gras (*e.g.,* $\mathbf{A}$ et $\mathbf{x}$) représentent respectivement des matrices et des vecteurs. $\mathbf{0}$ et $\mathbf{1}$ désignent les vecteurs entièrement composés de zéros ou de uns. 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. De plus, $\mathbf{x}_\mathcal{S}$ désigne la restriction de $\mathbf{x}$ à ses entrées indexées par $\mathcal{S}$ et $\mathbf{A}_\mathcal{S}$ correspond à la restriction de $\mathbf{A}$ à ses colonnes indexées par $\mathcal{S}$. Enfin, nous utilisons la notation $[x]_+ = \max(x, 0)$. Tout au long de cet article, nous supposons que $f(\cdot)$ est une fonction propre, semi-continue inférieure, convexe, différentiable et que ∇f est L-Lipschitz.

2 Structure du Branch-and-Bound

Une procédure de BnB énumère implicitement toutes les solutions réalisables d'un PMNL dans un arbre de décision et construit des règles pour éliminer les candidats non pertinents. Dans le cas de $(\mathcal{P})$, un nœud de l'arbre est défini comme un triplet $\nu = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_\bullet)$ où $\mathcal{S}_0$ et $\mathcal{S}_1$ sont les ensembles des entrées de $\mathbf{x}$ forcées à être nulles et non-nulles, et où $\mathcal{S}_\bullet$ correspond aux entrées pour lesquelles aucune décision n'a encore été prise à ce stage de l'arbre. L'algorithme de BnB initialise l'arbre avec le nœud $\nu = (\emptyset, \emptyset, \{1, \ldots, n\})$ puis alterne entre l'*évaluation*, le potentiel *élagage* et la création via un *branchement* de nœuds jusqu'à qu'il n'en reste plus à traiter.

2.1 Étapes d'évaluation et d'élagage

Lors du traitement d'un nœud $\nu = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_\bullet)$, on teste si tout candidat vérifiant les contraintes du nœud courant peut correspondre à un minimiseur de $(\mathcal{P})$. Pour cela, on regarde la plus petite valeur p^ν de l'objectif de $(\mathcal{P})$ permise lorsqu'on impose $\mathbf{x}_{\mathcal{S}_0} = \mathbf{0}$ et $\mathbf{x}_{\mathcal{S}_1} \neq \mathbf{0}$. Si $p^\nu > p^\star$, le nœud ν peut être élaguée car il ne peut pas mener à un optimiseur de $(\mathcal{P})$. Cependant, cette règle présente peu d'intérêt en pratique puisque $p^\star$ n'est pas connue et que l'évaluation de p^ν nécessite la résolution d'un nouveau problème NP-difficile. Pour contourner ces difficultés, on relâche la règle d'élagage. Plus précisément, soit $r^\nu \leq p^\nu$ et soit $\bar{p} \geq p^\star$, alors

$$r^\nu > \bar{p} \implies p^\nu > p^\star \tag{1}$$

Premièrement, on peut calculer une borne inférieure r^ν sur p^ν en remplaçant la somme de la pseudo-norme ℓ_0 et des indicatrices des contraintes par son *enveloppe convexe*. Cela donne

$$r^\nu = \min_{\mathbf{x} \in \mathbb{R}^n} \left\{ \mathrm{R}^\nu(\mathbf{x}) = f(\mathbf{A}\mathbf{x}) + g^\nu(\mathbf{x}) \right\} \tag{$\mathcal{R}^\nu$}$$

où l'on définit $g^\nu(\mathbf{x}) = \sum_{i=1}^n g_i^\nu(x_i)$ avec $g_i^\nu(x_i) = \eta(x_i = 0)$ si $i \in \mathcal{S}_0$, $g_i^\nu(x_i) = \eta(|x_i| \leq M) + \lambda$ si $i \in \mathcal{S}_1$ et $g_i^\nu(x_i) = \eta(|x_i| \leq M) + \frac{\lambda}{M}|x_i|$ si $i \in \mathcal{S}_\bullet$. Le problème $(\mathcal{R}^\nu)$ est convexe et on peut donc obtenir r^ν en temps polynomial. Deuxièmement, la borne supérieure $\bar{p}$ sur $p^\star$ peut être obtenue en gardant la trace de la meilleure valeur objectif de $(\mathcal{P})$ obtenue durant le processus de BnB. On peut affiner sa valeur à l'aide d'heuristiques à chaque nœud exploré.

