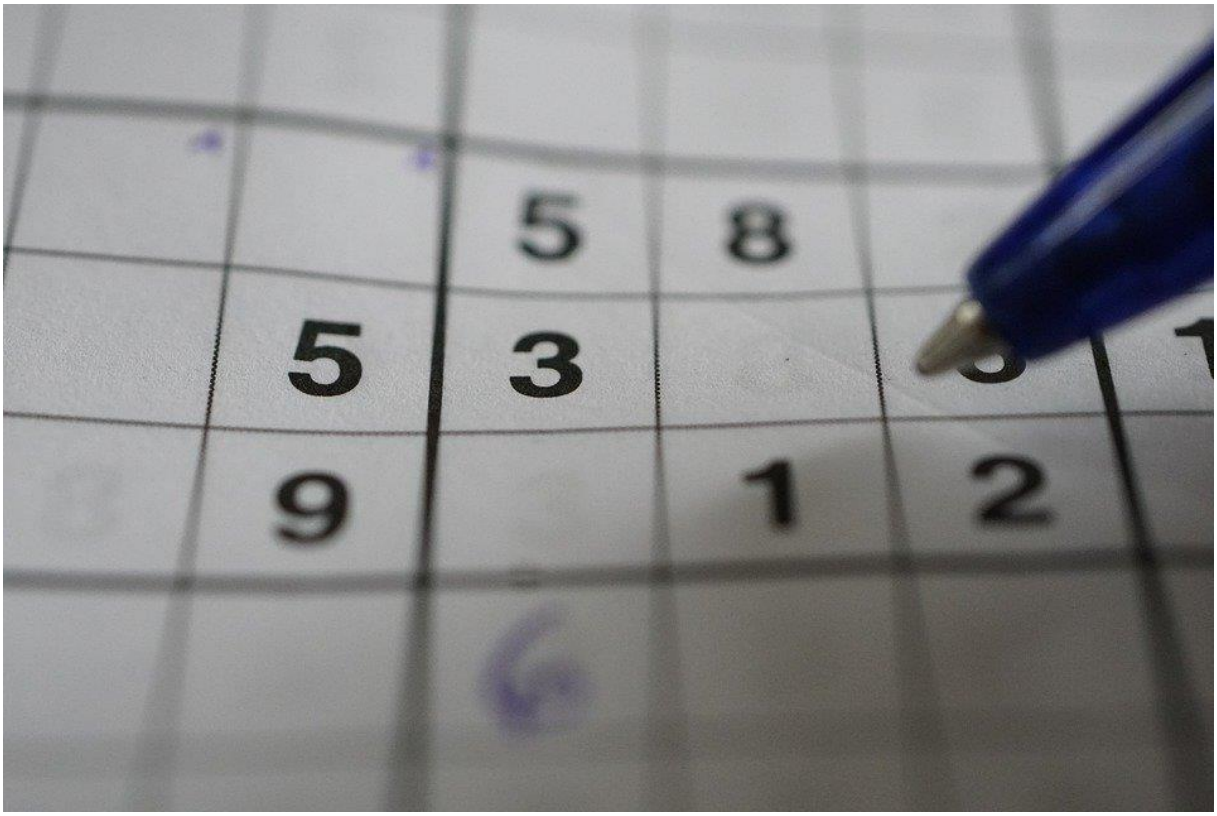
Faculté des Sciences
Aix*Marseille Université

Rapport Complexité TP2



Auteurs : THAI Ba Tuan / WERY Alexandre / LUU-VU Régis / SENES Jonathan
Enseignant responsable : M. PORRECA ANTONIO
Année : 2020 - 2021

Table des matières

Mini-Projet 1

Algorithme

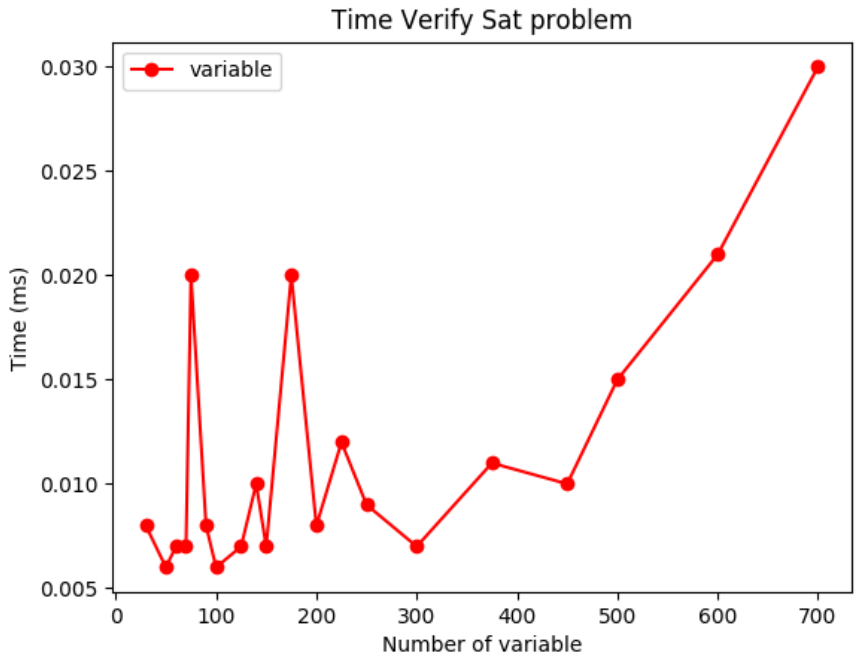
```
int verifySAT(struct SATProblem problem, struct Affectation affec) {
   int clauseIndex = 0;
   int variables = problem.variables;
   for(clauseIndex = 0; clauseIndex < problem.clauses; clauseIndex++) {
      int check = 0;
      for(int i = 0; i < problem.variables; i ++) {
         if(problem.value[clauseIndex*variables + i]==0) continue;
         if(problem.value[clauseIndex*variables + i]==1) {
            if(affec.affec[i] == 1) {
               check = 1;
               break;
            }
         } else if (problem.value[clauseIndex*variables+ i] == -1) {
            if(affec.affec[i] == 0) {
               check = 1;
               break;
            }
         }
      }
      if(check==0) return 0;
   }
   return 1;
```

