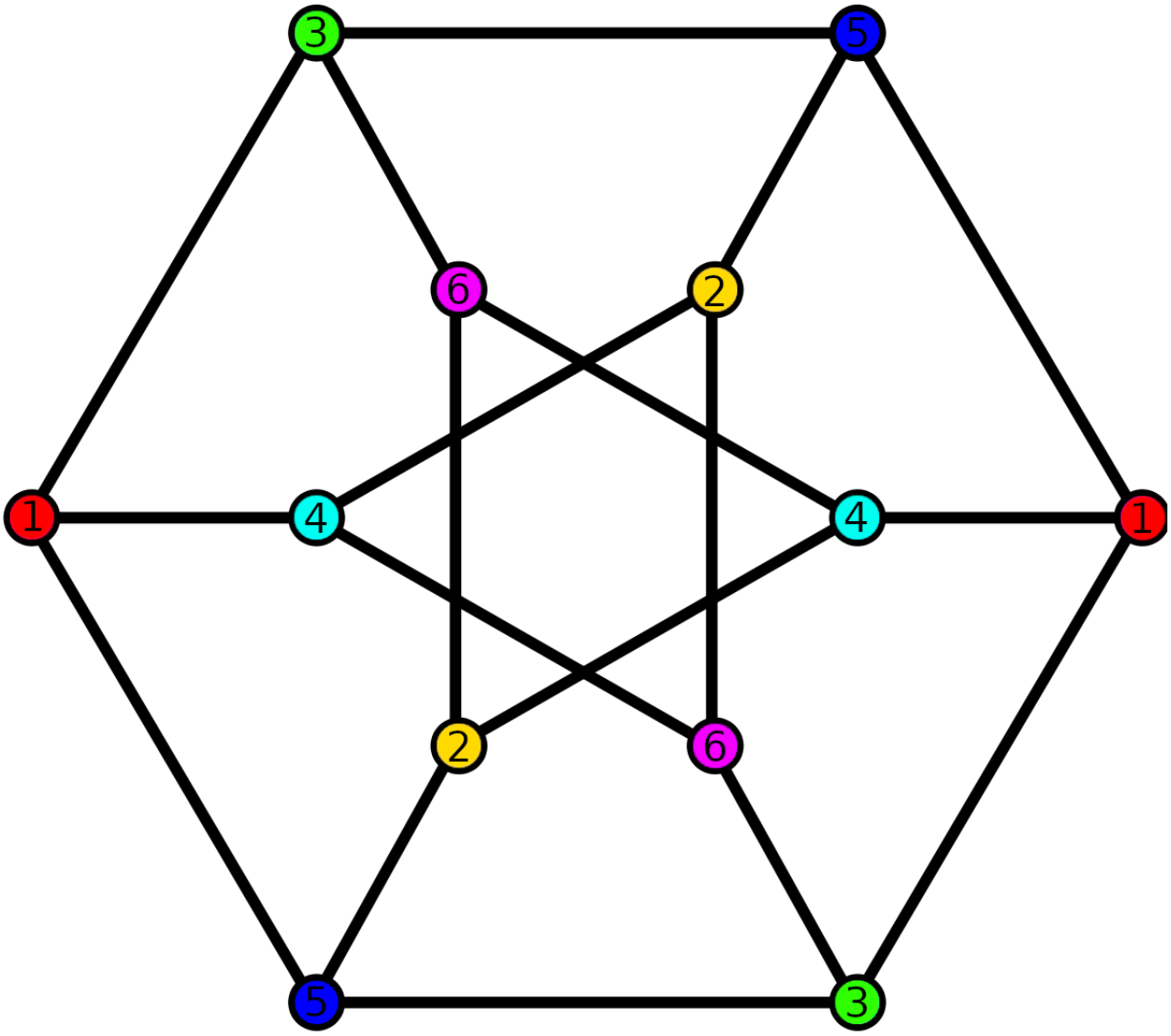
Compte rendu du projet de graphe

Clément Chupin Anonymity preserved

Avril 2021



Sommaire

I Introduction

1) Rappel de la problématique

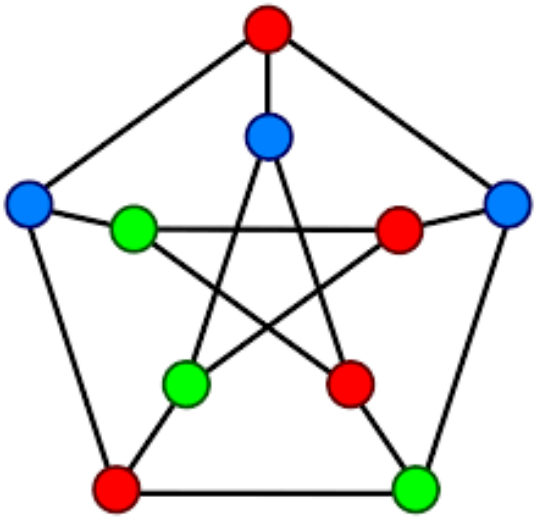
Nous devons à partir des algorithmes de coloration classiques ColorExact et DSATUR, créer une version de L(2,1)-coloration s'appliquant sur le graphe de Peterson. La difficulté de ce projet est de définir le graphe de Peterson mathématiquement via la fonction isVoisin(), ainsi que de parcourir les voisins de voisins pour attribuer une couleur à un sommet afin de satisfaire la L(2,1)-coloration.

II Le graphe et sa coloration

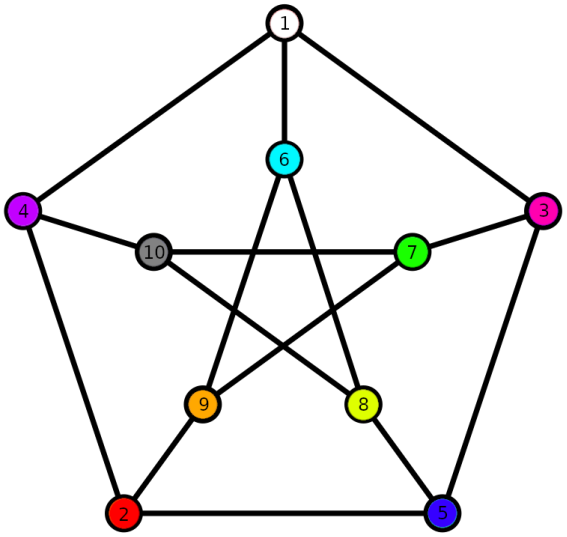
1) L(1,0)-coloration

Cette coloration est la plus classique dans les problématiques de coloration de graphe, elle est définie par :

- |couleur(a) - couleur(b)| >= 1 pour a et b voisins (couleur différente)
- |couleur(a) - couleur(b)| >= 0 pour a et b possèdant 1 voisin commun (pas de contrainte)



L(1,0)-coloration



L(2,1)-coloration

2) L(2,1)-coloration

Cette coloration présente pour caractéristique de devoir traiter les voisins de voisins d'un sommet pour pouvoir déterminer sa couleur :

- |couleur(a) - couleur(b)| >= 2 pour a et b voisins (couleur espacée d'au moins une couleur pour 2 voisins)
- |couleur(a) - couleur(b)| >= 1 pour a et b possèdant 1 voisin commun (couleur différente pour 2 sommets avec un voisin commun)