2.2 Étape de branchement

Si le nœud $\nu = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_\bullet)$ n'a pas été élagué, l'exploration de l'arbre se poursuit. Un élément $j \in \mathcal{S}_\bullet$ est sélectionné et deux nœuds sont créés sous ν en imposant soit $x_j = 0$, soit $x_j \neq 0$. Une stratégie efficace en pratique pour obtenir rapidement une bonne borne supérieure $\bar{p}$ consiste à explorer les nœuds où les variables sont fixées différentes de zéro en premier. On se rapproche alors des processus heuristiques gloutons qui permettent d'obtenir des solutions approchées de $(\mathcal{P})$. Pour choisir l'indice j à partir duquel créer une nouvelle branche, on peut par exemple sélectionner l'élément de $\mathcal{S}_\bullet$ ayant la plus grande valeur absolue dans la solution de $(\mathcal{R}^\nu)$.

3 Exploitation de la structure parcimonieuse du problème

L'efficacité de l'algorithme de BnB dépend essentiellement du nombre de nœuds explorés et de la capacité à les traiter rapidement. Les méthodes présentées dans cette section se focalisent sur ces deux points et exploitent la structure parcimonieuse de $(\mathcal{P})$ afin d'accélérer sa résolution. Elles reposent principalement sur des résultats découlant du théorème suivant.

Theorème 1. *Le problème dual associé à $(\mathcal{R}^\nu)$ est donné par*

$$d^\nu = \max_{\mathbf{u} \in \mathbb{R}^m} \left\{ \mathrm{D}^\nu(\mathbf{u}) = -f^\star(-\mathbf{u}) - \sum_{i \in \mathcal{S}_\bullet} [\boldsymbol{\mu}_i(\mathbf{u})]_+ - \sum_{i \in \mathcal{S}_1} \boldsymbol{\mu}_i(\mathbf{u}) \right\} \tag{$\mathcal{D}^\nu$}$$

où $f^\star(\cdot)$ est la conjuguée de Fenchel de $f(\cdot)$ et où l'on définit $\boldsymbol{\mu}_i(\mathbf{u}) = M|\mathbf{a}_i^{\mathrm{T}}\mathbf{u}| - \lambda$, $\forall i$. De plus, on a $\mathrm{R}^\nu(\mathbf{x}) \geq r^\nu = d^\nu \geq \mathrm{D}^\nu(\mathbf{u})$ pour tout $(\mathbf{x}, \mathbf{u})$. Enfin, tout couple d'optimiseurs $(\mathbf{x}^\nu, \mathbf{u}^\nu)$ de $(\mathcal{R}^\nu)$-$(\mathcal{D}^\nu)$ vérifie les conditions d'optimalité $\mathbf{A}^{\mathrm{T}}\mathbf{u}^\nu \in \partial g^\nu(\mathbf{x}^\nu)$ et $\mathbf{u}^\nu = -\nabla f(\mathbf{A}\mathbf{x}^\nu)$.

3.1 Accélération de la résolution de $(\mathcal{R}^\nu)$

Pour résoudre $(\mathcal{R}^\nu)$ plus rapidement, on peut tirer parti du fait qu'en pratique, de nombreuses entrées de sa solution valent 0 ou $\pm M$ en général. Connaître ces éléments au préalable permet d'éviter de dépenser inutilement de la ressource calculatoire en tentant de les mettre à jour dans la procédure itérative choisie pour traiter $(\mathcal{R}^\nu)$. On propose d'adapter à notre contexte de précédents travaux portant sur des problèmes parcimonieux *convexes*, comme le Lasso, afin d'identifier de telles composantes [3]. Plus précisément, on a le résultat suivant.

