Heuristique largest first

LABORATOIRE 1

Herzig Melvyn

SIO – 02.04.2021

Table des matières

Introduction

Le but de ce laboratoire est d'implémenter 4 versions de l'heuristique Largest First de Welsh et Powel.

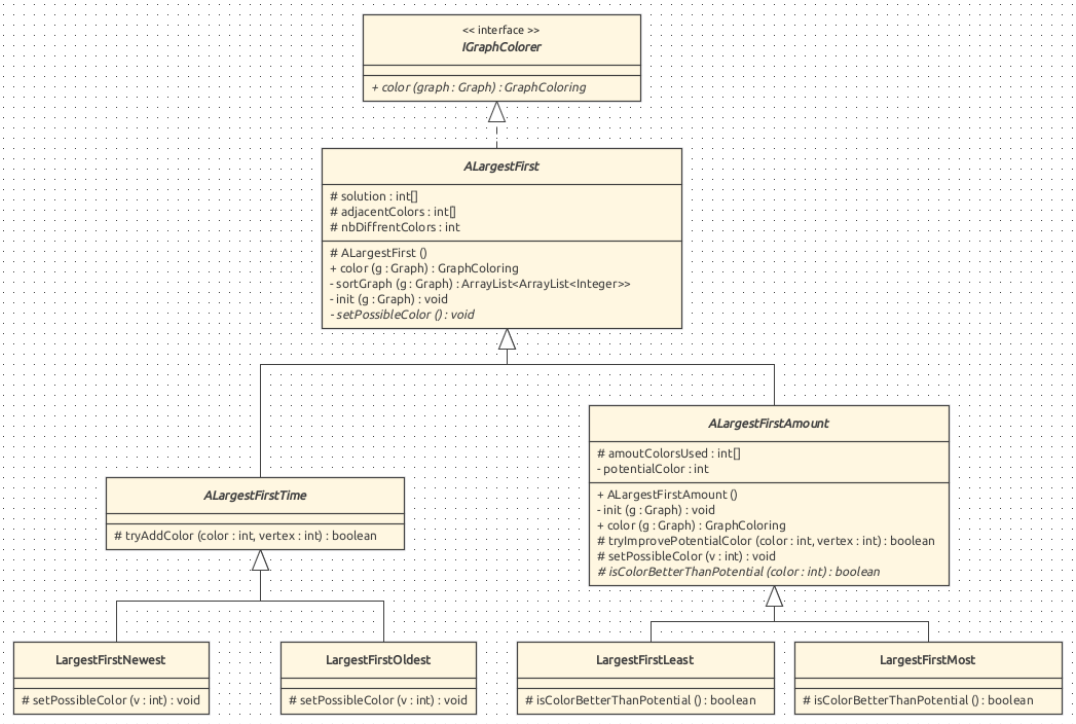
Les 4 versions sont :

- Oldest : La couleur choisie en priorité est la plus vieilles des couleurs.
- Newest : La couleur choisie en priorité est la plus récente des couleurs.
- Least : La couleur choisie en priorité est la moins utilisée des couleurs.
- Most : La couleur choisie en priorité est la plus utilisée des couleurs.

Ce document décrit l'implémentation ainsi que les résultats observés.

Architecture et algorithme

Le diagramme de classes du programme est le suivant :



(Les structures de données sont dimensionnées pour supporter le « pire » cas. De cette façon, nous évitons tout redimensionnement coûteux en cours d'exécution)

La classe *ALargestFirst* est le noyau des 4 algorithmes. Elle trie les sommets et elle vérifie quelles sont les couleurs adjacentes à un sommet donné. Elle stocke les adjacences dans adjacentColors.

Nous avons ensuite deux variantes :

Oldest/Newest

Pour chaque sommet, la fonction *color* de ALargestFirst, appelle *setPossibleColor* qui parcourt *adjacentColors* à dans un sens spécifique à la version de LF. Pour chaque couleur visitée, on tente de l'ajouter au sommet donné avec *TryAddColor* dans *ALargestFirstTime*.

Least/Most

La classe *ALargestFirstAmount* offre une structure supplémentaire. *AmountColorUsed* stocke le nombre d'utilisations de chaque couleurs. Le parcours de *adjacentColors* est semblable dans la version Least et Most de LF. De ce fait *setPossibleColor* est implémenté dans *ALargestFirstAmout*.

Dans tous les cas, on parcourt le tableau des couleurs utilisées à la recherche de la plus/moins affectée. On conserve la couleur candidate dans *potentialColor*. Les couleurs sont comparées à *potentialColor* dans *isColorBetterThanPotentialColor*. Une fois que toutes les couleurs sont parcourues, il est possible d'affecter une couleur au sommet courant.

Dans les 4 variantes, une nouvelle couleurs est ajoutée puis affectée au sommet courant si aucune couleur déjà employée est candidate.

Version alternative

Une variante alternative de Least/Most avait été implémentée en triant les couleurs dans des buckets en fonction de leur nombre d'utilisations. A l'initialisation, toutes les couleurs étaient dans le premier bucket (bucket 0 -> 0 utilisation). À chaque fois qu'une couleur était affectée, elle passait dans le bucket supérieur. Les couleurs au sein d'un même bucket étaient conservées dans l'ordre croissant. Sitôt qu'une couleur était candidate, nous pouvions arrêter la recherche. Cette version a été retirée car moins performante que celle en actuellement en place.

