8INF870 – Algorithmique

Présentation du problème du stable maximum

Fanny MONET	MONF10589604	
Mickaël REVILLON	REVM15119408	
Philippe ZANOLIN	ZANP21049609	01/04/2019

SOMMAIRE

Table des matières

I Présentation du problème

1 – Définitions préalables

Le problème du *stable maximum* s'inscrit dans le domaine de la théorie des graphes.

Un **graphe** est un couple (X,E) où X est un ensemble et E est un sous-ensemble de X x X.

Les éléments de X du graphe (X,E) sont appelés les **sommets** du graphe.

Dans un graphe G=(X,E), un sous-ensemble S de points de X est un **stable** si ses sommets ne sont pas adjacents entre eux.

Dans un graphe G=(X,E), un sous-ensemble stable S est **maximum** si sa cardinalité est supérieure ou égale à la cardinalité de tout autre stable de G.

Dans un graphe G=(X,E), un sous-ensemble stable S n'est pas **maximal** s'il est inclus strictement dans un autre stable.

Remarque : Tout stable maximum est maximal, mais la réciproque n'est pas vraie.

En théorie des graphes on dit que deux sommets d'un graphe non-orienté sont **voisins** ou s'ils sont reliés par une arête.

Le **degré** d'un sommet est la somme de ses voisins.

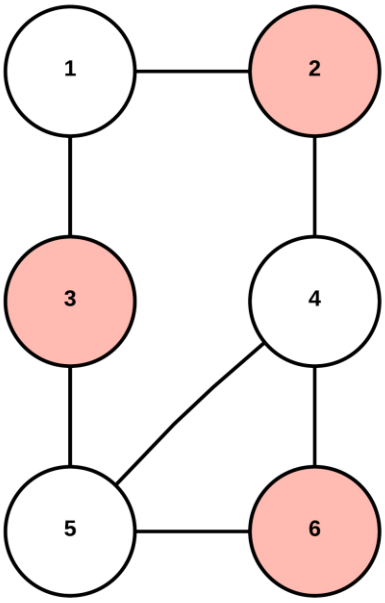
2 - Présentation

Le problème du **stable maximum** se pose dans un graphe que l'on peut considérer **simple** (graphe avec au plus un lien non orienté entre tout couple d'objets) puisque son éventuelle orientation ou la présence d'arêtes multiples ne jouent aucun rôle ici.

Trouver le stable maximum consiste à trouver dans un graphe le stable à la cardinalité la plus grande ⇔ trouver le plus grand sous ensemble de sommets non-adjacents.

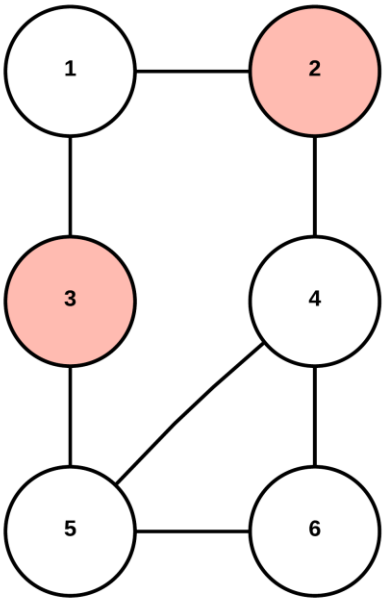
Le stable maximum peut être considéré comme le stable ayant la cardinalité la plus élevée parmi l'ensemble des stables maximaux d'un graphe.

Le stable maximum
(cardinalité 3)



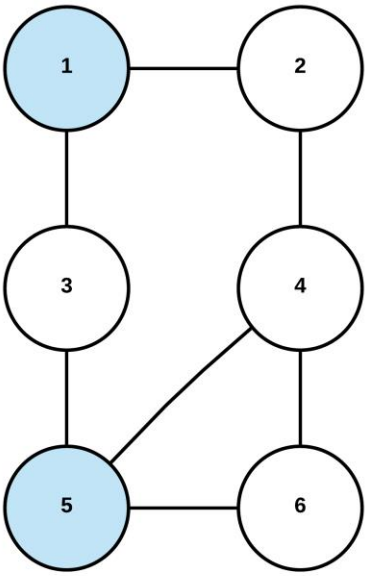
Il n'existe pas de stable avec une cardinalité supèrieure, il s'agit donc du maximum

Un stable simple
(cardinalité 2)



Ce stable est inclu dans le stable du schéma de gauche et n'est donc pas maximal

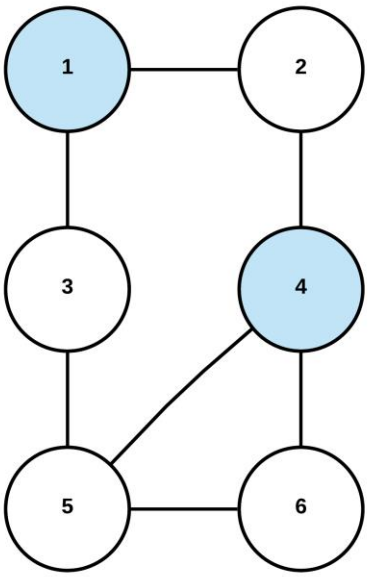
Un stable maximal
(cardinalité 2)



On ne peut pas inclure de nouveaux sommets dans l'ensemble déjà surligné, le stable est donc maximal.