Theorème 2. *Soit* $(\mathbf{x}^\nu, \mathbf{u}^\nu)$ *un couple de solutions de* $(\mathcal{R}^\nu)$-$(\mathcal{D}^\nu)$ *et soit* $\mathcal{S}(\mathbf{c}, r)$ *une sphère de centre* $\mathbf{c} \in \mathbb{R}^m$ *et de rayon* $r \geq 0$ *contenant* $\mathbf{u}^\nu$*. Alors, pour tout* $j \in \mathcal{S}_\bullet$*, on a*

$$|c_j| + r < \tfrac{\lambda}{M} \implies x_j^\nu = 0 \tag{2a}$$

$$|c_j| - r > \tfrac{\lambda}{M} \implies x_j^\nu = \operatorname{sign}(c_j) M \tag{2b}$$

Le Thm. 2 montre qu'il est possible d'identifier certaines composantes nulles ou valant $\pm M$ dans la solution de $(\mathcal{R}^\nu)$ si l'on connaît une sphère $\mathcal{S}(\mathbf{c}, r)$ contenant $\mathbf{u}^\nu$. Lorsque de telles composantes sont identifiées, on peut les fixer une fois pour toute lors de la résolution de $(\mathcal{R}^\nu)$ et ainsi effectuer une économie de ressource calculatoire. En pratique, on va effectuer les tests (2a)-(2b) dynamiquement lors de la résolution de $(\mathcal{R}^\nu)$. Autrement dit, on va construire une nouvelle région $\mathcal{S}(\mathbf{c}, r)$ à chaque itération se resserrant autour de $\mathbf{u}^\nu$. Ceci permet d'accroître la performance des tests (2a)-(2b). On va progressivement fixer de nouvelles variables de $(\mathcal{R}^\nu)$ et ainsi réduire son coût de résolution. Pour construire $\mathcal{S}(\mathbf{c}, r)$, on peut par exemple utiliser la méthode "GAP" qui permet d'implémenter les tests (2a)-(2b) sans aucun coût [4].

Une autre observation récurrente lors de la résolution de $(\mathcal{R}^\nu)$ est que de nombreux nœuds élagués vérifient en fait la règle (1) avec une grande marge. Résoudre ce problème exactement afin d'obtenir r^ν peut donc s'avérer être une dépense calculatoire importante mais inutile si l'on doit ensuite élaguer le nœud ν. Cependant, on ne peut pas simplement arrêter l'algorithme de résolution de $(\mathcal{R}^\nu)$ plus tôt et substituer la valeur de r^ν par la meilleure valeur objectif de $(\mathcal{R}^\nu)$ connue dans le test d'élagage (1) car cela le rendrait invalide. Pour répondre à ce problème, on propose de mettre en place une stratégie d'*élagage précoce* [5] se basant sur l'objectif *dual*. Celle-ci consiste à stopper la résolution de $(\mathcal{R}^\nu)$ lorsqu'au cours du processus itératif, on récupère un point dual $\mathbf{u} \in \mathbb{R}^m$ vérifiant

$$\mathrm{D}^\nu(\mathbf{u}) > \bar{p} \tag{3}$$

En effet, si cette inégalité est validée, on peut certifier que la condition $r^\nu > \bar{p}$ est vérifiée en conséquence du Thm. 1. On en déduit que le nœud ν sera obligatoirement élagué et qu'il n'est donc pas nécessaire de poursuivre la résolution de $(\mathcal{R}^\nu)$. L'évaluation de la condition (3) peut être effectuée avec une complexité négligeable en réutilisant des valeurs déjà calculées par la majorité des algorithmes permettant de résoudre $(\mathcal{R}^\nu)$. En pratique, appliquer le test (3) permet de concenter la puissance de calcul dans les parties prometteuses de l'arbre de décision.

3.2 Détection rapide de nœuds inutiles

Parmi tous les nœuds explorés dans l'arbre de décision, certains peuvent être élagués de manière évidente[1]. Le BnB étant un algorithme de résolution exacte, il faut cependant le prouver. Le processus classique d'élagage consistant à un résoudre un problème d'optimisation convexe par nœud traité peut donc sembler coûteux pour ces cas "simples" à traiter. Dans la suite, nous présentons une méthode pour identifier *sans aucun coût* des nœuds qu'il est inutile d'explorer. On peut voir celle-ci comme un complément à l'élagage standard du BnB. Plus spécifiquement, notons $\nu \cap \{x_j = 0\}$ et $\nu \cap \{x_j \neq 0\}$ les deux nœuds créés directement en dessous de ν en fixant $x_j = 0$ ou $x_j \neq 0$. Alors, on a le résultat suivant.

[1] Par exemple, un nœud où toutes les entrées de la variable sont fixées différentes de zéro ne donnera généralement pas une solution de $(\mathcal{P})$ car la pénalité ℓ_0 sera trop importante.