Complexité

Soit N le nombre de sommets d'un graphe et M son nom d'arêtes. On pose aussi C le nombre maximal de couleurs de ce graphe dans le pire des cas. On a donc C <= N.

Le tri des sommets est effectué à l'aide d'un tri bucket, donc en O(M+N)

Dans le cas des versions Oldest et Newest.

Dans ces deux versions, nous parcourons les couleurs utilisées et nous nous arrêtons sur la première candidate affectable. La recherche d'une couleur est effectuée en O(C).
On en déduit donc que **les versions Oldest et Newest sont amorties en O(M + N).**

Dans le cas des versions Least et Most.

Dans ces deux versions, l'ensemble des couleurs affectées est parcouru à la recherche de la meilleure candidate. Cette opération est faite en O(C).

Ainsi la recherche s'effectue en O(C). On en déduit donc que **les versions Least et Most sont amorties en O(M + N).**

Vérification des implémentations

Nous avons reçu trois graphes de tests : *EX_SERIE1_4_a.txt, EX_SERIE1_4_b.txt*, *EX_SERIE1_4_c.txt*. Les 4 versions de LF ont été testées sur les trois graphes. Afin d'alléger ce document, nous ne présenterons que les résultats du dernier test sur le graph *EX_SERIE1_4_c.txt*.

Les sorties des 4 algorithmes ont été comparées avec les solutions mises à disposition. Toutes correspondent.

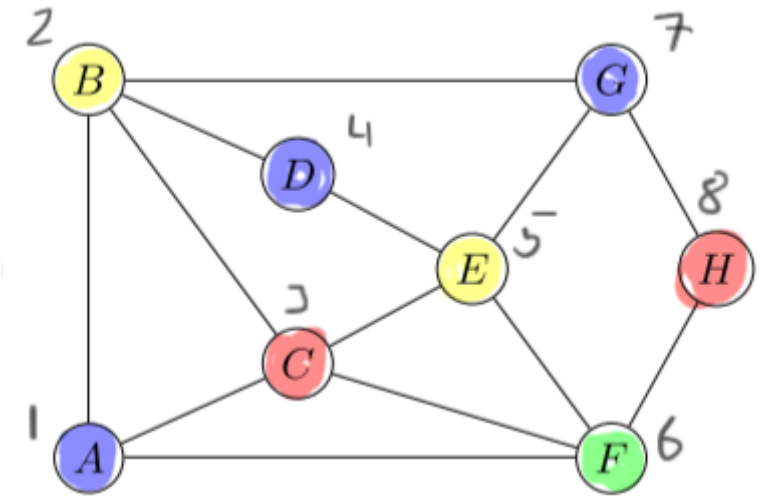
Newest

Voici une sortie console commentée, les deux colonnes de gauches sont le graphes colorié, la colonne à gauche de la matrice correspond au sommet traité, la matrice représente les couleurs adjacentes au sommet traité et la colonne à droite de la matrice est le nombre de couleurs utilisées.

Comme indiquée la première couleur est allouée au plus petit sommet de plus grand degré.

```
Newest
-------------------------------------------------
Résultat|sommet traité| couleurs adjacentes| nb couleurs utilisées
                      1  2  3  4  5
8 4                                    Sommet 2 automatiquement ajouté avec c1
1 4                3 [3, 0, 0, 0, 0] 1 c1 adjacente, ajout et assignation c2
2 1                5 [3, 5, 0, 0, 0] 2 c2 adjacente, assignation c1
3 2                6 [6, 6, 0, 0, 0] 2 c2, c1 adjacente, ajout et assignation c3
4 4                1 [1, 1, 1, 0, 0] 3 c3, c2, c1 adjacente, ajout et assignation c4
5 1                7 [7, 1, 1, 0, 0] 4 assignation c4
6 3                4 [4, 1, 1, 0, 0] 4 assignation c4
7 4                8 [4, 1, 8, 8, 0] 4 assignation c2
8 2
```

Représentation



Comme nous pouvons le voir, les sommets sont traités dans le bon ordre et la couleur assignée est toujours la plus récente ou une nouvelle.

Aucun sommet ne se voit assigner une couleur adjacente.

Au terme des trois testes, en première approche, l'algorithme LF version Newest est fonctionnel.

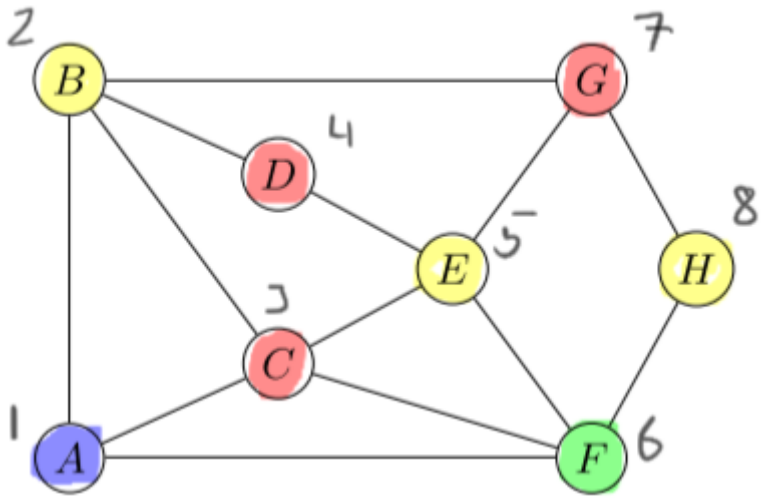
Oldest

Voici une sortie console commentée, les deux colonnes de gauches sont le graphes colorié, la colonne à gauche de la matrice correspond au sommet traité, la matrice représente les couleurs adjacentes au sommet traité et la colonne à droite de la matrice est le nombre de couleurs utilisées.

