|  |
| --- |
| **Le cutset** |
| Algorithmes et applications aux CSP |
|  |
| Présentation d’algorithmes pour trouver des cutset de petite taille dans un graphe non-orienté et leurs applications dans la résolution de CSP |
|  |
| **BENZINE Abdallah, MOKHTARI Anis, BELKACEMI Sabri** |
| **01/01/2016** |
|  |

Table des matières

Partie I : Algorithmes de recherche du cutset

[I) Algorithme naïf pour trouver le cutset minimal 4](#_Toc439171608)

[II) Heuristiques pour trouver le cutset 4](#_Toc439171609)

[III) Modified Greedy Algorithm 7](#_Toc439171610)

Partie 2 : Algorithmes de résolution de CSP en utilisant le cutset

[I) L’agorithme de cycle cutset 8](#_Toc439171611)

[a) Présentation de l’agorithme 8](#_Toc439171612)

[b) Performances 15](#_Toc439171613)

[II) Déroulement de l’algorithme cycle cutset **Erreur ! Signet non défini.**](#_Toc439171614)

[II) L’algorithme MACE 17](#_Toc439171615)

[a) Présentation de l’algorithme MACE 17](#_Toc439171616)

[b) Performances 21](#_Toc439171617)

Introduction

L’intelligence artificielle est une branche de l’informatique très prometteuse. Elle aspire en effet à doter les machines des facultés de l’intelligence humaine. Parmi ces facultés-là, la capacité de résoudre des problèmes est très importante. Pour cela, différents algorithmes furent développés. On peut citer les algorithmes de recherche classique comme la recherche en profondeur d’abord, la recherche en largeur d’abord ou la recherche à coût uniforme. Néanmoins, ces algorithmes sont dits non-informés. Cela signifie qu’ils n’ont aucune connaissance à priori sur le problème. Pour le résoudre, ils doivent souvent explorer un espace de recherche très important ce qui implique une grande complexité en temps et en mémoire. Des algorithmes de recherche informés furent alors développés dont A\*. Ils se basent sur l’utilisation de fonctions heuristiques pour se guider dans l’espace de recherche. Cependant, trouver une fonction heuristique n’est pas une tâche évidente, d’autant plus que chaque heuristique est propre à un problème donné.

Le CSP est une manière standard de formuler un problème. Une fois formalisé de la sorte, celui-ci peut-être résolu de la même manière que tous les autres CSP en utilisant des algorithmes et des heuristiques générales qui ne sont pas spécifiques à un problème donné. L’enjeu est alors de trouver des algorithmes pour résoudre les CSP. Les algorithmes de résolution en arbre sont les algorithmes avec la moindre complexité pour résoudre les CSP. Ils nécessitent cependant des CSP acycliques, ce qui est rarement le cas. D’où l’idée de trouver des cutset, c'est-à-dire un ensemble de sommets qui une fois retirés du graphe rendrait celui-ci acyclique. Le cutset pourrait alors être résolu indépendamment du reste du graphe et une fois celui-ci résolu les solutions sont propagées au reste du graphe qui pourra être résolu par l’algorithme de résolution en arbre.

Le problème reste de trouver ce fameux cutset, idéalement le cutset minimal c'est-à-dire celui qui contient le plus petit nombre de sommets. Ce problème, appelé problème du cycle-cuset ou du feedback vortex set est un problème NP-Complet [KAR72]. On s’intéressera ici à des algorithmes permettant de trouver des cutsets de petite taille avec une complexité raisonnable. On verra ensuite comment les cutsets sont utilisés dans des algorithmes de résolutions de CSP. On comparera et on verra ce qui influence la performance de ces algorithmes.

Algorithmes de recherche d’un cutset

# Algorithme naïf pour trouver le cutset minimal

Un algorithme naïf afin de trouver le cutset d’un CSP à n variables consiste à énumérer toutes les combinaisons possibles de i nœuds, i=1,2 , …. Pour chaque combinaison, on retire les nœuds du graphe initial et on teste si le graphe restant est acyclique.

Cet algorithme est complet, c'est-à-dire qu’il trouve le cutset minimal pour n’importe quel graphe mais il est d’une grande complexité ( ) k étant la taille du cutset ce qui le rend inutilisable en pratique sur des csp non triviaux.

# Heuristiques pour trouver le cutset

Plutôt que trouver un cutset de taille minimale, on préférera en pratique trouver un cutset de « petite » taille avec des algorithmes peu coûteux. On peut pour cela utiliser les heuristiques suivantes :

1. On trie les nœuds du graphe par ordre décroissant de leurs degrés. En suivant cet ordre, on retire un a un les nœuds avec leur arcs et on les ajoute au cutset jusqu’à ce que le graphe devienne sans cycle [SAB97].

Considérons l’exemple de la figure 2. On y trouve un graphe de neuf sommets. On se propose d’utiliser cette première heuristique afin de trouver un cutset dans ce graphe.

Dans le tableau suivant, on trouve tous les sommets avec leurs degrés respectifs.

|  |  |
| --- | --- |
| Sommet | Degré |
| W | 5 |
| S | 4 |
| U | 4 |
| Y | 4 |
| V | 2 |
| Q | 1 |
| R | 1 |
| T | 1 |
| X | 1 |
| Z | 1 |

Figure 1 tableau de tri décroissant des sommets

On retire W du graphe, on obtient le graphe de la figure 3.

Le graphe contient encore des cycles. On retire le sommet suivant, S comme montré à la figure 4.

La graphe contient encore des cycles, on retire alors U. On aboutit alors à un graphe sans cycle. Le cutset obtenu contient alors 3 sommets : W, S et U

|  |  |
| --- | --- |
| C:\Users\AG-TEC\Documents\exposé cutset moia\exempleCutSet.bmp  figure 2 : Graphe avec cycle | C:\Users\AG-TEC\Documents\exposé cutset moia\exempleCutSet2.bmp  figure 3 : On retire le sommet W |
| C:\Users\AG-TEC\Documents\exposé cutset moia\exempleCutSet3.bmp  figure 4 : On retire le sommet S | C:\Users\AG-TEC\Documents\exposé cutset moia\exempleCutSet4.bmp  figure 5 : Graphe sans cycle |

1. On peut obtenir un cutset plus petit en testant si un nœud appartient à un cycle avant de l’ajouter au cutset [SAB97].

Reprenons l’exemple précédent. Après avoir retiré W du graphe, S n’est pas retiré car il n’appartient à aucun cycle. On aboutit alors à un cutset composé des nœuds W et U.



Figure 6 : on remet S au graphe.

1. La troisième heuristique compte dynamiquement le nombre de cycles dans lequel intervient chaque nœud et ajoute au cutset les nœuds intervenants dans le plus de cycles [SAB97].

Dans notre exemple, on compte le nombre de cycles pour chaque nœud.

|  |  |
| --- | --- |
| Nœuds | Nombres de cycles |
| Q | 0 |
| R | 0 |
| S | 1 |
| T | 0 |
| U | 3 |
| V | 1 |
| W | 2 |
| X | 0 |
| Y | 2 |

Le nœud impliqué dans le plus de cycles est U. On le retire du graphe qui devient alors acyclique. On a ainsi obtenu **le cutset minimal**.



Figure 7 : Graphe acyclique.

La troisième heuristique permet en général de trouver des cutsets légèrement plus petits que la deuxième mais est beaucoup plus coûteuse. Pour cette raison, on utilise généralement la deuxième heuristique.

# Modified Greedy Algorithm

Le Modified Greedy Algorithm est un algorithme développé par Beicker et Geiger. Il s’agit d’une amélioration du Greedy Algorithm. Il permet de trouver un cutset dont la taille au pire des cas est deux fois supérieure à la taille du cutset minimal dans un graphe non orienté et pondéré avec une complexité de O(E+VLog(V)) (avec E étant le nombre d’arrêtes du graphe et V le nombre de sommets) . Dans notre exposé, on pourra considérer que tous les sommets sont affectés d’un poids égal [BEC94].