⇨ **Complexité** : O (clauses * variables)

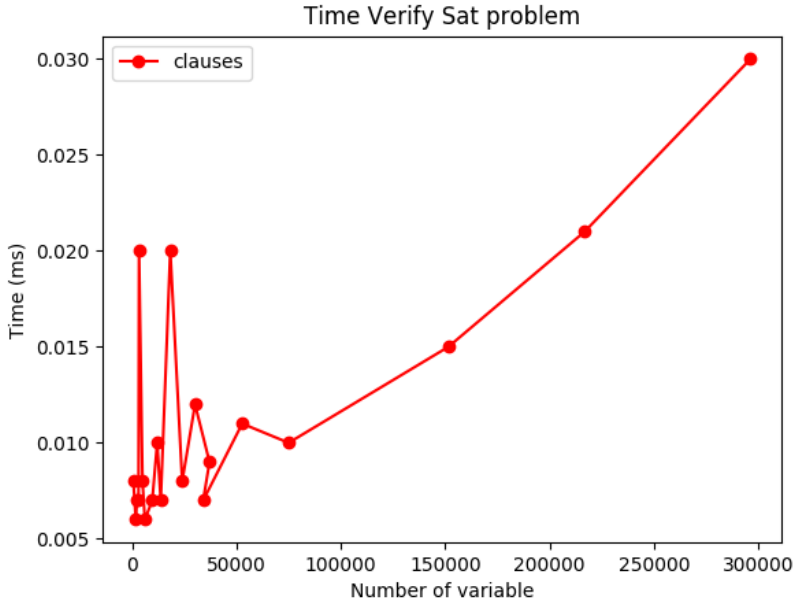
Résultats expérimentaux

Le nombre de variables varie entre 30 et 700.
Le nombre de clauses Clause varie entre 515 to 296305.

- Nombre de variables en fonction du temps.



- Nombre de clauses en fonctions du temps.



Mini-Projet 2

Algorithme

Entrée : Un graphe G = (V, E, k)
Sortie : Est-ce qu'il existe dans le Graphe G (V, E) une zone-vide de taille k ?

Nous pouvons définir une liste de variables xiv pour tout $1 \leq i \leq k$ et tout $v \in V$. Ainsi, nous pouvons interpréter chacune de ces variables xiv comme « v est le i -ème sommet de la Zone-Vide. xiv peut être VRAI (1) ou FAUX (0).

Contraintes : On sait que pour tout i (où $1 \leq i \leq k$), il existe un ième sommet dans Z (où Z est la Zone-Vide) :

$$\forall \{i \mid 1 \leq i \leq k\} \exists \text{ xiv}$$

Pour chaque arête $(v, w) \in E$, v ET w ne peut pas être à LA FOIS dans la Zone-Vide :

$$\forall \{i,\ j \mid i \neq j\}, \forall\{v, w \in V \mid (v, w) \in E \wedge v \neq w\} \neg xiv \vee \neg xjw$$

Pour chaque i, j (où $j \neq i$), nous faisons en sorte que le i-ème sommet est différent du j-ème sommet. Autrement dit, nous savons qu'un sommet v ne peut pas être à la fois le i-ème et le j-ème sommet de la Zone-Vide. Cela signifie également que deux sommets différents ne peuvent pas tous deux être le i-ème sommet de la Zone-Vide. Nous pouvons ainsi écrire ceci comme une contrainte en deux parties pour plus de simplicité :

$$\forall [i, j \mid i \neq j] \forall [v \in V] \neg xiv \vee \neg xjv$$

$$\forall i \forall [v, w \in V \mid v \neq w] \neg xiv \vee \neg xiw$$

Réduction réalisée : n sommets en n*k variables.
Complexité : $O(n*k + k^2 * n^2 + k * n^2 + k^2 * n) = 0(k^2 * n^2)$

Implémentation de l'algorithme de résolution

MINISAT

Après réduction du problème de Zone-vide dans un Graphe (V, E, K) vers un problème SAT encodée dans un fichier DIMACS CNF, nous pouvons utiliser le SAT Solver MINISAT pour résoudre ce problème en tapant la commande :

minisat cnf.txt sat.txt

Où **cnf.txt** est un fichier DIMACS CNF (au format txt) obtenue grâce à la précédente réduction et sat.txt est le résultat du problème résolu par MINISAT. Pour afficher ce résultat, nous utilisons un programme Python pour mettre en evidence une éventuelle solution :

python3 displayzonevide.py sat.txt k n

Où **sat.txt** est le fichier solution à traiter, k la taille de la zone testée et n le nombre de sommets.