Comme indiquée la première couleur est allouée au plus petit sommet de plus grand degré.

```
Oldest
-------------------------------------------------
Résultat|sommet traité| couleurs adjacentes| nb couleurs utilisées
                  1  2  3  4  5
8 4                                Sommet 2 automatiquement ajouté avec c1
1 4            3 [3, 0, 0, 0, 0] 1 c1 adjacente, ajout et assignation c2
2 1            5 [3, 5, 0, 0, 0] 2  assignation c1
3 2            6 [6, 6, 0, 0, 0] 2 c1, c2 adjacentes, ajout et assignation c3
4 2            1 [1, 1, 1, 0, 0] 3 c1, c2, c3 adjacentes, ajout et assignation c4
5 1            7 [7, 1, 1, 0, 0] 4 c1 adjacente, assignation c2
6 3            4 [4, 1, 1, 0, 0] 4 c1 adjacente, assignation c2
7 2            8 [4, 8, 8, 0, 0] 4 assignation c1
8 1
```

Représentation



Comme nous pouvons le voir, les sommets sont traités dans le bon ordre et la couleur assignée est toujours la plus ancienne ou une nouvelle.

Aucun sommet ne se voit assigner une couleur adjacente.

Au terme des trois testes, en première approche, l'algorithme LF version Oldest est fonctionnel.

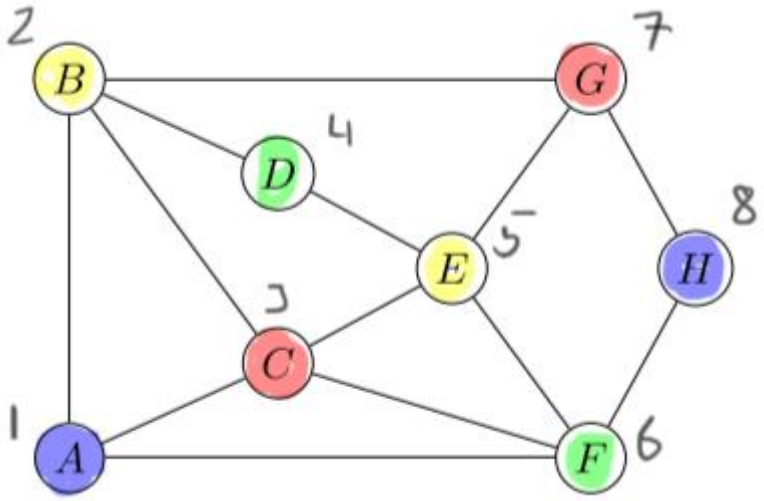
Least

Voici une sortie console commentée, les deux colonnes de gauches sont le graphes colorié, la colonne à gauche de la matrice correspond au sommet traité, la matrice représente les couleurs adjacentes au sommet traité et la colonne à droite de la matrice est le nombre de couleurs utilisées.

Comme indiquée la première couleur est allouée au plus petit sommet de plus grand.

```
Least
-------------------------------------------------
Résultat | sommet traité | couleurs adjacentes | nb couleurs utilisées
8 4                    1  2  3  4  5     c1 automatiquement ajouté au sommet 2
1 4                3 [3, 0, 0, 0, 0] 1 c1 adjacente, assignation c2
2 1                5 [3, 5, 0, 0, 0] 2 c1 plus vieille couleur la moins utilisée, assignation c1
3 2                6 [6, 6, 0, 0, 0] 2 c1 et c2 adjacentes, assignation c3
4 3                1 [1, 1, 1, 0, 0] 3 c1, c2 et c3 adjacentes, assignation c4
5 1                7 [7, 1, 1, 0, 0] 4 c2 plus vieille couleur la moins utilisées, assignation c2
6 3                4 [4, 1, 1, 0, 0] 4 c3 plus vieille couleur la moins utilisées, assignation c3
7 2                8 [4, 8, 8, 0, 0] 4 c4 plus vieille couleurs la moins utilisée assignation c4
8 4
```

Représentation



Sommet traité	Couleurs 0 utilisation	Couleurs 1 utilisation	Couleurs 2 utilisation
2	1, 2, 3, 4, 5		
3	2,3,4,5	1	
5	3,4,5	1,2	
6	3,4,5	2	1
1	4,5	2,3	1
7	5	2,3,4	1
4	5	3,4	1,2
8	5	4	1,2,3

Comme nous pouvons le voir, les sommets sont traités dans le bon ordre et la couleur assignée est toujours la première la moins utilisée et non adjacente. Les couleurs existantes sont privilégiées aux couleurs avec 0 utilisation. En cas d'égalité, la plus vieille couleur vient avant.