On note ici par v un sommet d’un graphe G, par w(v) son poids, d(v) son degré et par F le cutset à construire.

L’algorithme se déroule en deux phases. D’abord, l’algorithme choisit le sommet v ayant le rapport w(v)/d(v) minimal et l’ajoute à F. V est retiré du graphe, ainsi que tous ses arcs adjacents et tous ses sommets de degré 0 et 1. Pour chaque arrête retirée, un poids de w(v)/d(v) est retiré à l’autre sommet. On répète ces étapes jusqu’à ce que le graphe devienne vide. La deuxième phase enlève les sommets en trop du cutset [BEC94].

**MGA**

Entrée : Un graphe non orienté G(V, E, w)

Sortie : A cutset F

Retirer tous les sommets de degré 0 et 1 de de G et inséré le graphe obtenu dans Gi

Tant que Gi n’est pas vide

Choisir le sommet vi pour lequel w(vi)/d(vi) est minimal

Retirer tous les sommets de degré 0 et 1 de de G et inséré le graphe obtenu dans Gi

Pour chaque arrête e={u1, u2} retiré durant ce processus faire :

Pour a 1 faire

Si chaque cycle dans Gi qui coupe {vi} coupe aussi F\{vi] alors

Algorithmes de résolutions de CSP en utilisant le cutset

On a vu quelques algorithmes qui permettent de trouver des cutset dans des graphes. Nous présenterons dans cette partie des algorithmes qui permettent de résoudre les CSP de façon efficace une fois que le Cutset a été trouvé.

# L’algorithme de cycle cutset

## Présentation de l’algorithme

**L’algorithme du cycle-cutset** a été développé par Dechter et Perl en 1988. Il se base sur partition du csp en deux ensembles : le cutset et son complément. Le cutset est résolu, puis les solutions trouvées sont propagées et le reste du csp est résolu par l’algorithme de résolution en arbre. Si aucune solution n’est trouvée, l’algorithme réessaye avec une autre solution au cutset [DEC90].

* **Algorithme de cycle-cutset**

1) Diviser le graphe en deux ensembles

Un cutset C et son complément T

2) Trouver une (autre) solution uniquement avec les variables de C

Si aucune solution ne peut être trouvée alors retourner echec

3) Appliquer la consistance des arcs entre C et T

Si un des domaines devient vide alors retourner à l’étape 2

4) Utiliser l’algorithme de résolution en arbre pour aboutir à une solution complète

* **Algorithme de résolution en arbre**

**ResolutionArbre(d=X1, X2, …, Xn)**

SI directAcrConsistency(d)

TrouverSolution(d)

Sinon

Retourner echec

**directAcrConsistency(d)**

Pour i allant de n à 1

Pour chaque arc(Xj, Xi) ; j inferieur à i

Reviser(Xj, Xi)

Si Xj est vide

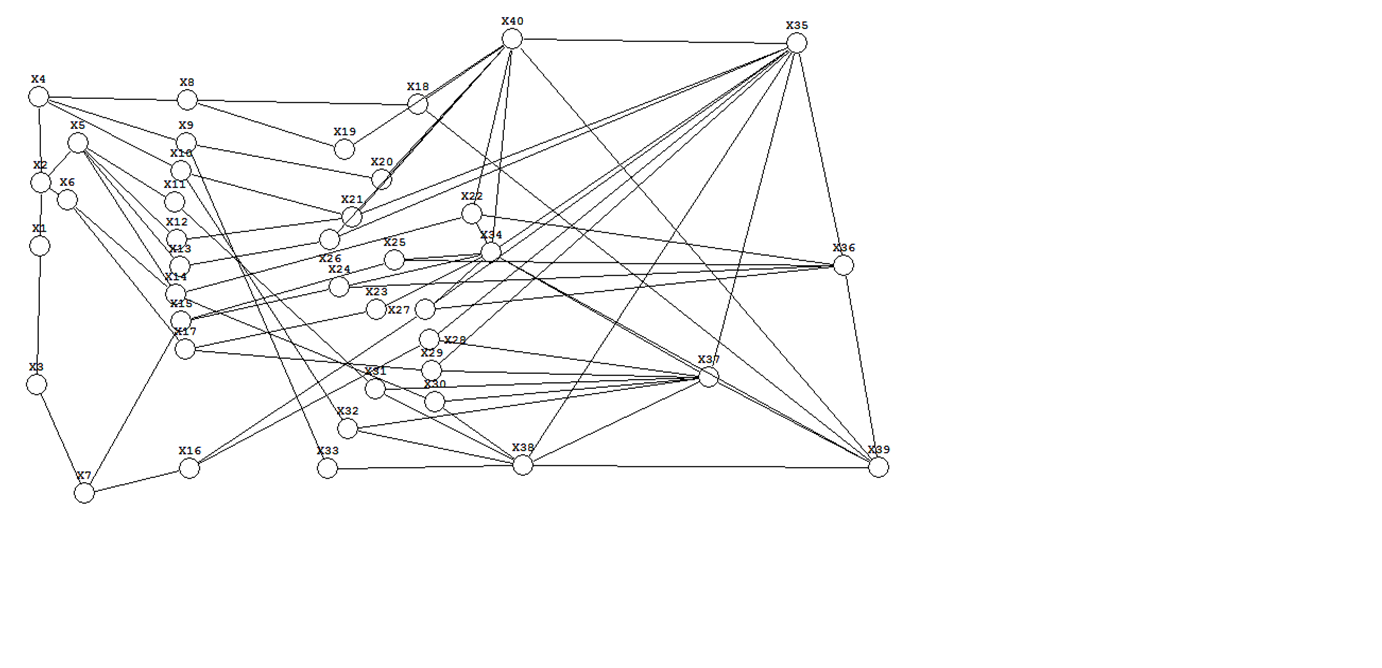
Retourner echec

La procédure TrouverSolution se contente d’affecter une valeur consistante à chacune des variables de d.

Reviser(Xj, Xi) supprime de Xj les valeurs qui ne sont pas consistantes avec Xi.

## Déroulement de l’algorithme

On va ici dérouler l’algorithme de cycle-cutset. On doit d’abord trouver un cutset de petite taille.

 Figure 08 : Exemple de graphe a colorier.

1. On trie les nœuds du graphe par ordre décroissant de leurs degrés.

|  |  |
| --- | --- |
| Point | Dégrée |
| X35 | 10 |
| X40 | 9 |
| X37 | 9 |
| X34 | 9 |
| X38 | 7 |
| X39 | 6 |
| X36 | 6 |
| X5 | 5 |
| X2 | 4 |
| X4 | 4 |
| X27 | 4 |
| X22 | 4 |
| X21 | 4 |
| X14 | 4 |
| X32 | 3 |
| X31 | 3 |
| X30 | 3 |
| X29 | 3 |
| X28 | 3 |
| X26 | 3 |
| X25 | 3 |
| X24 | 3 |
| X18 | 3 |
| X17 | 3 |
| X16 | 3 |
| X15 | 3 |
| X10 | 3 |
| X9 | 3 |
| X8 | 3 |
| X7 | 3 |
| X6 | 3 |
| X33 | 2 |
| X23 | 2 |
| X20 | 2 |
| X19 | 2 |
| X13 | 2 |
| X12 | 2 |
| X11 | 2 |
| X3 | 2 |
| X1 | 2 |

Figure 09: Tri décroissant des nœuds par leur degré.

2. Une fois que c’est fait on passe à la deuxième étape qui consiste à retirer les nœuds du plus grand degré un a un jusqu’à ce qu’il n y est plus de cycle.

