Sasmithan Satkunarajah 12403246 L2 Informatique,Groupe (TP) 5
Sabarisha Gnanabaskaran 12401817

RAPPORT PROJET "Jeu d'inondation" (Flood-It)

Présentation du projet :

Dans le cadre de ce projet d'informatique, nous avons implémenté en C différentes stratégies d'optimisation pour la résolution du jeu Flood-It.

Mode d'emploi pour compiler et exécuter le programme le programme :

Le fichier Projet_Floodit.zip contient les fichiers .c et .c du code, le fichier Makefile et un unique fichier .gp qui contient le script pour le tracé des courbes avec Gnuplot.

Après avoir extrait le dossier entier, ouvrez un terminal et assurez vous de vous trouver dans le bon dossier.

I)Pour compiler : Dans le terminale, la commande **make** suffit.

II)Pour exécuter le fichier *Flood-it.o* (après compilation): il suffit de rentrer dans le terminale l'appel

./Flood-it <dimension> <nb_de_couleurs> <niveau_difficulte> <graine> <exo> <aff > <graph >

en remplaçant les éléments entre < > par l'entier voulu :

- dimension : entier >0
- nb_de_couleur : entier>0
- niveau_de_difficulte : entier
- graine : entier
- exo : entier compris entre 0 à 4 (inclus) :
  - 0 : trouve_zone_rec (exo1)
  - 1 : sequence_aleatoire_rec (exo1)
  - 2 : sequence_aleatoire_rapide (exo2)
  - 3 : max_bordure (exo3)
  - 4 : max_bordure_zone (exo4)
- aff : 1 ou 0 :
  - 1 : affichage de la grille (on voit la grille initiale se résoudre)
  - 0 : pas de affichage
- graph 1 ou 0 :
  - 0: l'appel ne fait que la fonction de l'exercice

- 1 : l’appel ne fonctionnera pas, cette argument existe pour une commande que vous pouvez utiliser (voire III))

III) Pour avoir tt les données/courbes de toutes les fonctions en une commande , il suffit d'entrer la commande **make graphes**

 Cet appel prend plus de temps que l’appel avec graph = 0, notamment parce qu'elle est utilisée avec l’exo 1 (explication dans la suite) cela peut prendre beaucoup de temps (plusieurs longues minutes), si c’est le cas, merci d’être patient. (Nous nous excusons pour ce temps d’attente.)

Pour vous faciliter la tâche, et ne pas perdre votre temps nous avons déjà récupéré les donnés/graphes et nous avons inclus dans ce rapport les résultats les plus pertinents à analyser. Nous avons tout de même choisi de laisser cette commande et l’argument graph en temps que trace de notre méthode et au cas où vous voudriez l'exécuter.

Exercice 1 – Aléatoire récursif :

Q 1.3 :

Une fois avoir inséré trouve_zone_rec et sequence_aleatoire_rec, voici les temps cpu obtenus pour différentes grilles de 10 couleurs, de tailles ($n \times n$) et sans affichage:

n	temps cpu (s)
25	0.046427
45	1.023052
65	2.119114
85	12.533680
105	31.052490

On remarque ici, que plus la taille de la grille est grande, plus le temps d'exécution de sequence_aleatoire_rec est élevé. On constate en plus, que pour un même écart de taille (de +/-20), l’écart de temps n’est pas constant, au contraire, il croit très vite. (Entre n = 25 et n=45, il y a un écart de temps d’environ $\Delta \simeq 0{,}97$ s , alors que entre n=85 et n=105, $\Delta \simeq$ 18,51s). Ainsi la fonction sequence_aleatoire_rec fonctionne mais est plus efficace en termes de temps pour des grilles de petite taille et sans affichage.

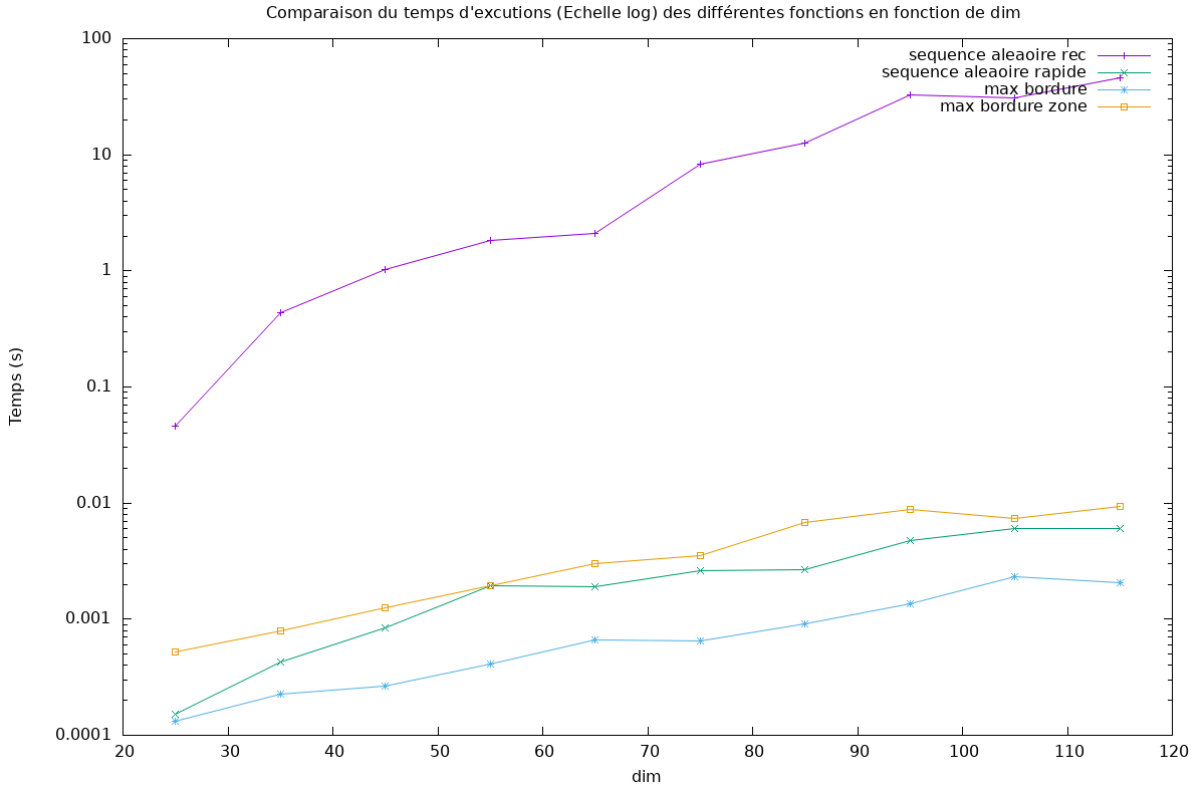
Voici les temps cpu obtenus pour différentes difficultés avec une grille de n = 10 et de 10 couleurs, sans affichage:

difficulté	temps cpu (s)
5	0.042308
30	0.017875
50	0.011017
80	0.010759
100	0.006806

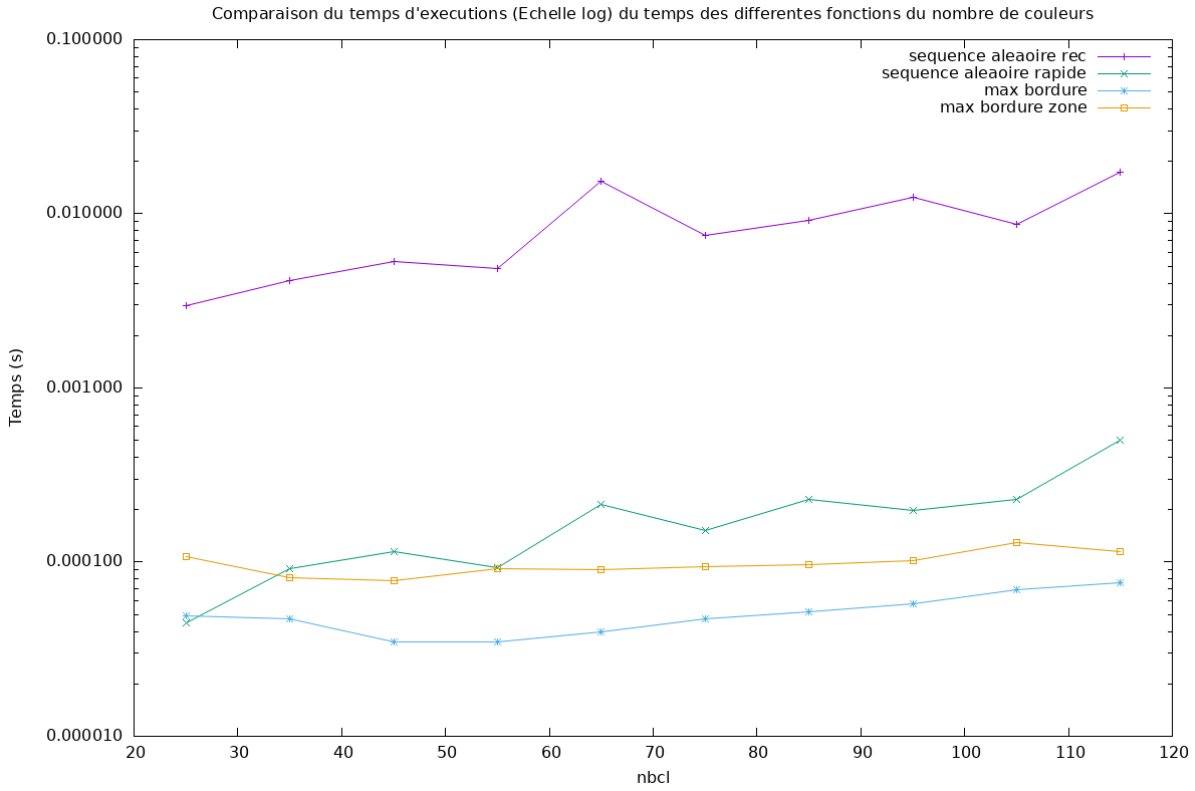
Nous pouvons observer que plus la difficulté augmente, plus le temps cpu diminue et il en est de même pour le nombre de coups. Cela s'explique par le fait que d'après le sujet, "le niveau de difficulté correspond à la taille moyenne d'une zone dans l'instance. Cette taille est ainsi un pourcentage du nombre de cases" . Ainsi, plus la difficulté est élevée, plus les zones sont grandes en moyenne et il faudra donc moins de changement de couleur pour remplir la grille. Ainsi la Zsg (calculée récursivement) s'agrandit plus rapidement et en moins d'itérations. De ce fait, plus le niveau de difficulté est élevé, plus le temps cpu diminue. A l'inverse, plus la difficulté est faible, plus les zones seront petites et plus il y aura de coups et d'appels récursifs pour recalculer la Zsg.

Exercice 2– Allons plus vite! (Structure acyclique)

Q 2.5 :



Ce graphe représente le temps d'exécution de toutes les fonctions du projet (sans affichage) en fonction de la dimension de la grille.