Au terme des trois testes, en première approche, l'algorithme LF version Least est fonctionnel.

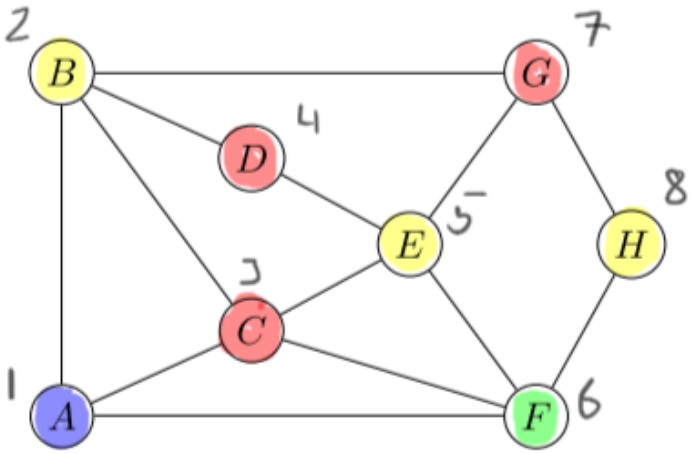
Most

Voici une sortie console commentée, les deux colonnes de gauches sont le graphes colorié, la colonne à gauche de la matrice correspond au sommet traité, la matrice représente les couleurs adjacentes au sommet traité et la colonne à droite de la matrice est le nombre de couleurs utilisées.

Comme indiquée la première couleur est allouée au plus petit sommet de plus grand degré.

```
Most
-------------------------------------------------
Résultat | sommet traité | couleurs adjacentes | nb couleurs utilisées
8 4                1  2  3  4  5     c1 automatiquement ajouté au sommet 2
1 4            3 [3, 0, 0, 0, 0] 1 c1 adjacente, assignation c2
2 1            5 [3, 5, 0, 0, 0] 2 c1 plus vieille couleur la  plus  utilisée, assignation c1
3 2            6 [6, 6, 0, 0, 0] 2 c1 et c2 adjacentes, assignation c3
4 2            1 [1, 1, 1, 0, 0] 3 c1, c2, et c3 adjacentes, assignation c4
5 1            7 [7, 1, 1, 0, 0] 4 c2 plus vieille couleur la  plus  utilisée, assignation c2
6 3            4 [4, 1, 1, 0, 0] 4 c2 plus vieille couleur la plus utilisée, assignation c2
7 2            8 [4, 8, 8, 0, 0] 4 c1 plus vieille couleur la plus utilisées, assignation c1
8 1
```

Représentation



Sommet traité	Couleurs 0 utilisation	Couleurs 1 utilisation	Couleurs 2 utilisations	Couleurs 3 utilisations
2	1, 2, 3, 4, 5			
3	2,3,4,5	1		
5	3,4,5	1,2		
6	3,4,5	2	1	
1	4,5	2,3	1	
7	5	2,3,4	1	
4	5	3,4	1,2	
8	5	4	1,3	2

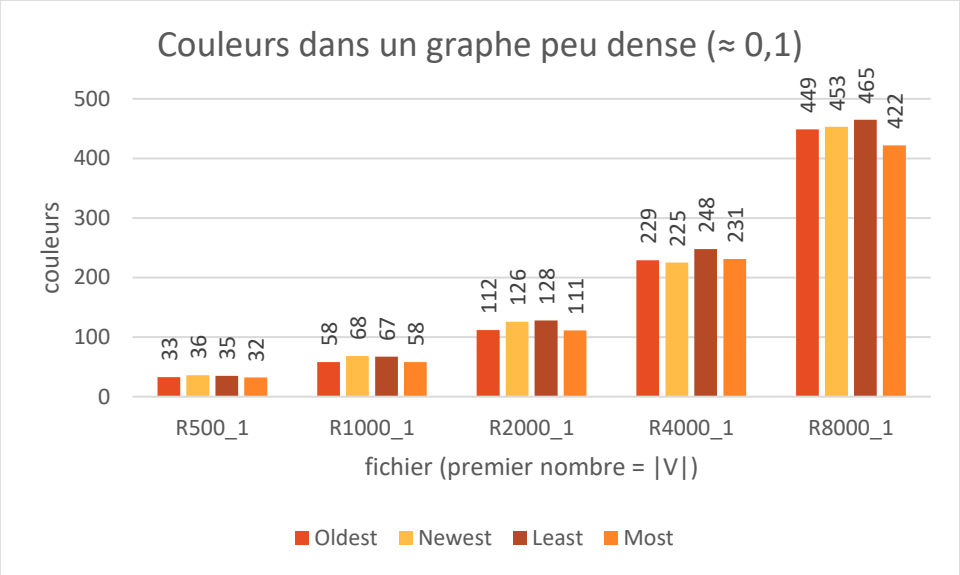
Comme nous pouvons le voir, les sommets sont traités dans le bon ordre et la couleur assignée est toujours la première la plus utilisée non adjacente. Les couleurs existantes sont privilégiées aux couleurs avec 0 utilisation. En cas d'égalité, la plus vieille couleur vient avant.

Au terme des trois testes, en première approche, l'algorithme LF version Least est fonctionnel.