Il ne s'agit pas du stable maximum car il existe au moins un stable de cardinalité supèrieure (cf 1er schéma)

Un stable maximal
(cardinalité 2)



Un stable différent du précédent, mais aussi maximal (de même cardinalité)

II Complexité et réductions

Le problème du stable maximum est un problème NPcomplet. En effet, il faut générer tous les stables maximaux d'un graphe afin de trouver celui avec la plus grande cardinalité, ce qui fait que la complexité n'est pas polynomiale. Le nombre d'ensemble des sommets générés augmentera d'autant plus que le nombre de sommets augmente. Néanmoins, sur de nombreuses classes de graphes, le problème est résolvable en temps polynomial.

III Critique de la littérature

Approximating Maximum Weighted Independent Set Using Vertex Support

Introduction

Cet article a été publié dans le *World Academy of Science, Engineering and Technology International Journal of Mathematical and Computational Sciences* par les auteurs S. Balaji, V. Swaminathan et K. Kannan. Il traite du problème de **stable de poids maximum**, qui est le cas général du stable maximum dans un graphe non orienté dont chaque sommet a un poids associé. Le stable maximum est un problème classique de la théorie des graphes, il a beaucoup d'applications comme les enchères combinatoires, la coloration de graphe, la théorie des codes, les faux diagnostics, la reconnaissance de pattern, la biologie moléculaire, et la bio-informatique.

Les auteurs définissent le stable de poids maximum (**MWIS** : Maximum Weighted Independent Set) d'un graphe (G = (V, E) avec V l'ensemble des sommets du graphe G et E l'ensemble des ponts) comme un ensemble de sommet S inclus dans V tel que la **somme des poids** des sommets de S est **maximum** et qu'**aucun couple de sommets de S n'est relié par un pont**. De plus, ils définissent la couverture de poids minimum par sommets (**MWVC** : Minimum Weighted Vertex Cover) comme un ensemble Vc de sommets de G à la **somme des poids minimum** tel que **pour chaque pont** de E, **au moins un des sommets du pont appartient à Vc**. Le raisonnement de sauteurs s'appuie sur le fait que le MWIS contient tous les sommets qui ne sont pas contenus dans le MWVC.

Objectif des travaux

L'objectif de ces travaux est de trouver un **algorithme de résolution approchée** du problème du stable de poids maximum à l'aide de la couverture de poids minimum par sommets du graphe. Les auteurs de l'article cherchent ensuite à comparer l'efficacité de leur algorithme (le SRA : **Support Ratio Algorithm**) avec ceux déjà existants dans la littérature scientifique en utilisant des graphes générés automatiquement à partir de deux méthodes, ainsi que des graphes spéciaux utilisés dans d'autres études.

Outils utilisés

L'algorithme SRA a été développé dans le langage C++ et les expériences ont été menées sur une machine Intel Pentium Core2 Duo 1.6 GHz CPU avec 1 GB de RAM.

Méthodologie

Les chercheurs ont effectué 3 expériences différentes, soit 117 instances en tout, pour évaluer l'efficacité de l'algorithme SRA.

Expérience 1 :

Pendant la première expérience, ils ont testé 44 instances de l'algorithme. Tout d'abord, ils l'ont testé sur 20 graphes générés automatiquement grâce au modèle G(n, p). Le modèle **G(n, p)** (aussi appelé Erdos Renyi random graph model) consiste à créer un graphe de **n sommets** et à parcourir l'ensemble des couples du graphe en **créant un pont avec une probabilité de p**. Les chercheurs ont choisi un p >> log(n)/n pour s'assurer que les graphes auto-générés soient presque toujours connectés. Le **poids** de chaque sommet a été sélectionné aléatoirement **entre 1 et 40**. Ils ont relevé les valeurs du **poids total** des stables trouvés ainsi que leur **cardinalité** pour chacun des graphes.

Ensuite, ils ont voulu tester le SRA sur **des graphes plus grands**, générés par la même méthode (avec la même intervalle de poids). Ils ont donc relevé, en plus du **poids total** et de la **cardinalité**, le **temps d'exécution** de l'algorithme puisque les solutions optimales n'étaient pas calculables facilement avec de plus grands graphes.

Expérience 2 :

Pendant la deuxième expérience, ils ont testé 53 instances du SRA sur des graphes de référence fournis par **DIMACS**. Les résultats de ces expériences sont ceux du clique maximum, qui est équivalent au stable maximum lorsqu'on utilise le graphe complémentaire. De plus, ils ont ajouté des poids aux sommets de ces graphes (aléatoirement entre 1 et 10). Ils ont relevé le **temps d'exécution** de l'algorithme ainsi que le **poids moyen** des sommets et la **cardinalité** de chaque stable. Ces résultats ont ensuite été comparés aux résultats expérimentaux reportés dans *Bomze et al 2000*, *Babel 1994* et *Pullan 2008*, en analysant le pourcentage de déviation de ces algorithmes par rapport au SRA.

Expérience 3 :

Pendant la troisième et dernière expérience, ils ont testé 20 instances du SRA sur des graphes générés grâce au modèle **G(n, m)**, avec un nombre n de sommets de 50 à 1000, dont les poids sont choisis entre 1 et 100. Ce modèle consiste à créer un graphe de n sommets et à générer m ponts de façon aléatoire entre les sommets. Le **temps** mis à trouver le stable de **poids maximum** ont été relevé pour chaque graphe et les chercheurs ont créé un graphique avec ce temps en en ordonnée et le nombre de sommets en abscisse.

Résultats

Expérience 1 :

Les chercheurs ont comparé le poids total et la cardinalité des stables au poids maximum résultant de l'algorithme SRA à ceux de la méthode Ostergard ainsi qu'aux solutions exactes. Les résultats de SRA sont meilleurs que ceux d'Ostergard, leur valeur différant du résultat optimal 1 seule fois contre 5 fois pour Ostergard. Avec des graphes plus grands, les chercheurs ont eu un maximum de temps d'exécution de 28 secondes.

