Rapport du projet PACE 2025

Hugo Barreiro M1 MPRI
Yoshimi-Théophile Étienne M1 MPRI

28 Mars 2025

Table des matières

1 Introduction

1.1 Avant-propos

Nous tenons à remercier *Jean-Christophe Filliâtre* pour nous avoir gentiment donné son implémentation des tableaux persistants en *OCaml*. L'archive du code est disponible sur GitHub.

Nous allons vous présenter notre algorithme pour résoudre le problème du *Dominating Set* de façon exacte. Nous avons comme hypothèse que les graphes traités sont planaires et particulièrement *sparse* (voir la prochaine sous-section).

Notre première idée a été de trouver des bornes sur la taille du *Dominating Set*. Ainsi, au lieu d'explorer les 2^n possibles solutions (avec n le nombre de nœuds du graphe), on pourrait alors se contenter d'un sous-ensemble bien plus petit. Nous avons donc recherché dans la littérature et nous nous sommes arrêtés sur le papier [Hen22].

Cela n'étant pas suffisant, on s'est alors demandé si trouver un *Dominating Set* de taille k serait un problème plus facile que de trouver le plus petit de taille au plus n (avec $k \leq n$). L'idée était alors d'effectuer une recherche dichotomique sur l'intervalle possible pour la taille du *Dominating Set*.

Néanmoins, cela n'est pas très réaliste pour un n trop grand puisque chaque k serait alors également trop grand (temps de calcul non négligeable pour chaque k). Notre nouvelle idée était alors de trouver le plus petit *Dominating Set* de taille au plus k. Ainsi, on pourrait lancer en parallèle chaque recherche avec un k différent uniformément réparti dans l'espace de recherche. Cela fonctionne puisque le premier appel à renvoyer une solution correcte renvoie également la solution optimale.

Enfin, notre dernière idée est de simplifier notre graphe à chaque fois qu'on ajoute un nœud dans la solution en cours. On peut voir cela comme si on appliquait un *pré-processing* à chaque tour. Ainsi, on peut espérer réduire la difficulté du problème au fur et à mesure. Nous nous sommes grandement inspirés du papier [Ste04].

Nous allons détailler chaque partie de l'algorithme ainsi que sa *correctness* dans les parties suivantes.

1.2 Problem Set

Les instances fournies par PACE pour le problème du *Dominating Set* présentent quelques avantages cruciaux comparés à des graphes quelconques.

En particulier, les graphes sont planaires et très *sparse*, c'est-à-dire qu'ils contiennent un nombre d'arêtes très réduit. En général, nous nous attendons à un nombre d'arêtes compris entre $\frac{3}{2}$ et 2 fois la taille du graphe.

Par conséquent, les degrés des nœuds dans le graphe sont très faibles, et le degré

moyen d'un nœud est compris entre 3 et 4. De manière cruciale, ce nombre reste plus ou moins constant quelle que soit la taille du graphe, une propriété que nous utiliserons à notre avantage.

Les graphes auxquels nous nous intéressons sont des graphes simples et non dirigés.

1.3 Notations

Nous utiliserons dans les sections qui suivent quelques notations pour les graphes.

Pour un graphe quelconque G, n indique la taille de G. $v \in G$ indique que le nœud v appartient au graphe G. Pour toute arête $e = \{u, v\}$, $e \in G$ signifie que les nœuds u et v sont connectés dans G.

Pour tout $v \in G$, $N(v)$ désigne l'ensemble des voisins de v, et $N[v]$ désigne ce même ensemble incluant v. Comme nous traitons des graphes simples, nous pouvons supposer que $N(v) \neq N[v]$ pour tout v.

Nous noterons le degré d'un nœud par $\delta(v) = |N(v)|$, ainsi que le degré maximal du graphe par $\Delta(G) = \max\{\delta(v), v \in G\}$.

Dans la suite du rapport, nous appellerons le problème qui consiste à trouver un *Dominating Set* minimal de taille inférieure ou égale à k, le problème *k-Dominating Set*.

2 Bornes

2.1 Borne inférieure

La seule borne inférieure qu'on a trouvée dans la littérature est la borne triviale :

$$\frac{n}{1 + \Delta(G)}$$

avec n le nombre de nœuds du graphe et $\Delta(G)$ le degré maximal du graphe G. Cette borne permettait déjà de réduire considérablement la taille minimale du *Dominating Set*, néanmoins on a trouvé une solution permettant d'obtenir une meilleure borne inférieure.

