Report
Complexite TP2

Auteurs : THAI Ba Tuan
Alexandre Wery

Table des matières

I. Miniproject 1 - Vérificateur déterministe pour SAT

1. Algorithm

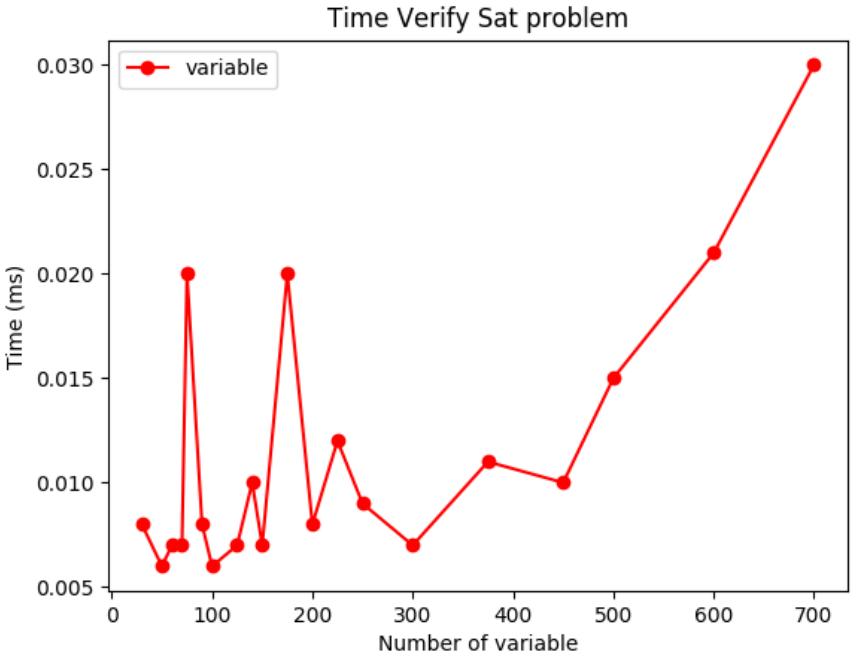
```
int verifySAT(struct SATProblem problem, struct Affectation affec) {
    int clauseIndex = 0;
    int variables = problem.variables;
    for(clauseIndex = 0; clauseIndex < problem.clauses; clauseIndex++) {
        int check = 0;
        for(int i = 0; i < problem.variables; i ++) {
            if(problem.value[clauseIndex*variables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

⇨ Complexity : O (clauses * variables)

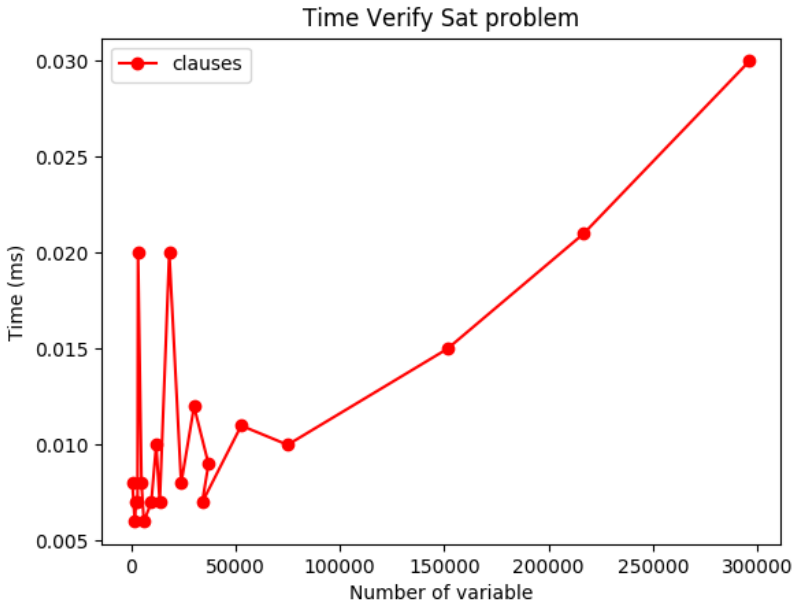
2. Testing

Variables is from 30 to 700
Clause is from 515 to 296305

- Variables vs time



- Clauses vs Time



II. Miniproject 2 - Réduction de Zone Vide à SAT

1. *Algorithm*

Input : Graph(V, E, k)
Output: Does a Zone-Vide (***Z***) of size k exist in a graph G = (V, E)?