3) Graphe de Peterson

Pour identifier les voisins de chaque sommet, nous créerons la fonction isVoisin(int a, int b), qui détermine en fonction des paramètres du graphe de Peterson si les sommets sont voisins dans celui-ci.

```
if (a < b){
  if (a < nPeterson){
    if (a + 1 == b && b < nPeterson || a + nPeterson == b || b == (nPeterson - 1)
        && a == 0)
    {
      return true;
    }
  }
  else{
    if (nPeterson + (a + kPeterson)%nPeterson == b  || nPeterson + (b + kPeterson
        )%nPeterson == a){
      return true;
    }
  }
}
else if (a > b){
  return isVoisin(b, a);
}
return false;
```

Cette méthode peut paraître complexe dans sa programmation, mais elle est simple pour le programme à calculer car elle repose sur une succession de si [...] alors [...] .

4) Test d'une couleur pour un sommet

Pour les deux algorithmes, on détermine la plus petite couleur possible à chaque sommet. ColorExact se rappelle récursivement en passant au sommet suivant, DSATUR sélectionne le sommet à traiter en fonction de la saturation du graphe.

```
for (int i = 0; i < nbSommetVerif; i++)
{
  if (isVoisin(x, i))
  {
    if (!(abs(tabCouleur[i] - c) >= coloration Voisin))
    {
      return false;
    }
    for (int k = 0; k < nbSommetVerif; k++)
    {
      if (isVoisin(k, i) && !(abs(tabCouleur[k] - c) >= coloration VoisindeVoisin)
          )
      {
        return false;
      }
    }
  }
}
return true;
```