Brute – Force

Comme nous le savons, transformer le problème Zone-Vide d'un Graphe (V, E, k) en forme standard SAT DIMACS aura en tout |V|*k variables. Il faut diviser toutes ces variables en k tableaux, chacun contenant en |V| éléments. Dans chacun de ces tableaux, il faut choisir un élément différent à entrer dans la zone vide (nous avons donc k sommets). Le nombre d'options possibles est : n^k, d'où la complexité de l'algorithme à $O(n^k)$
Le pseudo code de l'algorithme est le suivant :

```
Algo-Brute-Force (G, V, k):
    value = tableau of size k
    n = |V|
    for i:=0 to n^q do:
        temp = i
        q = k - 1
        # transform i to base n
        for j:=0 to k do:
            value[j] = temp/n^q
            temp = temp % n^q
            q = q - 1

        Affectation affec;
        for t := 0 to k do:
            affec[t*n+value[t]] = 1
        if verify(affec, problem) do
            return true
```

Nous avons également un autre algorithme brute-force avec une approche itérative en utilisant un tableau de n * k éléments que nous affectons avec des 0 et 1. Enfin, nous vérifions si la solution est satisfaisante. Cependant, la complexité d'un tel algorithme *brute force* est $2^{n * k}$. Il est en effet très élevé. Il est donc préférable d'utiliser l'algorithme *brute force* dont la complexité est n^k.

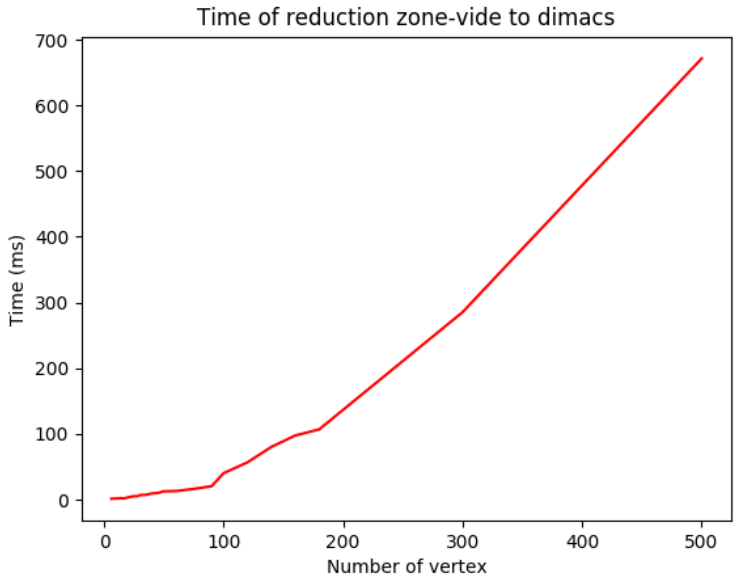
Analyses des courbes

Données

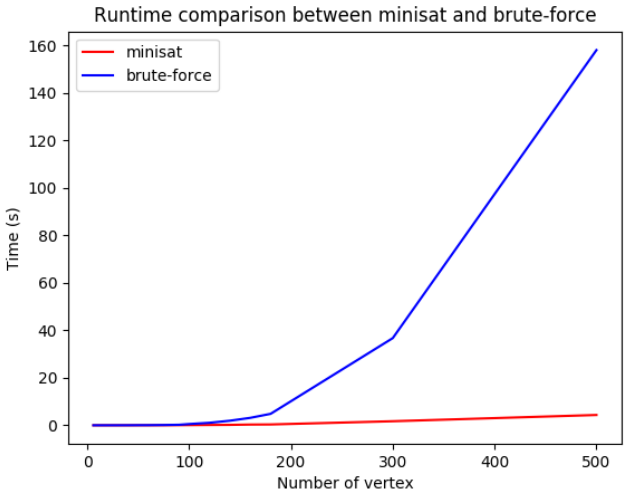
Pour générer des données de test de graphes, nous avons utilisé un outil qui des graphes et qui a été construit par Mark Brockington pour le concours DIMACS Challenge 1993. Les données générées seront constituées de graphes avec des sommets allant de 6 à 500 sommets. Les détails des données de test sont contenus dans le répertoire /TestData/datagraph.