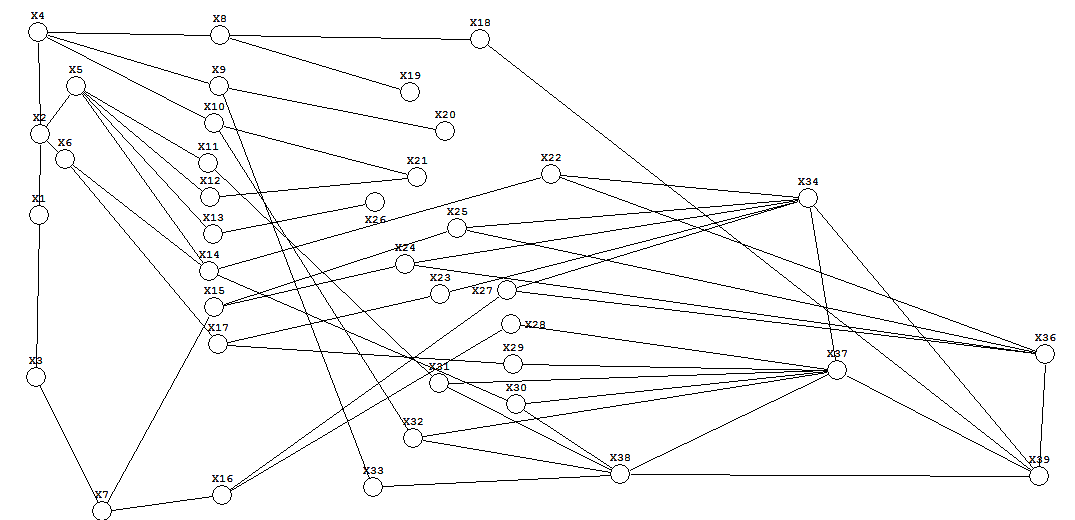
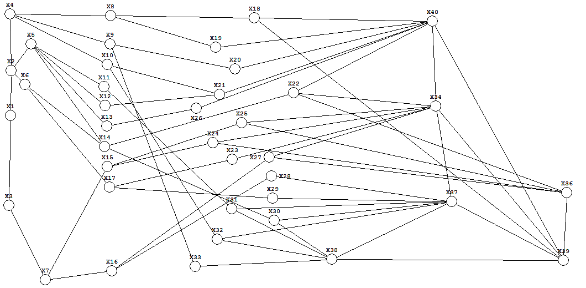


Figure 10 : on retire x35 . figure 11 : on retire X40.

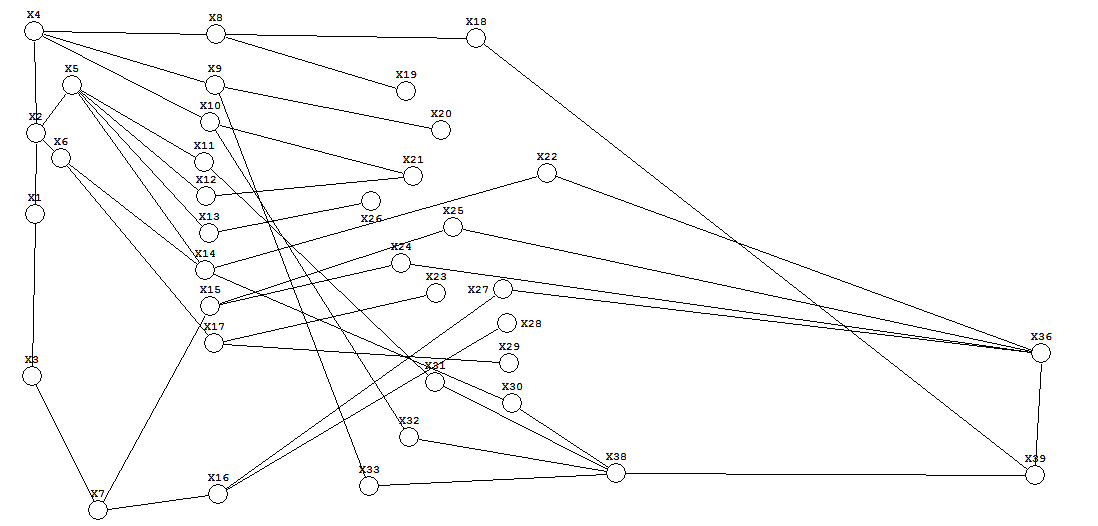
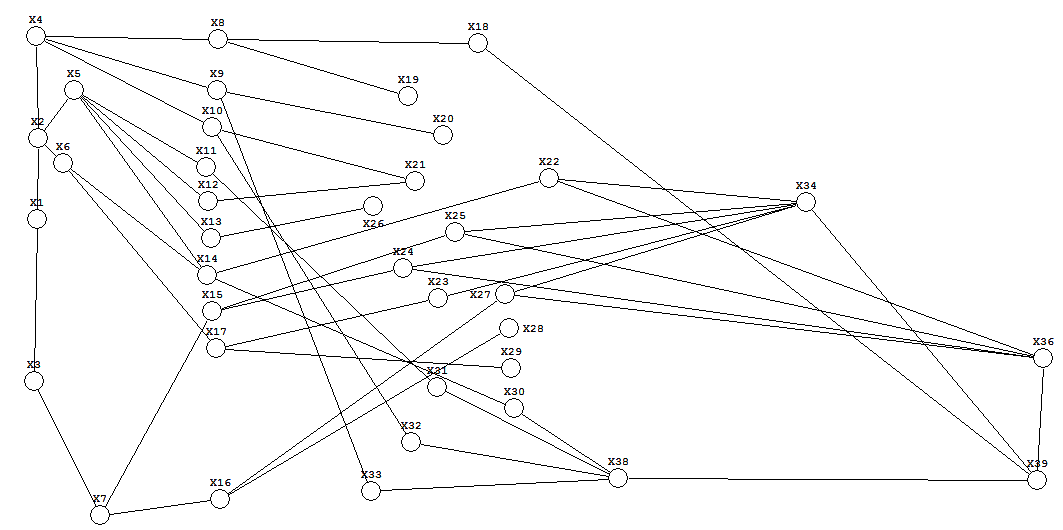


Figure 12 : on retire X37. Figure 13 : on retire X34.

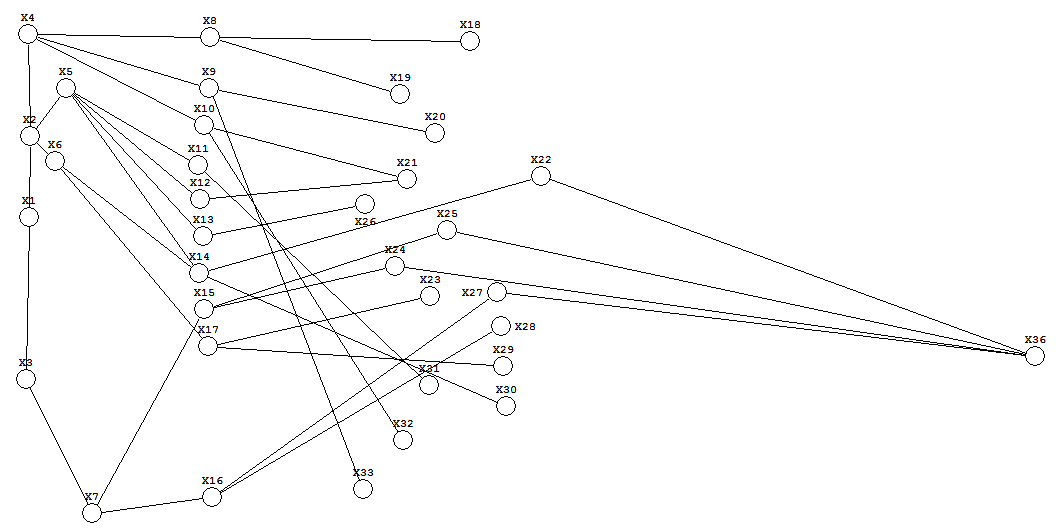
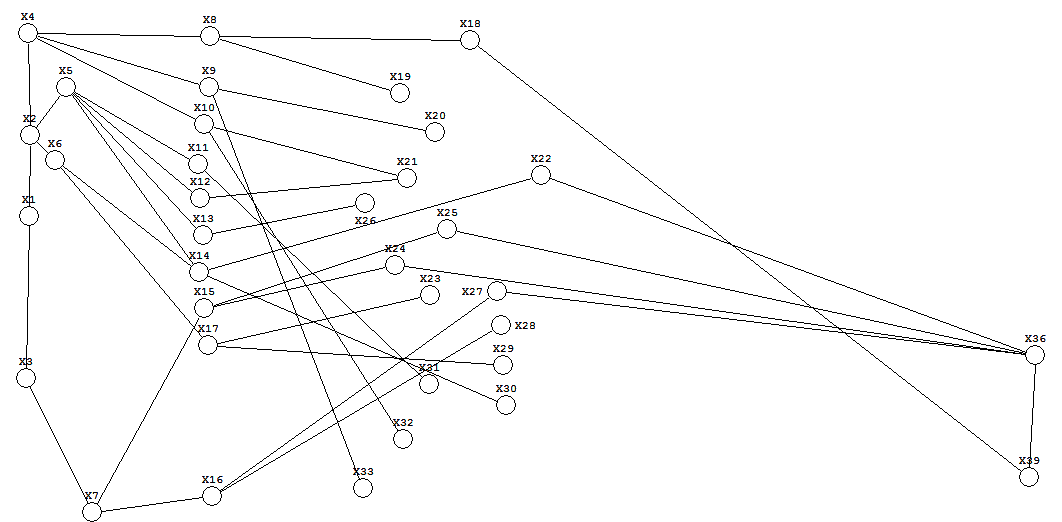