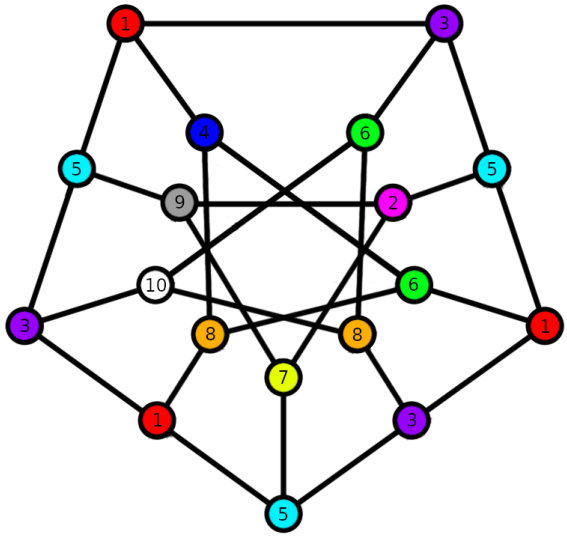
III Resultats obtenus

Voici quelques résultats de coloration obtenus :

n : 5 k : 2
– 1 – 3 – 5 – 2 – 4
– 6 – 7 – 8 – 9 – 10

n : 6 k : 1
– 1 – 3 – 5 – 1 – 3 – 5
– 4 – 6 – 2 – 4 – 6 – 2

n : 9 k : 3
– 1 – 3 – 5 – 1 – 3 – 5 – 1 – 3 – 5
– 4 – 6 – 2 – 6 – 8 – 7 – 8 – 10 – 9

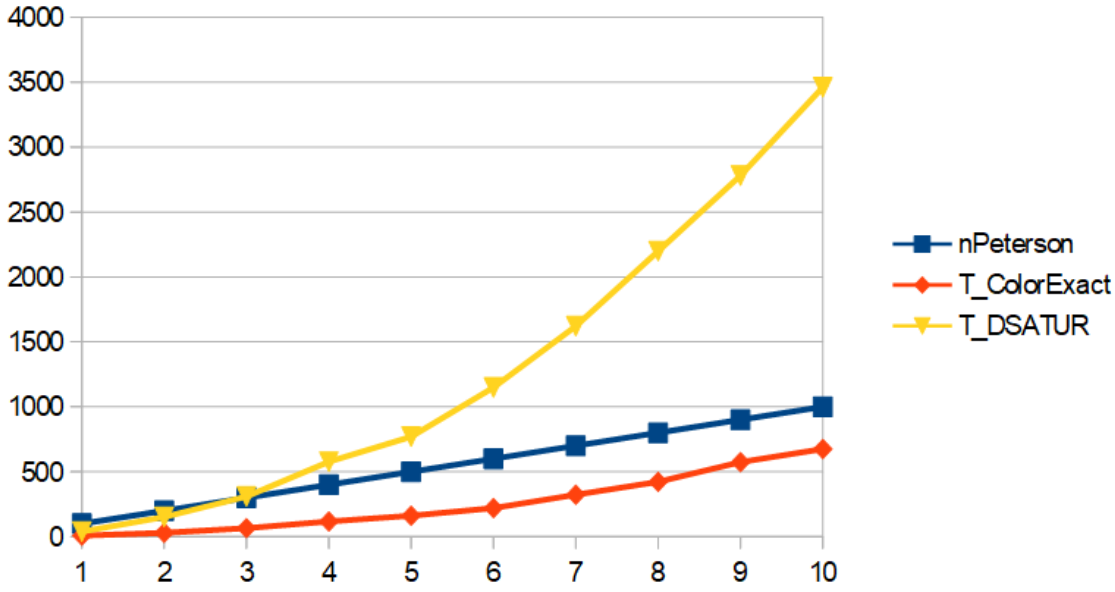


Les algorithmes nous donnent des résultats rapides et corrects pour des petites valeurs, mais augmentent rapidement en temps de calcul pour un plus grand nombre de sommets. On a compilé les résultats des deux algorithmes dans un tableau comparant plusieurs données pour vérifier la cohérence des résultats obtenus.

nPeterson	nbCouleurColorExact	nbCouleurDSATUR	tempsColorExact	tempsDSATUR	nbComparaisonColorExact	nbComparaisonDSATUR
100	269	185	63	96	23936	28584
200	280	191	208	319	47951	56049
300	271	182	459	654	71882	83183
400	272	185	921	1443	94139	111382
500	281	193	1327	1964	119528	138906
600	274	183	1857	2710	143645	165964
700	274	185	2589	3745	166446	194196
800	278	192	3361	4918	191313	221692
900	283	179	4290	6091	215851	248727
1000	279	185	5037	7535	236117	276970

Tableau de résultats de ColorExact vs DSATUR

Visuellement, malgré une différence de temps de calcul, les deux algorithmes évoluent de la même façon à l'augmentation du nombre de sommets.