Résultats expérimentaux

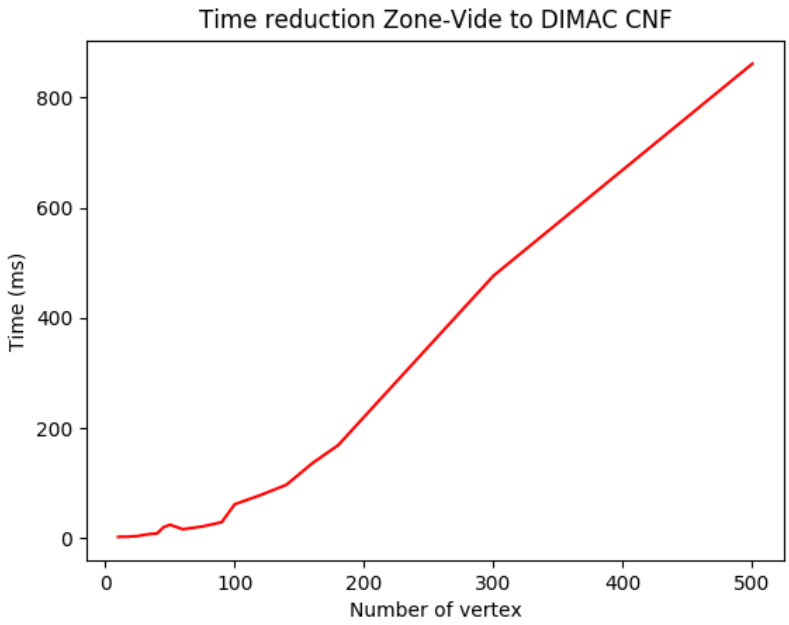
- Pour k=3
  - Temps de calcul de la réduction Zone Vide -> SAT :



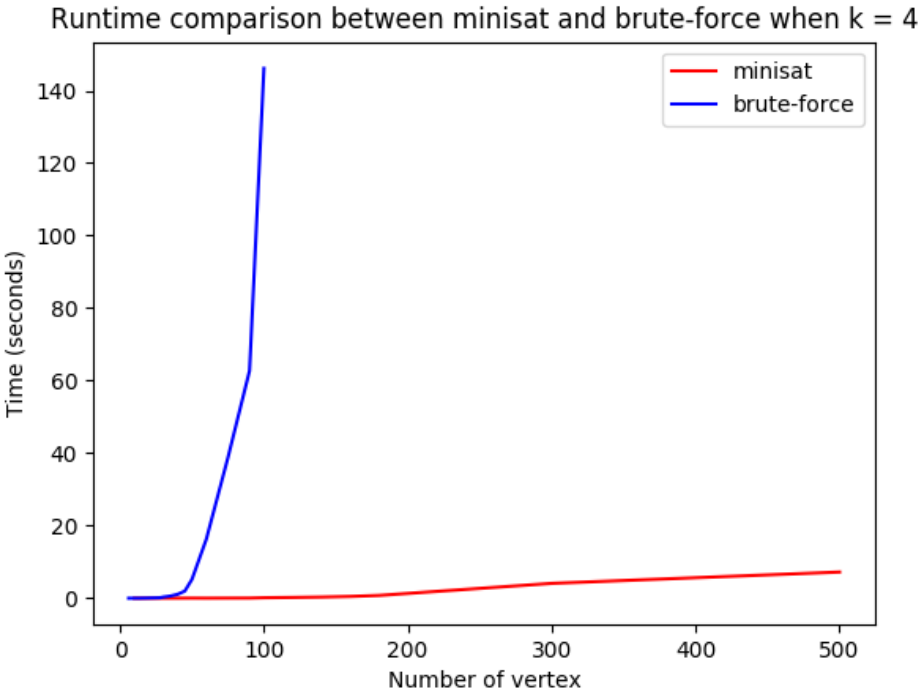
  - Temps de calcul total pour résoudre le problème :



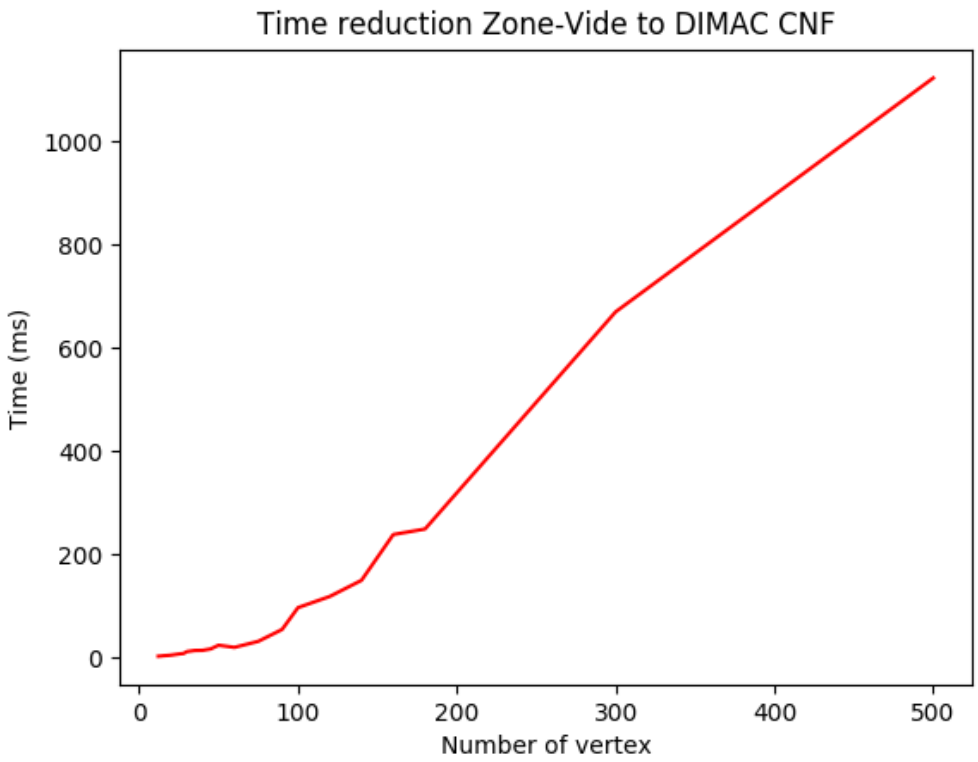
- Pour k=4
  - Temps pris pour la réduction Zone Vide -> SAT :