Ce graphes représente le temps d'exécution de toutes les fonctions du projet (sans affichage) en fonction du nombre de couleurs.

Pour séquence aléatoire_rec :

nbcl	Temps_cpu(s)
25	0.002957
35	0.004152
45	0.005330
55	0.004873
65	0.015419
75	0.007538
85	0.009204
95	0.012383
105	0.008664
115	0.017365

Dim	Temps_CPU (s)
25	0.046
35	0.438
45	1.023
55	1.834
65	2.119
75	8.262
85	12.534
95	32.984
105	31.052
115	45.683

Pour séquence aléatoire rapide :

dim	temps_cpu(s)
25	0.000151
35	0.000425
45	0.000840
55	0.001941
65	0.001914
75	0.002599
85	0.002681
95	0.004758
105	0.006020
115	0.006095

nbcl	temps_cpu(s)
25	0.000045
35	0.000092
45	0.000115
55	0.000093
65	0.000214
75	0.000151
85	0.000230
95	0.000198
105	0.000230
115	0.000501

Pour les deux fonctions :

ncbl	nombre de coups
25	235
35	457
45	401
55	444
65	564
75	751
85	813
95	840
105	1080
115	1119

dim	nombre de coups
25	128
35	257
45	159
55	165
65	126
75	189
85	146
95	159
105	164
115	174

Nous nous intéressons ici à la courbe séquence aléatoire rec (violet) et séquence aléatoire rapide (vert).

Il est très clairement visible que le temps d'exécution de séquence aléatoire rec croit beaucoup plus que celle de séquence aléatoire rapide lorsque dim ou nbcl augmente. De plus, comme le montre le graphe: quelle que soit dim ou le nombre de couleurs la fonction séquence aléatoire rapide se finit plus rapidement que la fonction séquence aléatoire rec.

Cela s'explique par le fait que la fonction séquence aléatoire rec doit parcourir toutes les cases de la Zsg a chaque coup. Ainsi on remarque qu' en fonction du nombre de case, le temps cpu de cette fonction croît de façon quadratique.

Quant à la fonction séquence aléatoire rapide, la structure S Zsg nous permet de garder en mémoire la Zsg et la bordure sans avoir à la re parcourir. De ce fait, à chaque coup, on explore uniquement les nouvelles cases ajoutées. Ainsi, chaque case n'est parcourue qu' une fois durant le jeu. On constate notamment qu' en fonction du nombre de cases, le temps cpu pour cette fonction croît quasiment de façon linéaire (beaucoup plus visible sur le graphe seul de la fonction en annexe)

Autre remarque : Lors de l'exécution des deux fonctions, on a remarqué en effet que en fonction du nombre de couleur et de dim,, le nombre de coups des deux fonction est identique (cela est du au fait que les deux fonctions on la même stratégie). Ainsi, pour les deux fonctions, on a le même nombre de coups, qui sont plus lent pour séquence aléatoire rec que pour séquence aléatoire rapide, expliquant la différence de temps cpu.

Exercice 3– Strategie max-bordure

Q 3.3

Nous analysons toujours les graphes de comparaisons (voir plus haut), en s'intéressant cette fois-ci à la courbe séquence aléatoire rapide (vert), et max bordure (bleu).

La stratégie max bordure est nettement plus rapide que la stratégie aléatoire. Notamment lorsque nbcl varie le temps d'exécutions de max bordure varie peu, contrairement à séquence aléatoire rapide qui croit plus lorsque nbcl augmente. Ceci est conforme à l'attente, parce que plus le nombre de couleurs est élevé, moins la probabilité de tomber sur la couleur la plus efficace de manière aléatoire diminue or avec la stratégie max bordure, peu importe le nombre de couleurs, la fonction choisira la couleur qui sera la plus efficace.

Pour l’augmentation de dim, max bordure croît nettement moins vite que séquence aléatoire rapide. (cf courbes plus haut)
On peut aussi remarquer que les temps d'exécutions des deux fonctions sont proches pour des petites valeurs de dim et nbcl et se distancient au fur et à mesure que dim et nbcl augmente (pour des petites valeurs, le nombre d'itérations reste faible dans les deux cas).

Ces différences s'expliquent du fait qu’il faut environ deux fois moins de coups (d’itérations) à max bordure qu’il en faut à séquence aléatoire rapide, pour finir la grille.

Pour séquence aléatoire rapide :

ncbl	nombre de coups
25	235
35	457
45	401
55	444
65	564
75	751
85	813
95	840
105	1080
115	1119

dim	nombre de coups
25	128
35	257
45	159
55	165
65	126
75	189
85	146
95	159
105	164
115	174

Pour max bordure :

nbcl	nb de coups
25	56
35	56
45	64
55	75
65	71
75	74
85	85
95	73
105	85
115	80

dim	nb de coups
25	75
35	129
45	86
55	107
65	82
75	102
85	89
95	102
105	92
115	99

Ces chiffres confirmes donc qu'il faut moins de coups avec la stratégie max bordure qu'avec la stratégie aléatoire.

