Satisfiabilité

Sylvain Salvati

Outline

La satisfiabilité de formule

Étant donnée une formule φ dans $\mathsf{Prop}(\mathbf{X})$, on se pose la question de savoir si il existe une valuation ν qui satisfait φ, i.e. telle que $[\![\varphi, \nu]\!] = 1$.

La satisfiabilité de formule

Étant donnée une formule φ dans $\mathsf{Prop}(\mathbf{X})$, on se pose la question de savoir si il existe une valuation ν qui satisfait φ, i.e. telle que $[\![\varphi, \nu]\!] = 1$. Ce problème est difficile dans le sens que l'on ne connaît pas d'algorithme pour le résoudre qui ne soit pas exponentiel en temps dans le pire cas.

La satisfiabilité de formule

Étant donnée une formule φ dans $\mathsf{Prop}(\mathbf{X})$, on se pose la question de savoir si il existe une valuation ν qui satisfait φ, i.e. telle que $[\![\varphi, \nu]\!] = 1$.
Ce problème est difficile dans le sens que l'on ne connaît pas d'algorithme pour le résoudre qui ne soit pas exponentiel en temps dans le pire cas.
En gros, on ne sait pas vraiment faire autrement que d'énumérer toutes les valuations possibles dans certains cas.

NP-complétude de la satisfiabilité

Plus précisément, ce problème est NP-complet (Cook-Levin 1971).
C'est-à-dire que

NP on peut le résoudre en temps polynomial avec un algorithme non-déterministe,

NP-complétude de la satisfiabilité

Plus précisément, ce problème est NP-complet (Cook-Levin 1971). C'est-à-dire que

NP on peut le résoudre en temps polynomial avec un algorithme non-déterministe,

complet tout problème NP, peut être codé en temps polynomial dans ce problème.

NP-complétude de la satisfiabilité

Plus précisément, ce problème est NP-complet (Cook-Levin 1971).
C'est-à-dire que

NP on peut le résoudre en temps polynomial avec un algorithme non-déterministe,

complet tout problème NP, peut être codé en temps polynomial dans ce problème.

Les algorithmes non-déterministes s'écrivent en ajoutant une instruction.

choisir entre :
code1
et :
code2

qui choisit d'exécuter *code1* et *code2* sans se tromper pour trouver la solution du problème que l'algorithme cherche à résoudre.

NP-complétude de la satisfiabilité

Plus précisément, ce problème est NP-complet (Cook-Levin 1971).
C'est-à-dire que

NP on peut le résoudre en temps polynomial avec un algorithme non-déterministe,

complet tout problème NP, peut être codé en temps polynomial dans ce problème.

Les algorithmes non-déterministes s'écrivent en ajoutant une instruction.

choisir entre :
code1
et :
code2

qui choisit d'exécuter *code1* et *code2* sans se tromper pour trouver la solution du problème que l'algorithme cherche à résoudre.
On ne sait pas si tout algorithme NP peut être traduit dans un algorithme polynomial déterministe. Généralement on pense que non. Les seuls algorithmes déterministes connus qui résolvent des problèmes NP-complets s'exécutent en temps exponentiel dans le pire cas.

Le problème SAT

On appelle SAT le problème qui consiste à résoudre des formules sous forme normale conjontive c'est-à-dire de la forme :

$$(l_{1,1} \vee \cdots \vee l_{1,n_1}) \wedge \cdots \wedge (l_{p,1} \vee \cdots l_{p,n_p})$$

où les formules $l_{i,j}$ sont des litéraux, c'est-à-dire des formules de la forme :

- x – variable propositionnelle,
- $\neg x$ – négation de variable propositionnelle.

Les formules $l_{i,1} \vee \cdots \vee l_{i,n_i}$ sont appelées des clauses.

Le problème SAT

On appelle SAT le problème qui consiste à résoudre des formules sous forme normale conjontive c'est-à-dire de la forme :

$$(l_{1,1} \vee \cdots \vee l_{1,n_1}) \wedge \cdots \wedge (l_{p,1} \vee \cdots l_{p,n_p})$$

où les formules $l_{i,j}$ sont des litéraux, c'est-à-dire des formules de la forme :

- x – variable propositionnelle,
- $\neg x$ – négation de variable propositionnelle.