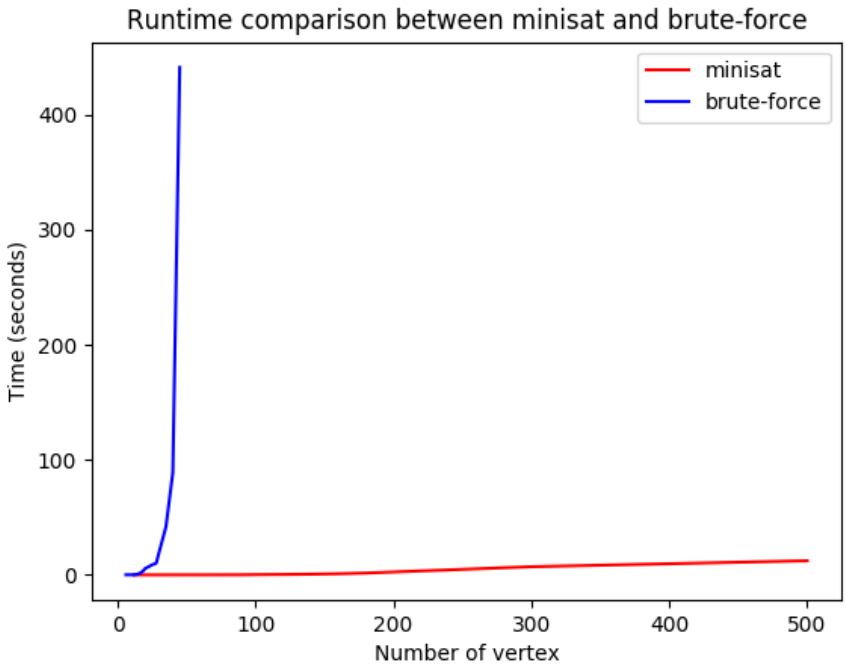
  - Temps de calcul total pour résoudre le problème :



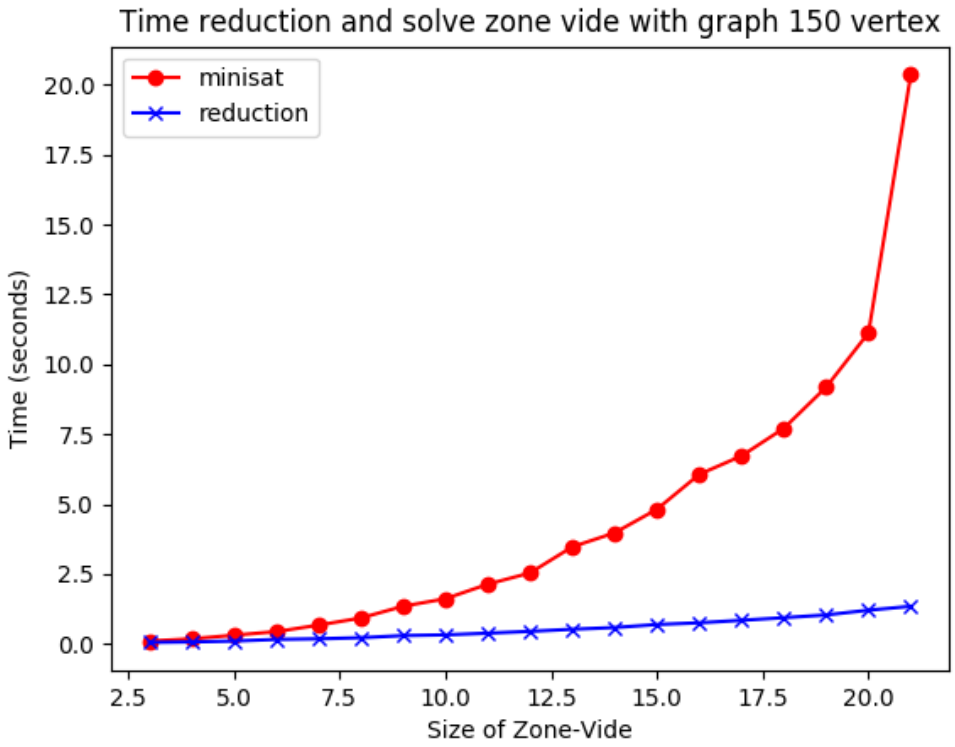
- k = 5
  - Temps de calcul de la réduction Zone Vide -> SAT :



  - Temps de calcul total pour résoudre le problème :



Temps de calcul de la réduction de la résolution du problème de Zone-vide avec un graphe de |V|=150, |E| = 2320. Nous choisissions ici, des zones vides dont la taille va de 3 à 21.