Ceci est dû à la stratégie des fonctions:

- stratégie max bordure : fait le choix de couleurs de sorte à ce qu'il y ait plus de cases qui s'ajoutent à Zsg (efficace), cela diminue donc le nombre de coups n'apportant qu'une ou deux cases dans la Zsg (peu efficace).
- stratégie aléatoire : choix des couleurs aléatoires, il y a donc plus de coups peu efficaces, donc la fonction nécessite plus de ces coups pour finir la grille

Q 4.2:

Pour max bordure zone:

Nombre de Couleurs (nbcl)	Nombre de Coups
25	56
35	57
45	64
55	75
65	71
75	74
85	85
95	73
105	86
115	80

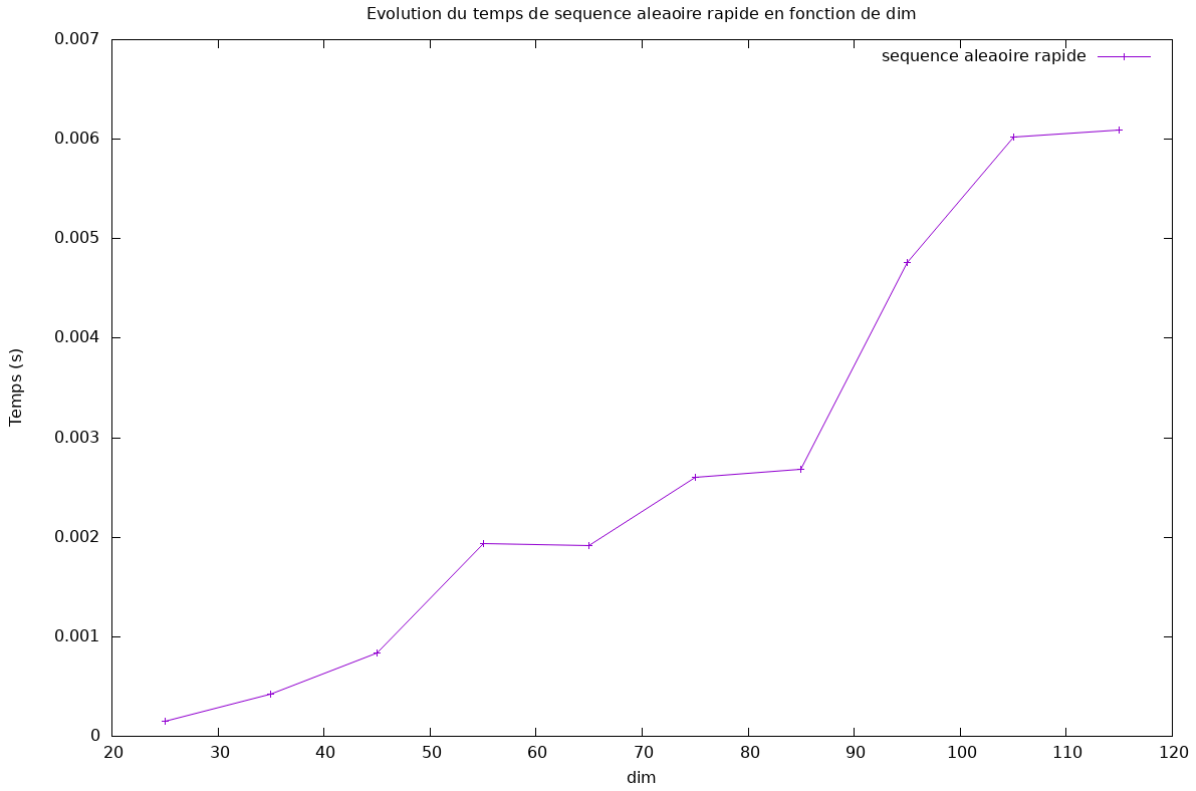
Dimension (dim)	Nombre de Coups
25	65
35	115
45	75
55	91
65	75
75	89
85	84
95	80
105	81
115	84

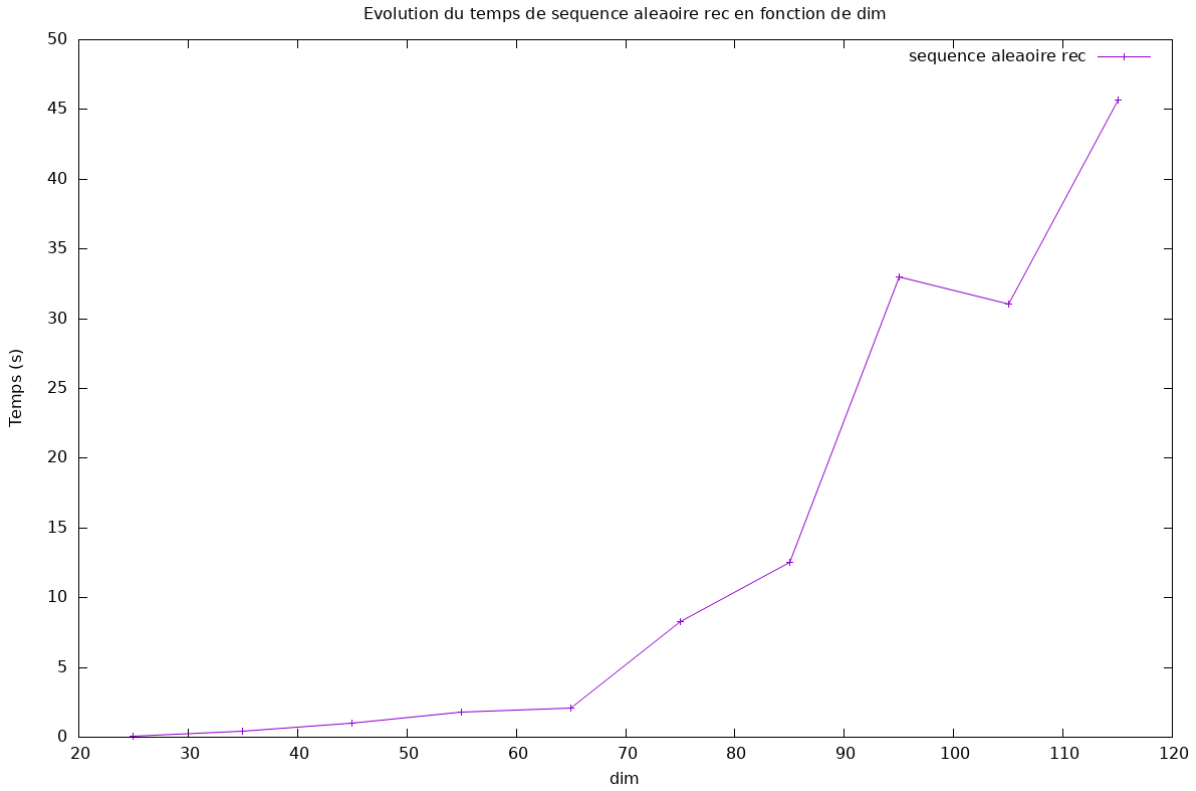
D’après les tableau en fonction du nombre de coups pour les deux fonctions, on constate que la stratégie MaxBordure Zone permet de terminer le jeu en moins de coups que la stratégie Max Bordure, tandis que le graphe nous révèle que la stratégie Max Bordure a un temps d'exécution plus rapide que celui de Max Bordure Zone. Cela s’explique par le fait que la stratégie max bordure zone choisit la couleur dans la bordure qui apporte la plus grande zone adjacente à la Zsg (et non seulement la couleur la plus présente dans la bordure immédiate). Ainsi, la stratégie max bordure zone prend une décision plus efficace, réduisant ainsi le nombre d'itérations nécessaires pour la terminer le jeu. Cela explique pourquoi la stratégie max bordure zone est plus efficace que max bordure en terme de nombre de coups.