Les formules $l_{i,1} \vee \cdots \vee l_{i,n_i}$ sont appelées des clauses.
Résoudre un problème SAT consiste à trouver une valuation qui rende vraie toutes les clauses.

Le problème SAT

On appelle SAT le problème qui consiste à résoudre des formules sous forme normale conjontive c'est-à-dire de la forme :

$$(l_{1,1} \vee \cdots \vee l_{1,n_1}) \wedge \cdots \wedge (l_{p,1} \vee \cdots l_{p,n_p})$$

où les formules $l_{i,j}$ sont des litéraux, c'est-à-dire des formules de la forme :

- x – variable propositionnelle,
- $\neg x$ – négation de variable propositionnelle.

Les formules $l_{i,1} \vee \cdots \vee l_{i,n_i}$ sont appelées des clauses.
Résoudre un problème SAT consiste à trouver une valuation qui rende vraie toutes les clauses.
SAT est également un problème NP-complet.

Outline

SAT comme langage de programmation

- Nous avons vu que dans *NP-complet*, *complet* signifie que tout problème que l'on peut résoudre avec un algorithme NP peut être traduit dans un problème NP-complet en temps polynomial.

SAT comme langage de programmation

- Nous avons vu que dans *NP-complet*, *complet* signifie que tout problème que l'on peut résoudre avec un algorithme NP peut être traduit dans un problème NP-complet en temps polynomial.
- Si on a un programme efficace pour résoudre un problème NP-complet, alors on peut s'en servir pour résoudre d'autres problèmes de NP.

SAT comme langage de programmation

- Nous avons vu que dans *NP-complet*, *complet* signifie que tout problème que l'on peut résoudre avec un algorithme NP peut être traduit dans un problème NP-complet en temps polynomial.
- Si on a un programme efficace pour résoudre un problème NP-complet, alors on peut s'en servir pour résoudre d'autres problèmes de NP.
- Le problème SAT a une structure très simple qui permet d'utiliser des instructions très rapides pour le résoudre.

SAT comme langage de programmation

- Nous avons vu que dans *NP-complet*, *complet* signifie que tout problème que l'on peut résoudre avec un algorithme NP peut être traduit dans un problème NP-complet en temps polynomial.
- Si on a un programme efficace pour résoudre un problème NP-complet, alors on peut s'en servir pour résoudre d'autres problèmes de NP.
- Le problème SAT a une structure très simple qui permet d'utiliser des instructions très rapides pour le résoudre.
- Les solveurs-SAT (SAT-solver) sont des logiciels qui résolvent ces problèmes très efficacement :

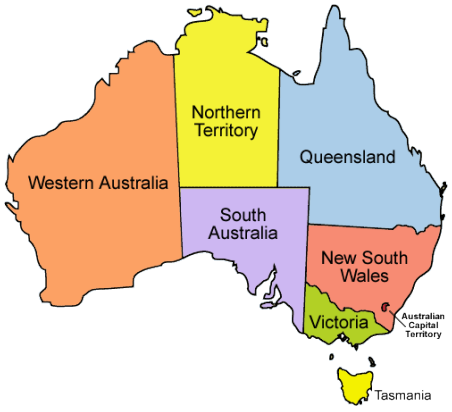
SAT comme langage de programmation

- Nous avons vu que dans *NP-complet*, *complet* signifie que tout problème que l'on peut résoudre avec un algorithme NP peut être traduit dans un problème NP-complet en temps polynomial.
- Si on a un programme efficace pour résoudre un problème NP-complet, alors on peut s'en servir pour résoudre d'autres problèmes de NP.
- Le problème SAT a une structure très simple qui permet d'utiliser des instructions très rapides pour le résoudre.
- Les solveurs-SAT (SAT-solver) sont des logiciels qui résolvent ces problèmes très efficacement :
  - Ils peuvent résoudre de très gros problèmes SAT (plusieurs millions de variables).

