République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Bejaia
Faculté des Sciences Exactes
Département d'Informatique

Mémoire de Master II en Informatique Professionnel

Option

Génie Logiciel

Thème

La Résolution de CSP par les Méthodes de Décompositions Arborescentes.

Présenté par :

- **ARAB Salim**
- **BOUCHEKHCHOUKH Mourad**

Dirigé par : Dr AMROUN Kamal

Devant le jury composé de :

Président : LEKADIR. O
Examinateur : SABRI. S
Examinateur: GHANEM. S

Année Universitaire 2016/2017

Juin, 2017

Table des Matières

Liste des Figures

Liste des Algorithmes

Liste des Abréviations

-AC: Arc-Consistance

-FC: Forward Checking

-MAC: Maintaining Arc-Consistance

-BJ: Back Jumping

-CBJ: Conflict Back Jumping

-TD: Tree Decomposition

-TC: Tree clustering

-LEX-BFS:

-LEX-M:

-MCS: Maximum Cardinality Search

-H-TD-WT:

-BTD: Backtrack on Tree Decomposition

Remerciements

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce travail.
En second lieu, nous tenons à remercier notre encadreur Dr AMROUN Kamal chef de département informatique, qui nous a bien accueillez depuis le jour du dépôt du dossier de passerelle du système classique vers le système LMD et pour ses précieux conseils ainsi pour son aide durant toute la période du travail.

Nos vifs remerciements vont également aux membres de jury pour l'intérêt qu'ils ont porté à notre mémoire en acceptant d'examiner notre travail et de l'enrichir par leurs propositions.

Nos remerciements à notre ami YOUS ATMANE qui nous à aider.

Je remercie ma femme qui est toujours à mes côtés dans des moments difficiles, qui ma soutenue à reprendre les études en sacrifiant tout son temps pour le bien être de notre petite famille.(Mourad.B)

Mes premières pensées vont à ma famille, qui m'a toujours soutenu et encouragé (Salim Arab).

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

.

Résumé

Résoudre un CSP constitue un problème NP-Complet. Devant cette difficulté, des recherches ont été faites et qui ont conduits à définir de nombreuses méthodes de résolution qu'on peut classer en deux approches : une repose sur l'exploration complète de l'espace de recherche, tandis que l'autre utilise des heuristiques. Parmi, on compte les méthodes de décomposition arborescente. De ces méthodes, il y a celles qui décomposent le graphe de contraintes associé au CSP sans le triangulé comme l'heuristique H-TD-WT proposée par Ciril Tireoux. L'inconvénient principale de celle-ci est le choix du premier Cluster et de limité sa taille maximale qui nécessite des heuristiques. Cela augmente le temps de résolution.

La résolution de CSP par les méthodes basées sur la triangulation fait le bon choix du premier cluster de départ, diminue la taille et donne des solutions proches de l'optimum.

LEX-M est t'une méthode de triangulation qui date de 1976 qui donne une triangulation minimale et de largeur minimale, le choix du sommet de départ est basé sue l'ordre lexicographique cela nous permis de triangulé le graphe de contraintes sans faire recours à des heuristiques. Le fait que LEX-M n'a pas d'autres méthodes concurrentes, en termes du temps d'exécution ou de complexité théorique, la triangulation par LEX-M est un bon choix.

L'implémentation de LEX-M nous a conduits à proposer de l'améliorée en éliminant les sommets isolés et d'arrêter le déroulement de l'algorithme aux deux derniers sommets non numérotés.

Le domaine de recherche est ouvert pour toute proposition.

Anglais

Introduction générale

Dans le monde réel plusieurs problèmes sont représentés par des contraintes, leur résolution consiste à les satisfaire et pour se faire plusieurs méthodes ont fait naissance.

Le problème de satisfaction de contraintes ou CSP, nom donné à tout problème qui est constitué de plusieurs contraintes et que sa résolution consiste à trouver une solution qui va satisfaire toutes ces contraintes, une formulation est donnée au CSP pour le simplifier et le modéliser sous forme mathématique.

Un CSP est constitué de variables qui peuvent recevoir des valeurs. Ces variables sont liées par des contraintes. Une solution d'un CSP est une affectation des valeurs aux variables qui ne viole aucune contrainte.

Le principal problème résidé dans le fait que résoudre un CSP constitue un problème NP-Complet. Devant cette difficulté, des recherches ont été faites et qui ont conduits à définir de nombreuses méthodes de résolution qu'on peut classer en deux approches : une repose sur l'exploration complète de l'espace de recherche, tandis que l'autre utilise des heuristiques.

La première approche utilise des techniques de recherche énumérative pour trouver la solution tel que l'algorithme BACKTRACK ou encore l'algorithme FORWARD CHECKING qui est basé sur le filtrage afin de réduire l'espace de recherche et simplifier le problème. La combinaison de ses différentes techniques permet de trouver des algorithmes plus performants.

La deuxième approche, cherche à décomposer le problème en sous problèmes qu'on appelle CLUSTER qui facilite la recherche dans ces ensembles réduits. Cette approche qu'on appelle Décomposition Arborescente utilise des techniques qui réduits les clusters à des cliques dans le seul but est de trouver une solution optimale au CSP et en un minimum de temps et d'espace.

Ce mémoire est organisé en trois chapitres :

Le chapitre I présent le formalisme CSP et exemple d'illustration ainsi que les différentes approches de résolution qui existe.

Le chapitre II est consacré à l'introduction relative aux différentes méthodes de décomposition arborescentes et présentation de méthodes exploitants la décomposition arborescente pour la résolution de CSP, des méthodes basées sur la triangulation et d'autres sans triangulation.

Le chapitre III est consacré à l'implémentation de l'algorithme LEX-M, présentation de la structure de données adoptée et un exemple de déroulement à titre d'illustration de notre application. On concluant par un algorithme amélioré de LEX-M qu'on a dénommé LEX-M SI-2

CHAPITRE -I- Les problèmes de satisfaction de contrainte

Introduction :

Les problèmes de satisfaction de contrainte sont un cadre générique introduits par MONTANARI dans les années 70 et qui permettent de formaliser un très grand nombre de problèmes tels que les problèmes de coloration de graphe, les problèmes d'ordonnancement, les problèmes de placement,…etc. Un CSP est défini par un ensemble de variables et de contraintes reliant les variables entre elle. Les variables prennent leurs valeurs dans un ensemble fini de valeurs ou domaine.

1. Les Problèmes de Satisfaction de Contraintes

Les problèmes de satisfaction de contraintes ou CSP (Constraint Satisfaction Problem) sont au cœur de nombreuses applications en Intelligence Artificielle et en Recherche Opérationnelle. Un CSP est un formalisme à la fois simple et puissant permettant de représenter et de résoudre un grand nombre de problèmes. Un CSP se définit comme un ensemble de contraintes impliquant un ensemble de variables.

L'objectif est de trouver une valeur pour chaque variable afin de satisfaire l'ensemble des contraintes ou bien de satisfaire le maximum de contraintes. La recherche dans le domaine des CSP est motivée par la découverte de méthodes toujours plus efficaces en pratique pour les résoudre. Mais, comme les CSP font partie de la classe des problèmes combinatoires NP-Complets, les algorithmes connus pour les résoudre nécessitent un temps exponentiel en fonction du nombre de variables.