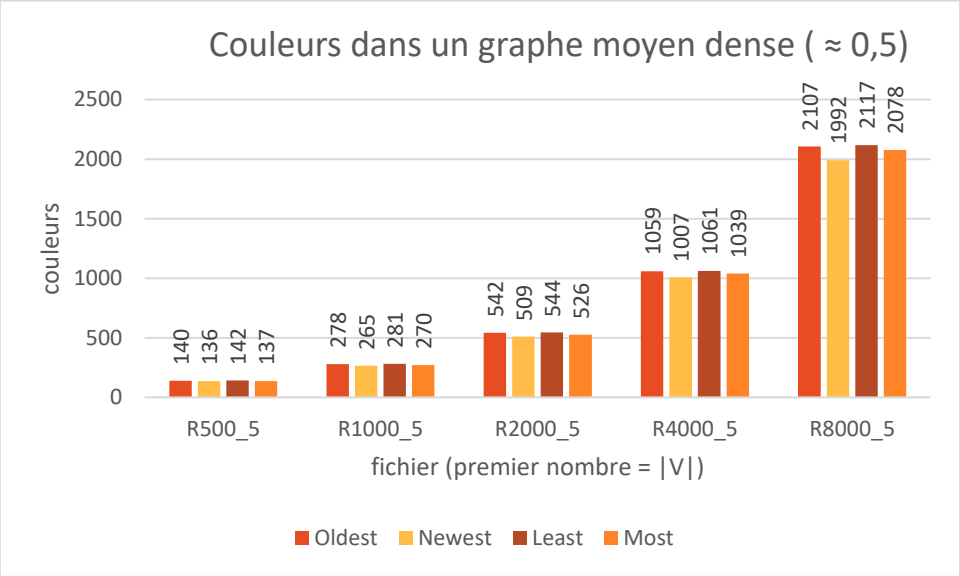
Analyse des résultats

Couleurs

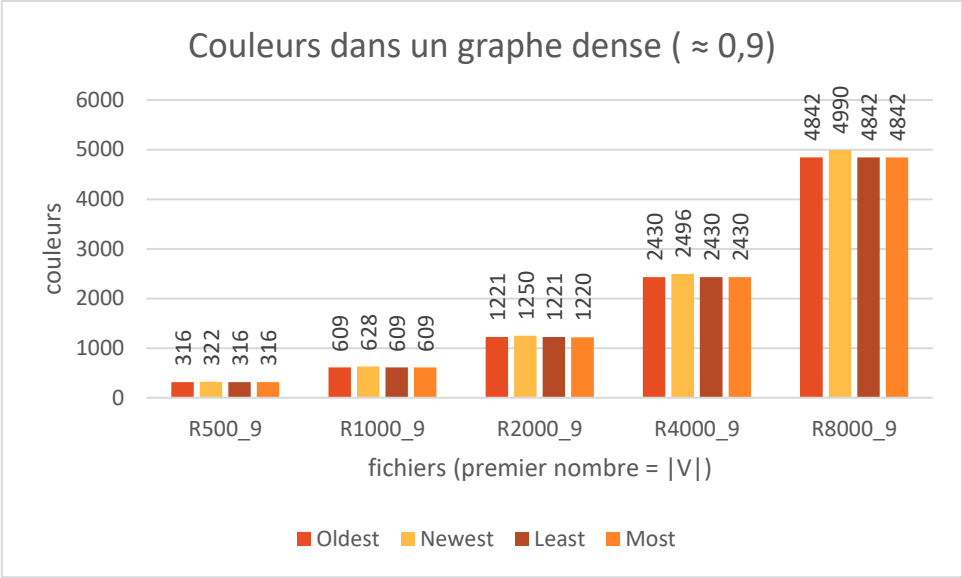
Les graphes ont été séparés en 4 familles : peu dense (0.1), moyennement dense (0.5), très dense (0.9) et VLSI.



Dans un graphe peu dense, l'algorithme LF Most est le plus efficace. Le pire est LF Least. La différence, entre le pire et le moins bon varie jusqu'à 10%. LF Oldest et LF Newest sont dans la moyenne mais LF Oldest est sensiblement plus performant que LF Newest. De plus, il est souvent proche de LF Most.