Elle reprend la même idée mais en effectuant une estimation plus fine. Pour ce faire, on trie de manière décroissante nos nœuds par rapport à leur degré. Puis, on ajoute le degré du premier nœud au compteur. Si le compteur est plus petit que k alors on passe au nœud suivant, et ainsi de suite. En d'autres termes, on cherche le plus petit k tel que k permet de dominer un nombre maximum de nœuds d avec $d \geq n$ et n le nombre de nœuds du graphe.

2.2 Borne supérieure

Les bornes supérieures sont tirées du papier [Hen22]. Tout d'abord, on considère la borne triviale

$$n - \Delta(G)$$

avec n le nombre de nœuds de G et $\Delta(G)$ le degré maximal de G. Cette borne n'est vraiment pas précise, on peut alors en considérer d'autres.

Le papier présente de meilleures bornes pour des graphes ayant un degré minimal supérieur ou égal à d avec $2 \leq d \leq 6$. Ces bornes sont des spécialisations plus précises de la formule générale

$$\frac{n}{\delta + 1} \times \sum_{j=1}^{\delta+1} \frac{1}{j}$$

avec $\delta \geq 1$ le degré minimal du graphe et n le nombre de nœuds du graphe.

Les bornes étaient assez satisfaisantes mais on a trouvé une meilleure borne supérieure. Ainsi, nous avons implémenté un algorithme glouton pour calculer une approximation du *Dominating Set* de notre graphe. L'algorithme est le suivant : on prend le nœud de degré maximal, on l'ajoute lui et ses voisins dans la solution, ensuite on retire toutes les arêtes entre ce nœud et ses voisins, puis on recommence tant que l'on n'a pas dominé tous les nœuds.

2.3 Efficacité

On présente ici les résultats obtenus pour nos bornes sur les instances fournies par PACE :

Nombre nœuds	Min degré	Max degré	Min dom	Max dom
7	2	3	2	2
32	2	6	6	10
63	2	4	13	22
109	2	4	22	34
164	1	4	33	51
216	1	6	43	68
270	1	7	53	89
311	1	7	61	101
8340	2	30	683	2448

On peut remarquer que la taille maximum du *Dominating Set* trouvé avec nos bornes est environ 3 fois plus petite que le nombre de nœuds du graphe. De plus, pour la taille minimum du *Dominating Set*, elle est selon les graphes, environ entre 1.5 et 3 plus petite que la taille maximum du *Dominating Set*.

3 Algorithme

3.1 Structure

L'algorithme de recherche pour le *k-Dominating Set* prend la forme d'un algorithme de parcours en profondeur.

En particulier, cet algorithme utilise des graphes où les nœuds sont coloriés de deux couleurs : en noir lorsque le nœud n'est pas dominé, et en blanc lorsqu'il est dominé. À chaque itération, nous pouvons affirmer que pour un nœud v quelconque qui n'est pas encore dominé, un nœud dans $N[v]$ appartient au *k-Dominating Set*. Cette observation nous permet d'itérer récursivement sur chacune de ces possibilités et de colorier en blanc les nœuds nouvellement dominés.

La structure de l'algorithme repose donc sur une fonction récursive qui prend en argument un graphe, une borne k et une solution temporaire, et retourne soit l'absence de solution, soit, si la solution temporaire aboutit à un *dominating set* de taille inférieure ou égale à la borne, la solution minimale possible.

3.2 Condition d'arrêt

L'algorithme requiert deux conditions d'arrêt : l'une lorsque l'on trouve une solution, et l'autre lorsqu'aucune solution n'existe.

Le premier cas est trivial à vérifier : on considère que l'on a trouvé une solution si le graphe ne contient plus de nœuds noirs, c'est-à-dire que tous les nœuds sont dominés. Cette condition est correcte tant que nous colorions en blanc les nœuds que nous dominons.

Pour le second cas, une condition d'arrêt triviale consiste à vérifier si l'on a atteint le seuil fixé pour la taille du *k-Dominating Set*. Autrement dit, si l'on cherche une solution de taille inférieure ou égale à k, on peut conclure à l'absence de solution lorsqu'une solution temporaire de taille k ne domine pas l'ensemble du graphe.