La méthode la plus employée pour les résoudre est la recherche énumérative qui consiste à explorer systématiquement un arbre de recherche.

1.1 Définitions de base :

Définition 1 :

Un problème de satisfaction de contrainte (***CSP***) est un quadruplet (X, D, C, R) où :

- $X = (x_1, x_2, \ldots, x_n)$: est un ensemble de n variables ;
- $D = (D_1, D_2, \ldots, D_n)$: est un ensemble de n domaines.
- $C = (C_1, C_2, \ldots, C_m)$: est un ensemble de m contraintes
- $R = (R_1, R_2, \ldots, R_m)$: est un ensemble de m relations. Chaque relation Ri définit L'ensemble des n_i-uplets sur $D_{i1} \times D_{i2} \ldots \times D_{ini}$ autorisés par la contrainte C_i.

Définition 2 : Le domaine d'une variable est l'ensemble des valeurs, discret ou continu que peut prendre celle-ci. Ainsi, une variable est dite numérique si les valeurs qu'elle peut prendre sont entières. Elle est dite booléenne si ses valeurs sont booléennes et symbolique dans le cas où les valeurs qu'elle prend sont un ensemble énuméré d'objets.

Définition 3 : une affectation ou une instanciation est un ensemble de couples (variable, valeur) exprimant l'association des valeurs aux variables. Cette affectation est dite totale si toutes les variables ont été instanciées et on parle de ***k-affectation*** si k variables ont été instanciés.

Définition 4 : Une contrainte est une relation logique ou une propriété devant être vérifiée par un sous ensemble de variables. Une contrainte permet de restreindre les valeurs pouvant être prises simultanément par un ensemble de variables. Une contrainte peut être implicite, définie par une expression logique. Elle peut être explicite et donc définie par une relation composée de tous les tuples autorisés. Ainsi, une contrainte est définie par un ensemble de variable et la relation entre ces variables.

Définition 5 : une k-affectation est dite ***consistante*** si elle ne viole aucune contrainte. Une ***solution*** d'un CSP est une affectation totale consistante, c'est à dire une assignation d'une valeur à chaque variable de X tel que toutes les contraintes soient satisfaites. Un CSP est dit ***consistant*** s'il possède au moins une solution, dans le cas contraire le CSP est dit ***inconsistant***.

Exemple : Problème des 4-reines

Le problème des 4 reines est un problème académique qui consiste à placer 4 reines sur un échiquier 4*4 de telle sorte que 2 reines ne soient pas en prise mutuelle. Deux reines ne peuvent être ni sur la même ligne, ni sur la même colonne, ni sur la même diagonale. Ce problème peut être modélisé comme un CSP de plusieurs façons. Ici on illustre l'exemple par une des façons possibles. P = (X, D, C, R) tel que

- $X = (x_1, x_2, x_3, x_4)$ où x_i représente le numéro de colonne de la reine placée sur la ligne i.
- $D = (D_1, D_2, D_3, D_4)$ tel que Di = {1, 2, 3, 4} (la colonne de la reine sur la ligne i)
- $C = (C_{12}, C_{13}, C_{14}, C_{23}, C_{24}, C_{34})$ tel que C_{ij} = {xi, xj}
- $R = (R_{12}, R_{13}, R_{14}, R_{23}, R_{24}, R_{34})$ tel que pour chaque R_{ij} :

$x_i \neq x_j$ (les deux reines ne se trouvent pas sur la même colonne)

$x_i \neq x_j + (j - i)$

$x_i \neq x_j - (j - i)$ (les deux reines ne se trouvent pas sur la même diagonale)

-

les relations R_{ij} de contraintes sont :

R_{12} = {(1,3),(1,4) ,(2,4),(3,1),(4,1),(4,2)}

R_{13} = {(1,2),(1,4) ,(2,1),(2,3),(3,4),(4,1) ,),(4,3)}

R_{14} = {(1,2),(1,3) ,(2,1),(2,3),(2,4),(3,1),(3,2),(3,4),(4,2),(4,3)}

R_{23} = {(1,3),(1,4) ,(2,4),(3,1),(4,1),(4,2)}

R_{24} = {(1,2),(1,4) ,(2,1),(2,3),(3,4),(4,1),(4,3)}

R_{34} = {(1,3),(1,4) ,(2,4),(3,1),(4,1),(4,2)}

Une solution a ce problème est $x_1 = 2$, $x_2 = 4$, $x_3 = 1$, $x_4 = 3$.

2. Résolution des CSP : différentes méthodes

Pour les problèmes CSP, il n'existe pas de méthodes universelles pour une résolution efficace. La résolution d'un CSP consiste à parcourir toutes les combinaisons possibles ou lors d'explorer un arbre de recherche (arbre de résolution) afin de trouver une ou toutes les solutions.

Pour résoudre les CSP plusieurs approches ont été explorées :

On identifie deux grandes familles au sein de ces techniques de résolution. D'une part, les méthodes complètes (ou exactes), d'autre part, les méthodes incomplètes (ou approchées) La figure (Fig. 1.1**) [1]** illustre les différentes approches de résolution des problèmes CSP.