Temps de calcul en fonction de nPeterson

1) ColorExact

Sommet par sommet, cet algorithme recherche la plus petite couleur convenant, et une fois trouvée, ColorExact s'appelle recursivement pour déterminer la couleur du prochain sommet.

```
int ColorExactRecursif(int x) // fonction recursive pour tester toutes les couleurs
    possible pour le sommet x
{
  if (x == nSommet)
  {
    return 0;
  }
  else
  {
    for (int c = 1; c <= nSommet; c++)
    {
      if (convient(x, c, 1))
      {
        couleur1[x] = c;
        return ColorExactRecursif(x + 1);
      }
    }
    return 1;
  }
  return 1;
}
```

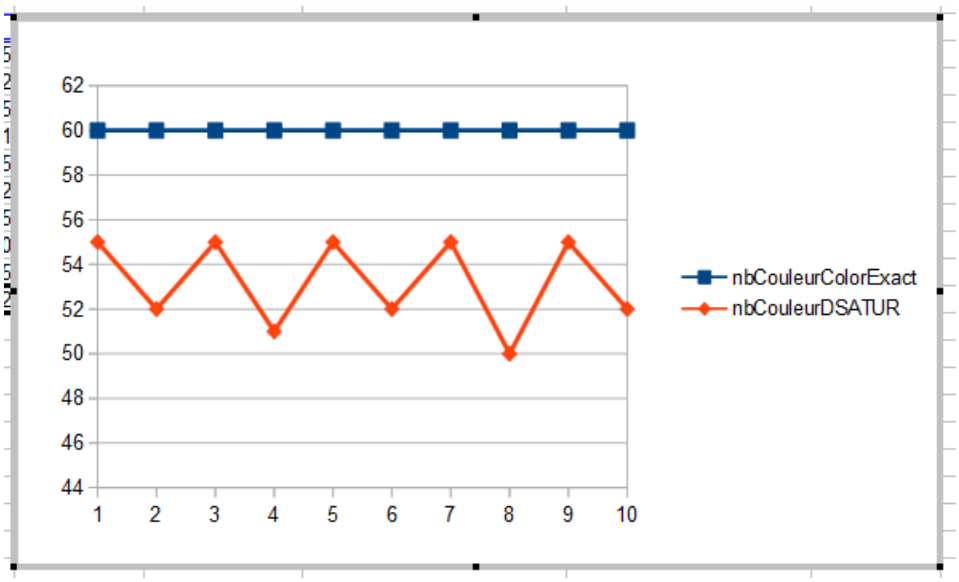
2) DSATUR

Cet algorithme est plus complexe mais permet, de déterminer l'ordre des sommets à colorier pour obtenir un resultat optimal. Par sa complexité, le temps de calcul est plus grand. En effet, pour sélectionner le prochain sommet à traiter, DSATUR met à jour la saturation des sommets en parcourant leurs voisins, puis il va de nouveau parcourir tous les sommets et sélectioner celui à la saturation maximale. ColorExact quant à lui se contente de prendre le sommet suivant.

```
for (int nb = 0;nb < nSommet;nb++) // tant qu'on a pas colorie tous les sommets
  {
    c = 1;
    x = dsatMax(); // on choisit le sommet de DSAT max non encore colorie
    while (!convient(x, c, 2))
    {
      c++;
    }
    [...] //mise a jour des saturations
  }
```