We can define a list of variables x_{iv} for every $1 \leq i \leq k$ and every $v \in V$. We can interpret each of these variables xiv as "v is the i th vertex in the Zone-Vide. x_{iv} can be **True (1)** or **False (0).**
Constraints:

1) We know that for every i (where $1 \leq i \leq k$), there exists an i^{th} vertex in Z (where Z is the Zone-Vide):

$$\forall\{i \mid 1 \leq i \leq k\} \exists x_{iv}$$

2) For every **edge** (v, w) ∈ E, v AND w cannot **BOTH** be in the Zone-Vide:

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

3) For every i, j (where $j \neq i$), the i^{th} vertex is different from the j^{th} vertex. That is, we know that a vertex v cannot be both the i^{th} and the j^{th} vertex in the Zone-Vide. It also means that two different vertices cannot both be the i^{th} vertex in the Zone-Vide. We can write this as a two-part constraint for simplicity:

$$\forall[i, j \mid i \neq j]\ \forall[v \in V\ ]\ \neg x_{iv} \vee \neg x_{jv}$$

$$\forall i\ \forall[v, w \in V \mid v \neq w]\ \neg x_{iv} \vee \neg x_{iw}$$

- ⇨ Transformation: n vertices to n * k variables.
- ⇨ Complexity: $O(n*k + k^2 * n^2 + k * n^2 + k^2 * n) = 0(k^2 * n^2)$

2. Implementation Solve Algorithm

a. Minisat

After Transform problem Zone-Vide with Graph(V, E, k) to the DIMACS CNF, we can using SAT Solver to solve this problem by the command:

```
minisat cnf.txt sat.txt
```

where **cnf.txt** is the DIMACS CNF file transformed form the Zone–Vide, and **sat.txt** is result of the problem solved by minisat.
For display the result, I have writed an tools with Python to display solution.