Miniproject 3 - Réduction de Sudoku à SAT

Algorithme

Un sudoku peut facilement être représenté par un problème SAT, même s'il requiert un nombre assez significatif de variables propositionnelles. Si par exemple, nous utilisions des variables avec des domaines finis arbitraires, alors n^2 x n^2 variables dans un domaine [1… n^2] semble être l'option la plus adéquate. Cependant, encoder un sudoku en CNF demanderait n^2 x n^2 x n^2 = n^6 variables propositionnelles. Pour chaque entrée dans la grille S de dimension n^2 × n^2, nous associons n^2 variables. Utilisons la notation s_{xyz} pour désigner des variables à manipuler. Une variable s_{xyz} est assigné à VRAI si et seulement si l'entrée à la ligne x et à la colonne y est assigné au nombre z. Par exemple, s_{559} = 1 signifies que l'élément S[5, 5] de la grille est 9. Naturellement, les entrées pré-assignées seront représentés par des clauses unitaires.

- Lorsqu'il s'agit d'une entrée pré-assigné :

$$\bigwedge_{x=1}^{n^2} \bigwedge_{y=1}^{n^2} S_{xyz}$$

  On fait ça pour chaque cellule (x, y) dont la valeur est déjà assignée à z.

- Lorsqu'il y a au moins un nombre à chaque entrée :

$$\bigwedge_{x=1}^{n^2} \bigwedge_{y=1}^{n^2} \bigvee_{z=1}^{n^2} S_{xyz}$$

- Lorsque chaque nombre apparait au plus une fois par ligne :

$$\bigwedge_{y=1}^{n^2} \bigwedge_{z=1}^{n^2} \bigwedge_{x=1}^{n^2-1} \bigwedge_{i=x+1}^{n^2} \neg S_{xyz} \vee \neg S_{iyz}$$

- Lorsque chaque nombre apparait au plus une fois par colonne :

$$\bigwedge_{x=1}^{n^2} \bigwedge_{z=1}^{n^2} \bigwedge_{y=1}^{n^2-1} \bigwedge_{i=y+1}^{n^2} \neg S_{xyz} \vee \neg S_{xiz}$$

- Lorsque chaque nombre apparait au plus une fois dans chaque n x n sous-grille :

$$\bigwedge_{z=1}^{n^2}\bigwedge_{i=0}^{n-1}\bigwedge_{j=0}^{n-1}\bigwedge_{x=1}^{n}\bigwedge_{y=1}^{n}\bigwedge_{k=y+1}^{n}(\neg S_{(3i+x)(3j+y)z} \vee \neg S_{(3i+x)(3j+k)z})$$

ET

$$\bigwedge_{z=1}^{n^2}\bigwedge_{i=0}^{n-1}\bigwedge_{j=0}^{n-1}\bigwedge_{x=1}^{n}\bigwedge_{y=1}^{n}\bigwedge_{k=y+1}^{n}\bigwedge_{l=1}^{n}\neg S_{(3i+x)(3j+y)z} \vee \neg S_{(3i+k)(3j+l)z}$$

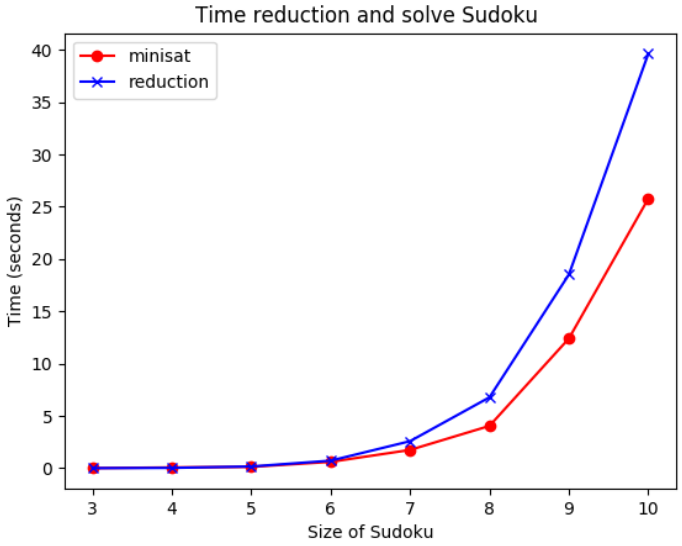
Complexité : $\boldsymbol{O(n^4 + n^8 + n^8 + n^8 + n^7 + n^8) = O(n^8)}$