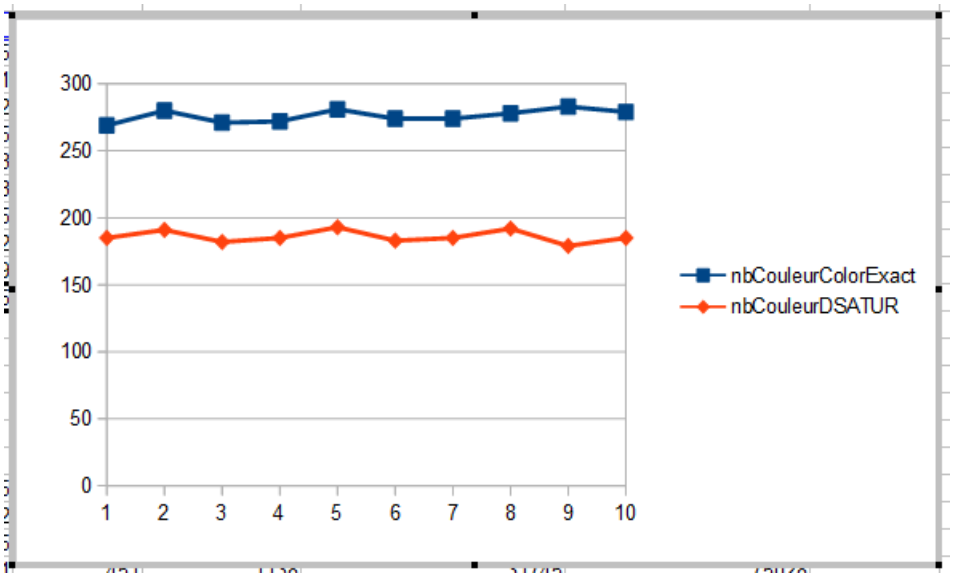
3) Efficacité générale :

Le résultat de la L(2,1)-coloration sur le graphe de Peterson est similaire à celle d'une coloration classique. DSATUR, par son fonctionnement, reste dans les deux colorations plus gourmand en calcul que ColorExact.



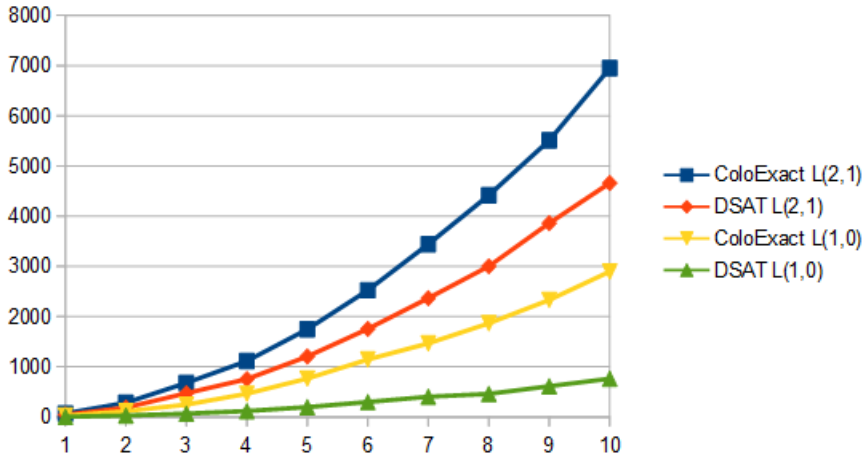
L(1,0)-coloration nb Couleur

Sur une coloration classique, DSATUR trouve moins de couleurs que ColorExact et a donc un meilleur résultat.



L(2,1)-coloration nb Couleur

DSATUR reste meilleur en colorations obtenues sur une L(2,1)-coloration.



Comparaion entre les temps de calcul des colorations et leurs algorithmes

Par le parcours de voisins de voisins, les deux algorithmes prennent plus de temps sur une L(2,1)-coloration. DSATUR est proportionellement plus long que ColorExact.

4) Limitations actuelles :

La L(2,1)-coloration implique une recherche sur les voisins de voisin, ColorExact trouve un moins bon résultat mais plus rapidement que DSATUR. Contrairement à une coloration classique, cette coloration est plus contraignante et teste plus de couleurs pour chaque sommet. Peu importe que l'on favorise la vitesse avec ColorExact, ou l'optimisation de la coloration via DSATUR, le temps de calcul évolue exponentiellement et ne permet un nombre de sommets conséquents.

IV Optimisation de ColorExact

1) Problématique générale des algorithmes

Par sa conception, l'augmentation du nombre de sommets augmente au carré le nombre de calculs. En effet, la L(2,1)-coloration implique de parcourir tous les sommets pour déterminer si ils sont voisins, puis de réitérer l'opération pour déterminer les voisins de voisins.
Attention la fonction isVoisin() ne change pas en temps de traitement selon nPeterson.

```
for (int i = 0; i < n; i++)
{
  if (isVoisin(x, i))
  {
    [...] voisin de x
    for (int k = 0; k < n; k++)
    {
      [...] voisin de voisin de x
    }
  }
}
```