```
python3 displayzonevide.py sat.txt k n
```

Where **sat.txt** is the solution and **k** is size of zone vide, **n** is the number of the vertices.

b. Brute – Force

As we know, transformer Graph (V, E, k) from Zone-Vide problem to standard form DIMACS CNF will have all n * k (n = |V|) variables. Divide all of these variables into k arrays, each consisting of n elements. In each of these arrays choose a different element to be input to the Zone-Vide (so we have k vertices). The number of options is : n^k, hence the complexity of the algorithm is $O(n^k)$
The pseudo code of the algorithm:

```
Algo-Brute-Force(G, V, k):
    value = tableau of size k
    n = |V|
    for i:=0 to n^k 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
```

We also have an iterative approach when assigning an array of n * k elements with 0 or 1, and then verify if the assgined is satisfied, howerver the complexity of this brute-force algorithm is 2^{n*k}, it is too big. So we chosing the brute-force algorithm with the complexity is n^k.

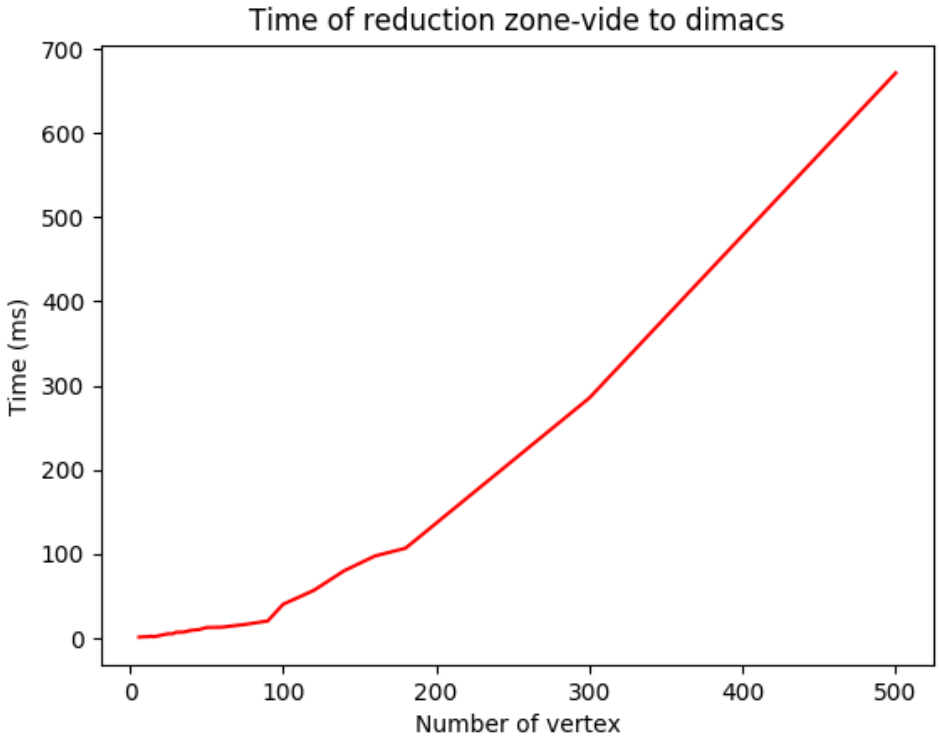
3. Testing

a. *Data*

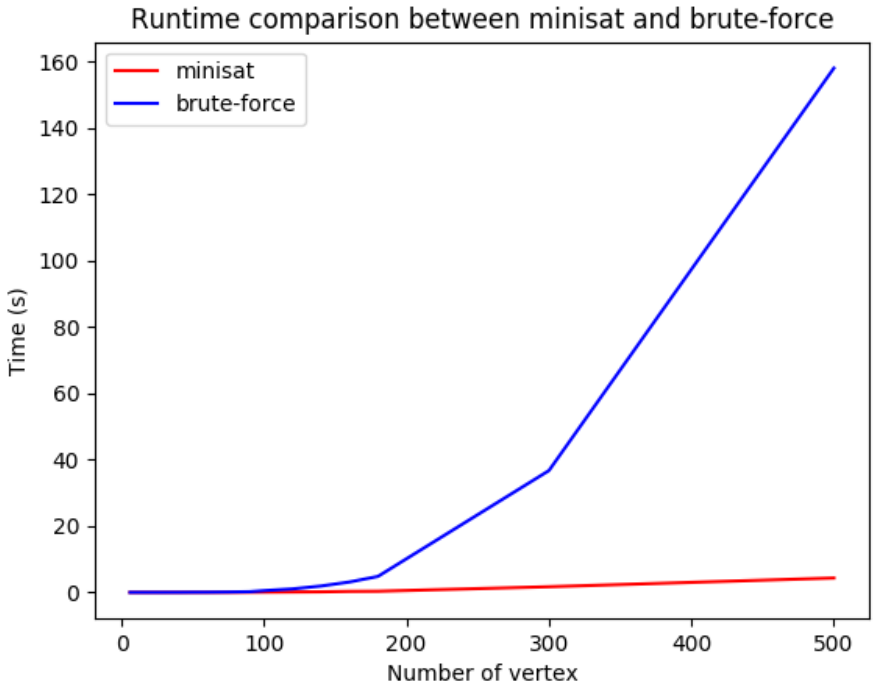
To generate graph test data, we used a graph generator tool which built by author **Mark Brockington** for the 1993 DIMACS Challenge competition.
The data generated will consist of graphs with vertices from 6 to 500 vertices. Details of the test data are contained in the directory **/TestData/datagraph.** The source code of this tool we also add in our source-code folder.

b. Result

- Testing with Zone-Vide size k = 3.
