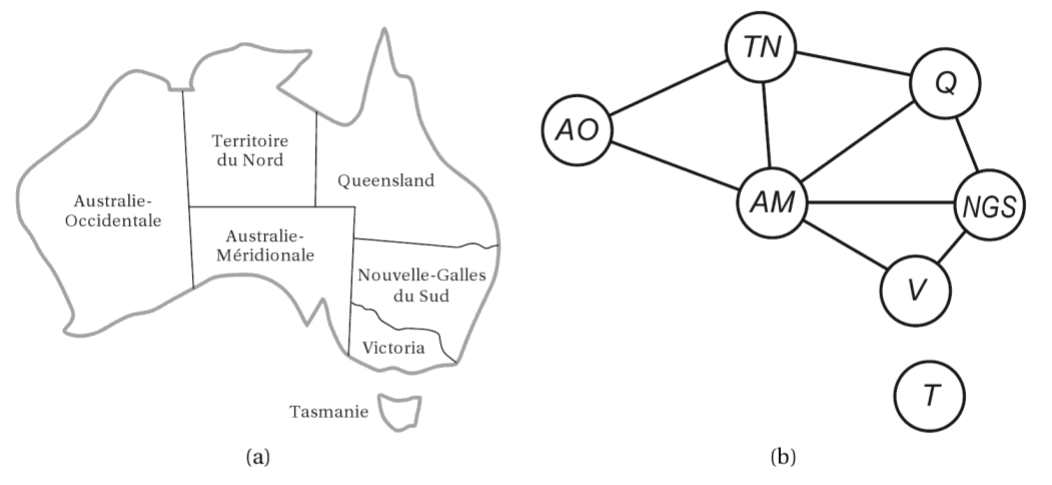
Backtracking intelligent : l'examen en amont

* L'algorithme EXPLORATION- BACKTRACKING applique une stratégie très simple pour savoir quoi faire en cas d'échec d’une branche de l'exploration: revenir à la variable précédente et essayer une autre valeur.
* Cette stratégie est appelée **backtracking chronologique**, car elle consiste à réviser le point de décision le plus récent. Dans cette sous-section, nous étudions d'autres possibilités.
* **Considérons ce qui se passe quand on applique le backtracking simple :**



En suivant un ordre fixe des variables : .

Supposons qu'on ait généré l’assignation partielle  
.

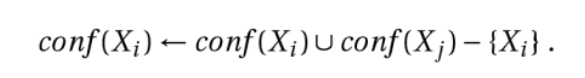
Quand on essaie la variable suivante , on s'aperçoit que toutes les valeurs violent une contrainte. On remonte alors à et on essaie une nouvelle couleur pour la Tasmanie !

C’est, de toute évidence, en pure perte : changer la couleur de la Tasmanie ne peut pas résoudre le problème de celle de l'Australie-Méridionale.

* Une approche plus intelligente du backtracking consiste à revenir à une variable qui pourrait régler le problème, c'est-à-dire qui puisse bloquer des valeurs possibles de .
* Pour cela, nous gardons une trace d’un ensemble d'assignations en conflit avec une valeur de .
* L'ensemble (dans ce cas ), est appelé **l'ensemble de conflit** pour .
* La méthode de **backjumping (« saut en arrière »)** remonte à l'assignation la plus récente dans l'ensemble de conflit.
* Dans le cas présent, le backjumping « sauterait » la Tasmanie et essaierait une nouvelle valeur pour .
* On met facilement en œuvre cette méthode en modifiant BACKTRACK afin qu’il stocke l’ensemble de conflit lorsqu'il recherche une valeur légale à assigner.
* S'il ne trouve pas de valeurs légales, l'algorithme doit retourner l'élément le plus récent de l’ensemble de conflit avec l'indicateur d'échec.
* **Le lecteur attentif aura remarqué que le forward checking peut fournir l'ensemble de conflit sans travail supplémentaire** : chaque fois que le forward checking, à la suite d'une assignation X = x, supprime une valeur du domaine de Y, il doit ajouter X = x à l'ensemble de conflit de Y.
  + Si la dernière valeur est supprimée du domaine de Y, les assignations de l'ensemble de conflit de Y sont ajoutées à l'ensemble de conflit de X.
  + Ensuite, lorsque l'algorithme passe à Y, il sait immédiatement où remonter en cas de besoin.
* **Le lecteur très attentif aura remarqué une bizarrerie :** le backjumping se produit lorsque toutes les valeurs d’un domaine sont en conflit avec l'assignation courante. Mais le forward checking détecte cet événement et empêche l'exploration d'atteindre un tel nœud!
* En fait, on peut montrer que toutes les branches élaguées par le backjumping sont aussi élaguées par le forward checking. Il en résulte que le backjumping est redondant dans une exploration avec forward checking ou même dans une exploration qui applique une vérification forte de la

cohérence, comme c'est le cas de MAC.