Expérience 2 :

Les chercheurs ont comparé les résultats de leur algorithme sur les références DIMACS avec les algorithmes Babel, RD et PLS. Dans la majorité des cas, le SRA obtient des résultats bien plus proches de la solution optimale que les autres algorithmes.

Expérience 3 :

Le graphique montre que le temps pour trouver le MWIS augmente fortement lorsque le nombre de sommets augmente, avec un maximum de 12.31 secondes pour un graphe avec 1000 sommets.

Conclusion

L'algorithme SRA a donc prouvé son efficacité à trouver un stable au poids maximum optimal pour différents graphes, avec des résultats bien meilleurs que les algorithmes évoqués dans l'article et une vitesse d'exécution plus rapide.

ANR AGAPE Implémentation d'algorithmes exacts pour le problème du stable maximum

Introduction

Ce rapport, rédigé par Vincent Levorato de l'Université d'Orléans, LIFO, a pour but de comparer trois algorithmes exacts pour le problème de recherche d'un stable maximum dans un graphe, ainsi que la comparaison d'un algorithme sous différents langages (c++, java), pour des graphes aléatoires binomiaux allant de 5 à 100 sommets (par pas de 5) et des probabilités d'arêtes entre sommets de 0 à 1 (par pas de 0,1).

Algorithmes testés

Algorithme basé sur le travail de Moon and Moser

Cet algorithme énumère tous les ensembles stables maximaux d'un graphe et permet donc d'en déduire un plus grand (maximum). Sa complexité en temps d'exécution est bornée en O∗(1, 4423^n). Le facteur polynomial le plus élevé que l'on peut trouver dans l'algorithme est celui de la copie de graphe qui se fait en O(n + m).

Algorithme Degré Maximum

Cet algorithme utilise une règle de branchement qui, pendant les appels récursifs, vérifie s'il ne reste plus que des sommets de degré 1 ou 2 (cycle ou chemin), auquel cas on sait trouver le stable maximal en temps polynomial (approche gloutonne vue dans le cours par exemple). Il contient une méthode qui renvoie un ensemble stable maximal de G. Comme cette méthode n'est appliquée qu'à des cycles ou des chemins, il suffit de commencer par le sommet de plus petit degré et de prendre un sommet sur 2 comme appartenant au stable. Le facteur polynomial le plus élevé que l'on peut trouver dans l'algorithme est également celui de la copie de graphe (O(n+m)). La légère différence avec l'algorithme précédent est que l'on réalise 2 fois cette copie pour un appel récursif.

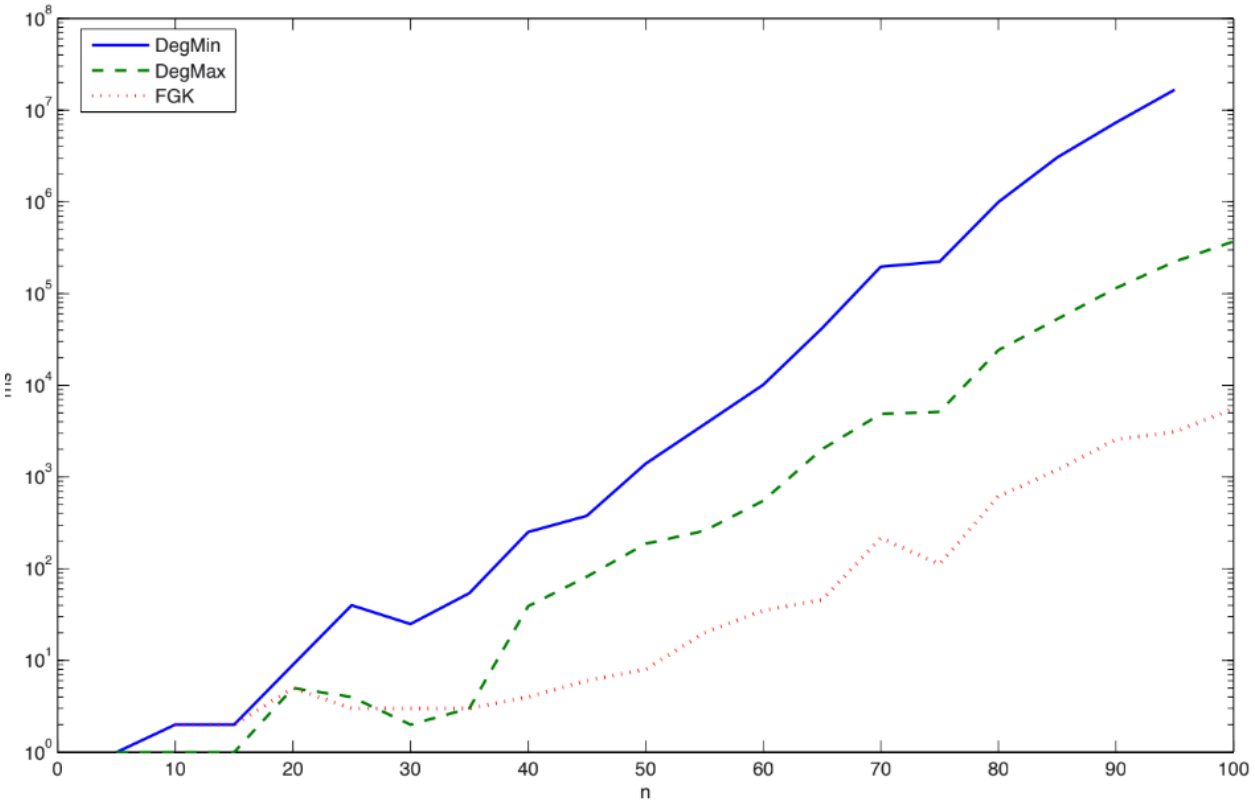
Algorithme Fomin, Grandoni and Kratsch (2009)

Cet algorithme utilise la méthode Mesurer pour conquérir pour établir le temps d'exécution au pire cas et permet de trouver un ensemble stable maximum dans un graphe avec une complexité en temps d'exécution en O∗(1, 2201^n). Celui-ci fait appel à certaines opérations sur les graphes comme la domination, le pliage de nœud (folding), ou les nœuds miroirs (mirroring). Comme l'algorithme original renvoie seulement la taille d'un stable maximum, elle a été modifiée afin de renvoyer le stable.