  - Time for reduction

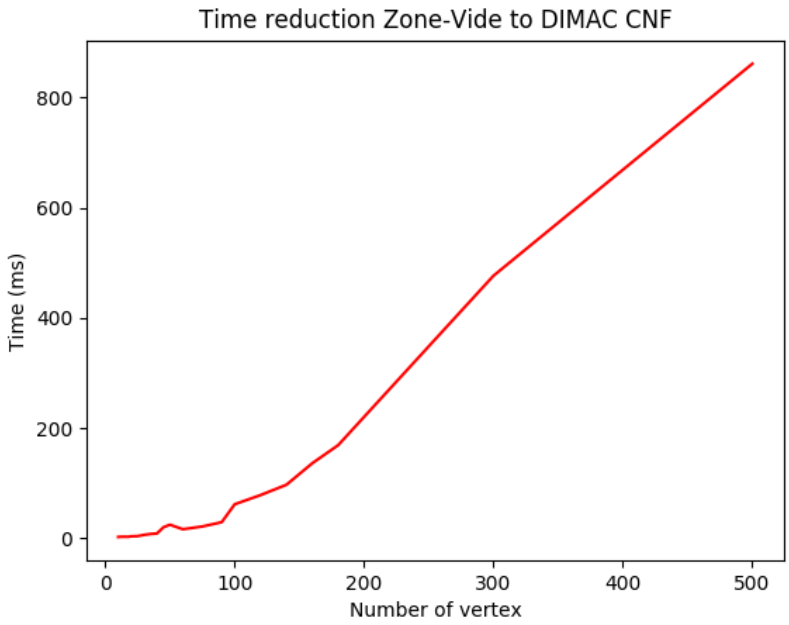


  - Time solve

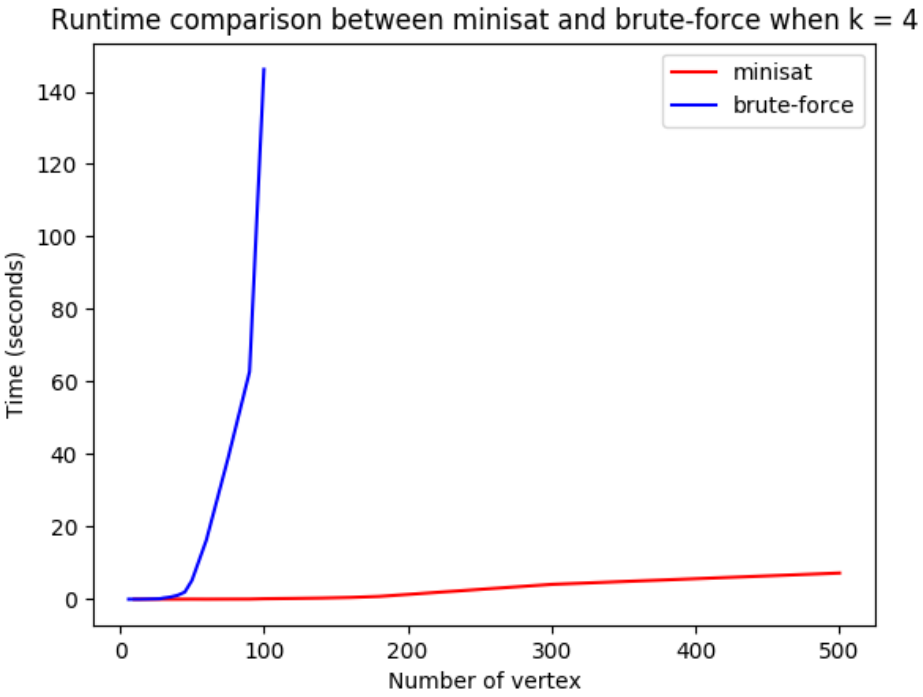


- Testing with Zone-Vide size k = 4

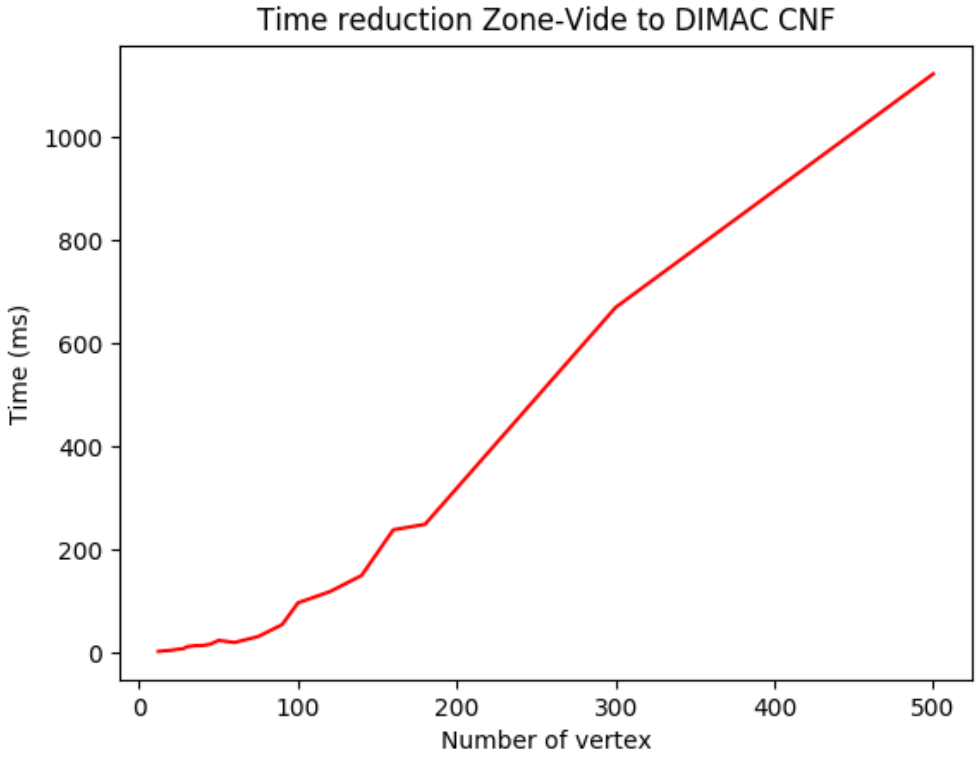
- Time for reduction



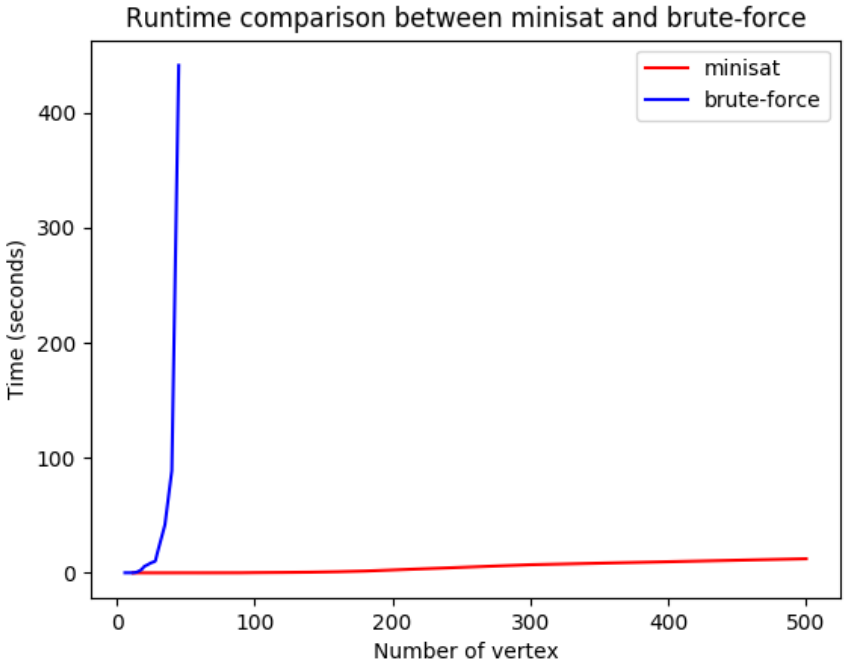