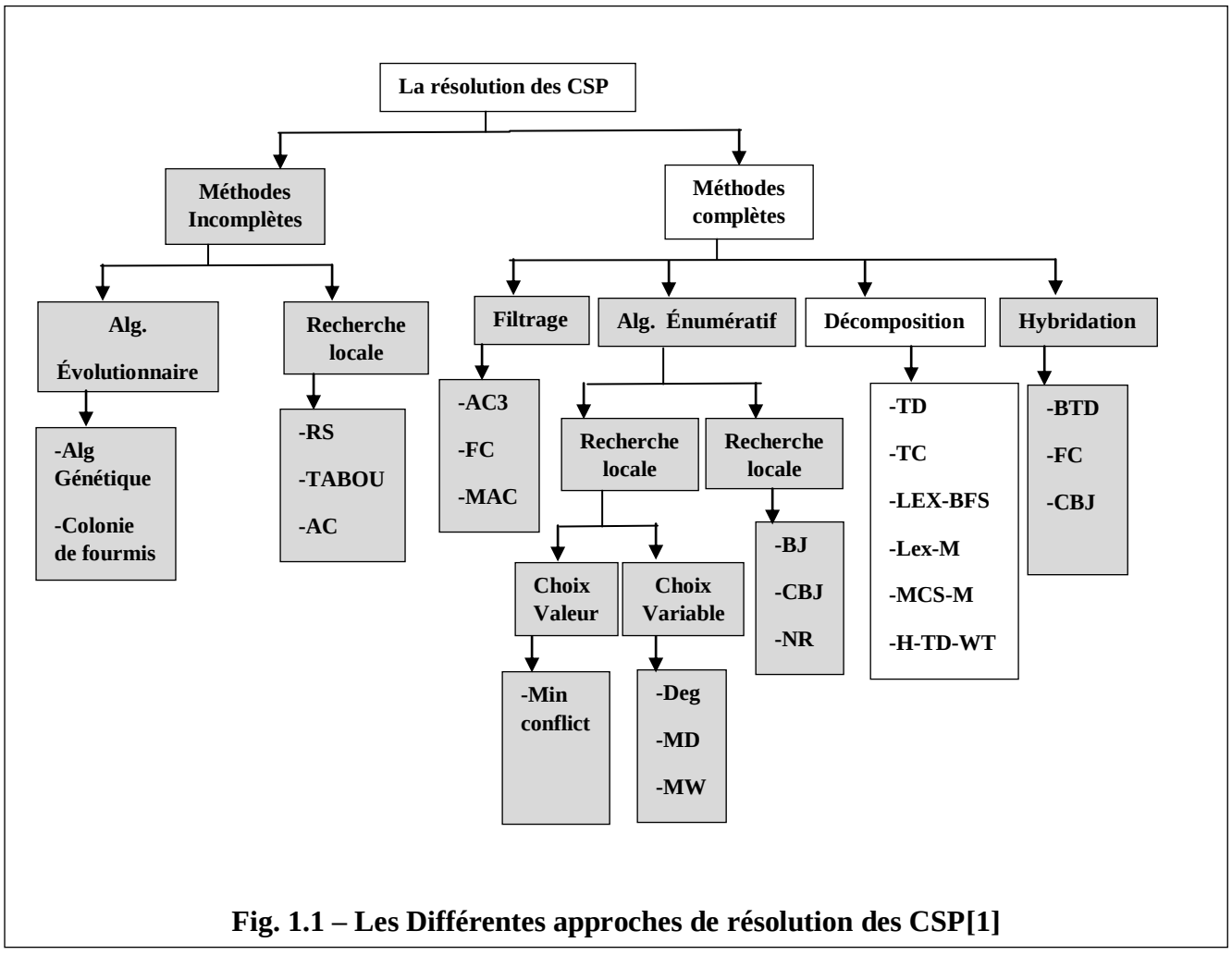


Fig. 1.1 – Les Différentes approches de résolution des CSP[1]

2.1 Les méthodes incomplètes

Les méthodes incomplètes considèrent l'espace de recherche dans sa totalité mais elles ne l'explorent qu'en partie en se dotant des combinaisons d'heuristiques pour choisir les zones d'exploration, ces combinaisons sont appelées méta heuristique. Ces méthodes visent à donner un résultat acceptable en un temps raisonnable mais elles ne peuvent prouver l'inexistence de solution d'un problème sur contraint. Autrement dit, elles abordent la résolution d'un CSP comme un problème d'optimisation combinatoire pour lequel il s'agit de calculer une affectation satisfaisant le plus grand nombre de contraintes, l'objectif final étant de les satisfaire toutes.

Les différentes approches possibles d'une résolution incomplètes appartiennent a deux grandes familles de méta-heuristiques, celle de la recherche locale avec ses algorithmes : recuit simulé (RS), la recherche Tabou, colonies de fourmis (AC, Ant Columns), et celle dites évolutionniste avec les algorithmes génétiques.

2.2 Les méthodes complètes

Les méthodes complètes explorent totalement l'espace de recherche et elles sont toujours capables de répondre par vrai ou faux concernant l'existence d'une solution. En effet, elles peuvent prouver la satisfiabilité d'un problème, tout comme déterminer l'ensemble des solutions de ce problème.

Les nombreux travaux qui ont été réalisés dans cet axe et qui tentent d'améliorer ces méthodes de résolution peuvent être classés en trois grandes classes :

2.2.3 Les méthodes énumératives

Ces algorithmes consistent à visiter toutes les configurations possibles de l'espace de recherche (l'ensemble des affectations possibles des variables). L'algorithme de type Backtracking constitue la méthode de base. Cet algorithme, beaucoup trop coûteux, a conduit à la recherche d'algorithmes intelligents les plus efficaces.

2.2.4 Les méthodes de consistance (filtrage par consistance)

Ces algorithmes visent à réduire l'espace de recherche à explorer pour simplifier les instances avant ou pendant la recherche d'une solution. Ils sont utilisés comme des algorithmes de prétraitement pour améliorer le travail des algorithmes de recherche en utilisant des techniques de consistance d'arc, de chemin,...etc.

2.2.5 Résolution par décomposition des CSP

Ces algorithmes exploitent le fait que la traitabilité d'un CSP est reliée à ses propriétés structurelles. Etant donné une instance CSP, ces algorithmes transforment cette instance en une instance acyclique constituée d'un ensemble de sous problèmes dont la complexité est inférieure à celle de l'instance originale.

Nous nous intéressons dans la suite de ce mémoire aux méthodes complètes c'est pourquoi nous détaillons dans ce qui suit ces algorithmes.

3. Résolution par des méthodes complètes

La résolution des problèmes CSP par des méthodes complètes se base, généralement, sur des algorithmes de recherche énumérative de type Backtrack. Le Backtrack combiné à des heuristiques (telles que les techniques de retour arrière intelligent et de l'ordre des variables), et à des techniques de filtrage, induit, généralement, de bonnes performances en pratique. La résolution par décomposition présente des bons résultats comme nous allons voir dans la suite.

3.1Résolution par des algorithmes énumératifs

Les algorithmes énumératifs effectuent un parcours systématique de l'espace de recherche. La manière la plus simple est de générer toutes les configurations possibles, c'est à dire toutes les combinaisons possibles de valeurs des variables, et de tester si elles vérifient les contraintes. Cette approche est connue sous le nom de Generate and test. Cependant, l'algorithme de base et le plus répandu pour une recherche systématique est celui du Backtrack chronologique [2].

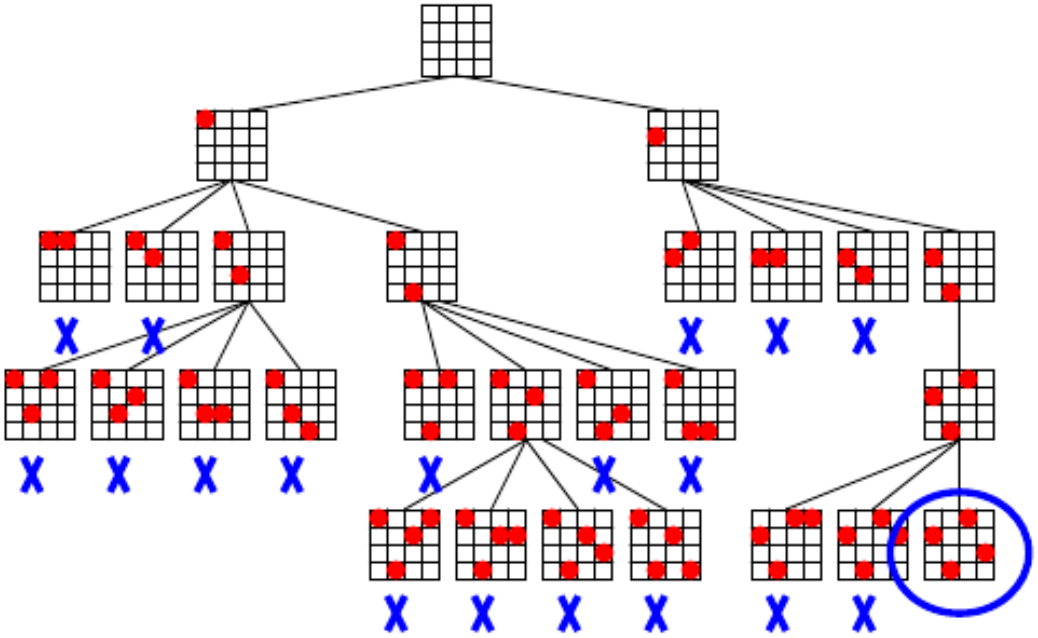
3.1.1 Algorithme Backtrack (BT)

Le *backtracking* est sans doute la méthode la plus répandue pour une recherche systématique. Pour cette méthode, les variables sont instanciées les unes après les autres et ce jusqu''à obtenir une affectation complète. Cette méthode teste la faisabilité à chaque étape de la résolution, c'est-`a-dire que pour chaque instanciation de variables, les contraintes dont les variables sont déjà instanciées sont vérifiées, sur l'affectation partielle courante.