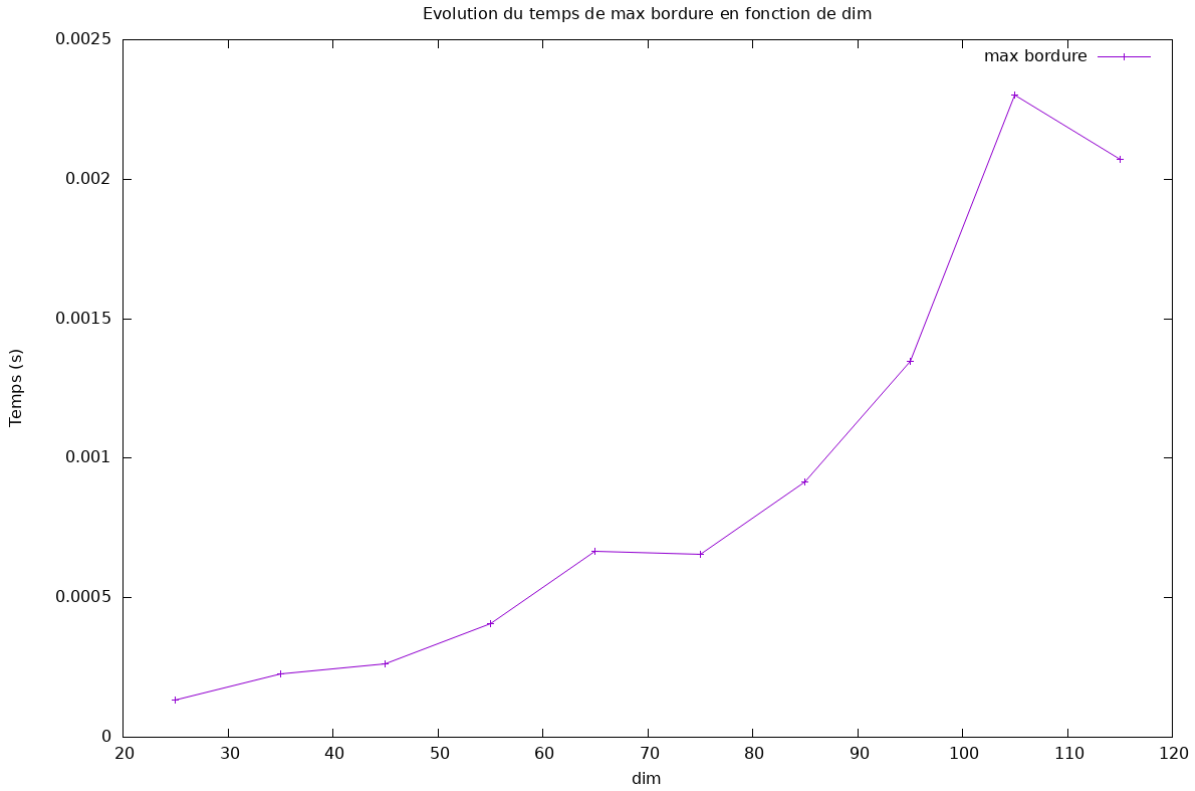
La différence de temps d'exécution entre les deux fonctions s'explique par le fait que la stratégie max bordure compte seulement les cases de la bordures, tandis que max bordure zone nécessite de calculer la zone de chaque couleur potentielle présente dans la bordure et ceux à chaque coup. Cette opération est plus coûteuse en temps cpu.