- Time solve



- k = 5
  - Time for reduction

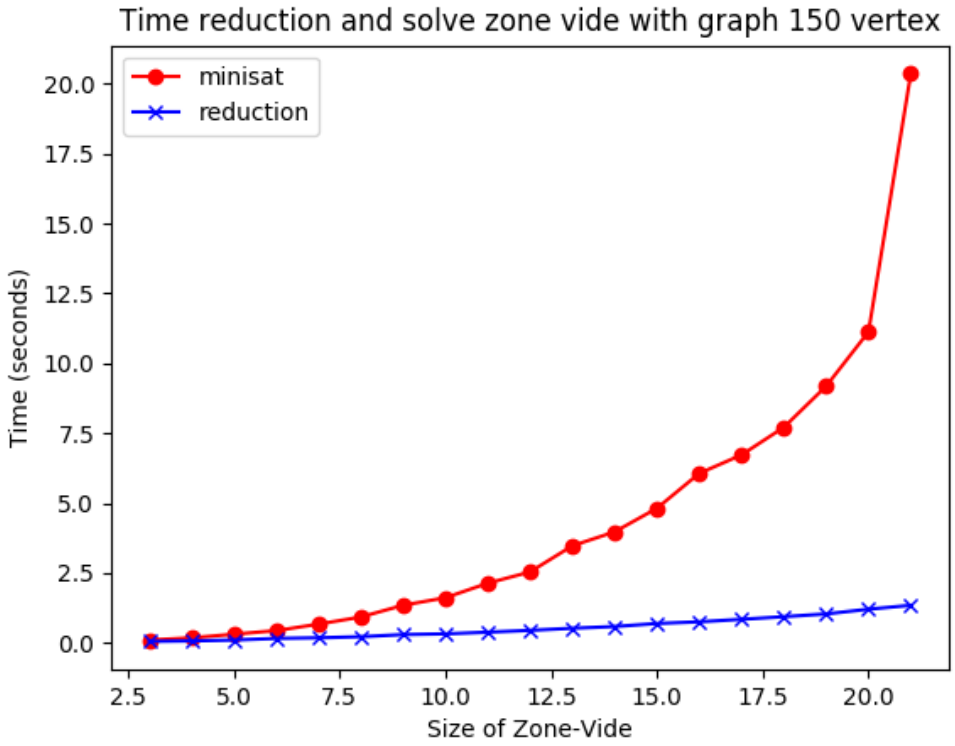


  - Time solve



⇨ Minisat >> Brute-Force

- Testing time excution of reduction and solve Zone-vide problem with a graph of |V|=150, |E| = 2320, we chose the Zone-Vide size is from 3 to 21.