Figure 14 : on retire X38 figure 15 : on retire X39

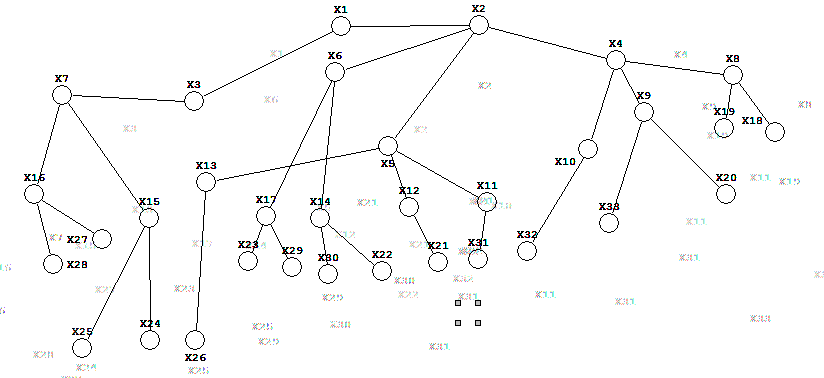


Figure 16 :Graphe acyclique.

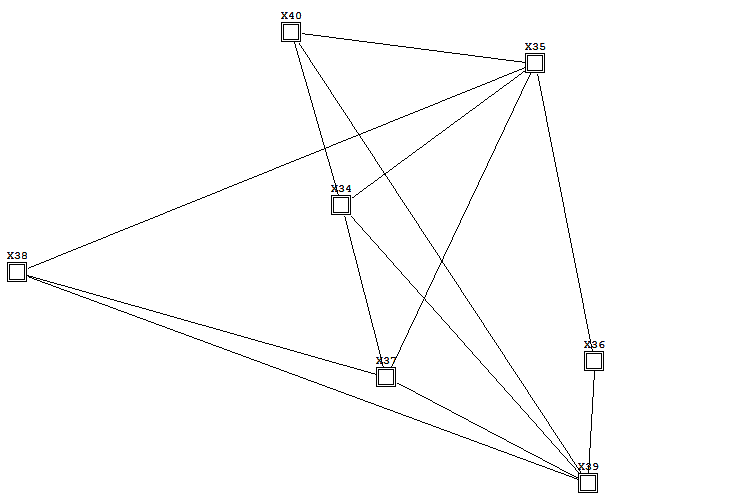


Figure 17 : le cutset.

Sur cet exemple on va appliquer le forward checking pour l’affectation des valeurs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K | X40 | X35 | X34 | X38 | X36 | X37 | X39 | L40 | L35 | L34 | L38 | L36 | L37 | L39 |
| 0 | - | - | - | - | - | - | - | RBJV | RBJV | RBJV | RBJV | RBJV | RBJV | RBJV |
| 1 | R | - | - | - | - | - | - | BVJ | BJV | BJV | RBJV | RBJV | RBJV | BVJ |
| 2 | R | V |  |  |  |  |  | BJ | BJ | BJ | RBJ | RBJ | RBJ | BVJ |
| 3 | R | V | B |  |  |  |  | J | J | J | RBJ | RBJ | RJ | VJ |
| 4 | R | V | B | R |  |  |  |  |  |  | BJ | RBJ | J | VJ |
| 5 | R | V | B | R | J |  |  |  |  |  |  | RB | J | VJ |
| 6 | R | V | B | R | J | J |  |  |  |  |  |  |  |  |
| 7 | R | V | B | R | J | J | V |  |  |  |  |  |  |  |

Figure 18 : Trace de recherche avec le forward checking.

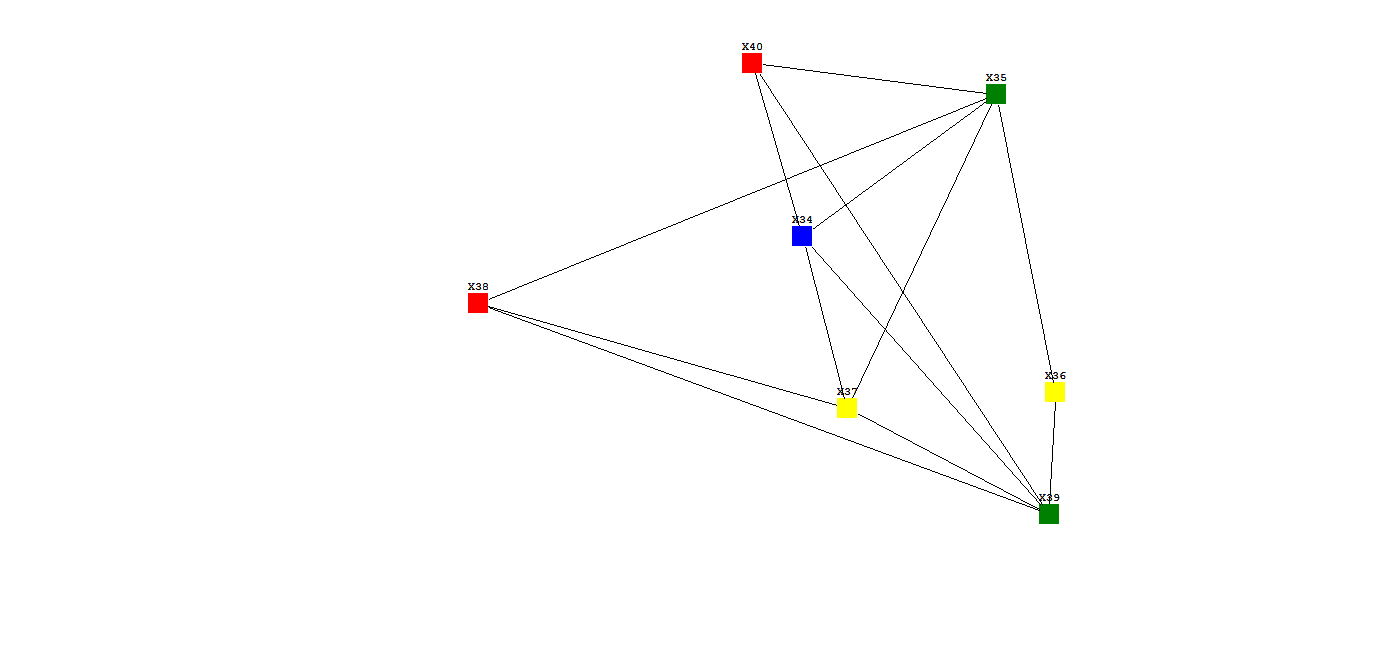


Figure 19 :Résolution du cutset.