SAT comme langage de programmation

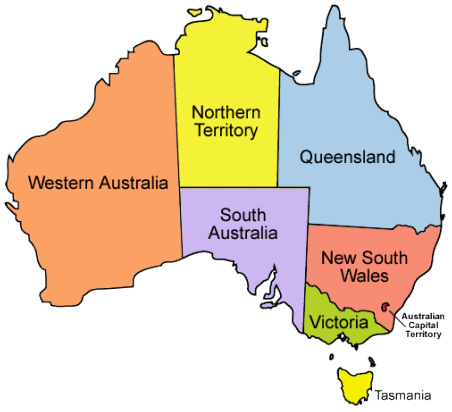
- Nous avons vu que dans *NP-complet*, *complet* signifie que tout problème que l'on peut résoudre avec un algorithme NP peut être traduit dans un problème NP-complet en temps polynomial.
- Si on a un programme efficace pour résoudre un problème NP-complet, alors on peut s'en servir pour résoudre d'autres problèmes de NP.
- Le problème SAT a une structure très simple qui permet d'utiliser des instructions très rapides pour le résoudre.
- Les solveurs-SAT (SAT-solver) sont des logiciels qui résolvent ces problèmes très efficacement :
  - Ils peuvent résoudre de très gros problèmes SAT (plusieurs millions de variables).
  - Mais ils sont parfois incapables de résoudre des problèmes avec quelques dizaines de variables seulement.

Exemple : Colorier l'Australie



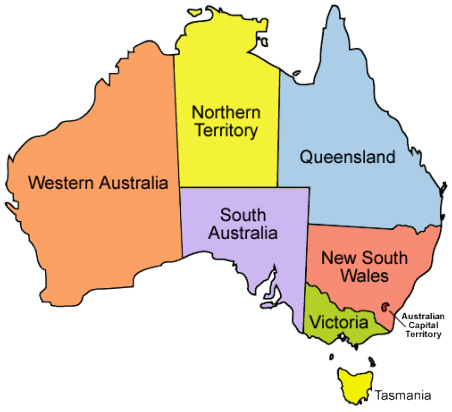
Peut-on colorier la carte de l'Autralie en n'utilisant que 3 couleurs de sorte que deux états limitrophes soient de couleurs différentes ?

Exemple : Colorier l'Australie



Peut-on colorier la carte de l'Autralie en n'utilisant que 3 couleurs de sorte que deux états limitrophes soient de couleurs différentes ?
On peut chercher à le faire à la main, mais on préfère utiliser un solveur-SAT.

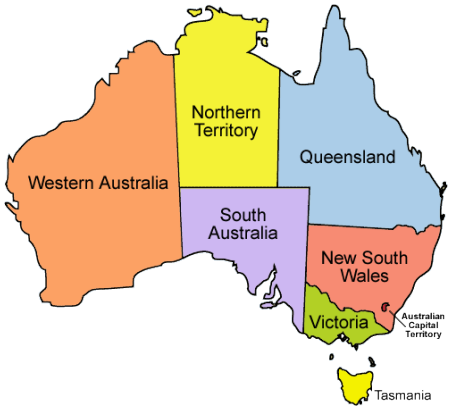
Exemple : Colorier l'Australie



$C = \{r, v, b\}$

- Nous utilisons les couleurs rouge, vert et bleu notées respectivement r, v, b. On appelle $C = \{r, v, b\}$ l'ensemble de ces couleurs.

Exemple : Colorier l'Australie

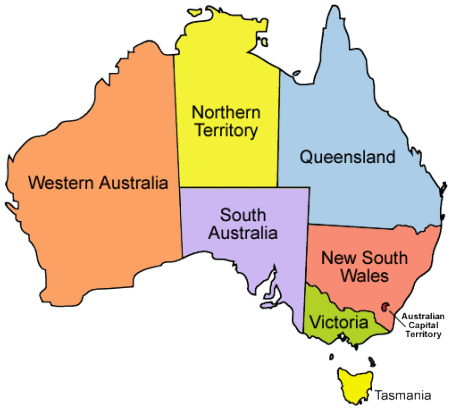


$C = \{r, v, b\}$

$E = \{WA, NT, Q, SA, NSW, V, T\}$

- Nous utilisons les couleurs rouge, vert et bleu notées respectivement r, v, b. On appelle $C = \{r, v, b\}$ l'ensemble de ces couleurs.
- On note les états par leurs initiales et l'ensemble des état est $E = \{WA, NT, Q, SA, NSW, V, T\}$.