Résultats

La première partie compare les trois algorithmes.

Pour une probabilité p de 0,1 on obtient la courbe suivante :



L'axe des abscisses correspond aux nombre de sommets des graphes,

L'axe des ordonnées correspond aux temps d'exécution (en ms) avec une échelle logarithmique.

DegMin correspond à l'algorithme basé sur le travail de Moon and Moser.

DegMax correspond à l'algorithme Degré Maximum.

FGK correspond à l'algorithme Fomin, Grandoni and Kratsch.

On constate que l'algorithme DegMin est celui qui prend le plus de temps, que pour un nombre de sommets inférieur à 20 l'algorithme DegMax est le plus rapide, entre 20 et 35 sommets les algorithmes DegMax et FGK sont à peu près équivalent et au-delà de 35 sommets l'algorithme FGK est le plus rapide.

Pour donner un ordre d'idée, pour n = 95 et p = 0.1 :

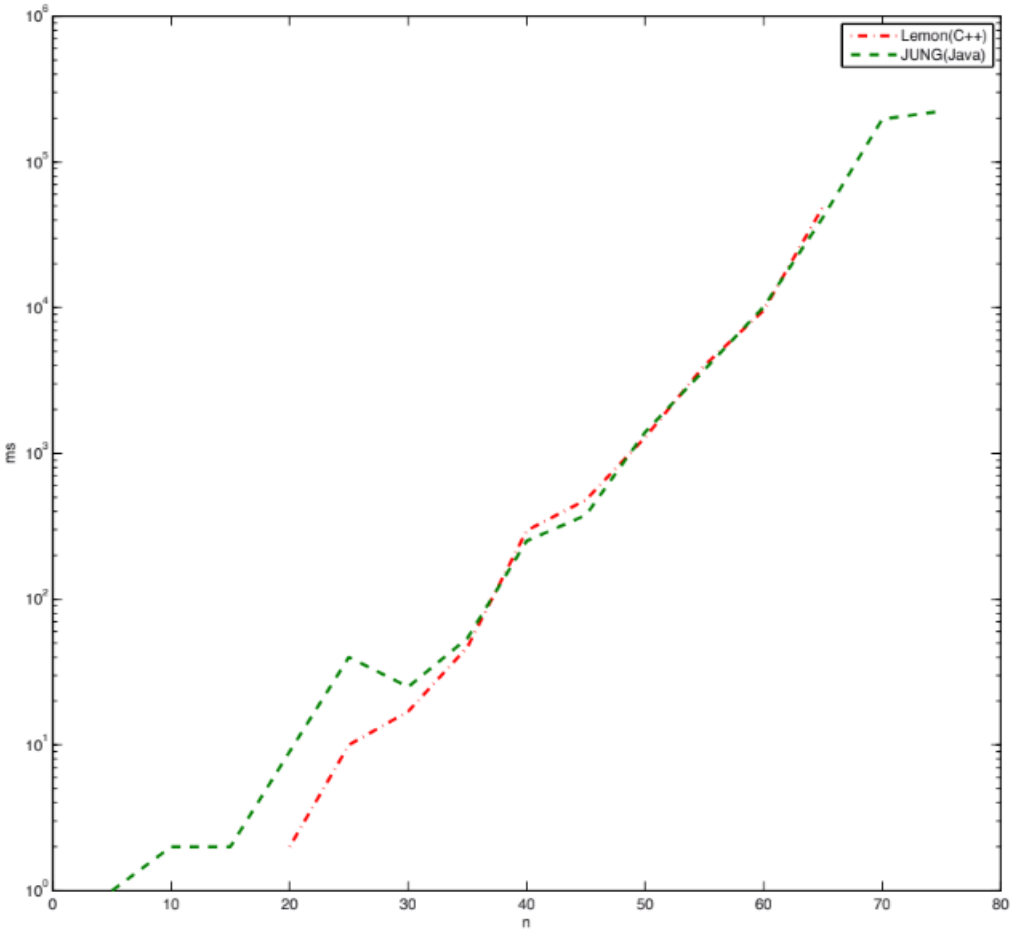
– t(DegMin) = 4h 38min

– t(DegMax) = 3min 43s

– t(FGK) = 3,1s

La seconde partie compare le temps d'exécution de l'algorithme DegMin en JUNG (JAVA) et Lemon (C++).

On obtient la courbe suivante :



On constate que les deux implémentation on des temps d'exécution à peu près semblable, l'auteur explique que l'implémentation en java est parfois meilleure car le Garbage Collector gère certainement mieux la mémoire que le langage c++.

IV Algorithmes exacte et rapproché

1 – Méthode exacte

Nous nous sommes inspirés de l'article *Implémentation d'algorithmes exacts pour le problème du stable maximum* (Levorato 2011) pour créer notre algorithme exact de résolution du problème. Cependant l'algorithme présenté dans cet article ne semblait pas fonctionner correctement, nous avons donc effectué des modifications sur celui-ci.