On passe maintenant à l’algorithme de résolution en arbre.

**Première étape** : faire la consistance des arcs.

On verra ici qu’il y a des sommets singleton, dans ce cas-là on introduira la notion de consistance directe.

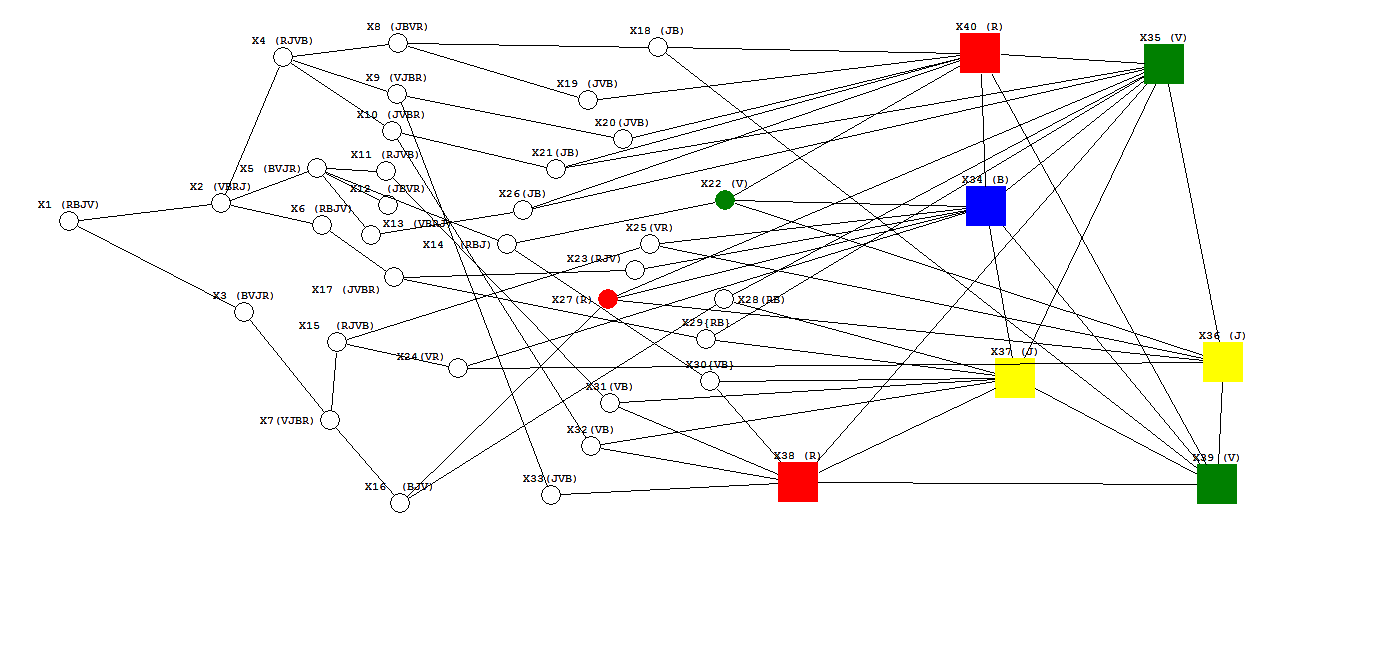


Figure 20 :Consistance directe des arcs X22 et X27.

Une fois celle-ci terminé on passera la dernière étape de la résolution.

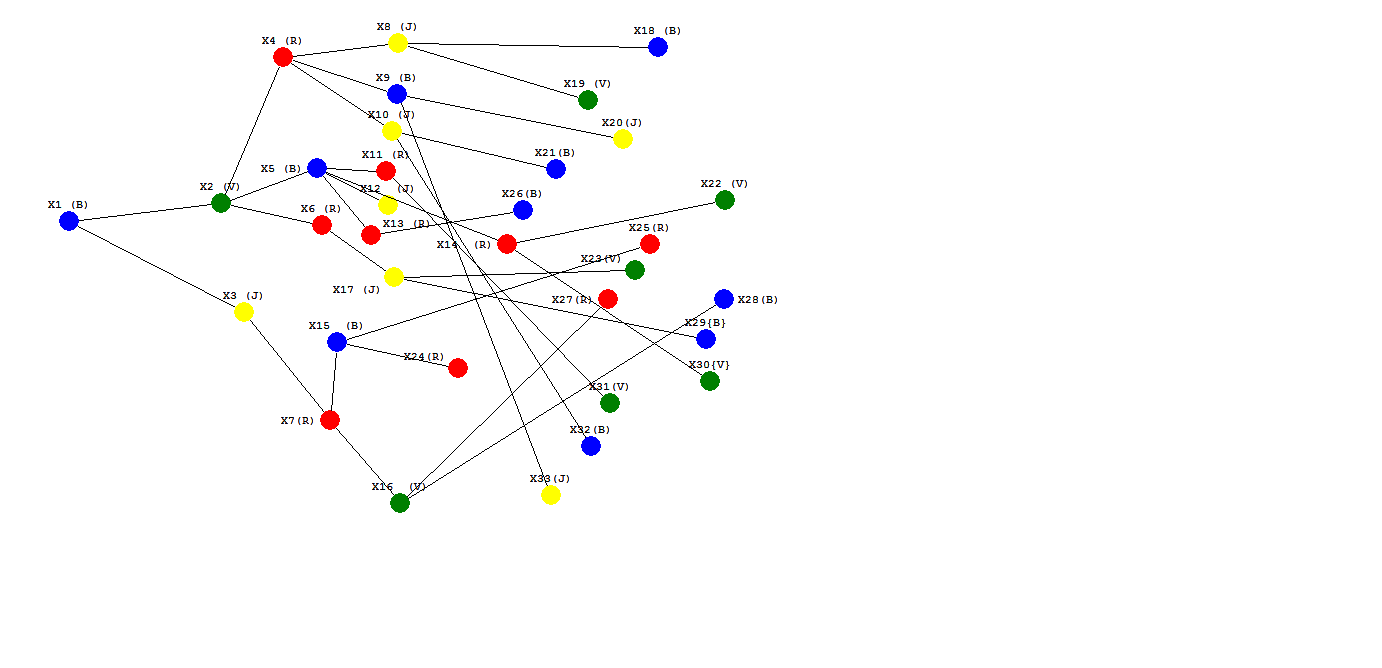


Figure 21 :Résolution de l'arbre.

## Performances

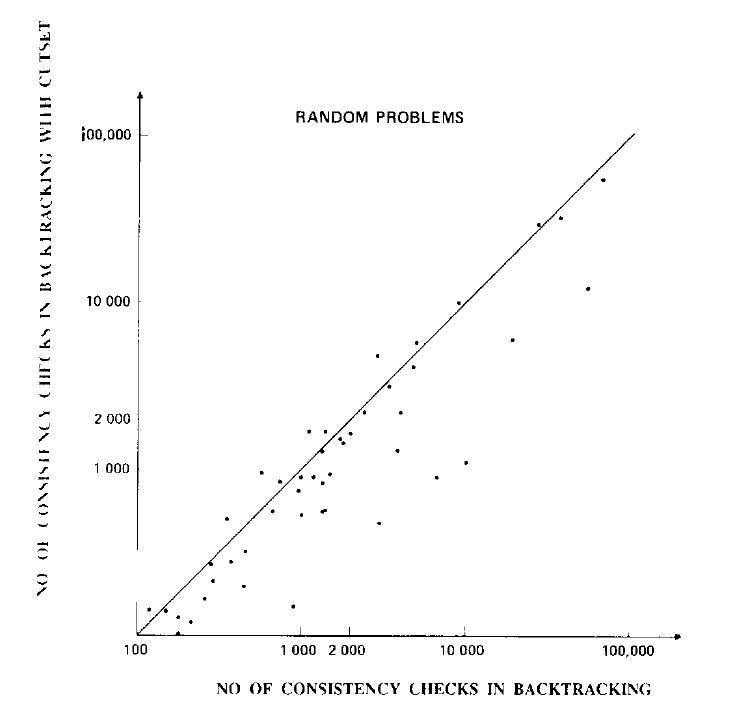
Dans [DEC90], on trouve une comparaison entre l’algorithme du cycle-cutset avec un algorithme de backtraking classique. 