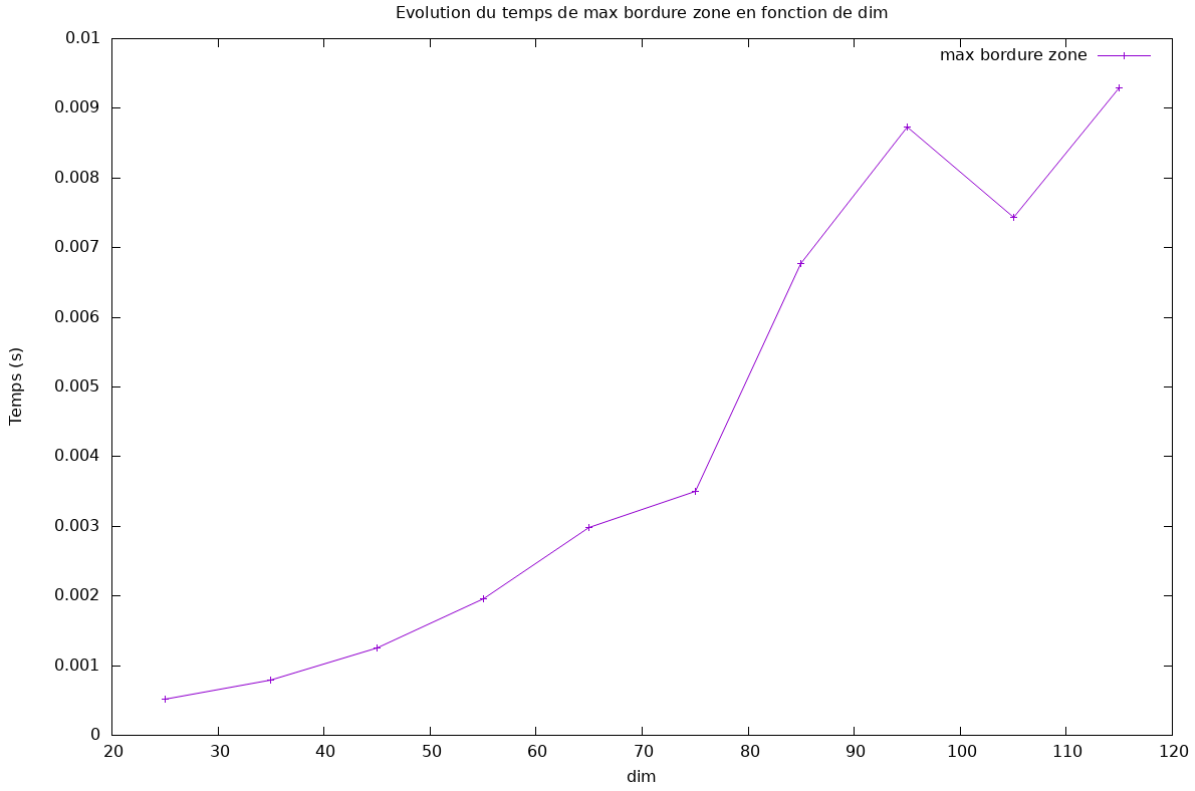
Pour conclure, on peut donc affirmer par ces deux fonctions que les critères nombre d’itération et vitesse d'exécution s’opposent souvent, notamment pour ces deux méthodes.

Annexes 1 : Courbes des temps d'exécution des fonctions (graphes individuels) lorsque dim varie