Theorème 3. *Soit $\nu = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_\bullet)$ un nœud et soit $j \in \mathcal{S}_\bullet$. Alors, $\forall \mathbf{u} \in \mathbb{R}^m$, on a*

$$\bar{p} < \mathrm{D}^\nu(\mathbf{u}) + [\boldsymbol{\mu}_i(\mathbf{u})]_+ \implies p^\star < p^{\nu \cap \{x_j = 0\}} \tag{4a}$$

$$\bar{p} < \mathrm{D}^\nu(\mathbf{u}) + [-\boldsymbol{\mu}_i(\mathbf{u})]_+ \implies p^\star < p^{\nu \cap \{x_j \neq 0\}} \tag{4b}$$

Les implications ci-dessus découlent du lien qu'on peut faire entre l'objectif de $(\mathcal{D}^\nu)$ à deux nœuds consécutifs. Le Thm. 3 montre que si l'on connaît un point dual $\mathbf{u} \in \mathbb{R}^m$ au nœud ν, on peut potentiellement élaguer des parties de l'arbre sous ν. En effet, lorsque la partie gauche de (4a) est vérifiée, on peut immédiatement imposer $x_j \neq 0$ au nœud ν car on ne pourra pas atteindre un optimiseur de $(\mathcal{P})$ autrement. De manière symétrique, lorsque la partie gauche de (4b) est vérifiée, on peut immédiatement imposer $x_j = 0$ au nœud ν. En pratique, fixer une nouvelle variable à un nœud ν donné permet d'élaguer la moitié de l'arbre restant sous ν. Notons que les implications (4a)-(4b) portent sur tous les éléments de $\mathcal{S}_\bullet$ et permettent donc potentiellement de fixer plusieurs variables *simultanément*. En utilisant le Thm. 3, on peut donc éliminer certaines parties de l'arbre de BnB sans avoir à passer par une résolution coûteuse de $(\mathcal{R}^\nu)$. Comme pour les méthodes présentées dans la Sec. 3.1, on peut appliquer (4a)-(4b) sans aucun frais calculatoire supplémentaire lors de la résolution de $(\mathcal{R}^\nu)$.

3.3 Détails d'implémentation

Dans les sections précédentes, nous avons souligné le fait que les méthodes d'accélération présentées pouvaient être appliquées gratuitement lors de la résolution de $(\mathcal{R}^\nu)$. Nous donnons maintenant des détails supplémentaires à propos de ce point. Notons tout d'abord que leur implémentation à une itération t donnée lors de la résolution de $(\mathcal{R}^\nu)$ nécessite uniquement[2] la connaissance d'un point dual $\mathbf{u}^{(t)} \in \mathbb{R}^m$, du vecteur $\mathbf{A}^\mathrm{T}\mathbf{u}^{(t)}$ ainsi que de la valeur de $\mathrm{D}^\nu(\mathbf{u}^{(t)})$.

Pour implémenter les méthodes d'accélération, nous proposons d'utiliser le point dual $\mathbf{u}^{(t)} = -\nabla f(\mathbf{A}\mathbf{x}^{(t)})$ où $\mathbf{x}^{(t)}$ est l'itéré généré par la méthode choisie pour résoudre $(\mathcal{R}^\nu)$. Premièrement, la suite $\{\mathbf{u}^{(t)}\}_{t \in \mathbb{N}}$ converge vers la solution de $(\mathcal{D}^\nu)$ de part les conditions d'optimalité énoncées dans le Thm. 1. Ceci suggère que les tests (2a)-(2b) vont permettre de fixer de plus en plus d'éléments lors de la résolution de $(\mathcal{R}^\nu)$ mais également que le test d'élagage précoce (3) ainsi que les tests (4a)-(4b) seront plus enclins à passer. Deuxièmement, les valeurs de $\mathbf{u}^{(t)}$ et $\mathbf{A}^\mathrm{T}\mathbf{u}^{(t)}$ son déjà calculées par la majorité des algorithmes de résolution adaptés au problème $(\mathcal{R}^\nu)$. On peut donc les récupérer sans frais. Enfin, la valeur de $\mathrm{D}^\nu(\mathbf{u}^{(t)})$ est déjà calculée si l'on décide de stopper la résolution de $(\mathcal{R}^\nu)$ lorsque la différence entre l'objectif primal et dual passe sous un certain seuil. Si ce n'est pas le cas, on peut de toute manière calculer $\mathrm{D}^\nu(\mathbf{u}^{(t)})$ à partir de $\mathbf{u}^{(t)}$ et $\mathbf{A}^\mathrm{T}\mathbf{u}^{(t)}$ avec une complexité $\mathcal{O}(n+m)$. Ceci illustre bien le fait que les méthodes d'accélérations proposées dans cette sections peuvent être appliquées avec un coût négligeable.