Analyse des courbes

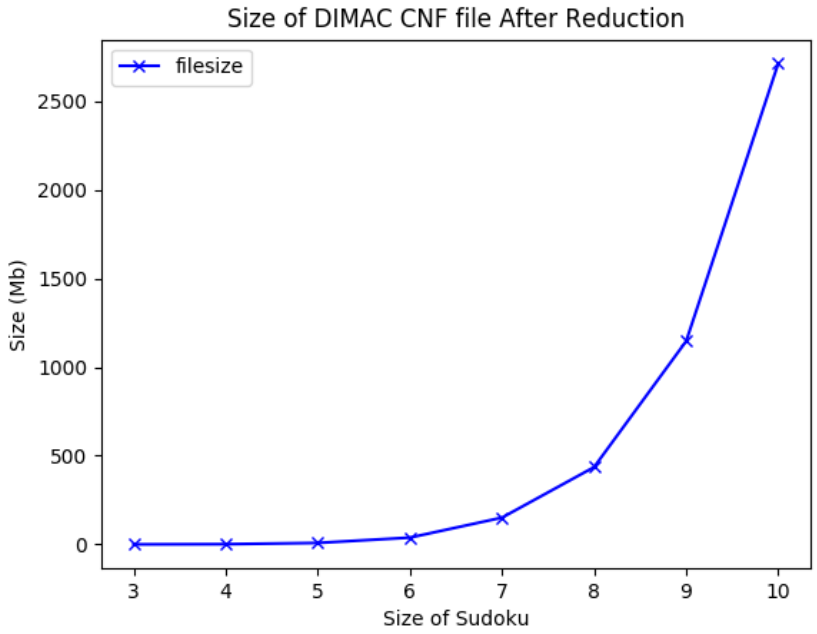
Données

Pour les problèmes de sudoku, les n nombres d'entrée utilisés sont 3, 4, 5, …, 10. Par conséquent, la taille du sudoku sera : 9 x 9, 16 x 16, …, 100 x 100. Les données proviennent du site Web : https : //sudokugeant.cabanova.com/ et se verront attribuer au préalable quelques numéros pré-assignés.

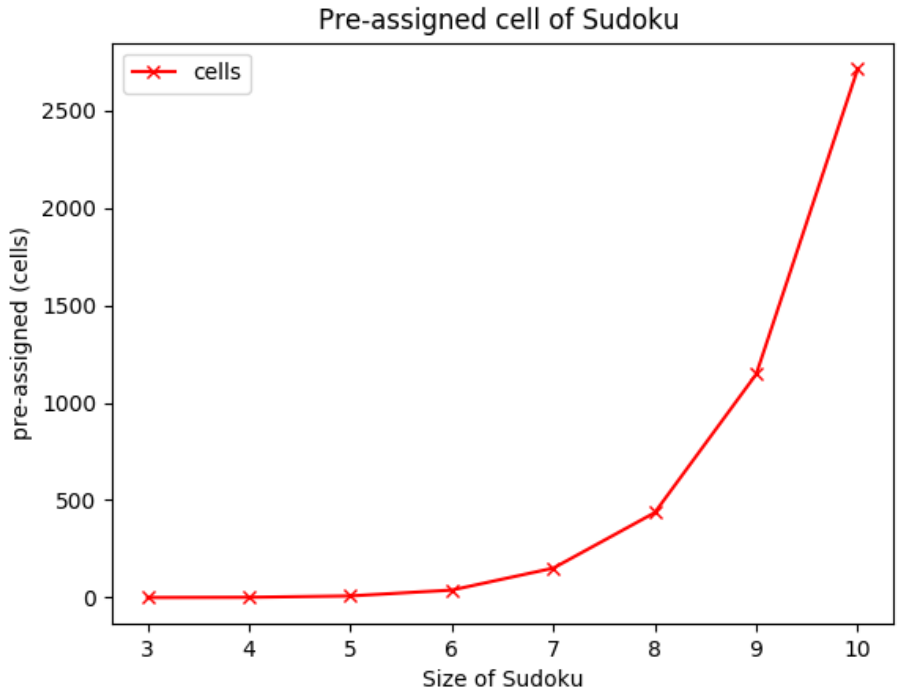
Résultats expérimentaux



- Taille du sudoku en fonction de la taille du fichier DIMAC :



- Taille du sudoku en fonction des cellules pré-assignées :



Bibliographie

[1] A. J. Hunt, « Polynomially Reducing the Clique Problem to the SAT Problem with DIMACS Encoding. » 2018.

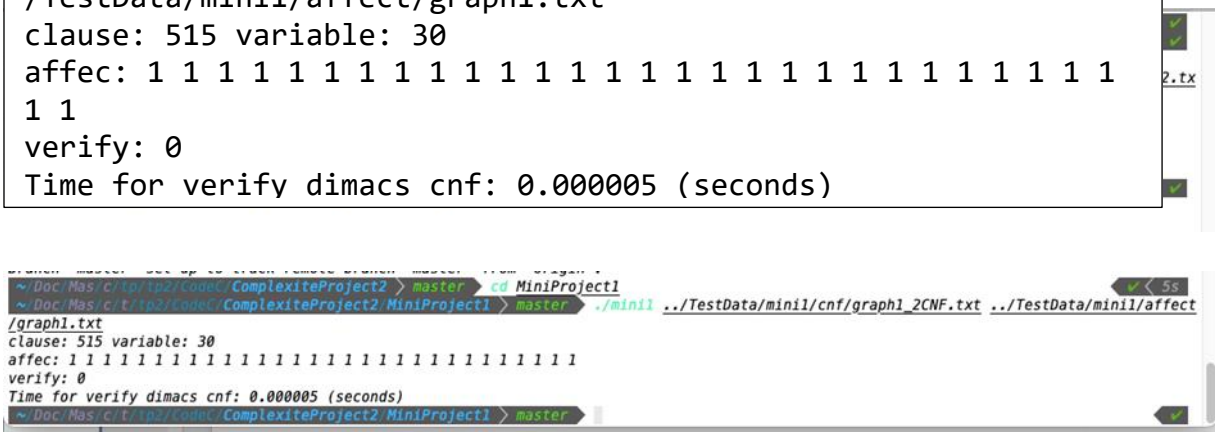
[2] L. Ines et O. Joel, « Sudoku as a SAT Problem. »