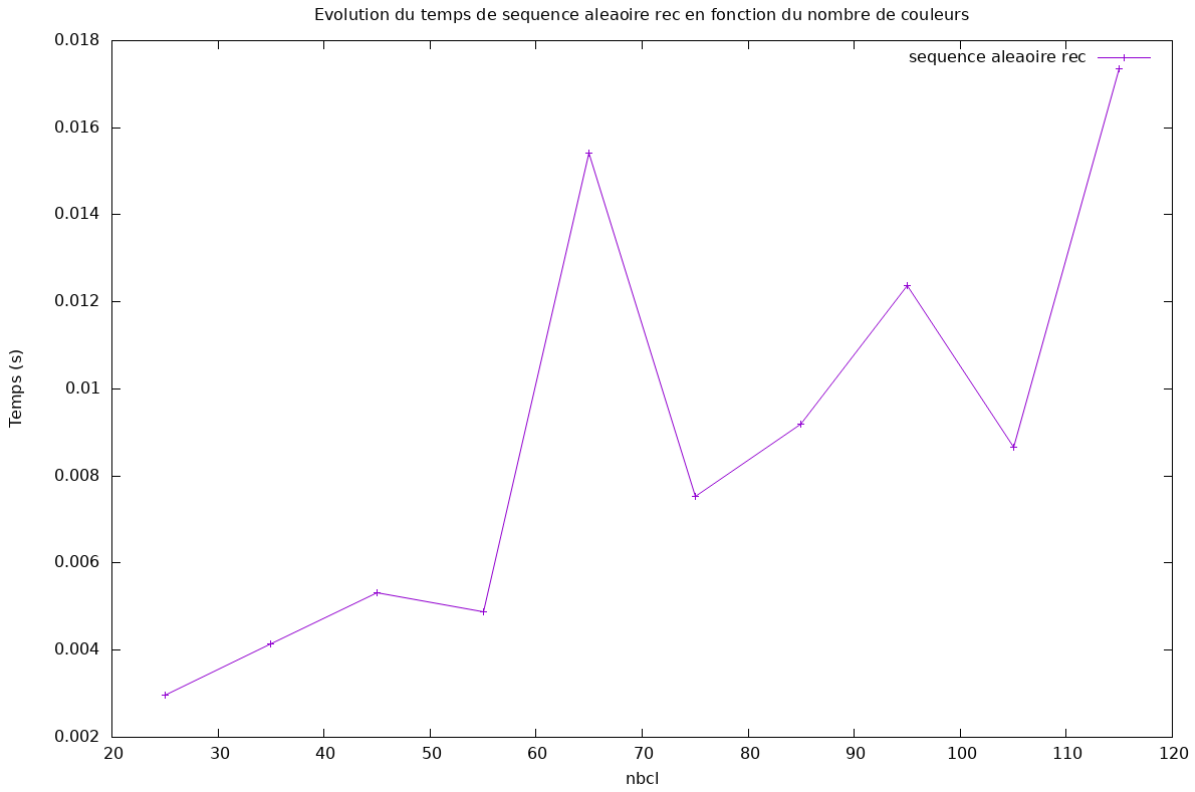


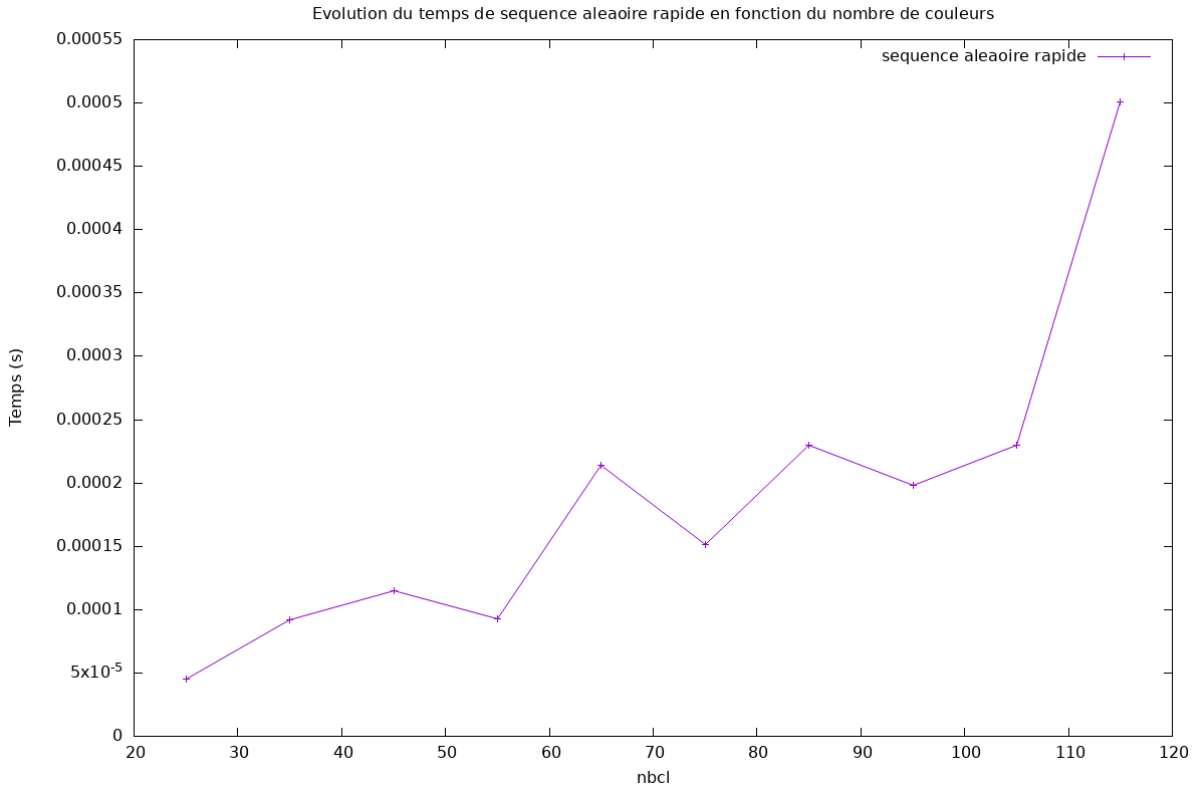


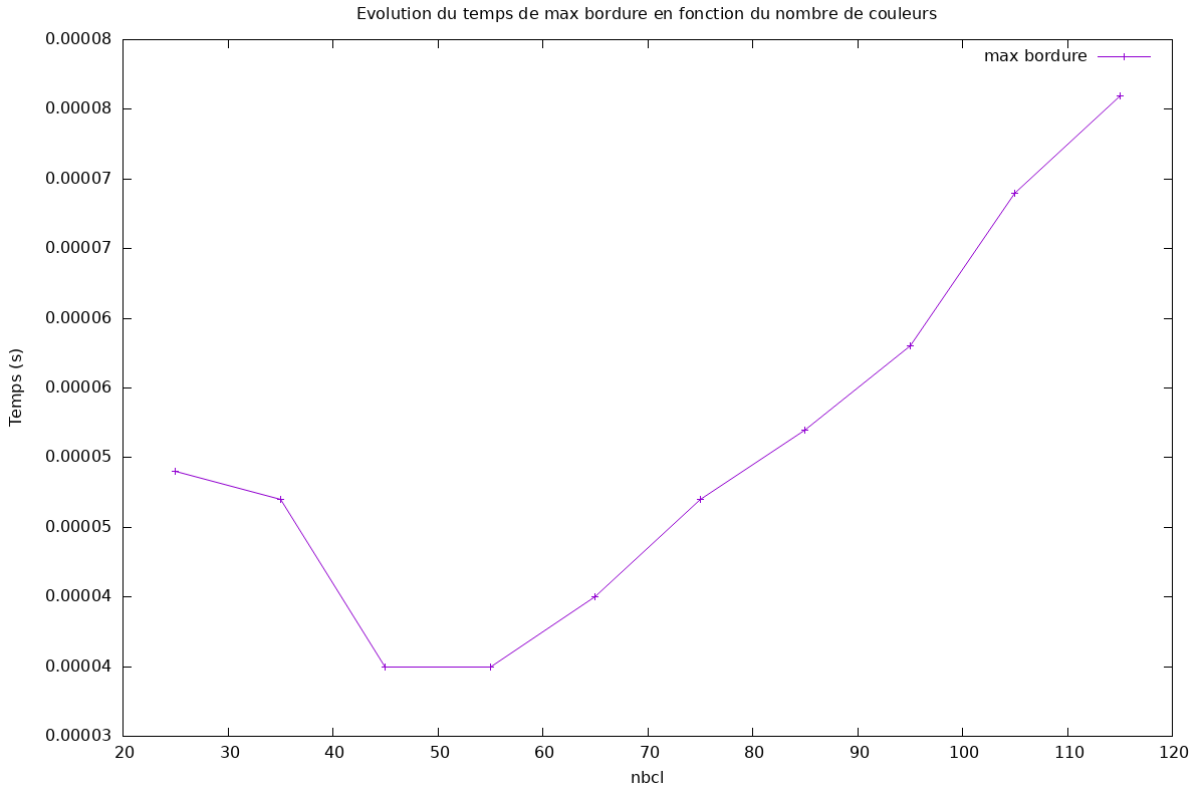

Evolution du temps de max bordure en fonction de dim
max bordure
Temps (s)
dim
0
0.0005
0.001
0.0015
0.002
0.0025
20
30
40
50
60
70
80
90
100
110
120

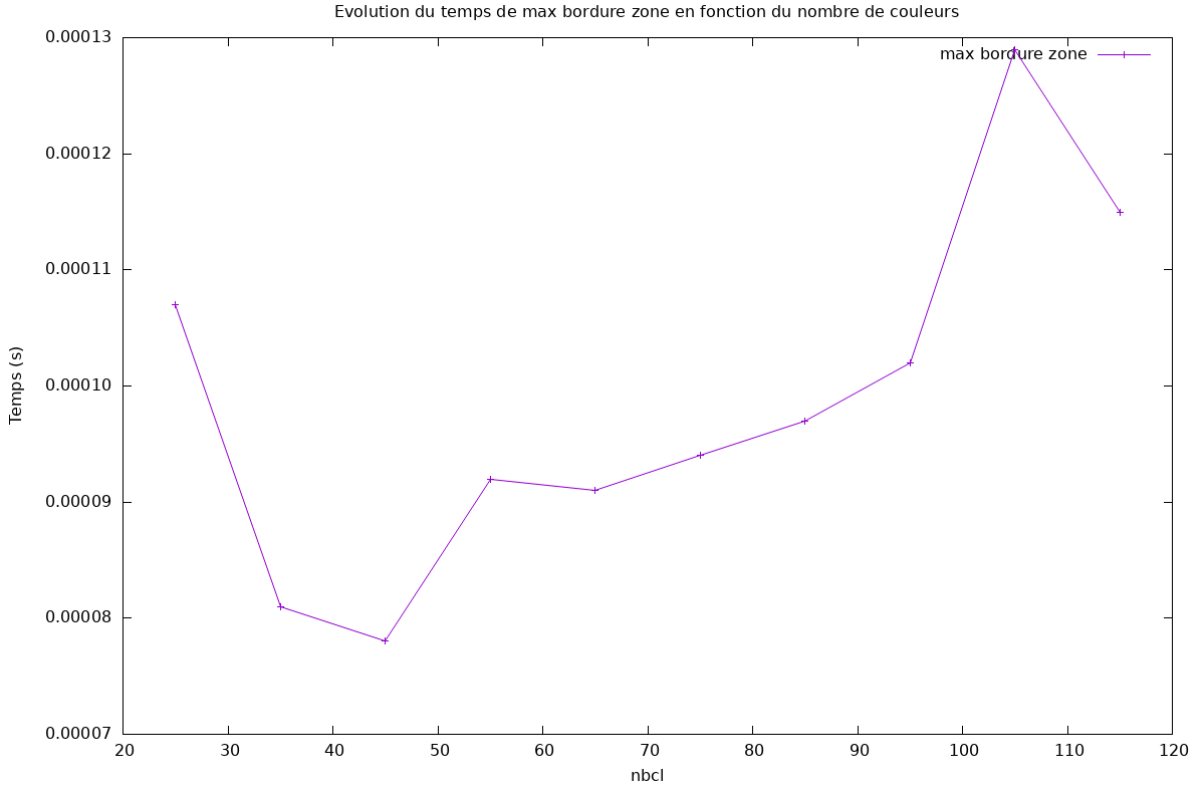

Evolution du temps de max bordure zone en fonction de dim
max bordure zone
Temps (s)
dim
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01
20
30
40
50
60
70
80
90
100
110
120

Annexes 2 : Courbes des temps d'exécution des fonctions (graphes individuels) lorsque nbcl varie






Evolution du temps de max bordure en fonction du nombre de couleurs
max bordure
Temps (s)
nbcl
0.00008
0.00008
0.00007
0.00007
0.00006
0.00006
0.00005
0.00005
0.00004
0.00004
0.00003
20
30
40
50
60
70
80
90
100
110
120


Evolution du temps de max bordure zone en fonction du nombre de couleurs
max bordure zone
Temps (s)
nbcl
0.00013
0.00012
0.00011
0.00010
0.00009
0.00008
0.00007
20
30
40
50
60
70
80
90
100
110
120