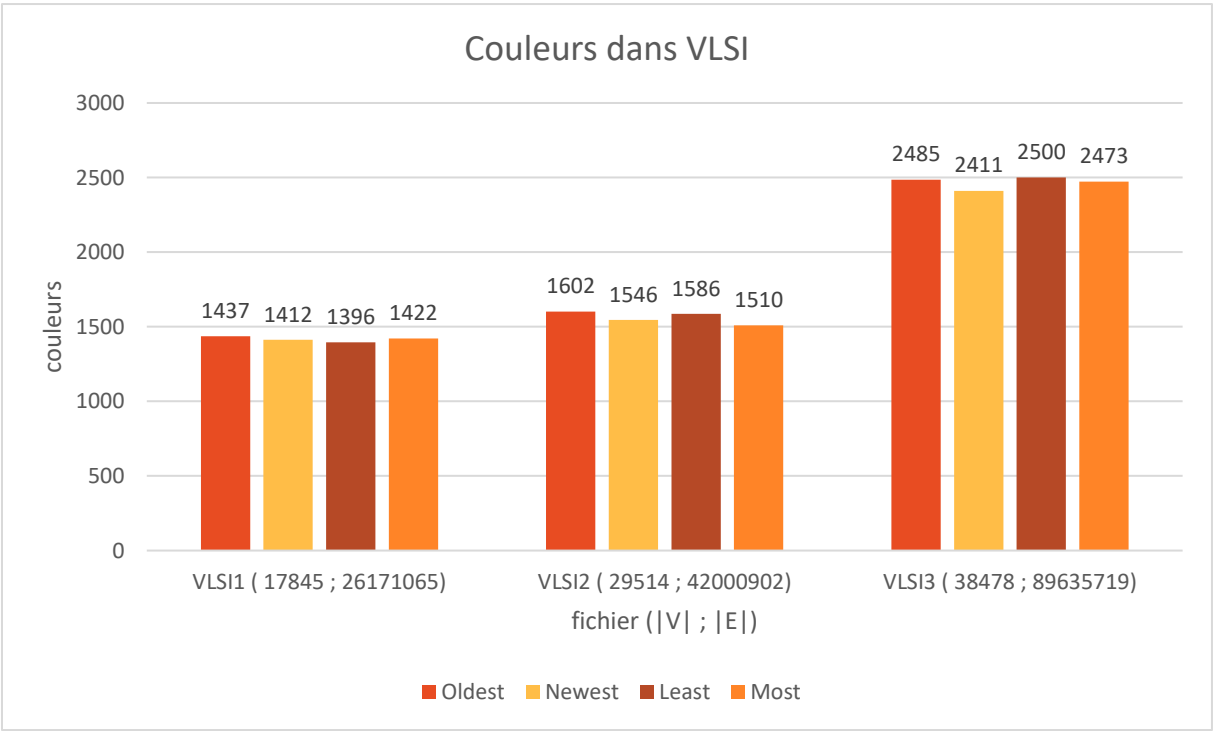


Dans le cas de graphes moyennement denses, LF Newest est le plus optimal, suivi de près par LF Most et derrière LF Oldest et LF Least. Tous sont très proches sauf la variante Newest qui tire son épingle du jeu.



Dans le cas de graphes denses, LF Most, LF Least et LF Oldest sortent du lot. Ils ont toujours le même nombre de couleurs à l'exception d'une fois. Avec 2000 sommets, LF Most a été meilleur d'une couleur par rapport à ses deux rivaux. Est-ce que cette exception unique est suffisante pour prouver la supériorité de la version Most ? Je ne le pense pas. Pour s'en assurer il faudrait effectuer d'autre tests. Contrairement aux graphes moyennement denses, LF Newest passe de premier à dernier.

Quoi qu'il en soit, dans des graphes très denses, l'écart se tasse. Oldest, Least et Most s'alignent.



Je ne savais pas comment traiter les graphes VLSI. Ils ont une densité qui s'étend de 0.9 à 0.16. De ce fait j'ai décidé de ne pas les inclure avec les graphes peu denses par manque d'homogénéité.

Difficile de déterminer un meilleure algorithme : dans VLSI 1 LF Least est meilleur, dans VLSI2 LS Most et dans VLSI3 LF Newest. On ne peut pas déduire de meilleur algorithme mais un moins bon : LF Oldest.

Conclusion sur la coloration

Après cet échantillon de tests, nous pouvons distinguer quelques tendances :

Dans le cas de graphes peu denses, l'heuristique Most semble être la plus fiable. Cela se comprend instinctivement. Comme le graphe est peu dense, il est raisonnable d'espérer pouvoir ré-utiliser la couleur la plus employée. Le risque de « collision » est faible puisque le nombre d'arêtes l'est aussi.

Dans le cas de graphes moyennement denses, Newest est le plus conseillé.

Finalement dans les graphes très denses, les algorithmes Oldest, Least et Most offrent un nombre de couleurs similaire. Comme le graphe est presque complet, les heuristiques se font rattraper par la logique de la coloration qui les pousse à choisir une couleur différentes des voisines. A cause de la densité, l'heuristique perd en efficacité.

Toutefois, cette logique ne s'applique pas pour les graphes VLSI. On ne tire aucune tendance de ces derniers. Ainsi pour compléter l'analyse des graphe à faible densité : À moins de connaître la nature exacte du graphe, il faut tester les différentes variantes afin de déterminer le meilleur résultat.

Vitesse

En parallèle à la couleur, le temps d'exécution des algorithmes est une donnée potentiellement intéressante à étudier. Est-ce qu'une heuristique est plus longue que les autres ? Y a-t-il des tendances ? Les temps mesurés sont des temps réels sujet aux variations de l'ordonnanceur. Nous nous concentrerons principalement sur les graphes avec plus de 2000 sommets pour minimiser l'impact de l'ordonnancement.