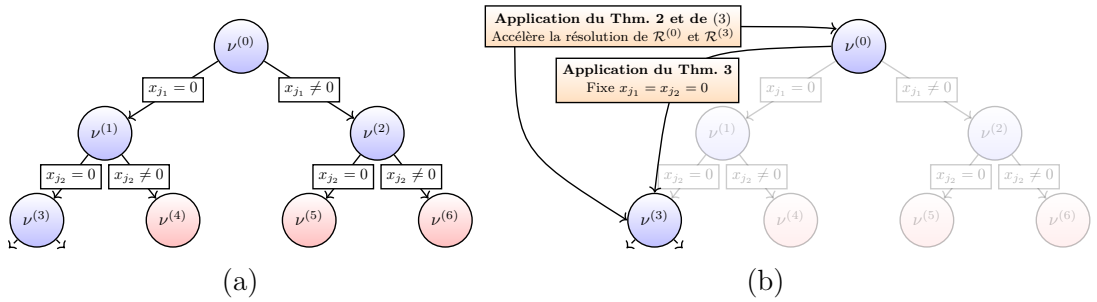


FIG. 1: (a) Exemple d'arbre de décision présenté dans la Sec. 2. Les nœuds rouges sont élagués et l'exploration se poursuit sous $\nu^{(3)}$. (b) Impact des accélérations présentées dans la Sec. 3. Au nœud $\nu^{(0)}$ on arrive à fixer $x_{j_1} = x_{j_2} = 0$ grâce au Thm. 3. On descend directement à $\nu^{(3)}$ en évitant de traiter les nœuds transparents et donc de résoudre leur relaxation. De plus, les méthodes présentées dans la Sec. 3.1 permettent d'accélérer la résolution de $\mathcal{R}^{(0)}$ et $\mathcal{R}^{(3)}$.

[2]On suppose que les tests (2a)-(2b) sont implémentés suivant la méthode "GAP" qui construit une sphère $\mathcal{S}(\mathbf{c}, r)$ avec $\mathbf{c} = \mathbf{A}^\mathrm{T}\mathbf{u}^{(t)}$ et $r = \sqrt{2L(\mathrm{R}^\nu(\mathbf{x}^{(t)}) - \mathrm{D}^\nu(\mathbf{u}^{(t)}))}$ où $\mathbf{u}^{(t)}$ est le point dual défini ci-dessus.

4 Résultats numériques

Dans cette dernière section, on montre que notre méthode de BnB permet de résoudre $(\mathcal{P})$ efficacement. On compare ses performances, avec ou sans les accélérations présentées dans la Sec. 3, à celles de solveurs génériques de PMNL. On considère des problèmes de *regression* parcimonieuse avec $f(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - \mathbf{w}\|_2^2$ ainsi que des problèmes de *classification* parcimonieuse avec $f(\mathbf{w}) = \mathbf{1}^{\mathrm{T}} \exp(\mathbf{1} + \log(\mathbf{y} \odot \mathbf{w}))$ où $\odot$ est le produit d'Hadamard et où les fonctions "exp" et "log" sont prises terme à terme. Notre algorithme de BnB est accessible via le package

`https://github.com/TheoGuyard/El0ps.jl`

et les solveurs de PMNL que nous considérons sont `Cplex`, `Mosek` et `Scip`.[3]

