Rapport Projet2 Complexité

Groupe 4, Wang Jinhua, Dauvier Nino, Borg Lucas, Nicolas Elie

December 7, 2022

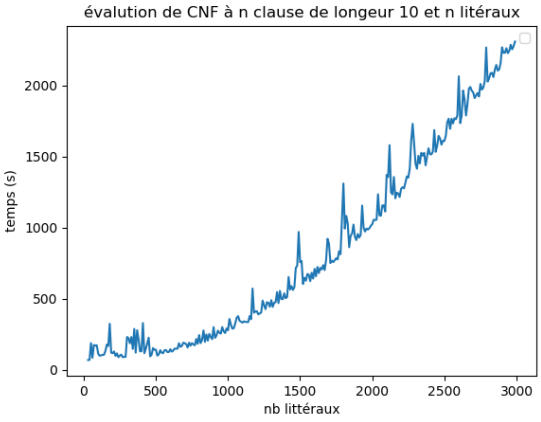
1 Introduction

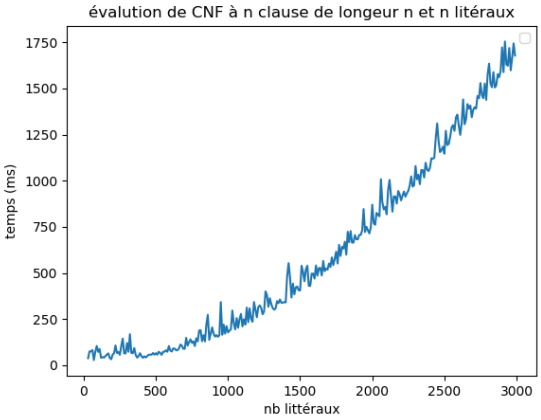
2 Mini-projet 1. Vérificateur déterministe pour SAT

L'algorithme SAT prend en entrée une famille d'ensemble F sous forme de <liste <liste <>>, contenant chacun une clause de la formule, et un ensemble de variable I sous forme de liste¡¿ et qui représente la liste des littéraux.

{F j } j $\in \{1, ..., m\}$ d'ensembles tels que $Fj \subseteq \{1, ..., n\}$ pour tout $j \in \{1, ..., m\}\{Ik\}K \in \{1, ..., l\}$

On a deux boucles for dans l'algorithme. La première boucle doit passer par tous les éléments de la liste donc $O(m)$. Et la deuxième boucle passe par tous les éléments de chacun des sous-Liste présent dans la liste F. Mais chaque sous-liste a une taille différente qui peut varier entre 1 et n. donc au maximum on a $O(n)$. Donc la complexité maximale est $O(nxm)$.





Sur la figure 2 le temps est en seconde et pas en milliseconde.

3 Mini-projet 2. Réduction de Zone Vide à SAT

4 Mini-projet 3. Réduction de Sudoku

4.1 Construction de propositions de Sudoku

Rappelons les exigences de tp2: une grille de taille $n^2 \times n^2$ où chaque case peut contenir un entier dans l'intervalle $[1, n^2]$, ou bien elle peut être vide. Donc les données d'entrée sont donc calculées en utilisant n.

Nous désignons par P(i, j, n) (exprimé dans le programme comme (i, j, k)) une proposition qui est vraie lorsque le nombre n se trouve sur la ligne i et la colonne j. Commençons par n=3 pour montrer comment mettre la clause "chaque cellule a au moins une valeur".
Chaque ligne contient tous les nombres de 1 à 9, et pour une ligne i et un nombre n fixes, il existe neuf possibilités pour j:
$P(i,1,n) \vee P(i,2,n) \vee P(i,3,n) \vee P(i,4,n) \vee P(i,5,n) \vee P(i,6,n) \vee P(i,7,n) \vee P(i,8,n) \vee P(i,9,n)$

Soit: $\vee_{j=1}^{9} p(i,j,n)$

Ensuite, on doit remplir les neuf chiffres de la ligne i:
$\vee_{j=1}^{9} p(i,j,1) \wedge \vee_{j=1}^{9} p(i,j,2) \wedge \vee_{j=1}^{9} p(i,j,3) \wedge \vee_{j=1}^{9} p(i,j,4) \wedge \vee_{j=1}^{9} p(i,j,5) \wedge \vee_{j=1}^{9} p(i,j,6) \wedge \vee_{j=1}^{9} p(i,j,7) \wedge \vee_{j=1}^{9} p(i,j,8) \wedge \vee_{j=1}^{9} p(i,j,9)$

Soit: $\wedge_{n=1}^{9} \vee_{j=1}^{9} p(i,j,n)$

Enfin, nous devons remplir chaque ligne avec des chiffres:
$(\wedge_{n=1}^{9} \vee_{j=1}^{9} p(1,j,n)) \wedge (\wedge_{n=1}^{9} \vee_{j=1}^{9} p(2,j,n)) \wedge (\wedge_{n=1}^{9} \vee_{j=1}^{9} p(3,j,n)) \wedge (\wedge_{n=1}^{9} \vee_{j=1}^{9} p(4,j,n)) \wedge$
$(\wedge_{n=1}^{9} \vee_{j=1}^{9} p(5,j,n)) \wedge (\wedge_{n=1}^{9} \vee_{j=1}^{9} p(6,j,n)) \wedge (\wedge_{n=1}^{9} \vee_{j=1}^{9} p(7,j,n)) \wedge (\wedge_{n=1}^{9} \vee_{j=1}^{9} p(8,j,n)) \wedge$
$(\wedge_{n=1}^{9} \vee_{j=1}^{9} p(9,j,n))$