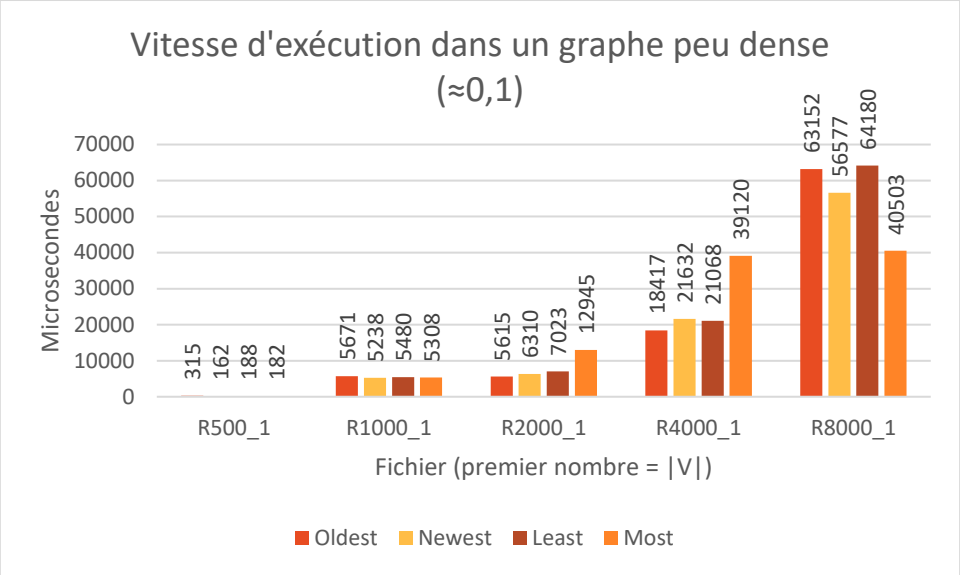
Pour ce faire, chaque algorithme a été exécuté trois fois puis la moyenne a été retenue.

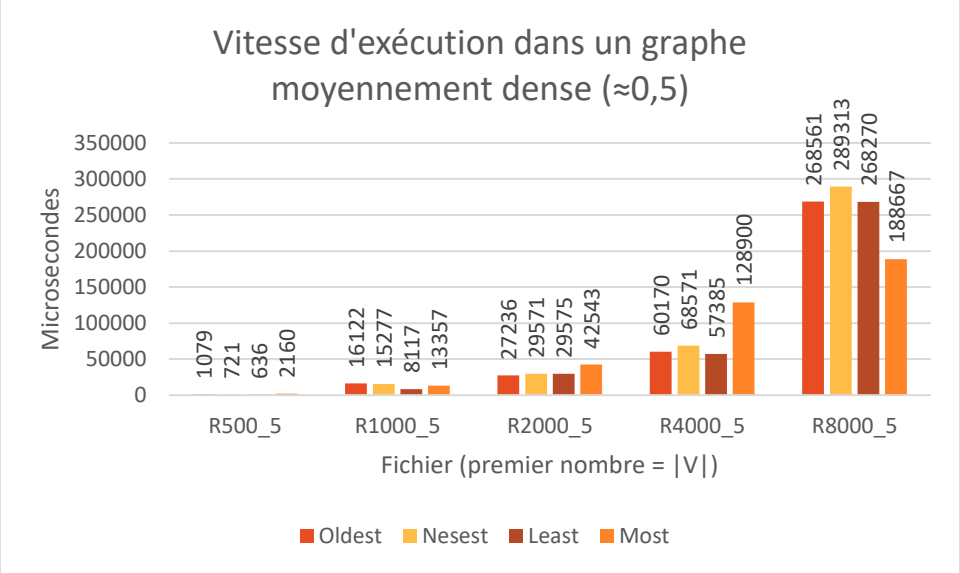
Soit les graphes de la *figure 1.* Si nous tentons de faire une lecture par densité (lecture horizontale), nous remarquons qu'il est impossible de définir une tendance.

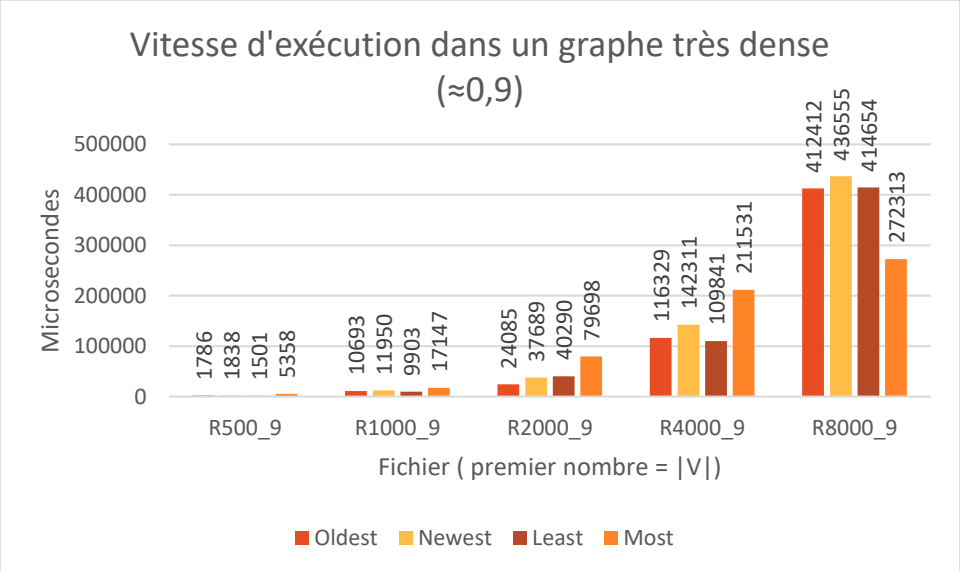
Au contraire, lorsque nous effectuons une lecture par nombre de sommets (lecture verticale), nous distinguons des récurrences.