Figure 22 : Comparaison de l'algorithme cycleCutset avec Backtracking.

Dans la **figure 22**, on compare le nombre de fois où la consistance d’un nœud est testée pour cycle-cutset par rapport au nombre de nombre de fois où elle est testée pour backtrack pour des csp générés aléatoirement.

La droite représente une performance égale pour les deux algorithmes. Les points en dessous la droite représente les csp où cycle-cutset est meilleur. On note que la plus part des points sont sous cette droite.

La figure ci-après **23** montre la relation entre la performance de cycle-cutset et la taille du cutset. Sur l’axe des x est représenté le rapport entre la taille du cutset et la taille du graphe et l’axe des y donne le rapport entre les performances de backtrack et de cycle-cutset. Les points en dessus de y=1 montre les CSP où cycle-cutset surpasse backtrack . On remarque que pour des cutsets de petite taille (ratio inférieur à 0.3), la plus part des points sont en dessous de cette droite. Dans certains cas cependant, surtout pour des ratios supérieurs, l’algorithme de backtrack se montre plus performant.

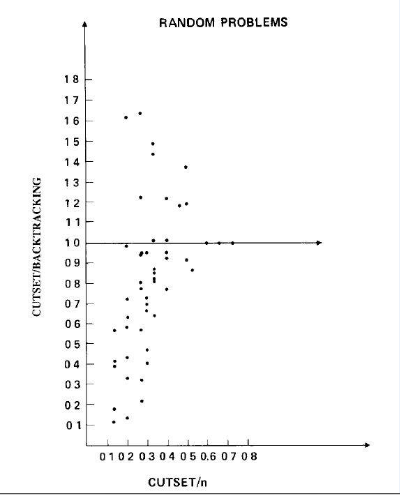


Figure 23 : Relation entre la performance de cycle-cutset et la taille du cutset.

# L’algorithme MACE

## Présentation de l’algorithme MACE

Tout comme backtrack ou Forward Cheking, Mac est un algorithme de résolution de CSP. Une fois une valeur affectée à une variable, Forward checking revise les domaines des noeuds adjacents en leur enlevant les valeurs inconsistantes avec cette affectation. L’algorithme MAC (Maintening Arc Consitency) ne se contente pas de vérifier la consistance des arcs des nœuds adjacents mais maintient la consistance des arcs sur tout le graphe. Cela permet de détecter des valeurs inconsistantes beaucoup plus tôt et de réduire le nombre de backtracks. Néanmoins, MAC a un overhead important. Maintenir la consistance des arcs sur tous les nœuds du graphe est coûteux ainsi que de restaurer le problème à un état antérieur en cas de mauvaise affectation [SAB97].

L’algorithme MACE (MAC extended) une amélioration d’algorithme MAC. Il repose sur deux principes : instancier moins et propager moins. Pour cela, il se base sur le cutstet du graphe. Au lieu de maintenir l’ensemble du graphe consistant, MACE maintient seulement une partie de celui-ci, le cutset en l’occurrence, consistant. Cela le fait un graphe plus petit à maintenir, avec moins de variables à instancier et moins de propagation ce qui permet un gain significatif en performances comme nous le verrons plus loin. De plus, lorsque durant la résolution le cutset contient des singletons ceux-ci sont retirés ainsi que toute composante acyclique qui pourra être résolue par l’algorithme de résolution en arbre [SAB97].

**L’algorithme MACE**

1) Appliquer la consistance des arcs (si ce n’est pas possible retourner échec)

2) Diviser les variables en deux ensembles:

Un cutset C et son complément U

Déconnecter U du graphe

3) Tant que C#0

3a) Affecter une valeur à une variable de C

3b) Appliquer la consistance des arcs

Si elle échoue retourner à 3a

3c) Déconnecter les singletons (les ajouter à U)

3d) Déconnecter les variables qui ne participent à aucun cycle (les ajouter à U)

4) reconnecter les variables à U

Appliquer la consistance des arcs directe de C à U

5) Appliquer l’algorithme de résolution en arbre pour une solution complète.

## Déroulement de l’algorithme MACE

On déroule l’algorithme sur l’exemple étudié précédemment :

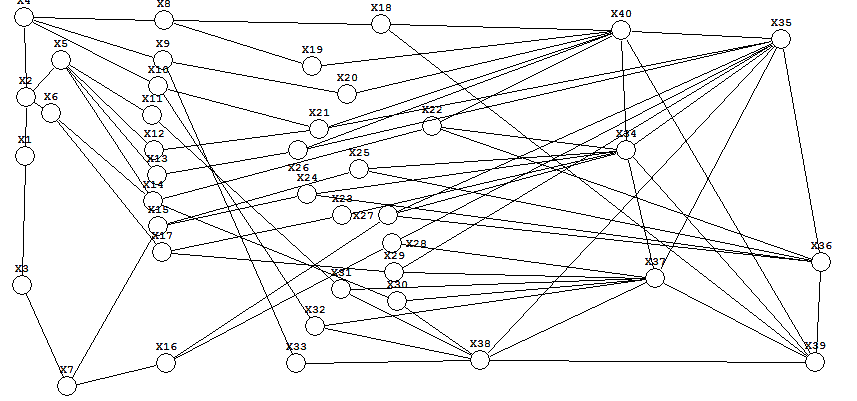


Figure 24 :Graphe d'origine

**Première étape :**

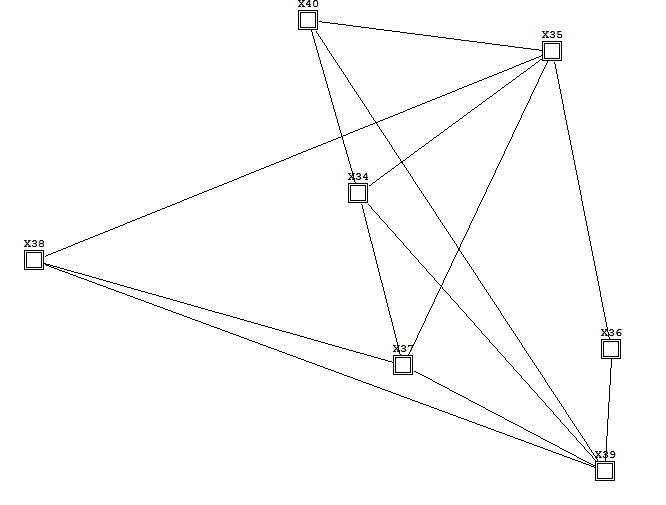
Trouver le CUTSET, on réutilise le cutset trouvé précédemment.

Figure 25 :représente le CUTSET.

**Deuxième étape :**

On procède à la résolution du cutset on utilisant l’algorithme AC-3.

Les figures suivantes représentent les étapes de résolution.