Notre algorithme consiste à générer tous les stables maximum du graphe et à prendre celui avec la cardinalité maximum. Pour ce faire, on parcourt tous les sommets du graphe, on crée un sous graphe en enlevant tous les voisins du sommet courant et en ajoutant ce dernier au stable. On effectue cette opération de manière récursive sur tous les sous graphes générés.

2 – Méthode approchée

Nous avons choisi de nous inspirer de l'article *Approximating Maximum Weighted Independent Set Using Vertex Support* (Balaji, Venkatasubramanian et al. 2009) pour l'algorithme de résolution approchée de notre problème. Leur algorithme permettant de trouver le stable de poids maximum, il suffit de l'utiliser avec un graphe dont tous les sommets ont le même poids afin de trouver le stable maximum, qui est un cas particulier du stable de poids maximum.

Notre algorithme consiste donc à trouver tout d'abord la couverture de poids minimum par sommets (**MWVC** : Minimum Weighted Vertex Cover). On ajoute alors à notre stable maximum tous les sommets du graphe n'étant pas compris dans la couverture minimum.

Afin de trouver le MWVC, on calcule pour chaque sommet son **degré**, son **support** et son **ratio**. Le degré d'un sommet est la somme de ses voisins, le support est la somme des degrés de ses voisins et le ratio est le produit du degré du sommet par son support. On ajoute alors le sommet au ratio le plus élevé au MWVC en supprimant ses liens avec les autres sommets, on le retire de la matrice d'adjacence, et on répète l'opération jusqu'à avoir une matrice d'adjacence vide.

V Instances des graphes

Afin de tester nos deux algorithmes de résolution du stable maximum, nous avons utilisé des graphes générés automatiquement avec la méthode G(n,p) expliquée dans l'article (Balaji, Venkatasubramanian et al. 2009).

Le modèle **G(n, p)** (aussi appelé Erdos Renyi random graph model) consiste à créer un graphe de **n sommets** et à parcourir l'ensemble des couples du graphe en **créant un pont avec une probabilité de p**.

Pour analyser les résultat, nous avons sélectionné 4 x 3 modèles G(n,p), en faisant varier le nombre de sommets créés (5,6,8,10) et la probabilité de création d'un pont pour chaque nombre de sommets (0.25, 0.5, 0.75). Nous avons effectué 10 générations aléatoires de graphes pour chacun des modèles proposés

Nous avons relevé le temps d'exécution pour chacun des algorithmes (exact et approché), la cardinalité de la solution trouvée. Dans le cas de l'algorithme approché, la validité de solution à aussi été indiquée.

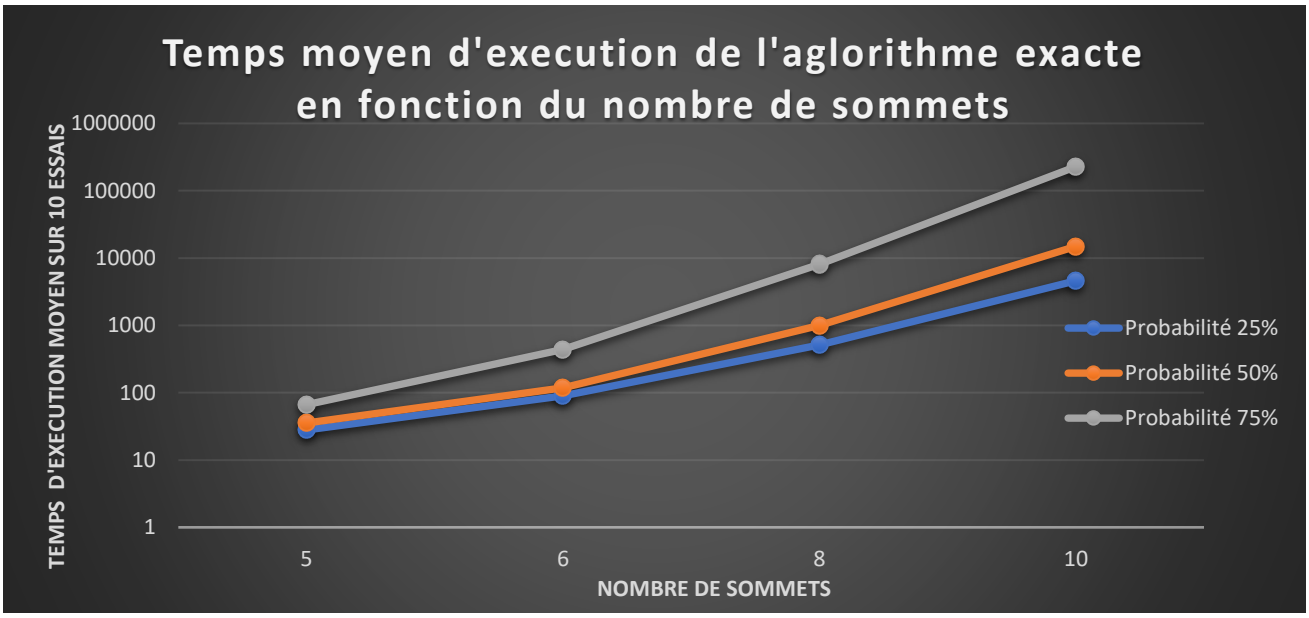
➔ Toutes les valeurs sont disponibles dans le tableau de résultats en Annexe.