Ainsi, lorsqu''a une étape donnée, l'affectation partielle courante viole une contrainte, le *backtracking* supprime le sous-espace de recherche en dessous du point de choix.

La figure 1.2 représente l'arbre de recherche du *backtracking* applique au problème des 4-reines. Dans un premier temps, l'algorithme place la première reine en haut à gauche de l'échiquier, ensuite la deuxième reine ne trouve de place viable qu'en troisième ligne.

La position de la 2e reine en troisième ligne rend impossible le placement d'une nouvelle, ce qui entraine un retour sur sa position et la place en 4e ligne et ainsi de suite.



Algorithme 1 : Algorithme Backtrack

1 : Entrée : Un CSP P = (X, D, C, R), et une affectation A de ce CSP. Initialement A = **Ø.**
2 : Sortie : Une solution du CSP s'il est consistant, déterminer qu'il est inconsistant.
3 : **si** A est non consistant **alors**
4 : Retourner Faux.
5 : **sinon**
6 : si A est une affectation totale **alors**.
7 : Retourner vrai /* A est une solution */
8 : **sinon**
9 : choisir une nouvelle variable X_i de X
10 : **pour** toute valeur V_i de $D(X_i)$ **faire**
11 : **si** Backtrack (A U $\{(X_i, V_i)\}$, (X, D, C, R)) = vrai **alors**.
12 : Retourner vrai.
13 : finsi.
14 : fin pour.
15 : finsi
16 : Retourner Faux.
17 : Finsi

3.1.2 Algorithme Forward checking (FC) [3]

Algorithme pour éviter les tests inutiles cet algorithme est assez proche du BACKTRACK, il affecte des valeurs aux variables au fur et à mesure. La différence avec la procédure, vue précédemment, se trouve dans la gestion des impasses, quand l'algorithme ne trouve plus de solutions.

Le FORWARD CHECKING, avant de choisir une valeur pour une variable x_n, vérifie que cette affectation correspond aux contraintes et vérifie que les autres variables x_i (i>n) pourront être affectées. Si l'affectation de x_i ne peut être faite, l'algorithme choisi une autre valeur pour x_n. Si jamais aucune solution, pour x_n, ne permet l'affectation des autres variables, la procédure fera un retour en arrière sur x_{n-1} pour changer sa valeur. Ce point reste identique au BACKTRACK. Si le CSP n'a pas de solution, on recule jusqu'à la variable x_0. Voici l'algorithme du FORWARD CHECKING.

Algorithme 2 Algorithme Forward Checking (FC)

1 : Entrée : pour un CSP P = (X, D, C, R), l'entrés est le quadruplet (A, NA, D, C) avec A est
Une affectation, initialement vide (A= {}), NA est l'ensemble des variables non
Encore instancier, initialement NA= X, Dest l'ensemble des domaines et C celui
Des contraintes.
2 : Sortie : Une solution du CSP si le problème est consistant, sinon déterminer que le CSP
Est inconsistant.
3 : si NA = {} **alors**
4 : Retourner A /*A est une solution*/
5 : sinon
6 : Choisir x de NA
7 : répéter
8 : Choisir une valeur v de D_x : $D_x = D_x - \{v\}$;
9 : **si** A ∪ {x, v} ne viole aucune contrainte **alors**
10: $D^{'}$ = Revise (NA – {x}, D, C, <x, v>)
11 : **si** aucun domaine de $D^{'}$ n'est vide **alors**
12: Result ←FC (NA – {x}, A+ <x, v>, $D^{'}$, C)
13 : **si** Result ≠ Null **alors**
14 : Retourner (Result)
15 : **finsi**
16 : **finsi**
17 : **finsi**
18 : jusqu'à D_x = {}
19 : Retourner (Null)
20 : finsi

CONCLUSION :

Dans ce chapitre, nous avons présenté le cadre CSP ainsi quelques approches de résolution proposées dans la littérature , les méthodes dites énumératives à savoir la méthode Backtrack et la méthode Forward checking. Même si ces méthodes sont relativement efficaces en pratique, leurs bornes de complexité théorique sont de l'ordre exponentiel. Afin de borner cette complexité théorique, nous allons présenter dans le chapitre qui suit des techniques qui permettent de réduire l'espace de recherche de manière significative basées sur la décomposition arborescente du graphe de contraintes.

2. Méthodes de décomposition arborescente

La résolution par décomposition est une technique qui analyse les CSP et définit un schéma de décomposition avant la recherche d'une solution. Autrement dit, elle utilise la décomposition comme un prétraitement pour la résolution.

La décomposition arborescente vise à partitionner un graphe en groupes de sommets, qu'on appelle clusters. Ces clusters forment un graphe acyclique qui constitue un arbre de jonction du graphe original. Plusieurs algorithmes de résolution, exploitant la structure du graphe de contraintes, utilisent la notion de décomposition arborescente. L'intérêt de cette approche vient du fait que beaucoup de problèmes difficiles peuvent être résolus efficacement si leur largeur de décomposition est faible **[4] [5]**.

Plan du chapitre. Tout d'abord, nous introduisons les notions relatives à la décomposition arborescente et les différentes méthodes de calcul de ces décompositions. Ensuite, nous présentons plusieurs méthodes exploitant les décompositions arborescentes pour la résolution de CSP, des méthodes basée sur la triangulation et d'autres sans triangulation. Enfin, nous concluons et motivons le choix de la méthode de décomposition que nous avons retenue.

2.1 Définitions

Les notions de cliques et de clusters permettent d'identifier des sous-parties du graphe de contraintes.

Définition 6 (Cluster, clique)**.** Soit G = (V, E) un graphe. Tout sous-ensemble C ⊆ V de l'ensemble des sommets de G est appelé cluster de G. On dit que le cluster **C** est une clique si et seulement si ∀ {u, v} ∈ C^2 |u ≠ v, {u, v} ∈ E. Si de plus ∀z ∈ V \C, C ∪ {z} n'est pas une clique, alors **C** est une clique maximale de G.

La décomposition arborescente vise à diviser un graphe en clusters et à organiser ces clusters sous la forme d'un arbre de jonction (ou arbre de clusters).

Définition 7 (Décomposition arborescente)**.** Soit G = (V, E) un graphe, on appelle décomposition arborescente de G un couple ($C_{T,\ T}$) ou` T = (*I*, F) est un arbre avec *I* l'ensemble des sommets, F l'ensemble des arêtes et C = {C_i|i ∈ *I* } est une famille de sous-ensembles de V qui vérifie :

- $\mathbf{U_{i\in\ I}\ C_i = V}$ **;**
- **∀{v, w} tel que {v, w} ∈ E, ∃ i ∈ I avec v ∈ C_i et w ∈ C_i ;**
- **∀(i, j, k) ∈ I^3 , si j est sur le chemin de i à k dans T alors $C_i \cap C_k \subseteq C_j$.**

Définition 8 (Largeur d'arbre d'un graphe G)**.** On définit par $\max_{i\in I}$ ($|C_i| - 1$) la largeur d'une décomposition. La largeur d'arbre d'un graphe G est la largeur minimale sur toutes ses d´décompositions arborescentes.