Cependant, on peut affiner cette condition. En effet, nous pouvons utiliser le même algorithme que celui servant à déterminer une borne inférieure pour la taille minimale d'un *Dominating Set*. Cet algorithme permet d'estimer précisément le nombre maximal de nœuds que l'on peut encore dominer en ajoutant k' nœuds au *k-Dominating Set*. Le calcul de cette borne s'effectue en temps $O(\Delta(G))$.

3.3 Appels récursifs

À chaque itération de notre recherche, nous choisissons un nouveau nœud v et appelons récursivement la recherche pour l'ajout de chaque nœud de $N[v]$ à la solution temporaire.

Il est donc essentiel de sélectionner, à chaque itération, un nœud de degré minimal afin de minimiser le nombre d'appels récursifs. En pratique, dans notre implémentation, nous recherchons un nœud de degré au moins égal à 2, car nous supposons que tous les nœuds de degré 1 ont déjà été éliminés (voir la section suivante sur les réductions). Cette recherche s'effectue en temps $O(n)$.

Alors que l'algorithme présenté dans [Ste04] se contente de trouver une solution quelconque de taille inférieure à k, notre approche vise à identifier une solution minimale. Pour cette raison, nous explorons tous les appels récursifs, les comparons, puis sélectionnons la solution minimale, plutôt que de nous arrêter dès que nous en trouvons une.

Pour améliorer les performances, si un appel récursif trouve une solution, nous réduisons la borne k afin de chercher dorénavant une solution strictement plus petite que la dernière trouvée. Cela nous permet d'éliminer plus de cas et d'éviter de comparer directement les solutions : toute nouvelle solution trouvée sera nécessairement plus petite que la précédente.

4 Réductions

4.1 Règles de Réduction

L'algorithme, jusqu'à présent, a une complexité en $O(n\Delta(G)^k)$. Pour améliorer ses performances, nous effectuons un *pré-processing* sur le graphe à chaque itération de notre recherche en profondeur. Pour cela, nous utilisons sept règles de réduction.

L'intérêt de ces règles de réduction (à l'exception de la règle 3) est de réduire le graphe (en supprimant des arêtes et des nœuds) tout en garantissant que le *Dominating Set* du graphe original et celui du graphe réduit (après application exhaustive des réductions) restent identiques. Les preuves de ces propriétés sont données dans [Ste04].

La motivation initiale de ces règles de réduction est de garantir qu'après réduction du graphe, il est toujours possible de trouver un nœud sur lequel effectuer la recherche du *k-Dominating Set* récursivement, avec un degré inférieur ou égal à 7. Cela permet aux auteurs de [Ste04] de donner une complexité pour l'algorithme *k-Dominating Set* de l'ordre de $O(n8^k)$.

La règle 3 est particulière, car elle modifie le *Dominating Set* : elle est utilisée lorsqu'il est certain que l'ajout d'un nœud v au *Dominating Set* est optimal. Par "optimal", nous entendons que, parmi l'ensemble des *Dominating Sets* obtenus en ajoutant des nœuds à une solution partielle, il existe forcément une solution minimale contenant v. Bien que la complexité théorique de l'algorithme reste inchangée avec cette règle, en pratique, elle accélère considérablement l'exécution.

4.1.1 Règles de Réduction 1, 2, 4, 5, 6, 7

Les règles de réduction qui préservent le *Dominating Set* sont les suivantes :
— **Règle 1** : Supprimer toutes les arêtes reliant deux nœuds blancs.

— **Règle 2** : Supprimer tous les nœuds blancs w tels que $\delta(w) = 1$.
— **Règle 4** : Pour un nœud blanc w tel que $\delta(w) = 2$ et $N(w) = \{u_1, u_2\}$, supprimer w si $\{u_1, u_2\} \in G$.
— **Règle 5** : Pour un nœud blanc w tel que $\delta(w) = 2$ et $N(w) = \{u_1, u_3\}$, supprimer w s'il existe un nœud noir u_2 tel que $\{u_1, u_2\} \in G$ et $\{u_2, u_3\} \in G$.
— **Règle 6** : Pour un nœud blanc w tel que $\delta(w) = 2$ et $N(w) = \{u_1, u_3\}$, supprimer w s'il existe un nœud blanc $u_2 \neq w$ tel que $\{u_1, u_2\} \in G$ et $\{u_2, u_3\} \in G$.
— **Règle 7** : Pour un nœud blanc w tel que $\delta(w) = 3$ et $N(w) = \{u_1, u_2, u_3\}$, supprimer w si au moins deux arêtes du triangle $\{u_1, u_2, u_3\}$ figurent dans G.