Annexes

https://github.com/batuan/TP2Complexite

- Exemple de lancement de commandes pour Mini Projet 1 :

```
~/MiniProject1 ./mini1 ../TestData/mini1/cnf/graph1_2CNF.txt ..
/TestData/mini1/affect/graph1.txt
clause: 515 variable: 30
affec: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
verify: 0
Time for verify dimacs cnf: 0.000005 (seconds)
```

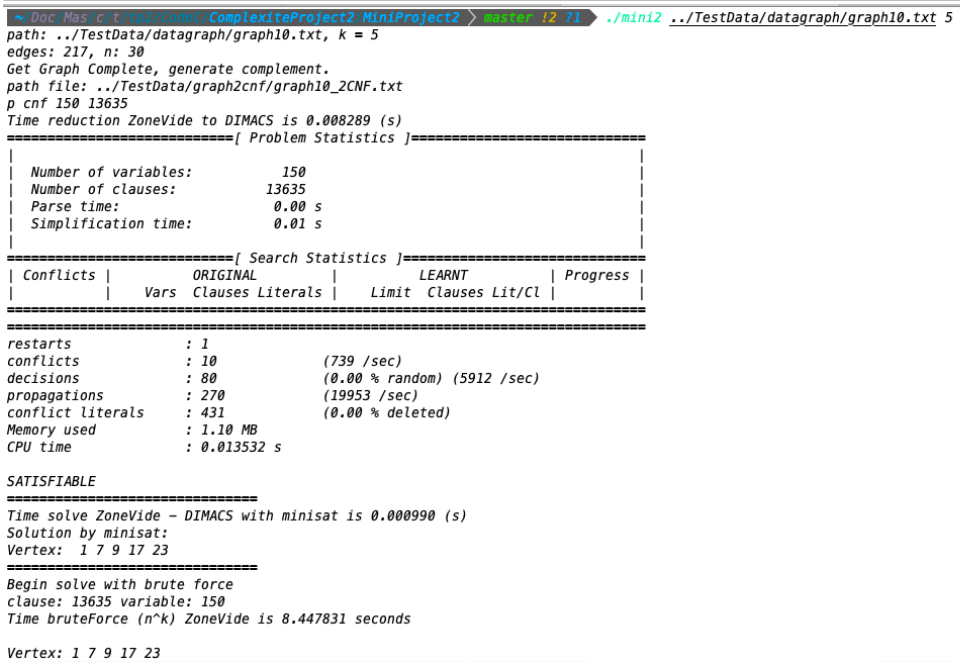


- Exemple de lancement de commandes pour Mini Projet 2 :

```
~/MiniProject2 ./mini2 ../TestData/datagraph/graph10.txt 5
path: ../TestData/datagraph/graph10.txt, k = 5
edges: 217, n: 30
Get Graph Complete, generate complement.
path file: ../TestData/graph2cnf/graph10_2CNF.txt
p cnf 150 13635
Time reduction ZoneVide to DIMACS is 0.008289 (s)
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:           150                                         |
|  Number of clauses:           13635                                         |
|  Parse time:                   0.00 s                                       |
|  Simplification time:          0.01 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
===============================================================================
restarts              : 1
conflicts             : 10             (739 /sec)
decisions             : 80             (0.00 % random) (5912 /sec)
propagations          : 270            (19953 /sec)
conflict literals     : 431            (0.00 % deleted)
Memory used           : 1.10 MB
CPU time              : 0.013532 s

SATISFIABLE
================================
Time solve ZoneVide - DIMACS with minisat is 0.000990 (s)
Solution by minisat:
Vertex:  1 7 9 17 23
================================
Begin solve with brute force
clause: 13635 variable: 150
Time bruteForce (n^k) ZoneVide is 8.447831 seconds

Vertex: 1 7 9 17 23
```



- Exemple de lancement de commandes pour Mini Projet 3 :

```
~/MiniProject3 ./mini3 ../TestData/sudoku/sudoku2.txt 3
total pre-assign: 31
total line: = 8859
Time for redution sudoku to Dimacs is 0.004449 seconds

 ~/MiniProject3 minisat ../TestData/sudoku2sat/sudoku2_2CNF.txt sat.txt
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:           729                                         |
|  Number of clauses:            3624                                         |
|  Parse time:                   0.00 s                                       |
|  Eliminated clauses:           0.00 Mb                                      |
|  Simplification time:          0.00 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
===============================================================================
restarts              : 1
conflicts             : 0              (0 /sec)
decisions             : 6              (0.00 % random) (726 /sec)
propagations          : 608            (73617 /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 0.56 MB
CPU time              : 0.008259 s

SATISFIABLE
 ~/MiniProject3 python3 display_sudoku.py sat.txt 3

S O L U T I O N
- - - - - - - - -
8  6  5  9  3  7  2  4  1
9  1  7  2  4  8  5  6  3
3  2  4  6  1  5  7  9  8
2  5  9  4  7  1  8  3  6
7  4  6  3  8  9  1  5  2
1  8  3  5  2  6  4  7  9
5  3  2  1  9  4  6  8  7
6  7  1  8  5  3  9  2  4
4  9  8  7  6  2  3  1  5
```