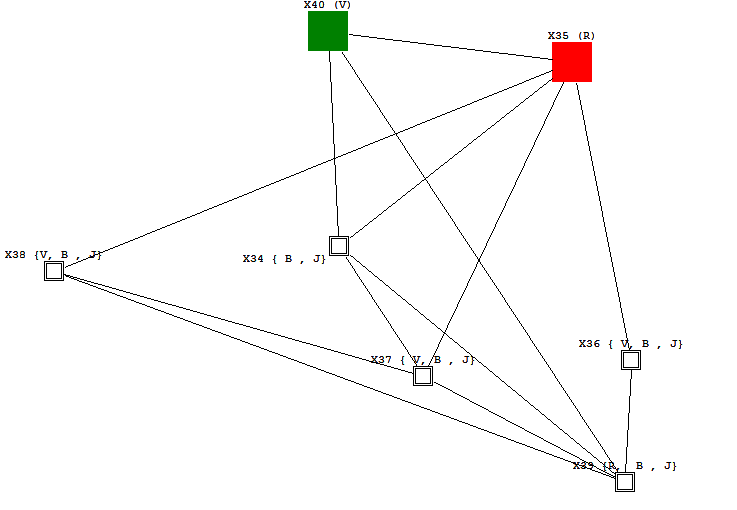
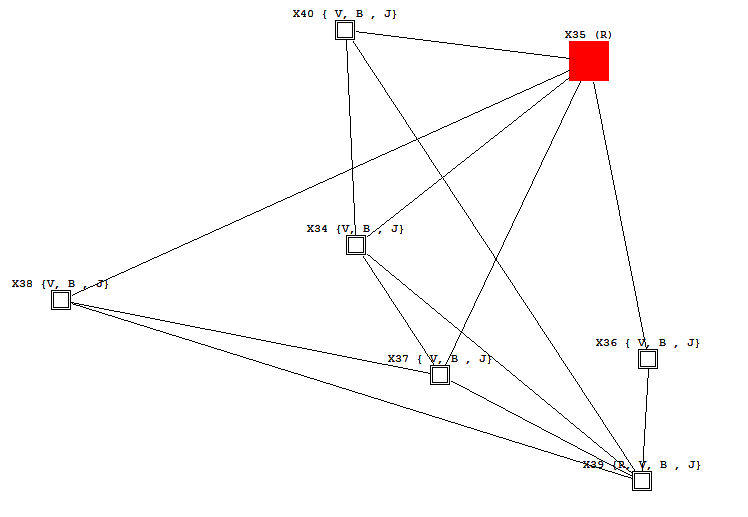


Figure2  : affectation du sommet X35. Figure2 affectation du sommet X40.

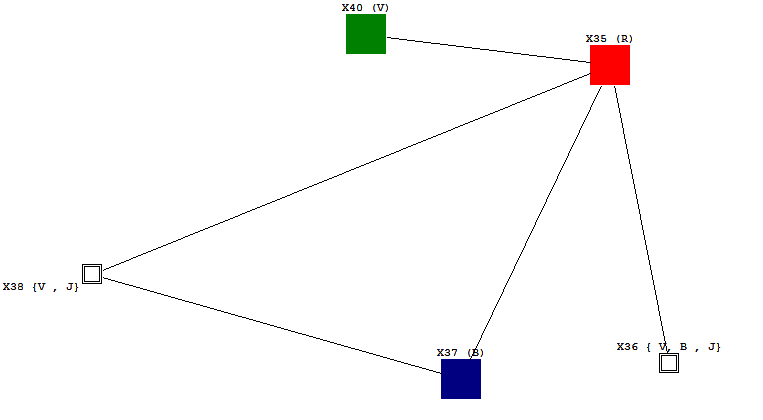
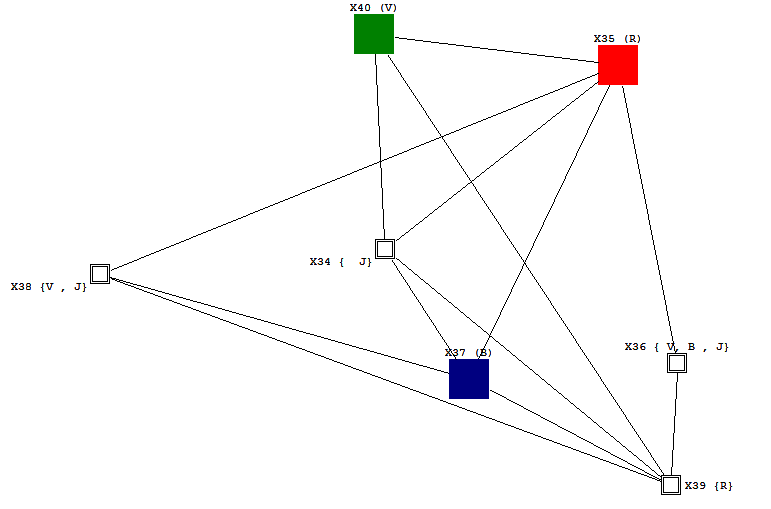


Figure 28 :affectation du sommet X37.

Figure 29 :affectation du sommet X37.

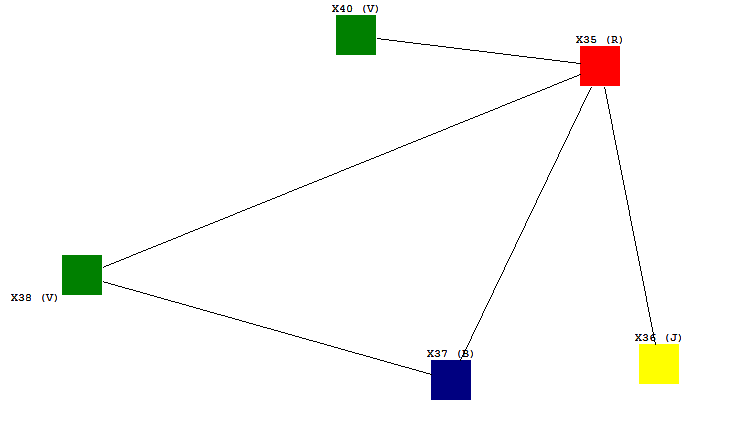


Figure 30 : affectation du sommetX38 et X36.

En cours de la résolution on retrouvera des nœuds **singleton** on les ajoutes au graphe d’origine.

Une fois le cutset résolus on reconnecte ce dernier au complément.

On passe maintenant a l’algorithme de résolution de l’arbre.

**Premiere étape :**

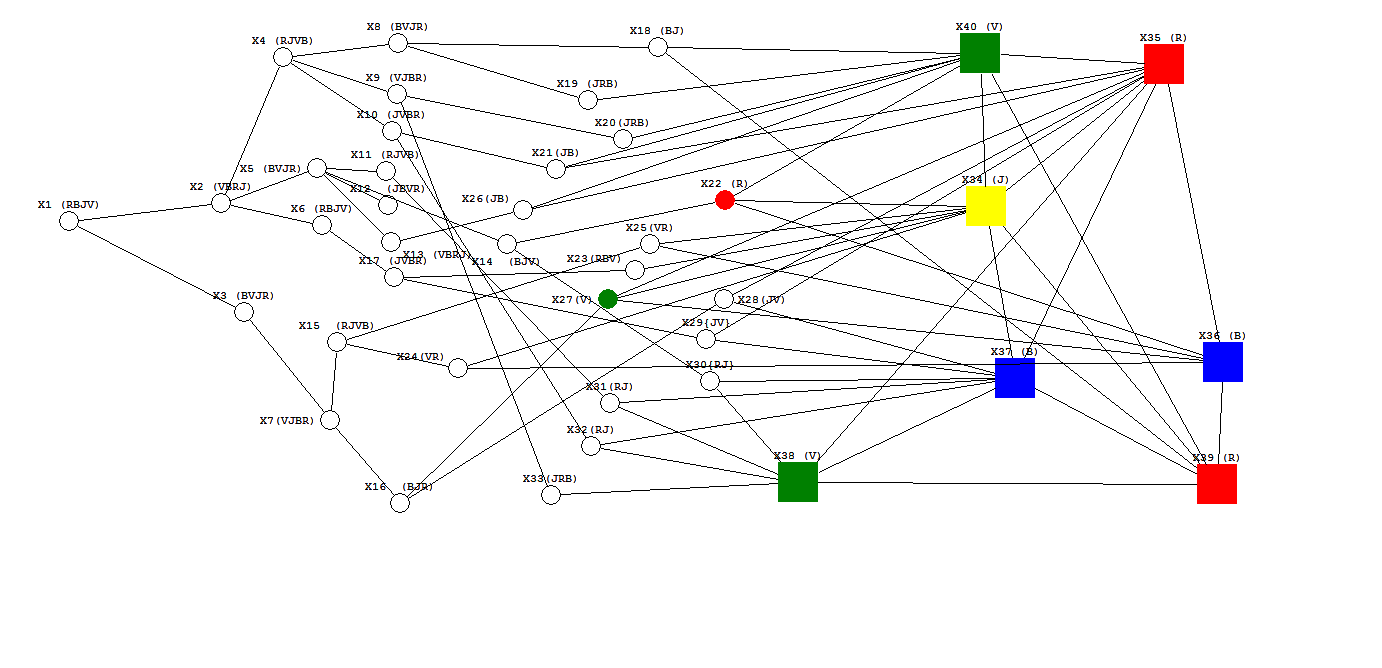


Figure 31 :Consistance des arcs.