Dans un graphe avec 2000 sommets, Most est la version la plus lente et Oldest la plus rapide. Avec 4000 sommets, la version Most est toujours la plus lente. Oldest et Least sont les plus performants. Finalement avec 8000 sommets et contre toute attente, Most est la version la plus rapide. Difficile de déterminer le moins rapide car il fluctue. Toutefois, nous remarquons que Oldest et Least produisent des temps similaires.

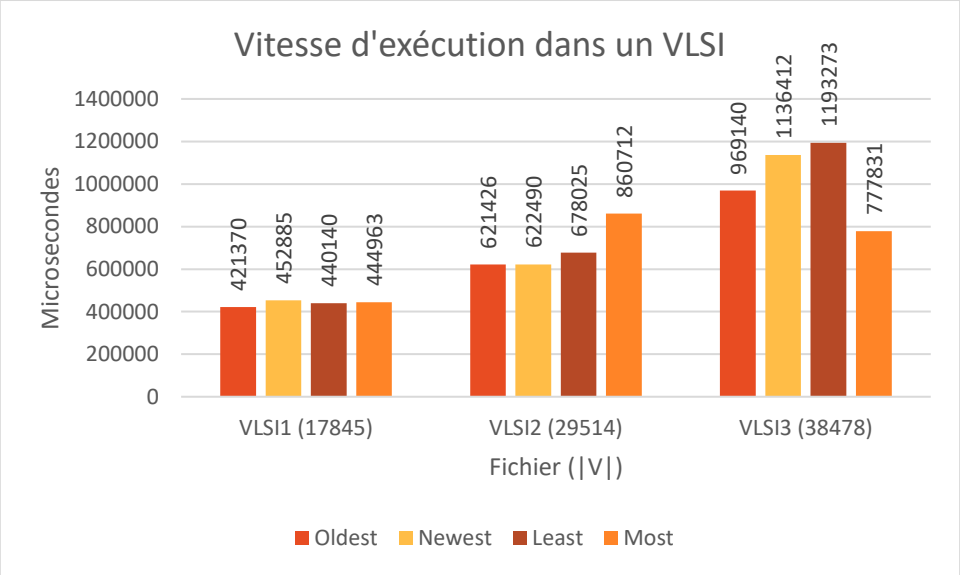
Figure 1


Vitesse d'exécution dans un graphe peu dense (≈0,1)
Microsecondes
70000
60000
50000
40000
30000
20000
10000
0
315
162
188
182
5671
5238
5480
5308
5615
6310
7023
12945
18417
21632
21068
39120
63152
56577
64180
40503
R500_1
R1000_1
R2000_1
R4000_1
R8000_1
Fichier (premier nombre = |V|)
Oldest
Newest
Least
Most


Vitesse d'exécution dans un graphe moyennement dense (≈0,5)
Microsecondes
350000
300000
250000
200000
150000
100000
50000
0
1079
721
636
2160
16122
15277
8117
13357
27236
29571
29575
42543
60170
68571
57385
128900
268561
289313
268270
188667
R500_5
R1000_5
R2000_5
R4000_5
R8000_5
Fichier (premier nombre = |V|)
Oldest
Nesest
Least
Most


Vitesse d'exécution dans un graphe très dense (≈0,9)
Microsecondes
500000
400000
300000
200000
100000
0
1786
1838
1501
5358
10693
11950
9903
17147
24085
37689
40290
79698
116329
142311
109841
211531
412412
436555
414654
272313
R500_9
R1000_9
R2000_9
R4000_9
R8000_9
Fichier (premier nombre = |V|)
Oldest
Newest
Least
Most

Par souci de complétude, voici les vitesses d'exécution sur les graphes VLSI.



Une fois n'est pas coutume, aucune tendance notable. Parfois les vitesses sont égales, parfois un algorithme sera le meilleure et d'autres fois le pire. C'est le cas pour Most dans VLSI2 et VLSI3.

Conclusion sur les temps d'exécution

D'après moi, le temps d'exécution n'est pas un critère de sélection recevable. La priorité est de minimiser les couleurs. Les temps mesurés varient mais ils sont tous du même acabit. Aucun algorithme n'a été à répétition trois fois plus lent ou trois fois plus rapide que les autres. Les 4 versions ont été estimées avec la même complexité. Ceci explique pourquoi les différences ne sont pas suffisantes pour préférer un algorithme.

Fonctionnement du programme

Pour utiliser et tester mes implémentation, il est possible de passer des arguments à mon programme.

Le premier argument peut prendre 5 formes :

- *oldest*/*newest*/*least*/*most* : exécute l'algorithme concerné et affiche la coloration dans le flux de sortie.
- Benchmark : Lance un benchmark qui teste chaque algorithme sur une série de graphes.

Le second argument peut prendre deux formes qui dépendent du premier :

- Dans le cas de *oldest*/*newest*/*least*/*most*, passer un chemin vers un fichier contenant un graphe.
- Dans le cas de benchmark, passer un chemin contenant une série de fichier graphes. Il ne doit contenir que des fichiers graphes car le benchmark tente de lire tous les fichiers du répertoire.

Exemple

```
benchmark data/
```

```
least EX_SERIE1_4_a.txt
```