4.1 Données synthétiques

Premièrement, on considère des données synthétiques de *regression* afin de tester l'influence de différents paramètres sur le comportement des méthodes de résolution considérées. Pour générer des instances du problème, on commence par construire un vecteur parcimonieux $\mathbf{x}^\dagger$ avec k entrées non-nulles espacées uniformément et avec une valeur tirée dans $\{-1, +1\}$ aléatoirement. Ensuite, on construit $\mathbf{A} \in \mathbb{R}^{m \times n}$ avec des lignes générées suivant une loi $\mathcal{N}(\mathbf{0}, \Sigma_\rho)$ où la matrice de corrélation Σ_ρ est choisie avec des entrées $\Sigma_\rho(i, j) = \rho^{|i-j|}$. Le paramètre $\rho \in [0, 1]$ permet de contrôler la corrélation entre les colonnes de $\mathbf{A}$. On génère $\mathbf{y} = \mathbf{A}\mathbf{x}^\dagger + \boldsymbol{\epsilon}$ où $\boldsymbol{\epsilon}$ est un bruit blanc Gaussien avec un SNR[4] égal à τ. Enfin, on calibre les paramètres λ et M en utilisant le package `L0Learn`, l'objectif étant de retrouver $\mathbf{x}^\dagger$ à partir des données $\mathbf{A}$ et $\mathbf{y}$ en résolvant $(\mathcal{P})$. En général, augmenter k, m, n, ρ ou diminuer τ mène à des instances du problème plus compliquées.

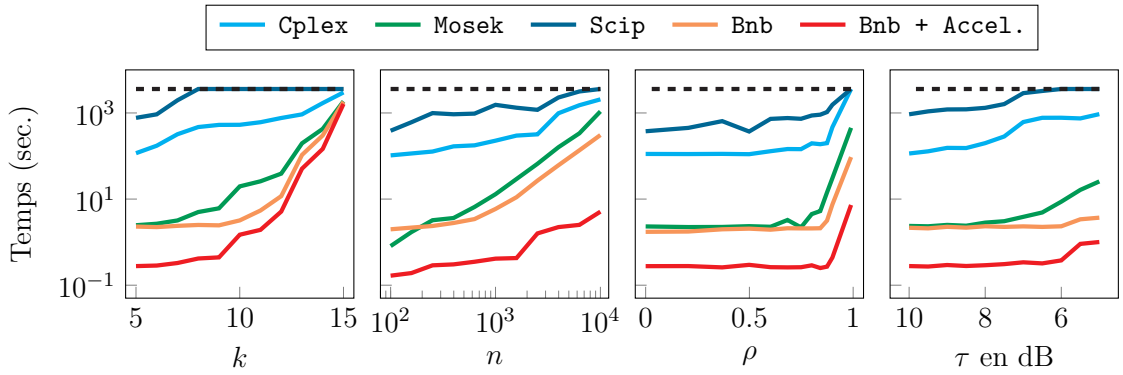


FIG. 2: Performances de différentes méthodes de résolution sur des données synthétiques. Temps maximum autorisé : 3600 secondes (ligne pointillée noire).

Dans la Fig. 2, on génère des instances synthétiques de $(\mathcal{P})$ en fixant $k = 5$, $m = 100$, $n = 200$, $\rho = 0.1$ et $\tau = 10$dB. On fait ensuite varier ces paramètres un à un et on observe le temps de résolution des différentes méthodes considérées. Premièrement, on remarque bien l'influence des paramètres sur la difficulté du problème. Aussi, on note que l'implémentation sans accélération de notre algorithme de BnB semble déjà compétitive au regard des solveurs génériques de PMNL. L'ajout des méthodes proposées dans la Sec. 3 permettent un facteur d'accélération significatif. De manière globale, on peut espérer au minimum un ordre de grandeur d'accélération par rapport aux solveurs génériques de PMNL. Ces gains peuvent même aller jusqu'à quatre ordres de grandeur (accélération $\times 10,000$) contre certains des solveurs de PMNL qui ne semblent pas du tout adaptés pour résoudre $(\mathcal{P})$. Ceci illustre bien la nécessité d'exploiter la structure de $(\mathcal{P})$ si l'on veut résoudre ce problème efficacement.

[3] Les problèmes de classification parcimonieuse sont modélisés à l'aide de contraintes coniques non supportées par `Cplex` et `Scip`. On compare donc uniquement notre BnB à `Mosek` dans ce cas.

[4] Le "Signal-to-Noise Ratio" correspond au ratio $\tau = \|\mathbf{A}\mathbf{x}^\dagger\|_2^2 / \|\boldsymbol{\epsilon}\|_2^2$ et est souvent exprimé en décibel.