VI Présentation et analyse des résultats

1 – Méthode exacte

Moyenne des temps d'exécution

Afin d'analyser l'évolution du temps d'exécution de notre algorithme exact en fonction de la complexité des graphes (nombre de sommets et de ponts), nous avons dressé le graphique suivant : (le temps d'exécution est sur une échelle logarithmique)



On voit que le temps d'exécution augmente de façon exponentielle lorsque le nombre de sommets augmente et lorsque le nombre de ponts augmente. Il nous a été difficile d'effectuer des tests avec des valeurs plus élevées puisque la résolution était beaucoup trop longue.

Test unitaires

Afin de tester notre algorithme exact sur des graphes plus complexes, nous avons effectué quelques tests supplémentaires résumés dans ce tableau, avec une probabilité d'apparition de ponts de 30% :

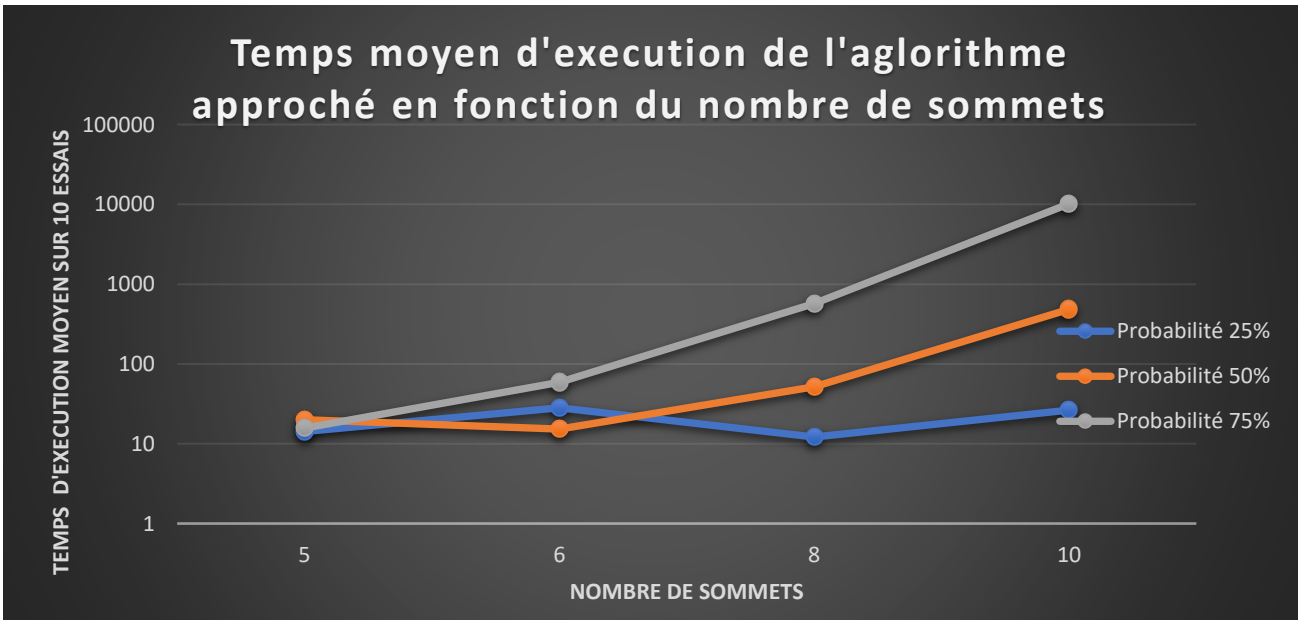
Sommets	Temps d'exécution (en secondes)
10	6
11	12
12	45
13	224

Afin de tester les limites de notre algorithme exact, nous avons essayé de trouver le stable maximum d'un graphe de 15 sommets avec une probabilité d'apparition de ponts de 10%, cependant au bout de 42 minutes nous n'avions toujours pas de résultat.

2 – Méthode approchée

Moyenne des temps d'exécution

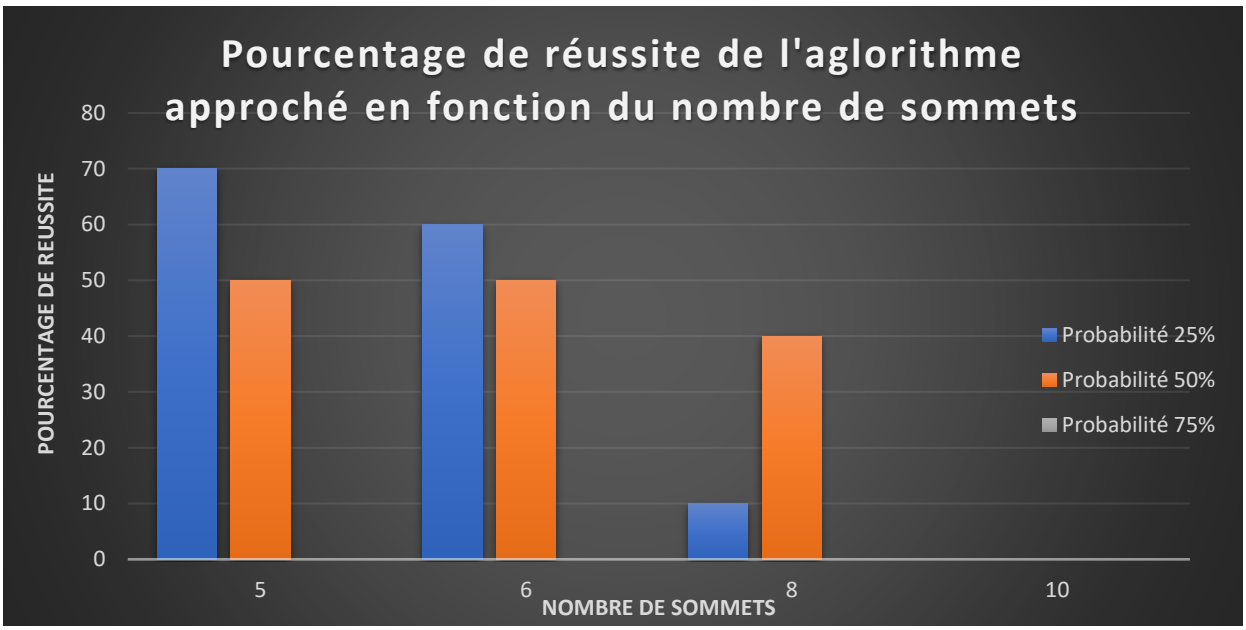
Afin d'analyser l'évolution du temps d'exécution de notre algorithme approché en fonction de la complexité des graphes (nombre de sommets et de ponts), nous avons dressé le graphique suivant : (le temps d'exécution est sur une échelle logarithmique)