Définition 9 (séparateur)**.** L'ensemble des variables partagées entre deux clusters C_i et C_j, noté sep (C_i, Cj), est appelé séparateur : sep (C_i, Cj) = $C_i \cap C_j$

Définition 10 (Voisinage, degré)**.** Soit C_i un cluster de C_T. On appelle voisinage de C_i, noté vois(C_i), l'ensemble des clusters C_j qui partagent des variables avec C_i.
Vois(C_i) = {C_j|sep (C_i, Cj) $\neq \emptyset$}
Le degré d'un cluster C_i est le cardinal de vois(C_i).

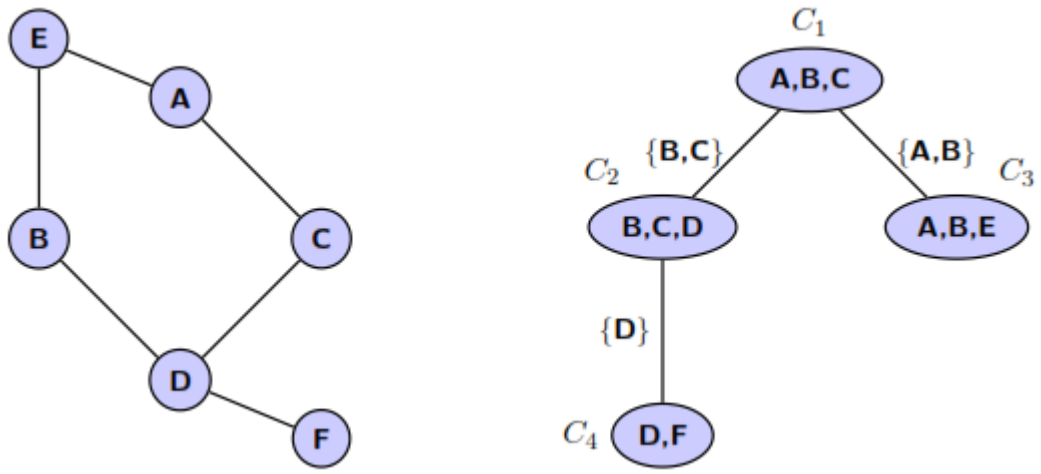
Définition 11 (Variable propre)**.** On appelle variables propres de Ci l'ensemble $V_p(C_i)$ des variables qui n'appartiennent qu'au cluster C_i :

$$\mathbf{V_p(C_i) = C_i \setminus \bigcup_{C_j \in vois(C_i)} sep(C_i, C_j)}$$

Nous noterons parent(C_i) (resp. fils(C_i)) le parent (resp. l'ensemble des fils) de C_i dans T.

CHAPITRE -II- TREE DECOMPOSITION

La figure 2.1 donne un exemple de décomposition arborescente. Sa largeur d'arbre vaut 2. Chaque C_i correspond à un ensemble de sommets du graphe qu'on appelle **clusters**. Les arêtes entre les clusters sont étiquetées par l'ensemble des sommets communs aux clusters qu'elles relient. On appelle cet ensemble un **séparateur**.



(a) Graphe original G = (V, E). (b) Décomposition arborescente de G de Largeur 2.

Fig 2.1 – Exemple de décomposition arborescente.[22]

Pour un graphe donné, il y a un grand nombre de décompositions arborescentes possibles. Calculer une décomposition de largeur minimale est un problème NP-difficile. Seules les décompositions de largeur proche de la largeur d'arbre vont nous intéresser. Il existe deux types de méthodes pour calculer une décomposition arborescente d'un graphe :

- La première méthode cherche à identifier les composants fortement connectes du graphe qui vont constituer les clusters de la décomposition, puis à former un arbre vérifiant les propriétés de la définition 8 (qu'on appelle méthode de décomposition arborescente avec triangulation).
- La seconde méthode se base sur l'identification des séparateurs. Elle vise à trouver des ensembles de variables qui vont former des **coupes** minimales du graphe. Ces ensembles constitueront les séparateurs entre les clusters de la décomposition. (méthode de décomposition arborescente sans triangulation).

Nous allons présenter ces deux types de méthodes dans les sections suivantes.

2.2 Méthodes de décomposition basées sur la triangulation

2.2.1 Triangulation de graphe

La triangulation consiste à transformer un graphe quelconque en un graphe cordal.

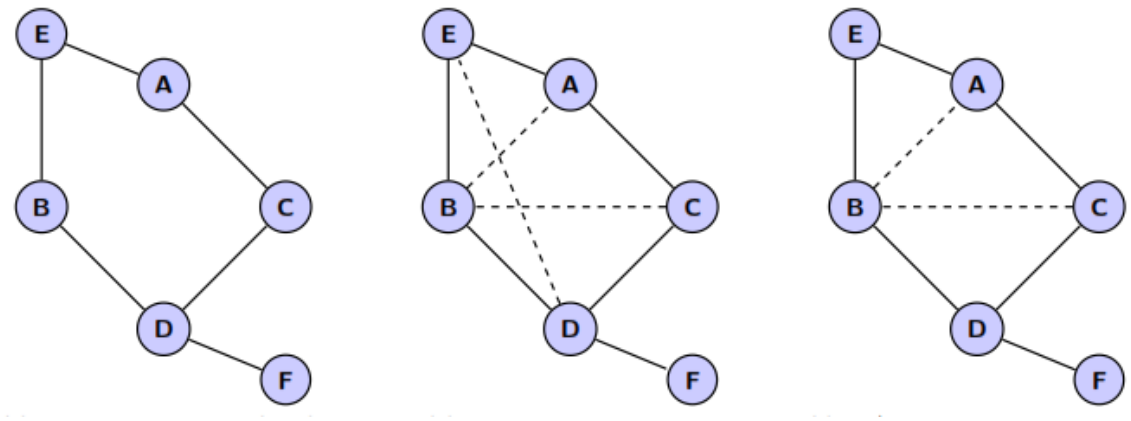
Définition 12 (Graphe cordal)**.** Soit un graphe G = (V,E), on dit que G est cordal si et seulement-si tout k-cycle ($k \geq 4$) de G possède une ≪ corde ≫, c'est-`a-dire une arête reliant deux sommets non consécutifs.

Définition 13 (Graphe triangulé)**.** Un graphe est dit triangulé s'il est cordal.
Une telle triangulation est notée G^*. Elle est obtenue en ajoutant des arêtes a` G afin de casser tous les cycles de taille supérieure à 3. Une triangulation est dite minimum si elle ajoute un minimum d'arêtes au graphe G. Ce problème est NP-difficile **[6]**. On s'intéresse donc a une variante polynomiale de ce problème, qui est la production de triangulations minimales.

Définition 14 (Triangulation minimale)**.** Soit G^+ une triangulation de G, G^+ est minimale si, et seulement si :

$$\nexists G'^{+} \subseteq G^+, \text{ tel que } G'^{+} \text{ est une triangulation de G}$$

La figure 2.2 présente, pour un graphe G (Fig. 2.2a), un exemple de triangulation G^*(Fig.2.2b) et un exemple de triangulation minimale G^+(Fig. 2.2c).