En réalité, les règles 2, 4, 5, 6 et 7 nécessitent également que les voisins de w soient noirs, mais cette condition est implicite, car nous supposons que la règle 1 a été appliquée de manière exhaustive avant.

Dans notre implémentation, "supprimer un nœud v de G" signifie que nous supprimons toutes les arêtes incidentes à v : ainsi, la taille n du graphe G ne change jamais.

4.1.2 Règle 3

La règle 3 permet de réduire le graphe lorsqu'il contient des nœuds noirs b tels que $\delta(b) = 1$ (c'est-à-dire lorsque b est une feuille). Intuitivement, cette règle simplifie les sous-graphes de G qui forment des arbres enracinés dans G.

Si b est une feuille, alors on peut ajouter son voisin u au *Dominating Set*. Cette décision est justifiée, car soit b, soit u appartient au *Dominating Set* (sinon, b ne serait pas dominé). Or, comme $N[b]$ est inclus dans $N[u]$, u est toujours un meilleur choix.

On peut généraliser ce raisonnement : considérons $\{v_1, \ ... \ , v_p\}$ formant une clique de p nœuds dans G pour un $n \geq p > 1$. S'il existe un $i < p$ tel que v_i est noir, mais que $\delta(v_1) = \ ... \ = \delta(v_{p-1}) = p - 1$, alors on peut ajouter v_p au *Dominating Set*. Le cas précédent correspond au cas $p = 2$.

Comme nous travaillons sur des graphes planaires, les cas $p > 4$ sont impossibles. Nous éliminons également le cas $p = 4$, qui semble très rare, étant donné que les degrés moyens des graphes qui nous intéressent sont inférieurs à 4.

Nous avons donc ajouté une règle 3 bis pour le cas $p = 3$, qui stipule que si $\{v_1, v_2, v_3\}$ forme un triangle, mais que deux des trois nœuds du triangle ont un degré de 2, alors on peut ajouter le troisième au *Dominating Set*.

En pratique, si les autres règles de réduction sont appliquées de manière exhaustive, il n'est pas nécessaire de vérifier la couleur des nœuds, car le triangle ne peut contenir que 0 ou 1 nœud blanc. Sinon, la règle 1 supprimera une arête du triangle.

4.2 Complexité

Dans [Ste04], la complexité des réductions est donnée comme étant linéaire en la taille du graphe (en réalité, les règles 5 et 6 sont modifiées pour permettre cette complexité). En pratique, il est possible d'appliquer ces réductions de manière exhaustive en un temps polynomial en $\Delta(G)$, ce qui, dans notre cas, est préférable, puisque $\Delta(G)$ est très petit.

4.2.1 Règle 1

À chaque itération de notre recherche, on choisit un v à ajouter à notre solution temporaire du *k-Dominating Set*, et on colorie $u \in N[v]$ en blanc.

Si l'on suppose que, dans toutes les itérations précédentes, la règle 1 a été appliquée de manière exhaustive, alors à cette itération, la règle 1 ne concerne que les arêtes issues des nœuds de $N[v]$.

Il y a au plus $O(\Delta(G)^2)$ de telles arêtes, ce qui nous donne une complexité pour la règle 1 de $O(\Delta(G)^2 \log \Delta(G))$, car notre implémentation nécessite $O(\log \Delta(G))$ pour supprimer une arête.

4.2.2 Règles 2, 4, 5, 6 et 7

De manière similaire, les seuls nœuds blancs à considérer à chaque itération pour ces règles sont ceux dont une arête a été supprimée depuis la dernière itération. Ce sont donc ceux affectés par la règle 1, et nous savons qu'il y a $O(\Delta(G)^2)$ de tels nœuds.