On voit que le temps d'exécution augmente de façon exponentielle lorsque le nombre de sommets augmente et lorsque le nombre de ponts augmente. Il nous a été difficile d'effectuer des tests avec des valeurs plus élevées puisque la résolution était beaucoup trop longue. Cependant, lorsque le nombre de ponts est peu élevé (ici avec une probabilité d'apparition de 25%), le temps d'exécution semble rester à une constante peu élevée.

Efficacité

Nous avons cherché à évaluer l'efficacité de notre algorithme approché en le comparant sur les mêmes graphes aux résultats optimaux de notre algorithme exact. Nous avons relevé pour chaque nombre de sommet et chaque probabilité le pourcentage de stable maximum valides en solution de notre algorithme approché :



Pour des graphes avec peu de sommets ou une probabilité faible de création de ponts, notre algorithme approché semble performant. Cependant, plus on augmente la complexité des graphes, plus l'algorithme approché donne des résultats erronés.

Nous avons vérifié pour le cas particulier d'un graphe à deux sommets avec ou sans pont que notre algorithme approché donnait bien la bonne solution, les résultats sont visibles dans le tableau en annexe.

VII Conclusion

Pour conclure, l'analyse de nos deux algorithmes de résolution a montré que l'algorithme exact est opérationnel, mais très lent par rapport à l'algorithme approché et les algorithmes que nous avons vu dans la littérature scientifique, notamment lorsque les graphes gagnent en complexité. Cependant, l'algorithme approché donne des résultats bons dans des petits graphes mais la validité de ses solutions est rarement observée dans des graphes plus complexes.

ANNEXE

Sommets	Probabilité	Exacte		Approchée				
		Temps (millisecondes)	Cardinalité	Temps	Cardinalité	Stable	Validité	Ecart temps (milli secondes)
2	0	1	2	7,3	2	o	o	
		9,6	2	11,7	2	o	o	
		3	2	0,8	2	o	o	
		4,8	2	1,6	2	o	o	
		5,3	2	4,2	2	o	o	
		5,6	2	7,2	2	o	o	
		5,2	2	4,2	2	o	o	
		6,4	2	7,1	2	o	o	
		10,6	2	10,4	2	o	o	
		7,7	2	3,4	2	o	o	
	Moyenne	5,92		5,79			100	0,13
	100	26,2	1	16,4	1	o	o	
		34,2	1	6	1	o	o	
		50,5	1	2,9	1	o	o	
		6,4	1	1,4	1	o	o	
		10,7	1	31	1	o	o	
		11,3	1	19,8	1	o	o	
		20,6	1	8,4	1	o	o	
		7,7	1	10,2	1	o	o	
		4,3	1	4,7	1	o	o	
		7,3	1	11,3	1	o	o	
	Moyenne	17,92		11,21			100	6,71
5	25	23,1	3	53,3	4	n	o	
		16,4	3	18,1	4	n	n	
		27,1	3	11,1	3	o	o	
		24,4	3	3,5	3	o	o	
		28,5	3	3,2	3	o	o	
		56,8	5	9,7	5	o	o	
		20,5	3	4,4	3	o	o	
		32,4	3	6,6	2	o	n	
		20,4	4	5,8	3	o	n	
		30,2	4	26	4	o	o	
	Moyenne	27,98		14,17			70	13,81
	50	33,3	3	83,8	1	o	o	
		49,1	3	9,4	3	n	n	
		40,1	2	4,2	1	o	n	
		39,4	3	4,1	2	o	n	

		28,3	2	19,5	2	o	o	
		34,5	3	17	3	o	o	
		43,6	2	15,1	3	n	n	
		44,7	2	24,4	2	o	o	
		22,9	3	9,3	3	n	n	
		18,9	3	12,5	3	o	o	
	Moyenne	35,48		19,93			50	15,55
	75	78,3	2	19,8	1	o	n	
		97,7	2	23,5	1	o	n	
		33,8	2	9,1	1	o	n	
		40,1	2	6,4	1	o	n	
		75	2	18,2	1	o	n	
		113,4	2	24,1	1	o	n	
		47	2	12	1	o	n	
		100	2	21,1	1	o	n	
		51	2	14,8	1	o	n	
		28	2	8,2	1	o	n	
	Moyenne	66,43		15,72			0	50,71
6	25	107,7	5	53,3	5	o	o	
		68	4	4,1	4	o	o	
		57,4	4	15,1	3	o	n	
		86,1	4	21	4	n	n	
		107,5	3	20	4	n	n	
		159,2	5	122,4	4	o	n	
		66,1	3	16,5	3	o	o	
		100,3	3	12,3	3	o	o	
		97	4	11	4	o	o	
		53,3	4	6,7	4	o	o	
	Moyenne	90,26		28,24			60	62,02
	50	94,5	3	11,8	1	o	n	
		55,2	3	9,2	1	o	n	
		41,6	3	4,2	2	o	n	
		62,5	3	13,4	3	o	o	
		100,3	3	12,9	3	o	o	
		156,6	3	21,1	2	o	n	
		137	3	14	3	o	o	
		437	2	50,6	2	o	o	
		37,6	3	7,9	2	o	n	
		59,5	4	7,5	4	o	o	
	Moyenne	118,18		15,26			50	102,92
	75	299	2	49,1	1	o	n	
		572,4	2	110,3	1	o	n	
		281,7	2	32,6	1	o	n	
		156,8	2	24,3	1	o	n	
		436,6	2	56,6	1	o	n	