Exemple : Colorier l'Australie



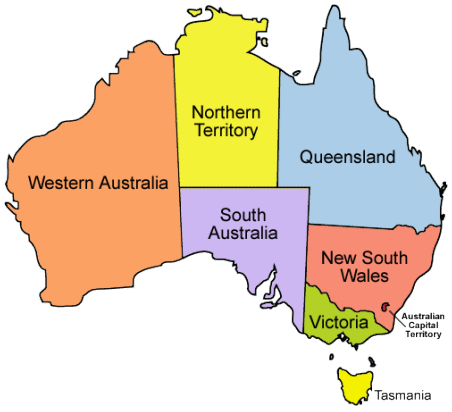
$$C = \{r, v, b\}$$

$$E = \{WA, NT, Q, SA, NSW, V, T\}$$

$$L = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$$

- Nous utilisons les couleurs rouge, vert et bleu notées respectivement r, v, b. On appelle $C = \{r, v, b\}$ l'ensemble de ces couleurs.
- On note les états par leurs initiales et l'ensemble des état est $E = \{WA, NT, Q, SA, NSW, V, T\}$.
- On note les paires d'états limitrophes :
  $L = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$

Exemple : Colorier l'Australie



$$C = \{r, v, b\}$$

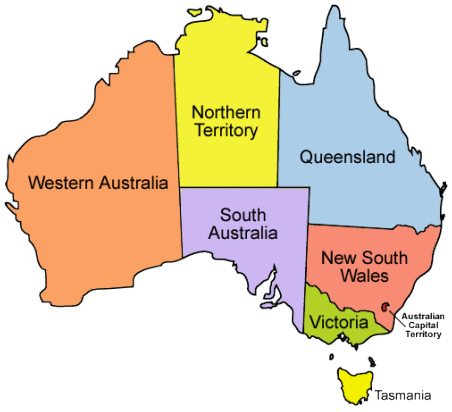
$$E = \{WA, NT, Q, SA, NSW, V, T\}$$

$$L = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$$

- Nous utilisons les couleurs rouge, vert et bleu notées respectivement r, v, b. On appelle $C = \{r, v, b\}$ l'ensemble de ces couleurs.
- On note les états par leurs initiales et l'ensemble des état est $E = \{WA, NT, Q, SA, NSW, V, T\}$.
- On note les paires d'états limitrophes :
  $L = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$
- Les variables propositionnelles sont $[e, c]$ pour toute paire d'état et de couleur, soit : $[WA, r], [WA, v], [WA, b], [NT, r], [NT, v] \ldots$

  La variable $[e, c]$ sera vraie ssi on peut colorier l'état e avec la couleur c.

Exemple : Colorier l'Australie



$$C = \{r, v, b\}$$

$$E = \{WA, NT, Q, SA, NSW, V, T\}$$

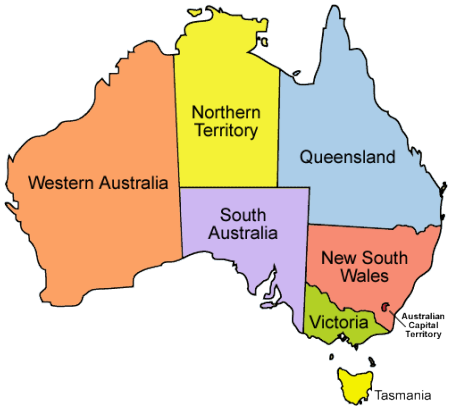
$$L = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$$

Nous allons utiliser les contraintes suivantes :

- tout état a au moins une couleur :

$$\varphi_1 = \bigwedge_{e \in E} \bigvee_{c \in C} [e, c] = ([WA, r] \vee [WA, v] \vee [WA, b]) \wedge ([NT, r] \vee [NT, v] \vee [NT, b]) \ldots$$

Exemple : Colorier l'Australie



$$C = \{r, v, b\}$$

$$E = \{WA, NT, Q, SA, NSW, V, T\}$$

$$L = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$$

Nous allons utiliser les contraintes suivantes :

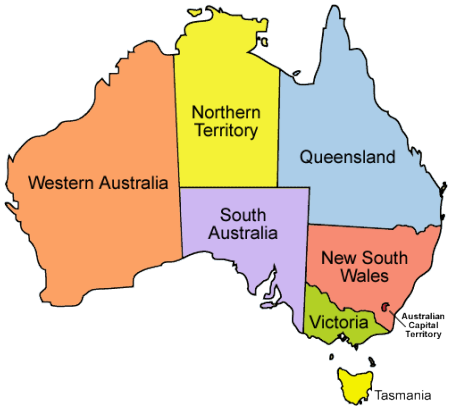
- tout état a au moins une couleur :

$$\varphi_1 = \bigwedge_{e \in E} \bigvee_{c \in C} [e, c] = ([WA, r] \vee [WA, v] \vee [WA, b]) \wedge ([NT, r] \vee [NT, v] \vee [NT, b]) \ldots$$