(a) Graphe original G = (V,E). **(b) G^*, triangulation de G.** **(c) G^+, triangulation minimale de G.**

Fig 2.2 – Exemple de triangulation [22].

2.2.2 De la triangulation a la décomposition arborescente

Le lien entre triangulation et décomposition arborescente est donné par la notion **d'arbre de jonction** :

Définition 15 (Arbre de jonction)**.** Soit G = (V, E) un graphe. Un arbre T = (I, F), ou` chaque nœud i ∈ I est associé a un sous-ensemble de sommets $V_i \subset V$, est un arbre de jonction de G si et seulement-si :

- $\{V_i | i \in I\}$ est l'ensemble des cliques maximales de G,
- pour tout sommet $v \in V$, $T_v = \{i | v \in V_i\}$ est un sous-arbre connexe de T.

Le théorème suivant donne une caractérisation des graphes triangulés basée sur les arbres de jonction :

Théorème 1 [7]. Un graphe G est triangulé si, et seulement si, il possède un arbre de jonction.

Ce théorème fournit un moyen simple de trouver une décomposition arborescente dans le cas d'un graphe triangulé. En effet, un arbre de jonction d'un graphe G est une décomposition arborescente de G dont la largeur est égale à la taille de la plus grande clique de G. De plus, cette décomposition est de largeur minimale car la largeur d'arbre d'un graphe est minorée par la taille de sa plus grande clique. Ainsi, nous avons le théorème suivant :

Théorème 2. Trouver la largeur d'arbre d'un graphe G est équivalent à trouver la triangulation de G dont la taille de la clique maximale est minimale.

2.2.3 Décomposition arborescente d'un graphe quelconque

Le théorème 2 donne une indication pour aboutir à une d´décomposition à partir d'un graphe quelconque. D'après le théorème 1, un graphe non triangulé ne possède pas d'arbre de jonction. Cependant, dans **[8]**, les auteurs montrent que pour toute décomposition arborescente d'un graphe G quelconque, il existe une triangulation G^* de G qui admet la même décomposition. Ainsi, calculer la largeur de décomposition d'un graphe G quelconque revient à trouver une triangulation de G ayant une clique maximale de taille minimale. La largeur de décomposition d'une triangulation d'un graphe G est donc un majorant de la largeur de décomposition de G. On peut donc construire une décomposition arborescente d'un graphe non triangulé en produisant une triangulation de ce graphe. Cette décomposition est obtenue en trois phases :

1. triangulation du graphe.
2. construction du graphe de clusters.
3. construction de l'arbre de jonction à partir du graphe de clusters.

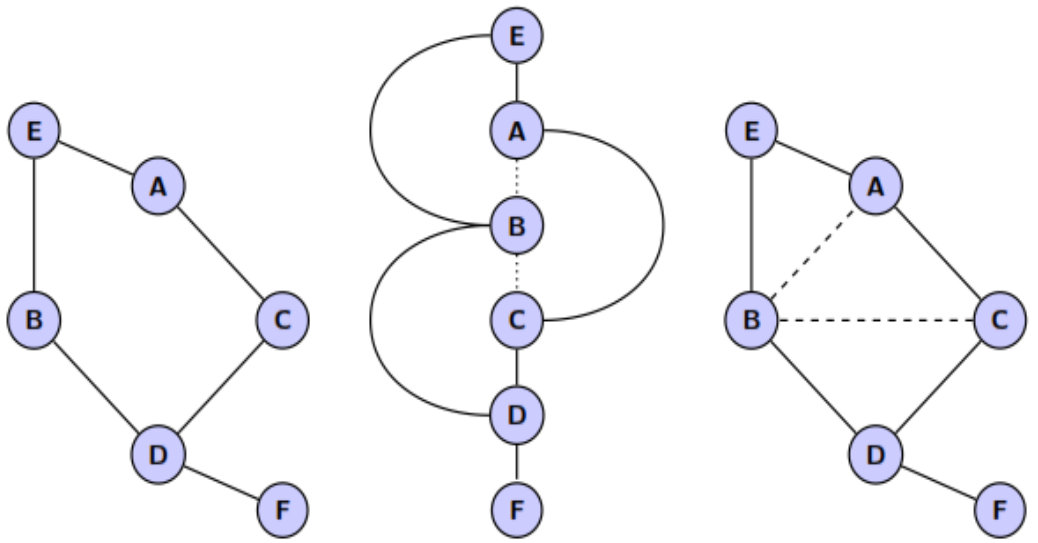
2.2.4 Algorithmes de triangulation

La décomposition arborescente d'un graphe consiste à calculer une triangulation de ce graphe. Un des moyens pour y parvenir est l'utilisation d'un ordre des sommets de G et la création d'un graphe d'élimination.

Définition 16 (Graphe d'élimination)**.** Soit G = (V, E) un graphe et α un ordre des sommets de G. Le graphe d'élimination de G selon α est le graphe produit en retirant successivement chaque sommet v de G dans l'ordre α et en formant une clique avec les voisins restants de v dans G (i.e. les sommets situent après v dans α).

Ce procédé désigne le jeu d´élimination et les arêtes ainsi ajoutées sont appelées arêtes de fill-in. Un ordre d'élimination de G est dit parfait si aucune arête n'est ajoutée lors de l'exécution du jeu d'élimination.

Il est montre dans [Fulkerson et Gross, 1965] [21] que la classe des graphes possédant un ordre parfait est exactement la classe des graphes triangulés.



(a)Graphe original G= (V,E) **(b) Graphe d'élimination selon α=(E ,A,B,C,D,F)** **(c) Graphe triangulé $G^* = (V,E')$**

Fig 2.3 – Exemple de graphe d'élimination [22].

Algorithme 3 : Jeu d'élimination :

Données : un graphe G = (V, E), un Ordre **α**
Résultat : une triangulation G_α^* de G
1 : début
2 : $G^0 = G$;
3 : pour i = 1 à n faire
4 : soit v le sommet tel que $\alpha(v) = i$;
5 : soit F^i l'ensemble des arêtes nécessaires pour faire de $N_{Gi-1}(v)$ une clique dans G^{i-1}
6 : $G^i = G^{i-1}$ (V\ {v}, E **U** F^i) ;
7 : fin pour
8 : $G_\alpha^* = (V, E \cup F^i)$;
9 : fin

Théorème 3 : Un graphe est triangulé si, et seulement si, il a un ordre d'élimination parfait. Pour trianguler un graphe, il suffit donc de calculer son graphe d'élimination. L'algorithme 3 décrit le pseudo-code du jeu d'élimination. Il reçoit en entrée un graphe G et un ordre d'élimination α et produit en sortie un graphe G^*, triangulation de G.

Définition 17 :

Un ordre minimum (resp. Minimal) est un ordre produisant une triangulation minimum (resp. Minimal) par le jeu d'élimination. À chaque itération, l'algorithme sélectionne le sommet **v** d'indice **i** dans l'ordre α (ligne 4).

Une Clique est ensuite formée à partir des sommets voisins de v dans G^{i-1} (linge 5) en ajoutant les arêtes de fill-in (i.e. F^i). Enfin, le sommet v est retiré du graphe G^{i-1} (linge 6) l'algorithme s'arrête lorsque tous les sommets ont été traites et renvoie G^* une triangulation de G (linge7)

La figure 2.3 présente un exemple de graphe d'élimination. Partant d'un graphe G (Fig.2.3a) et d'un ordre α = {E, A, B, C, D, F}, le jeu d'éliminations produit un graphe d'éliminations (Fig. 2.3b) en ajoutant les arêtes {A, B} et {B, C}. On obtient alors le graphe G^* (Fig. 2.3c), triangulation de G.