4.2 Données réelles

Maintenant, on considère des jeux de données réelles de régression (`pyrim` et `triazines`) et de classification (`ionosphere` et `a1a`) tirés de `LIBSVM` et choisis avec des difficultés et des tailles d'instance différentes. On résout $(\mathcal{P})$ sur une grille de valeurs $\lambda \in [0.01, 1] \times \lambda_{\max}$ où $\lambda_{\max}$ est choisi tel que l'unique solution de $(\mathcal{P})$ soit $\mathbf{x}^\star = \mathbf{0}$. Lorsque λ diminue, la solution de $(\mathcal{P})$ devient moins parcimonieuse mais permet de décomposer $\mathbf{y}$ de manière plus fidèle. En pratique, cela permet de choisir la solution la plus pertinente parmi toutes celles calculées.

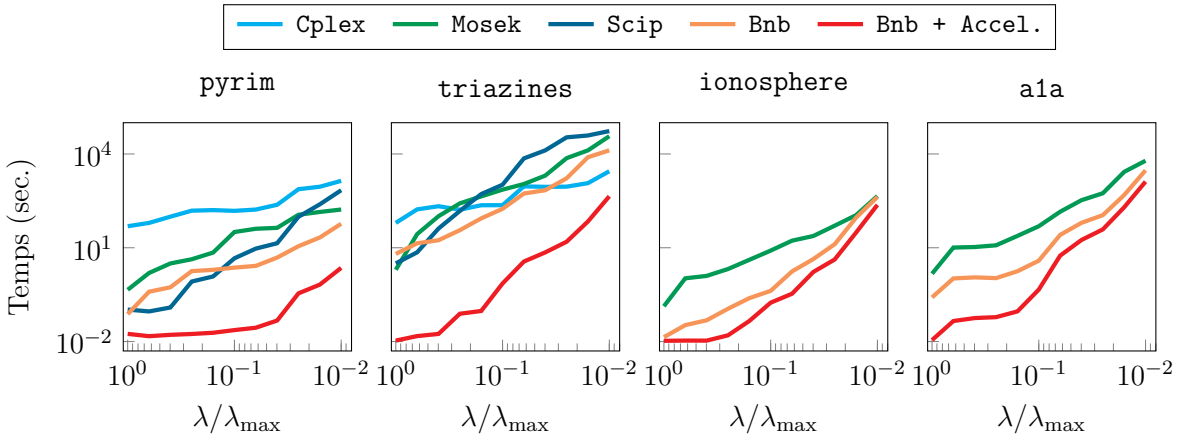


FIG. 3: Performances de différentes méthodes de résolution sur des jeux de données réelles.

Dans la Fig. 3, on remarque premièrement que la difficulté du problème augmente lorsque λ diminue. En effet, un λ plus petit mène à des solutions moins parcimonieuses, ce qui nécessite généralement un plus grand déploiement de l'arbre de décision du BnB. Comme pour les données synthétiques, l'algorithme de BnB a des performances comparables à celles des solveurs génériques de PMNL. Cependant, ajouter les accélérations de la Sec. 3 permet un gain de temps non négligeable. Celui-ci atteint jusqu'à quatre ordres de grandeur (accélération $\times 10,000$) sur les problèmes de regression et atteint jusqu'à trois ordres de grandeur (accélération $\times 1,000$) sur les problèmes de classification comparé aux solveur génériques de PMNL.

5 Conclusion

Dans cet article, nous avons montré comment tirer partie de la structure parcimonieuse de $(\mathcal{P})$ afin de construire une algorithme de résolution efficace. Nous fournissons une implémentation de notre méthode via le package `El0ps.jl`. Nos simulations numériques suggèrent que notre algorithme de BnB permet traiter des instances qui étaient jusque-là hors d'atteinte en un temps raisonnable pour des solveurs génériques de PMNL.

References

[1] J.A. Tropp and S.J Wright. "Computational methods for sparse solution of linear inverse problems". In: *Proceedings of the IEEE* 98.6 (2010), pp. 948–958.

[2] D. Bertsimas, A. King, and R. Mazumder. "Best subset selection via a modern optimization lens". In: *The annals of statistics* 44.2 (2016), pp. 813–852.

[3] Z. J. Xiang, Y. Wang, and P. J. Ramadge. "Screening tests for lasso problems". In: *IEEE transactions on pattern analysis and machine intelligence* 39.5 (2016), pp. 1008–1027.

[4] E. Ndiaye et al. "Gap safe screening rules for sparsity enforcing penalties". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 4671–4703.

[5] G. Samain, S. Bourguignon, and 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*. 2022.