- aucun état n'a deux couleurs (chaque état au plus une couleur) :

$$\varphi_2 = \bigwedge_{e \in E, c_1, c_2 \in C | c_1 \neq c_2} \neg[e, c_1] \vee \neg[e, c_2]$$

Exemple : Colorier l'Australie



$$C = \{r, v, b\}$$

$$E = \{WA, NT, Q, SA, NSW, V, T\}$$

$$L = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$$

Nous allons utiliser les contraintes suivantes :

- tout état a au moins une couleur :

$$\varphi_1 = \bigwedge_{e \in E} \bigvee_{c \in C} [e, c] = ([WA, r] \vee [WA, v] \vee [WA, b]) \wedge ([NT, r] \vee [NT, v] \vee [NT, b]) \ldots$$

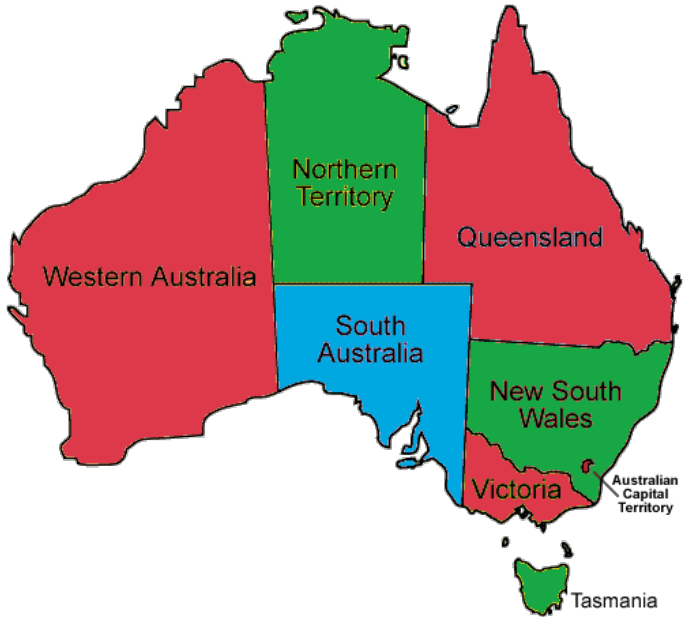
- aucun état n'a deux couleurs (chaque état au plus une couleur) :

$$\varphi_2 = \bigwedge_{e \in E, c_1, c_2 \in C | c_1 \neq c_2} \neg[e, c_1] \vee \neg[e, c_2]$$

- les états limitrophes n'ont pas la même couleur :

$$\varphi_3 = \bigwedge_{\{e_1, e_2\} \in L, c \in C} \neg[e_1, c] \vee \neg[e_2, c]$$

Exemple : Colorier l'Australie



Western Autralia	$\rightarrow$	rouge
Queensland	$\rightarrow$	rouge
Victoria	$\rightarrow$	rouge
Nothern Territory	$\rightarrow$	vert
New South Wales	$\rightarrow$	vert
Tasmania	$\rightarrow$	vert
South Australia	$\rightarrow$	bleu

Contraintes de comptage

Parfois on veut exprimer qu'au moins k variables propositionnelles d'un sous-ensemble de variables $\mathbf{X} = \{x_1, \ldots, x_n\}$ sont vraie :

- pour $k = 1$, il nous suffit d'utiliser la clause $x_1 \vee \cdots \vee x_n$,

Contraintes de comptage

Parfois on veut exprimer qu'au moins k variables propositionnelles d'un sous-ensemble de variables $\mathbf{X} = \{x_1, \ldots, x_n\}$ sont vraie :

- pour $k = 1$, il nous suffit d'utiliser la clause $x_1 \vee \cdots \vee x_n$,
- pour $k = 2$, pour pourrait écrire

$$\bigvee_{1 \leq i < j \leq n} x_i \wedge x_j = (x_1 \wedge x_2) \vee \ldots (x_1 \wedge x_n) \vee \ldots$$

Contraintes de comptage

Parfois on veut exprimer qu'au moins k variables propositionnelles d'un sous-ensemble de variables $\mathbf{X} = \{x_1, \ldots, x_n\}$ sont vraie :

- pour $k = 1$, il nous suffit d'utiliser la clause $x_1 \vee \cdots \vee x_n$,
- pour $k = 2$, pour pourrait écrire

$$\bigvee_{1 \leq i < j \leq n} x_i \wedge x_j = (x_1 \wedge x_2) \vee \ldots (x_1 \wedge x_n) \vee \ldots$$

- Cela se généralise facilement pour k quelconque. Malheureusement la formule obtenue n'est pas sous forme normale conjonctive.