Parmi ces règles de réduction, les règles 5 et 6 sont les plus complexes : elles nécessitent $O(\Delta(G))$ vérifications pour vérifier l'existence d'une arête entre $u_2 \in N(u_1) \backslash \{w\}$ et u_3.

Les autres règles de réduction nécessitent seulement $O(1)$ vérifications.

On remarque également qu'un seul passage sur les nœuds concernés suffit pour appliquer les règles de manière exhaustive. Cela nous donne une complexité pour ces règles de réduction de $O(\Delta(G)^3 \log \Delta(G)^2)$.

4.2.3 Règle 3

De la même manière, seuls les nœuds noirs dont une arête a été supprimée par les règles précédentes nous intéressent. Il y a $O(3\Delta(G)^2) = O(\Delta(G)^2)$ de tels nœuds.

Cela nous donne une complexité pour cette règle de réduction de $O(\Delta(G)^2 \log \Delta(G)^2)$.

5 Parallélisme

On va présenter plus en détails notre utilisation du parallélisme. Tout d'abord, comme dit en intro, nous utilisons le parallélisme pour lancer en parallèle plusieurs calculs du *Dominating Set* avec des k uniformément répartis dans l'espace des solutions.

En réalité, c'est sensiblement plus complexe. En effet, il peut arriver qu'en séparant uniformément les k, en partant de la borne supérieure, on dépasse la borne inférieure. Dans ce cas, pour le nombre de core encore non-utilisés on part des k les plus grands et on en ajoute des nouveaux au milieu de ces k. Par exemple, si on a 8 cores et que l'on dispose des k suivants $2, 4, 6, 8, 10, 12$ alors on ajoute 9 et 11 pour obtenir $2, 4, 6, 8, 9, 10, 11, 12$.

Ensuite, dans une première version, dès qu'un calcul terminait, si on avait trouvé un résultat, alors on savait qu'il était optimal et on pouvait renvoyer ce dernier, sinon, on arrêtait le core puis, attendait qu'un core renvoie un résultat valide. Ce n'est plus le cas maintenant.

À présent, on garde en mémoire les k avec lesquels on a commencé à rechercher un *Dominating Set* en plus d'un k minimal. Par exemple, si on lance avec $2, 4, 6, 8$ alors, peut chercher dans un encadrement [0, 2] pour le premier core, [2, 4] pour le deuxième, et ainsi de suite. Cela nous permet d'arrêter un core si la solution globale minimale est strictement plus petite que le k minimal de notre intervalle.

Une autre action que l'on effectue est de relancer avec un intervalle $[k, k+1]$ si on n'a rien trouvé. Cela peut s'avérer utile par exemple si on lance avec $5, 10, 15, 20$ et que la solution optimale est à $k = 6$. Il est peut-être plus rapide d'échouer avec $k = 5$ puis, relancer avec $k = 6$ que d'attendre que le $k = 10$ trouve la solution.

Ce qu'on fait en réalité est un peu plus subtil puisque l'on relance avec comme k minimal le plus grand k maximal avec lequel nous n'avons rien trouvé et comme k maximal le plus petit k strictement supérieur au plus grand k maximal avec lequel nous n'avons rien trouvé et que nous n'avons jamais considéré jusqu'à présent.

De plus, si notre nouveau k avec lequel on veut relancer est plus grand que la meilleure solution actuelle, alors on arrête le core au lieu de repartir dans des calculs.

6 Benchmarking

Pour ce benchmark, nous utilisons un processeur *Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz* à 8 cœurs. Nous testons notre implémentation sur les instances fournies par PACE. Voici le récapitulatif des performances :

Nom instance	Nombre nœuds	Nombre arêtes	Taille solution	Temps de calcul
test	7	9	2	instantané
bremen20	32	48	9	instantané
bremen50	63	98	17	instantané
bremen100	109	173	29	2min30
bremen150	164	259	42	11min
bremen200	216	338	57	inconnu

On peut expliquer pourquoi le *bremen*200 ne termine pas rapidement. Un core commence l'exécution avec $k = 56$. Il doit d'abord constater qu'aucune solution n'existe pour ce k avant de passer à $k = 57$. Or, le temps de calcul pour effectuer ces deux passes était trop long pour qu'on puisse le calculer. En effet, au bout de 3 heures, le calcul n'était toujours pas terminé.