2) Optimisation possible

On pourrait stocker les voisins de chaque sommet dans un tableau de tableaux, ce qui permettrait de parcourir seulement les sommets voisins sans avoir à vérifier si chaque sommet est voisin à X.

```
int voisinV=0,voisinDeVoisinV=0;
for (int i = 0; i < degreMax; i++)
{
    voisinV=voisin[x][i];
    if(voisinV<x){
      if (!(abs(couleur2[voisinV] - c) >= coloration Voisin))
      {
          return false;
      }
      for (int k = 0; k < degreMax; k++)
      {
          voisinDeVoisinV=voisin[voisinV][k];
          if (voisinDeVoisinV<x && !(abs(couleur2[voisinDeVoisinV] - c) >=
              coloration VoisindeVoisin))
          {
              return false;
          }
      }
    }
}
return true;
```

3) Efficacité

ColorExact : crash à nPeterson = 1500
Raison : un temps de calcul trop long et exponentiel.

ColorExactOpti : crash à nPeterson = 20000 (+ de 10 fois sa version antérieure)
Raison : un stockage mémoire trop important :
Pour nPeterson = 20 000 :

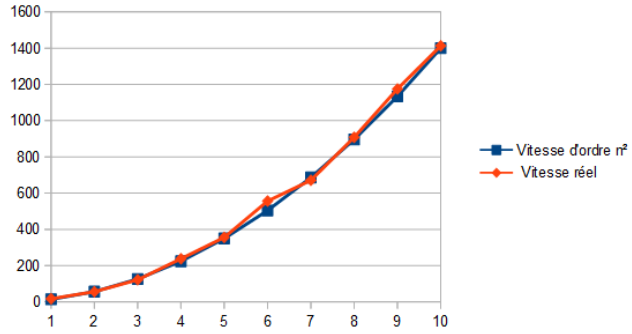
avec nSommet = 40 000, degréMax du graphe Peterson = 3
taille de stockage = nSommet × degréMax + stockageColoration
= 40 000 × 3 + 40 000
= 160 000 int

Soit 640.000 octets alloués et libérés plusieurs fois à la suite, la programmation en C n'étant pas optimale, le programme a du mal à aller plus loin.

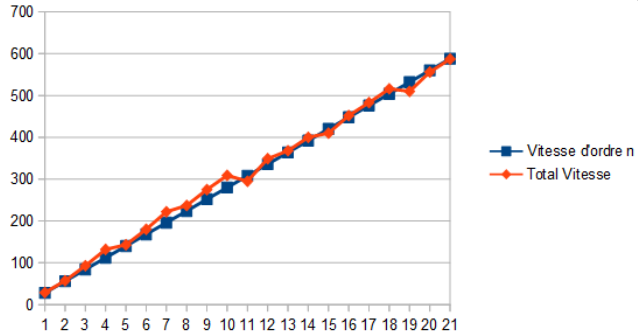
4) Complexité

De manière expérimentale, on a pour complexité :

Pour ColorExact O (n^2) **Pour ColorExactOpti O (n)**



L(1,0)-coloration



L(2,1)-coloration

La différence est énorme mais s'explique par un parcours similaire de chaque sommet, un graphe plus grand n'augmente pas la difficulté pour traiter chaque sommet dans la version optimisée de ColorExact.

5) Vérifications

Devant un tel résultat, nous avons manuellement vérifié les colorations une par une pour les petits graphes de Peterson, puis créé une méthode comparant les colorations obtenues, les deux algorithmes donnent un même résultat, mais dans des temps de calculs totalement différents.

nPeterson	Temps_ColorExact	Temps_ColorExact_opti	Différence entre les deux colorations
100	66	0	0
200	245	3	0
300	602	6	0
400	1021	3	0
500	1614	3	0
600	2167	4	0
700	2843	8	0
800	3575	10	0
900	4578	11	0
1000	5900	13	0

Comparaison du temps de calcul et de la coloration entre ColorExact et son optimisation

Les deux algorithmes produisent la même coloration, mais la manière optimisée permet un résultat plus rapide, au prix d'un espace mémoire occupé plus grand et d'un enregistrement au préalable des voisins de chaque sommet.

V Conclusion

Ce projet nous a permis d'appliquer différentes méthodes au traitement d'un graphe, chacune d'entre elles possèdant ses avantages et ses faiblesses. On a pu réaliser qu'il était important de discerner clairement les contraintes et spécificités des problèmes présentés pour pouvoir concevoir des algorithmes de manière à avoir un résultat optimal dans un temps de calcul raisonnable.