Une fois cette étape terminé on enchaine avec la résolution.

**Le résultat final :**

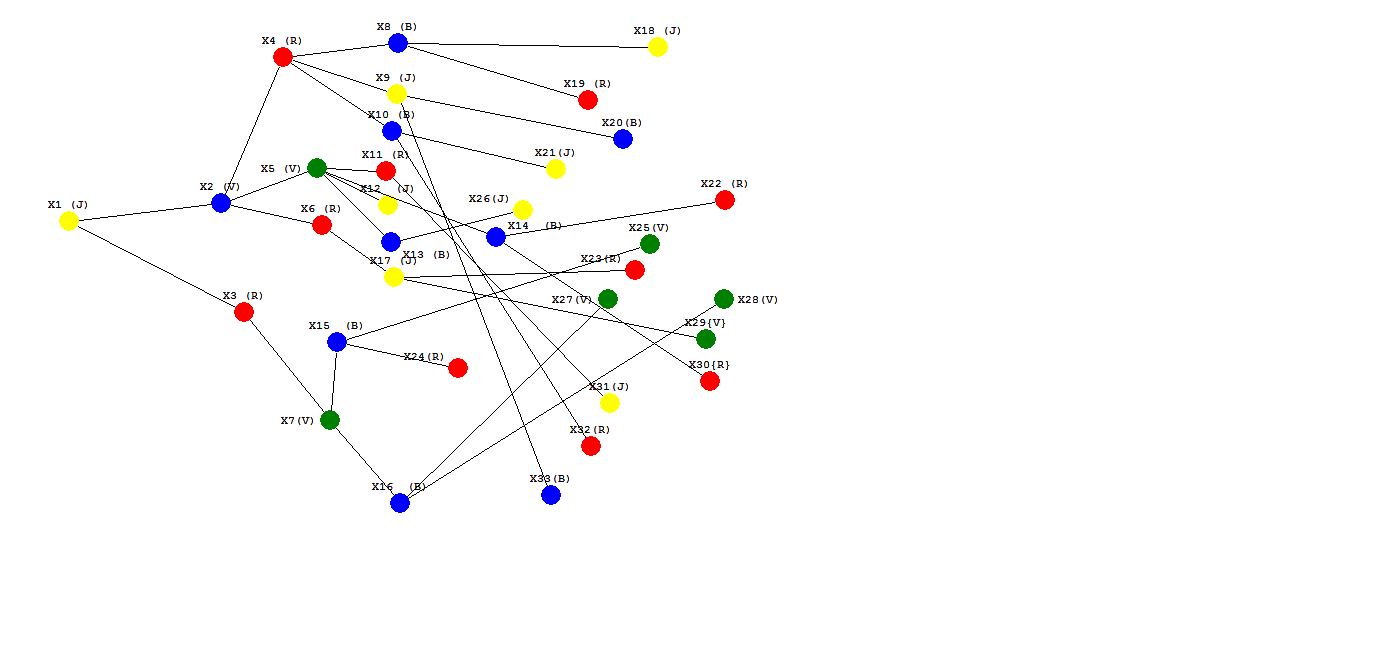


Figure 32 : Résolution de l'algorithme Mace.

## Performances

Dans [SAB97] , les auteurs comparent les performances de leur algorithme MACE avec celui du cycle-cutset et l’algorithme MAC.

Dans la figure suivante 32 , on trouve le rapport entre les performances de cycle-cutset et de MACE en fonction de la *tightness* de CSP choisis aléatoirement*.* La tightness est la fréquence de toutes les valeurs du domaines de deux variables qui ne sont pas autorisées par une contrainte [SAB97].

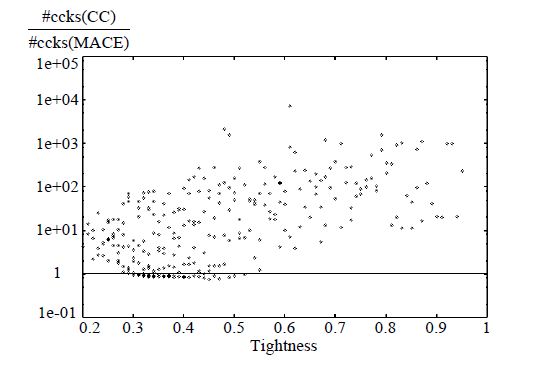


Figure 33 : Rapport entre les performances de cycle-cutset et de MACE

On remarque que dans la plus part des cas, MACE outrepasse cycle-cutset. Même conclusion pour le comparatif avec MAC :

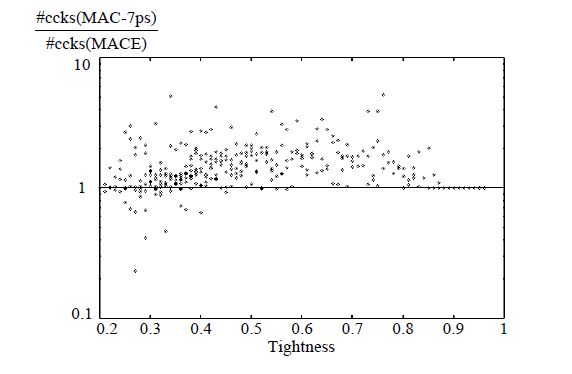


Figure 34 : comparatif avec MAC.

Conclusion

Trouver un cutset de taille minimale est un problème délicat mais des algorithmes d’approximation ont été développés pour trouver des cutsets de petite taille comme les heuristiques vues ici ou le MGA. Un autre algorithme de 2-approximation a été développé par Bafna et al avec une complexité de [BAF99] Les chercheurs se penchent encore sur ce problème.

Une fois le cutset trouvé, des algorithmes spécifiques peuvent être utilisés pour résoudre le CSP. L’algorithme du cycle-cutset de Dechter se montre dans la plus part des cas plus performant que l’algorithme de backtrack. Les expérimentations montrent aussi que cet algorithme se montre plus efficace quand le cutset est de petite taille par rapport au graphe entier. Cette efficacité peut-être accrue en utilisant des algorithmes plus performants comme le montre MACE et les expérimentations menées.

Références

**[KAR72]**Karp, Richard M. *Reducibility among combinatorial problems*. springer US, 1972.

**[SAB97]** Sabin, Daniel, and Eugene C. Ereuder. "Understanding and improving the MAC algorithm." *Principles and Practice of Constraint Programming-CP97*. Springer Berlin Heidelberg, 1997. 167-181.

**[BEC94]**BECKER, Ann. Approximation algorithms for the loop cutset problem. In :*Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1994. p. 60-68.

**[DEC90]** Dechter, Rina. "Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition." *Artificial Intelligence* 41.3 (1990): 273-312.

**[BAF99]**Bafna, Vineet, Piotr Berman, and Toshihiro Fujito. "A 2-approximation algorithm for the undirected feedback vertex set problem." *SIAM Journal on Discrete Mathematics* 12.3 (1999): 289-297.