La qualité de la triangulation obtenue dépend uniquement de l'ordre d'élimination α. Trouver un ordre conduisant à une triangulation minimum est un problème d'optimisation à part entière. Plusieurs approches ont été proposées pour la résolution de ce problème. Elles peuvent se partitionner en 3 classes :

-Les méthodes complètes basées sur l'algorithme de branch-and-bound **[10]**.

-Les méthodes incomplètes de type algorithme génétique **[11]**, recuit simulé **[12]**, etc.

-Les méthodes gloutonnes.

La complexité temporelle des deux premières classes d'algorithmes ne permet pas leur exploitation pratique. La troisième classe vise à donner une solution approchée de bonne qualité en un temps raisonnable. Ces algorithmes construisent l'ordre d'élimination en choisissant, à chaque étape, un sommet du graphe qui minimise la valeur d'une fonction heuristique. On s'intéressera donc à ces approches afin de produire des triangulations en des temps raisonnables.

2.2.4.1 Lex-BFS

Cet algorithme utilise un marquage lexicographique pour produire un ordre d'élimination.

À chaque sommet v du graphe est associée une étiquette e(v) représentant la liste de ses voisins déjà étiquetés, liste ordonnée de manière décroissante vis-à-vis de la position de ses voisins dans l'ordre d'élimination partiel. Son pseudocode est décrit par l'algorithme 4.

Au départ chaque sommet reçoit une étiquette vide. A chaque itération, le sommet possédant la première étiquette dans l'ordre lexicographique reçoit l'indice i dans l'ordre α (i.e. α(v) = i). L'indice i est ensuite ajouté à la fin de l'étiquette de tous les sommets non numérotés voisins de v. Enfin, pour produire une triangulation de G, on applique le jeu d'élimination en utilisant l'ordre d'élimination ainsi obtenu. La complexité de la triangulation est en O (n + m), avec n le nombre de sommets et m le nombre d'arêtes.

Algorithme 4 : Lex-BFS

Données : un graphe G = (V, E)
Résultat : un ordre α et G_α^*
1 : début
2 : pour tout sommet v de V, initialiser son étiquette e(v)=ø
3 : pour i = n à 1 faire
4 : Choisir le sommet v d'étiquette e(v) maximale dans l'ordre lexicographique :
5 : pour tout sommet non numéroté u de V faire
6 : si {u, v} ∈ E alors
7 : e(u) = e(u) U {i} ;
8 : finsi
9 : α (v) = i
10 : finpour
11 : jeu d'élimination (G, α)
12 : fin

2.2.4.2 MCS

MCS (Maximum Cardinality Search) **[13]** s'affranchit du marquage lexicographique en associant à chaque sommet un poids égal au nombre de ses voisins déjà visités. Ceci permet de réduire la complexité du tri des nœuds.

L'algorithme 5 d'écrit le pseudocode de **MCS** à chaque itération, le sommet (**noté v**) qui a le plus grand nombre de voisins déjà choisis et sélectionné. Ensuite, le poids de chaque sommet non numérote voisin de **v** est incrémente de 1. L'algorithme s'arrête quand tous les sommets de **G** ont été numérotés la complexité de la triangulation est en **O (n+m)**, avec n le nombre de sommets et m le nombre d'arrêtes.

Algorithme 5 : MCS

Données : un graphe G = (V, E)

Résultat : un ordre minimal α et G_{α}^{*}

1 : début

2 : F = ø ;

3 : Pour tout sommet v de V, initialiser son poids w(v) = 0 ;

4 : **pour** i = n à 1 **faire**

5 : Choisir le sommet v de poids w(v) = 0 ;

6 : **pour tout** sommet non numéroté u de V **faire**

7 : **si** {u, v} ∈ E **alors**

8 : incrémenté w(u) ;

9 : **finsi**

10 : **finpour**

11 : α(v) = i

12 : **finpour**

13 : jeu.élimination (G, α)

14 : Fin

2.2.4.3 LEX-M

LEX-M [14] est une extension de Lex-BFS qui vise à produire des triangulations minimales. Pour cela, il s'appuie sur la notion de fill-path.

Définition 18 (fill-path). On appelle fill-path un chemin u, x_1, x_2, . . ., x_k, v constitué uniquement de sommets non numérotés tel que $e(x_i) < e(u)$. L'arête {u, v} est dite de fill-in.

Comme Lex-BFS, à chaque itération, LEX-M (cf. Algorithme 6) sélectionne le sommet v ayant l'étiquette lexicographique la plus élevée (ligne 5). Toutefois, contrairement à Lex-BFS, Lex-M ajoute l'indice i dans l'ordre α (i.e. α(v) = i) aux étiquettes de tous les sommets non numérotés voisins de v ou reliés à v par un fill-path (l'ensemble S, ligne 9). Ensuite chaque arête de fill-in {u, v} est ajoutée à l'ensemble F. La complexité de la triangulation est en O(n × m), avec n le nombre de sommets et m le nombre d'arêtes.

Algorithme 6 : LEX-M

Données : un graphe G = (V, E)
Résultat : un ordre minimal α et G_α^+
1 : **début**
2 : F = ø ;
3 : pour chaque sommet v de V, initialiser son étiquette e(v) = ø ;
4 : **pour** i = n à 1 **faire**
5 : choisir le sommet v d'étiquette (e(v)) maximale dans l'ordre lexicographique ;
6 : S = ø
7 : **pour tout** sommet non numéroté u de V **faire**
8 : **si** {u, v} ∈ E ou ∃ un chemin u, x_1,…, x_k, v tel que $e(x_i) < e(u)$ pour tout $1 \leq i \leq k$
9 : **alors**
10 : S = S ∪ {u} ;
11 : **finsi**
12 : **pour tout** sommet u ∈ S **faire**
13 : e(u) = e(u) ∪ {i} ;
14 : **si** {u, v} ∉ E **alors**
15 : F = F ∪ {{u, v}} ;
16 : **finsi**
17 : **finpour**
18 : α(v) = i
19 : **finpour**
20 : G_α^+ = (V, E∪F)
21 : fin

2.2.4.4 MCS-M

Partant de LEX-M et MCS, le but de l'algorithme MCS-M **[15]** (algorithme 7) est de simplifier la production de triangulation en utilisant le marquage numérique des sommets. Il reprend le principe de LEX-M en remplaçant le tri lexicographique par une sélection par poids.

Soit **F** l'ensemble des arrêtes de fill-in. A chaque itération, l'algorithme sélection le sommet v de poids maximal (ligne5), puis construit l'ensemble S constitué des voisins de v et sommets reliés à v par un fill-path (ligne 9). Le poids de chaque sommets de S est incrémenté de 1 et l'arrête **{u, v}** est ajoutée à **F** (ligne 11).

A la fin du marquage des sommets, les arrêtes de **F** (de fill-in) sont ajoutées à **G** (ligne-15). La complexité de la triangulation est en O (n x m), avec n le nombre de sommet et m le nombre d'arrêtes.