III. Miniproject 3 - Réduction de Sudoku à SAT

1. *Algorithm*

A Sudoku puzzle can easily be represented as a SAT problem, albeit one requiring a significant number of propositional variables. If we were using variables with arbitrary finite domains, then n^2 x n^2 variables with domain [1..n^2] would be the most adequate option. Nevertheless, encoding Sudoku puzzles into CNF requires n^2 x n^2 x n^2 = n^6 propositional variables. For each entry in the $n^2 \times n^2$ grid S, we associate n^2 variables. Let us use the notation s_{xyz} to refer to variables. Variable s_{xyz} is assigned true if and only if the entry in row x and column y is assigned number z. Hence, s_{559} = 1 means that S[5, 5] = 9. Naturally, the pre-assigned entries of the Sudoku grid will be represented as unit clauses.

- Pre-assigined

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

For cell (x, y) already assigned with value z

- There is at least one number in each entry:

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

- Each number appears at most once in each row:

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

- Each number appears at most once in each column:

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

- Each number appears at most once in each ***nxn*** sub-grid:

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

And

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

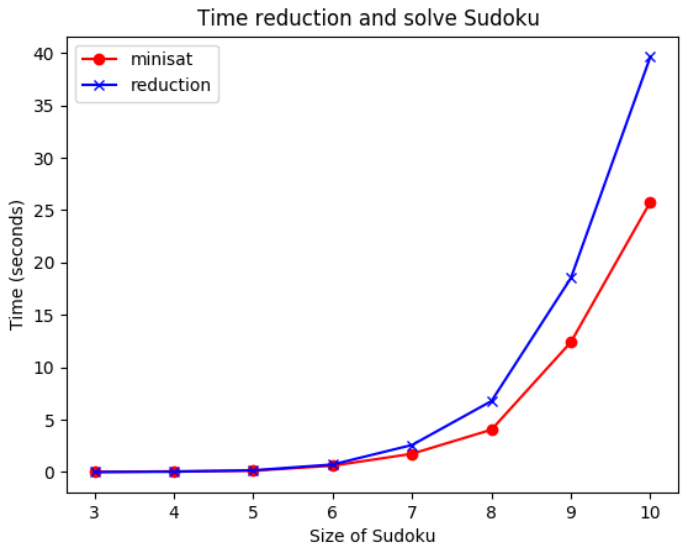
Complexity : $O(n^4 + n^8 + n^8 + n^8 + n^7 + n^8) = O(n^8)$

2. ***Testing***

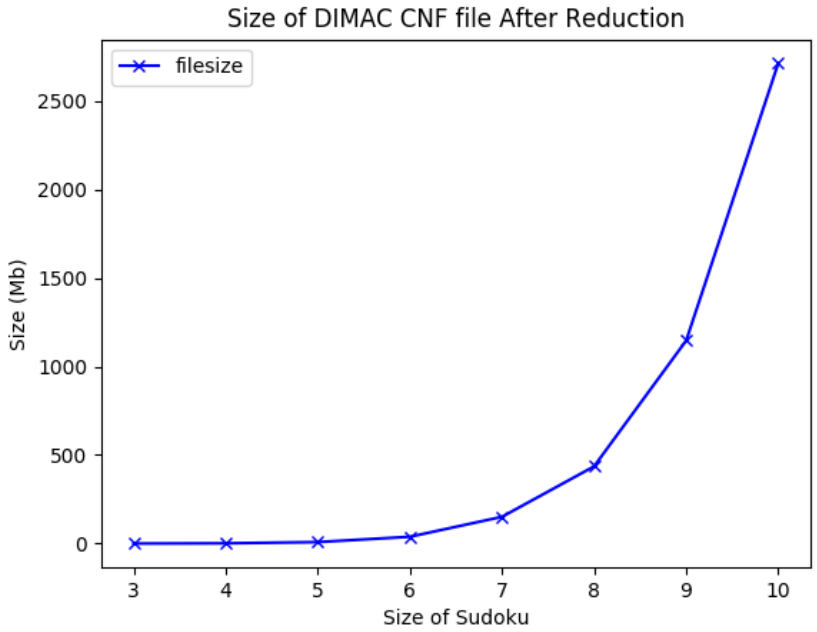
a. Data

For sudoku problems, the input n numbers are 3, 4, 5,…, 10 => the size of the sudoku will be: 9 x 9, 16 x 16,…, 100 x 100. Data is taken from the website: https://sudokugeant.cabanova.com/ and will be pre-assigned a few numbers.