Contraintes de comptage

Parfois on veut exprimer qu'au moins k variables propositionnelles d'un sous-ensemble de variables $\mathbf{X} = \{x_1, \ldots, x_n\}$ sont vraie :

- pour $k = 1$, il nous suffit d'utiliser la clause $x_1 \vee \cdots \vee x_n$,
- pour $k = 2$, pour pourrait écrire

$$\bigvee_{1 \leq i < j \leq n} x_i \wedge x_j = (x_1 \wedge x_2) \vee \ldots (x_1 \wedge x_n) \vee \ldots$$

- Cela se généralise facilement pour k quelconque. Malheureusement la formule obtenue n'est pas sous forme normale conjonctive.
- On pourrait chercher à mettre la formule obtenue en FNC, mais on préfère une approche directe.

Comptage : approche directe

On utilise un petit résultat combinatoire :

Lemma

Soit $\mathbf{X}$ *un ensemble de n élément, si* $\mathbf{Y} \subseteq \mathbf{X}$*, alors les deux propositions suivantes sont équivalentes :*

- $\mathbf{Y}$ *contient au moins k éléments,*
- *pour tout* $\mathbf{Z} \subseteq \mathbf{X}$ *qui contient* $n - k + 1$ *éléments intersecte* $\mathbf{Y}$*.*

Démonstration.

Si $\mathbf{Y}$ et $\mathbf{Z}$ étaient disjoints alors $\mathbf{Y} \cup \mathbf{Z}$ contiendrait au moins $n + 1$ éléments distincts de $\mathbf{X}$. Or $\mathbf{X}$ contient n éléments : contradiction. □

Comptage : approche directe

On utilise un petit résultat combinatoire :

Lemma

Soit $\mathbf{X}$ *un ensemble de n élément, si* $\mathbf{Y} \subseteq \mathbf{X}$*, alors les deux propositions suivantes sont équivalentes :*

- $\mathbf{Y}$ *contient au moins k éléments,*
- *pour tout* $\mathbf{Z} \subseteq \mathbf{X}$ *qui contient* $n - k + 1$ *éléments intersecte* $\mathbf{Y}$*.*

Démonstration.

Si $\mathbf{Y}$ et $\mathbf{Z}$ étaient disjoints alors $\mathbf{Y} \cup \mathbf{Z}$ contiendrait au moins $n + 1$ éléments distincts de $\mathbf{X}$. Or $\mathbf{X}$ contient n éléments : contradiction. □

On peut maintenant formuler qu'au moins k variables de $\mathbf{X} = \{x_1, \ldots, x_n\}$ sont vraies en utilisant la formule :

$$\bigwedge_{1 \leq i_1 < \cdots < i_{n-k+1} \leq n} \bigvee_{1 \leq j \leq n-k+1} x_{i_j}$$

Comptage : au plus

Si on veut qu'au plus k variables de $\mathbf{X}$ sont vraies, cela signifie qu'au moins $n-k$ variables de $\mathbf{X}$ sont fausses. En utilisant la même idée, on obtient qu'il faut que dans tout ensemble de $n-(n-k)+1 = k+1$ variables au moins une soit fausse :

$$\bigwedge_{1 \leq i_1 < \cdots < i_{k+1} \leq n} \bigvee_{1 \leq j \leq k+1} \neg x_{i_j}$$

Exemple : mieux Colorier l'Australie

On souhaite en plus d'utiliser 3 couleurs, que chacune soit utilisée au moins deux fois :