Soit: $\wedge_{i=1}^{9} \wedge_{n=1}^{9} \vee_{j=1}^{9} p(i,j,n)$

Sur les conditions ci-dessus, nous avons maintenant terminé la première étape, chaque cellule ayant au moins une valeur, comme une triple boucle est utilisée, la complexité de l'algorithme est de $\Theta(n^2)^3$,
Soit : $\Theta(n^6)$.

4.2 Plus de contraintes

<Etape 1> $\wedge_{i=1}^{9} \wedge_{j=1}^{9} \vee_{n=1}^{9} p(i,j,n)$ $\Theta(n^6)$

On a besoin d'un total de quatre étapes (ou cinq car la quatrième étape nécessite deux pièces).Comme indiqué dans la première partie, on va pas entrer dans le détail de la dérivation de i,j,n.
Ensuite, nous voulons que chaque nombre apparaisse au maximum une fois dans chaque colonne. On utilise quatre boucles, donc la complexité est de $\Theta(n^8)$.

<Etape 2> $\wedge_{i=1}^{9} \wedge_{n=1}^{9} \wedge_{j=1}^{8} \wedge_{m=j+1}^{9} (\neg p(i,j,n) \vee \neg p(i,m,n))$ $\Theta(n^8)$

Ensuite, nous voulons que chaque chiffre apparaisse au maximum une fois sur chaque ligne. La complexité comme précedent.

<*Etape*3> $\wedge_{j=1}^{9} \wedge_{n=1}^{9} \wedge_{i=1}^{8} \wedge_{m=i+1}^{9} (\neg p(i,j,n) \vee \neg p(m,j,n))$ $\Theta(n^8)$

Enfin, chaque chiffre n'apparaisse qu'au maximum une fois dans une grille 3X3.
La complexité de chacune des deux parties est $\mathrm{n}^{(2+1+1+1+1+1)} = n^7, etn^{(2+1+1+1+1+1+1+1+1)} = n^8$.

<Etape 4> $\wedge_{n=1}^{9} \wedge_{x=0}^{2} \wedge_{y=0}^{2} \wedge_{i=1}^{3} \wedge_{j=1}^{3} \wedge_{m=y+1}^{3} (\neg p(3x+i, 3y+j, n) \vee \neg p(3x+i, 3y+j, m))$ $\Theta(n^7)$
et $\wedge_{n=1}^{9} \wedge_{x=0}^{2} \wedge_{y=0}^{2} \wedge_{i=1}^{3} \wedge_{j=1}^{3} \wedge_{m=y+1}^{3} \wedge_{l=1}^{3} (\neg p(3x+i, 3y+j, n) \vee \neg p(3x+i, 3y+l, m))$ $\Theta(n^8)$

Sur la base du sudoku déjà saisi, on doit également attribuer des valeurs à la grille qui existe déjà. Dans cette étape, on utilise deux boucles pour attribuer des valeurs à i,j, donc la complexité est de $\Theta(n^4)$

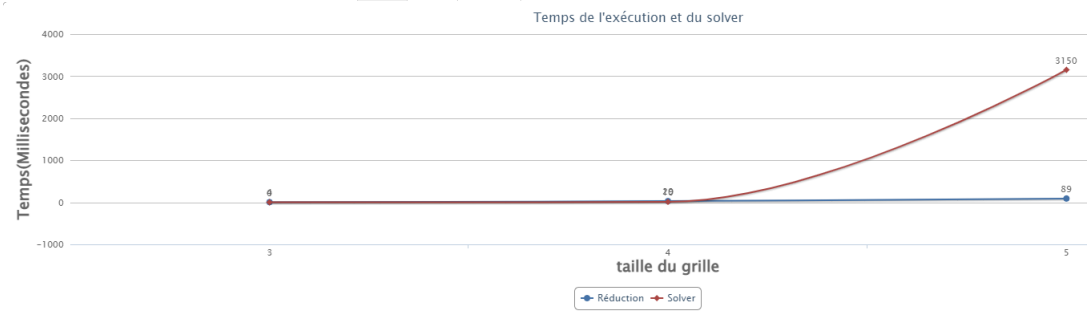
<Etape 5> $\wedge_{i=1}^{9} \wedge_{j=1}^{9} p(i,j,n)$ $\theta(n^4)$

4.3 Complexité

En résumé, la complexité algorithmique globale est $\Theta(n^6 + n^8 + n^8 + n^8 + n^7 + n^8 + n^4) = \Theta(n^8)$.

4.4 Test

	A	B	C
1	taille du grille	Réduction	Solver
2	3	4	0
3	4	29	10
4	5	89	3150



Comme le montre la figure, à mesure que la taille augmente, le temps d'exécution du programme augmente considérablement, par contre le temps de résolution du solveur ne change pas beaucoup.