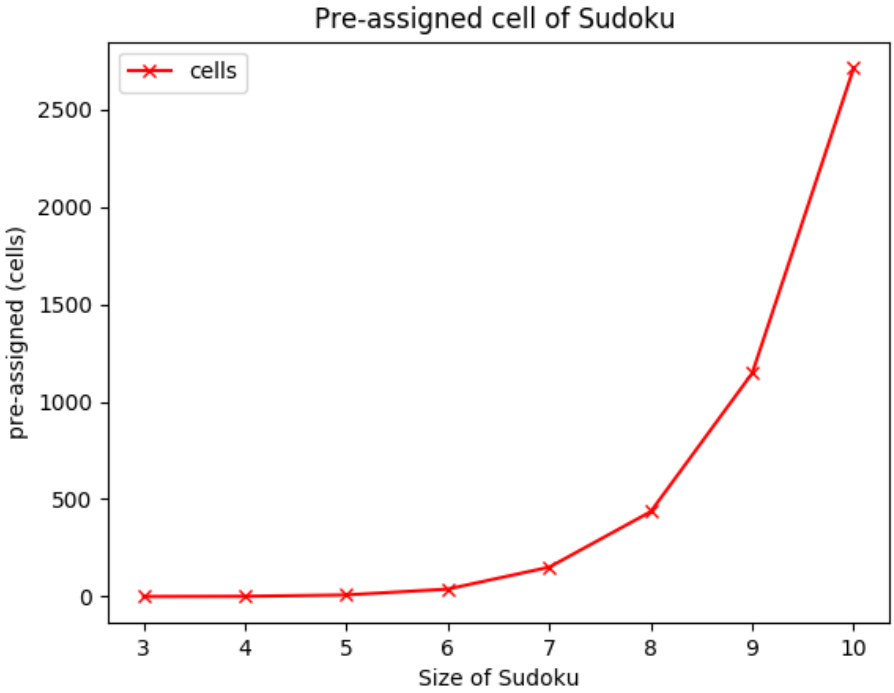
b. Experimental results



- Size of file



- Pre-assigned



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».

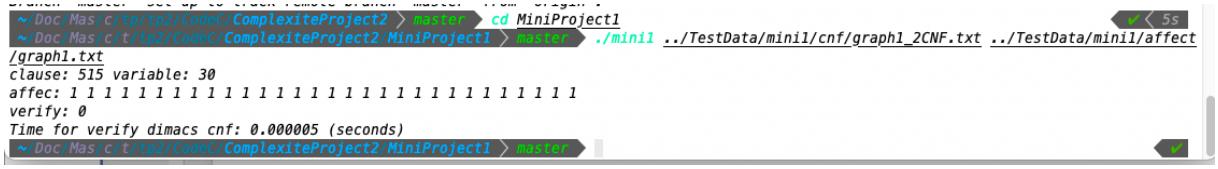
ANNEXE

Github link: https://github.com/batuan/TP2Complexite

Run command example :

MiniProject1 :

```
~/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 1
verify: 0
Time for verify dimacs cnf: 0.000005 (seconds)
```



MiniProject2 :

```
~/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
```

```
~/Doc/Mas/S1/2021/Code/ComplexiteProject2/MiniProject2 master !2 ?1 ./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
```

Miniproject3 :

```
~/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
```