		431,2	2	58,7	1	o	n	
		645,2	2	84	1	o	n	
		1000,9	2	104,7	1	o	n	
		401,7	2	46	2	n	n	
		153	3	21,1	1	o	n	
	Moyenne	437,85		58,74			0	379,11
8	25	621	5	9,6	4	o	n	
		519	5	10,8	4	o	n	
		284,6	4	6	3	o	n	
		661,5	5	5,3	4	o	n	
		574,1	5	17,5	5	o	o	
		568	5	12,2	6	n	n	
		307,1	5	11,1	4	n	n	
		592,4	4	25,2	5	n	n	
		577,2	5	8,3	4	o	n	
		433,4	5	15,5	5	n	n	
	Moyenne	513,83		12,15			10	501,68
	50	1480,2	4	79,6	2	n	n	
		912,3	3	63	2	o	n	
		1601,7	4	59,6	2	o	n	
		864	5	23,2	5	o	o	
		853,5	3	44,5	3	o	o	
		1174,2	3	83,5	3	o	o	
		459,5	3	28,3	3	n	n	
		1151,5	3	78,5	4	n	n	
		755	5	20	4	n	n	
		737,7	3	36,3	3	o	o	
	Moyenne	998,96		51,65			40	947,31
	75	5214,6	3	378,1	2	n	n	
		6575,5	3	487,8	1	o	n	
		12335,3	3	802,6	1	o	n	
		1210,7	3	86,7	2	o	n	
		2975,7	3	255,3	1	o	n	
		7744,4	2	647,3	1	o	n	
		13787,2	2	952,4	1	o	n	
		10594,8	2	789,3	1	o	n	
		11680,4	3	753,2	1	o	n	
		8662,6	3	542,6	1	o	n	
	Moyenne	8078,12		569,53			0	7508,59
10	25	5579,6	6	17	4	o	n	
		8025,6	5	27,7	4	n	n	
		4597,6	5	52	2	o	n	
		2432	5	8,2	3	n	n	
		3434,9	5	42	3	o	n	
		3383,9	5	26,5	4	n	n	

		7532,8	6	16	5	n	n	
		2433,4	4	33,9	3	o	n	
		6393,3	6	19,2	5	o	n	
		2127,9	5	20,7	3	o	n	
	Moyenne	4594,1		26,32			0	4567,78
	50	16779,1	4	597,6	2	o	n	
		31067,9	3	1221,5	1	o	n	
		11484,8	4	322,8	3	o	n	
		8782,1	5	140,2	2	o	n	
		16668,7	4	593,1	2	o	n	
		6599,7	3	237	2	o	n	
		4207,7	4	65	5	n	n	
		37936,9	3	1349,6	2	n	n	
		10539,3	5	153,2	5	n	n	
		4960	4	151,5	3	n	n	
	Moyenne	14902,62		483,15			0	14419,47
	75	56519	3	2768,4	2	n	n	
		292474,1	3	10885	1	o	n	
		183023,7	3	6679,2	2	o	n	
		58255	3	2548,7	1	o	n	
		435545,8	2	28308	2	n	n	
		165731,2	2	7216	1	o	n	
		194150,3	2	8587,9	1	o	n	
		387895,4	3	13810,7	2	n	n	
		339090,5	3	13269,2	1	o	n	
		162550,5	3	7435,7	1	o	n	
	Moyenne	227523,55		10150,88			0	217372,7

Balaji, S., et al. (2009). Approximating Maximum Weighted Independent Set using vertex support.

The Maximum Weighted Independent Set (MWIS) problem is a classic graph optimization NP-hard problem. Given an undirected graph G = (V, E) and weighting function defined on the vertex set, the MWIS problem is to find a vertex set ⊆ V whose total weight is maximum subject to no two vertices in S are adjacent. This paper presents a novel approach to approximate the MWIS of a graph using minimum weighted vertex cover of the graph. Computational experiments are designed and conducted to study the performance of our proposed algorithm. Extensive simulation results show that the proposed algorithm can yield better solutions than other existing algorithms found in the literature for solving the MWIS.

Levorato, V. (2011). Implémentation d'algorithmes exacts pour le problème du stable maximum, LIFO, Université d'Orléans.

Sewell, E. C., et al. (2011). "Reductions for the Stable Set Problem." Algorithmic Operations Research; Vol 6, No 1 (2011).

One approach to finding a maximum stable set (MSS) in a graph is to try to reduce the size of the problem by transforming the problem into an equivalent problem on a smaller graph. This paper introduces several new reductions for the MSS problem, extends several well-known reductions to the maximum weight stable set (MWSS) problem, demonstrates how reductions for the generalized stable set problem can be used in conjunction with probing to produce powerful new reductions for both the MSS and MWSS problems, and shows how hypergraphs can be used to expand the capabilities of clique projections. The effectiveness of these new reduction techniques are illustrated on the DIMACS benchmark graphs, planar graphs, and a set of challenging MSS problems arising from Steiner Triple Systems.