* En dépit des observations faites au paragraphe précédent, il n'en demeure pas moins que l'idée sous-jacente au backjumping reste bonne : tenir compte des raisons de l'échec avant de revenir en arrière.
* Le backjumping rend compte d’un échec quand le domaine d'une variable devient vide, mais il arrive souvent qu'une branche soit condamnée longtemps avant qu’un tel phénomène ne survienne.
* Considérez à nouveau l'assignation partielle   
  , dont on a montré précédemment qu'elle n'était pas cohérente.
* Supposez qu'on essaie ensuite , puis qu’on assigne .
* On sait qu'aucune assignation ne peut fonctionner pour ces quatre variables et nous épuisons les valeurs disponibles pour .
* La question est alors : où remonter ? Le backjumping ne peut pas fonctionner parce que en effet possède des valeurs cohérentes avec les variables précédemment assignées : ne possède pas un ensemble de conflit complet contenant les variables précédentes qui ont causé son échec.
* On sait toutefois que les quatre variables et considérées dans leur ensemble échouent en raison d'un ensemble de variables précédentes, qui doivent être celles qui entrent directement en conflit avec ces quatre-là.
* Cela conduit à l’approfondissement de la notion d'ensemble de conflit pour une variable telle que : il s'agit de l’ensemble des variables précédentes qui empêchent , prise avec toutes les variables suivantes, d'avoir une solution cohérente.
* Dans le cas présent, l'ensemble est et .
* Il en résulte que l'algorithme devrait remonter vers et « sauter » la Tasmanie.
* On parle d'algorithme de backjumping orienté conflit (conflict-directed) pour désigner un algorithme de backjumping qui utilise des ensembles de conflit définis de cette manière.
* La méthode pour calculer ces nouveaux ensembles conflictuels est très simple.
* L'échec «terminal» d'une branche est toujours lié au fait que le domaine d'une variable devient vide.
* Cette variable a un ensemble de conflit standard.
* Dans notre exemple, échoue.
* Admettons que son ensemble de conflit soit .
* On remonte par backjumping à et absorbe l'ensemble de conflit de (à l'exception de , bien entendu) dans son propre ensemble de conflit direct, qui est .
* Le nouvel ensemble de conflit est .
* Autrement dit, il n'y a pas de solution possible à partir de compte tenu des assignations précédentes de .
* On remonte danc à , la plus récente de celles-ci.
* absorbe dans son propre ensemble de conflit direct, , ce qui donne , comme on l’a vu au paragraphe précédent.
* À présent, l'algorithme remonte à , comme on le souhaitait.
* **En résumé,** soit ***Xj*** la variable courante, et ***conf(Xj)***, son ensemble de conflit : si toutes les valeurs possibles de ***Xj*** échouent, on remonte par backjumping à la variable la plus récente ***Xi*** de ***Xj*** et on définit :



* Quand nous rencontrons une contradiction, le backjumping peut nous dire jusqu'où revenir, pour que nous ne perdions pas temps à modifier des variables qui ne régleront pas le problème.
* Mais il serait bon, aussi, d'éviter de se heurter encore au même problème.
* Quand l'exploration arrive à une contradiction, nous savons qu'un sous-ensemble de l’ensemble de conflit est responsable du problème.
* L'apprentissage de contraintes consiste à trouver un ensemble minimal de variables de l’ensemble de conflit qui cause le problème.
* Ce sous-ensemble de variables, avec leurs valeurs, est appelé variables inutiles.
* Nous enregistrons alors la liste des variables inutiles, soit en ajoutant une nouvelle contrainte au CSP soit en la gardant en mémoire cache.
* Prenons par exemple l’état à la ligne du bas de la figure 6.6.
* Le forward checking avant peut nous indiquer que cet état est inutile parce qu'il n’y a aucune assignation valide à .
* Dans ce cas particulier, il ne servirait à rien d'enregistrer les variables inutiles, parce que, une fois que nous aurions élagué cette branche de l’arbre d'exploration, nous ne rencontrerions plus jamais cette combinaison.
* Mais supposons que l’arbre d'exploration de la figure 6.6 fasse partie d’un gros arbre d'exploration qui a commencé par l’attribution de valeurs à et .
* Il serait intéressant d'enregistrer

comme inutiles, parce qu'on va rencontrer de nouveau le même problème pour chaque ensemble d’assignations possibles à et à .

* La liste des variables inutiles peut aider efficacement le forward checking ou le back-jumping.
* L'apprentissage de contraintes est une des plus importantes techniques utilisées actuellement par les résolveurs de CSP pour résoudre des problèmes complexes.