Ainsi, on peut conclure que nos réductions ne sont pas assez efficaces pour un k trop grand et qu'on se fait rattraper par la dimension exponentielle du problème, qu'on semblait contourner jusqu'alors en obtenant des performances plutôt encourageantes.

7 Conclusion

Pour conclure, nous pouvons dire que les différentes avenues que nous avons explorées pour ce projet ont abouti à de nombreux résultats, avec un réel effet sur le temps d'exécution de notre algorithme. Nous tirons grandement parti de la structure du problème qui nous est donné, ainsi que de la théorie existante traitant des problèmes *Dominating Set* sur les graphes planaires, afin de trouver une solution exacte sur des graphes de taille allant jusqu'à 150 nœuds en un temps raisonnable.

Cependant, de nombreuses optimisations restent nécessaires afin de pouvoir résoudre le problème *Dominating Set* sur des graphes plus grands. Nous listons dans cette section quelques idées d'améliorations qui n'ont pas été implémentées, soit par manque de temps, soit en raison d'une incertitude quant à leur efficacité.

7.1 Bornes

De meilleures bornes, surtout une meilleure borne supérieure, seraient intéressantes à considérer pour rétrécir l'intervalle de recherche du *Dominating Set*.

Cette borne supérieure peut être trouvée avec l'aide de meilleurs algorithmes approximatifs pour le *Dominating Set*, qui remplaceraient l'algorithme glouton que nous utilisons à présent.

7.2 Algorithme

7.2.1 Condition d'arrêt

La condition d'arrêt que nous utilisons prend en compte le degré de chaque nœud pour déterminer le nombre maximum de nœuds que nous pouvons dominer si nous ne

choisissons que les nœuds à plus grand degré. Par exemple, deux nœuds de degré 3 et 4 respectivement peuvent dominer au plus 9 nœuds.

Cependant, cette condition peut encore être affinée, car le degré d'un nœud ne correspond pas forcément au nombre de nouveaux nœuds que ce nœud domine s'il est ajouté au *Dominating Set* : ces deux chiffres sont égaux pour un v si et seulement si v et ses voisins sont tous noirs.

Cependant, garder l'information pour cette nouvelle condition rend quelques opérations élémentaires sur les graphes plus compliquées (colorier un nœud en blanc requiert une mise à jour de tous ses voisins), ce qui empêche son implémentation.

7.2.2 Appels récursifs

Nous remarquons dans la section sur l'algorithme que l'ordre dans lequel nous itérons de façon récursive importe. En effet, si le premier appel récursif trouve une solution, il permet aux autres branches de chercher moins de cas (car on ne prend que les cas strictement meilleurs que la solution trouvée).

Il serait donc intéressant, lors de la récursion, de mieux ordonner les appels récursifs afin d'avoir une meilleure chance de trouver une solution quelconque plus rapidement.

Similairement, nous choisissons lors de notre appel récursif un nœud à degré minimal. Dans notre graphe, nous pouvons supposer que, l'intervalle des degrés possibles étant très petit, il existe un grand nombre de tels nœuds. Il serait alors intéressant de réfléchir à un choix parmi ces nœuds à degré minimal (par exemple, prendre le nœud à degré minimal dont les voisins ont un degré maximal).

7.3 Parallélisme

Des optimisations sont possibles dans l'implémentation du parallélisme, permettant un meilleur usage des instances parallèles qui ne servent plus : on pourrait envisager de meilleures communications entre les cœurs pour effectuer une recherche plus organisée. Notamment, à chaque solution trouvée, l'idéal serait une réallocation dynamique des cœurs cherchant une solution plus petite. Cependant, une telle réallocation garantissant une amélioration de l'algorithme n'est pas triviale à trouver.

Similairement, comme les réductions sont déterministes, chaque cœur en parallèle effectue les mêmes opérations (jusqu'à une certaine profondeur), créant une redondance qui pourrait en théorie être grandement réduite.

Références

[Hen22] Michael A. Henning. Bounds on domination parameters in graphs : A brief survey, 2022.

[Ste04] Jochen Alber ; Hongbing Fan ; Michael R.Fellows ; Henning Fernau ; Rolf Niedermeier ; Fran Rosamond ; Ulrike Stege. A refined search tree technique for dominating set on planar graphs, 2004.