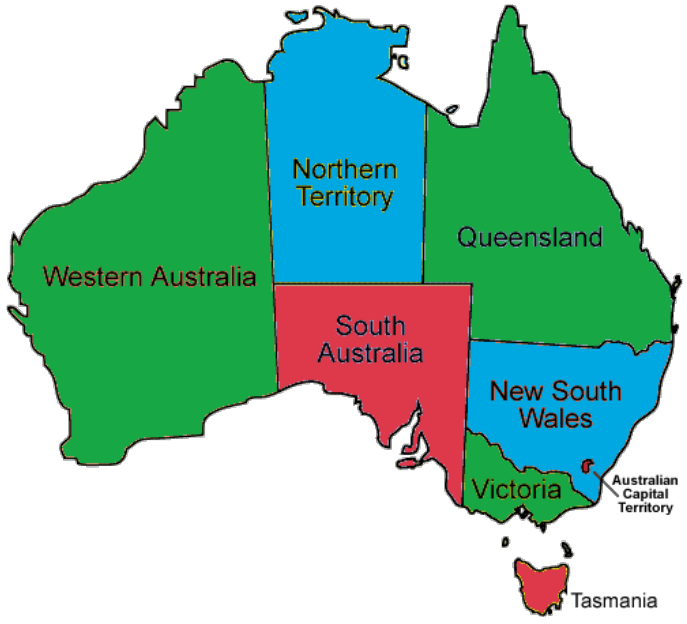
- pour tout c dans C, il nous faut donc au moins deux variables de l'ensemble $\{[e, c] \mid e \in E\}$ qui soient vraies.

Exemple : mieux Colorier l'Australie

On souhaite en plus d'utiliser 3 couleurs, que chacune soit utilisée au moins deux fois :

- pour tout c dans C, il nous faut donc au moins deux variables de l'ensemble $\{[e, c] \mid e \in E\}$ qui soient vraies.
- Comme il y 7 états en Australie, cela revient à considérer tous les ensembles de variables de taille $7 - 2 + 1 = 6$: $\bigwedge_{E' \subseteq E || E'| = 6} \bigvee_{e \in E'} [e, c]$.

Exemple : mieux Colorier l'Australie



South Australia	$\rightarrow$	rouge
Tasmania	$\rightarrow$	rouge
Western Autralia	$\rightarrow$	vert
Queensland	$\rightarrow$	vert
Victoria	$\rightarrow$	vert
Nothern Territory	$\rightarrow$	bleu
New South Wales	$\rightarrow$	bleu

Le format DIMACS

- Nous allons interagir avec des solveurs-SAT par l'intermédiaire de python.

Le format DIMACS

- Nous allons interagir avec des solveurs-SAT par l'intermédiaire de python.
- Les solveurs-SAT représentent les variables par des entiers strictement positifs : k représente la variable x_k

Le format DIMACS

- Nous allons interagir avec des solveurs-SAT par l'intermédiaire de python.
- Les solveurs-SAT représentent les variables par des entiers strictement positifs : k représente la variable x_k
- Le litéral $\neg x_k$ est représenté par l'entier $-k$.

Le format DIMACS

- Nous allons interagir avec des solveurs-SAT par l'intermédiaire de python.
- Les solveurs-SAT représentent les variables par des entiers strictement positifs : k représente la variable x_k
- Le litéral $\neg x_k$ est représenté par l'entier $-k$.
- Nous passerons aux solveurs-SAT une liste de listes d'entiers $l = [C_1, \ldots, C_n]$.
  - Chaque liste C_i représentera une clause. Par exemple la liste $[1, -2, 3]$ représente la clause $x_1 \vee \neg x_2 \vee x_3$.
  - La liste l représente la conjonction de ces clauses.

Le format DIMACS

- Nous allons interagir avec des solveurs-SAT par l'intermédiaire de python.
- Les solveurs-SAT représentent les variables par des entiers strictement positifs : k représente la variable x_k
- Le litéral $\neg x_k$ est représenté par l'entier $-k$.
- Nous passerons aux solveurs-SAT une liste de listes d'entiers $l = [C_1, \ldots, C_n]$.
  - Chaque liste C_i représentera une clause. Par exemple la liste $[1, -2, 3]$ représente la clause $x_1 \vee \neg x_2 \vee x_3$.
  - La liste l représente la conjonction de ces clauses.

En général les solveurs-SAT reçoivent les problèmes à résoudre par l'intermédiaire d'un fichier au format DIMACS de la forme :

```
p cnf 5 2
1 -5 0
2 5 -1 0
```

où 5 sur la première ligne donne le nombre de variables, 2 le nombre de clauses. Les lignes suivantes représentent les clauses.