Algorithme 7 : MCS-M

Données : un graphe G = (V, E)
Résultat : un ordre minimal α et G_α^+

1 : **début**
2 : F = ø ;
3 : pour chaque sommet v de V, initialiser son poids w(v) = 0 ;
4 : **pour** i = n à 1 **faire**
5 : choisir le sommet v de poids (w(v)) maximale ;
6 : S = ø
7 : **pour tout** sommet non numéroté u de V **faire**
8 : **si** {u, v} ∈ E ou ∃ un chemin u, $x_1,\ldots, x_k$, v tel que $w(x_i) < w(u)$ pour tout $1 \leq i \leq k$
9 : **alors**
10 : S = S ∪ {u} ;
11 : **finsi**
12 : **pour tout** sommet u ∈ S **faire**
13 : incrémenté w(u) ;
14 : **si** {u, v} ∉ E **alors**
15 : F = F ∪ {{u, v}} ;
16 : **finsi**
17 : **finpour**
18: α (v) = i
19: **finpour**
20: G_α^+ = (V, E∪F)
21 : fin

2.2.4.5 Min-Fill

Avec cette heuristique, les nœuds qui ajoutent le moins d'arêtes de Fill-in sont choisis en premier. Les arêtes de Fill-in sont cette fois prises en compte dans le calcul du voisinage des Nœuds.

Le calcul de l'ordre nécessite une complexité de O (n^2d^2). Cette heuristique donne
En général de bien meilleurs résultats mais au prix d'un temps de calcul plus élevé.

Min-Fill procède en numérotant et donc en ordonnant les sommets de G de 1 a n tout en rajoutant les arêtes nécessaires de sorte que cet ordre soit un ordre d'élimination parfait pour le graphe résultant G' qui sera triangule. Dans un tel ordre, les voisins ultérieurs de chaque sommet (c'est-à-dire les voisins apparaissant après ce sommet dans l'ordre) forment une clique. La triangulation va être réalisée sur un graphe G' initialise a G. A chaque étape, Min-Fill choisit Parmi les sommets non numérotes, le sommet qui a besoin du minimum d'arêtes à rajouter pour compléter le sous-graphe induit par ses voisins non encore ordonnes. Le processus continu jusqu'à ce que tous les sommets de G soient numérote **[16]**.

Algorithme 8 : Min-Fill

Entrées : Un graphe G = (X ; C)
Sorties : Un graphe G_0 triangulation de G et un peo = (un ordre d'élimination parfait)
1- Calcul du nombre d'arêtes nécessaires pour la complétion du voisinage de chaque sommet
2- tant que ∃ un sommet non numérote faire /* choix du prochain sommet dans l'ordre */
3- Choix d'un sommet **v** non numérote nécessitant un minimum d'arêtes pour compléter son voisinage ultérieur
4- Numéroter **v**
5- Compléter le sous-graphe induit par les voisins de **v** non numérotes
6- Mettre à jour le nombre d'ajouts d'arêtes nécessaires pour chaque sommet non numéroté.

La complexité en temps de MIN-Fill est O (n (n + e')) ou e' le nombre d'arêtes du graphe triangulé G'.

Remarque. Aucun algorithme ne précise comment choisir le premier sommet de l'ordre. Cette question reste ouverte et, à notre connaissance, aucun critère n'a été proposé pour assurer une décomposition de largeur minimale.

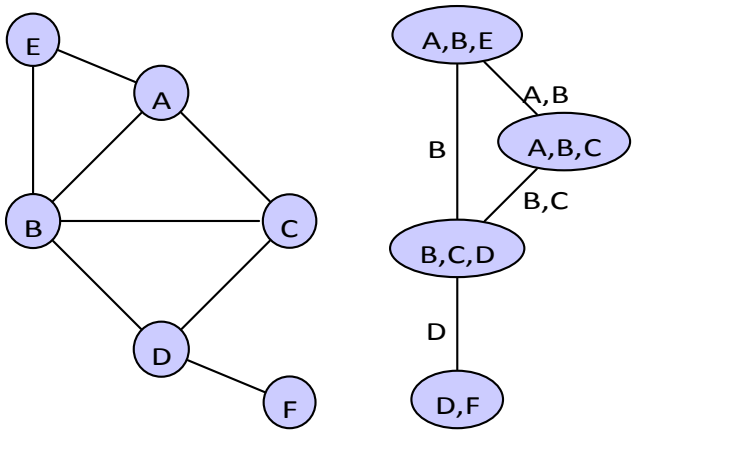
2.2.5 Recherche des cliques maximales.

La seconde étape de l'algorithme de recherche d'une décomposition arborescente consiste à construire le graphe de jonction (le graphe de clusters) :

Définition 19 : (Graphe de jonction). Soit G^* une triangulation de G et C l'ensemble de ses cliques maximales. Un graphe de jonction de G^* est un graphe étiqueté, noté $G_c = (C, E)$, dont les sommets sont les éléments de C et il existe une arête entre deux sommets C_i et C_j si et seulement si $sep(C_i, C_j) = \emptyset$. Les arêtes sont étiquetées par les sommets en commun.

La figure **2.5b** donne le graphe de jonction associé au graphe triangulé de la figure **2.5a**. Pour identifier l'ensemble des cliques maximales, il suffit d'exploiter l'ordre calcule par l'heuristique de triangulation et le graphe triangulé G^*.

En effet, pour un graphe triangulé et son ordre associé, tout sommet forme une clique avec tous ses voisins le suivant dans l'ordre. Il ne reste plus ensuite qu'à faire un test de maximalité.



(a) Graphe triangulé G^* **(b) Graphe de jonction associé G^***

Fig 2.4 – Exemple de graphe de jonction[22].

2.2.6 Calcul de l'arbre de jonction

L'obtention de l'arbre de jonction à partir du graphe de jonction se fait grâce à la propriété suivant :

Théorème 4 : Un arbre couvrant de poids maximal d'un graphe de jonction est un arbre de jonction. Figure 2.6 présente un exemple d'arbre de jonction associé au graphe de clusters de la Figure 2.5b.

Le principe de l'algorithme est de choisir, au fur et à mesure de la construction, l'arête contenant le plus grand nombre de sommets (cf. algorithme 9). On sélectionne tout d'abord un sommet u au hasard (ligne 2). Tant qu'il reste des sommets qui n'appartiennent pas à **I**, on sélectionne l'arête {v, w} de poids maximal dont un seul sommet v appartient à **I**.On ajoute l'arête {v, w} à F (l'ensemble des arêtes de l'arbre de jonction) et w à **I**.A la fin de l'algorithme, on obtient $T = (\mathbf{I}, F)$, l'arbre de jonction de Gc. Cet algorithme est linéaire en le nombre de sommets du graphe Gc.

Algorithme 9 : Algorithme de calcul d'arbre couvrant de poids maximal

Données : un graphe étiqueté $G_C = (C, E)$

Résultat : Un arbre $T = (\mathbf{I}, F)$ couvrant de poids maximal

1 : début
2 : soit I = {u} ou u est un sommet choisi au hasard ;
3 : soit F = ø ;
4 : **tant que** il reste des sommets non sélectionnés dans Gc **faire**
5 : choisir une arête {v, w} telle que $\mathbf{v} \in \mathbf{I}$ et $w \notin \mathbf{I}$ de poids maximal ;
6 : ajouter {v, w} à F ;
7 : ajouter w à I ;

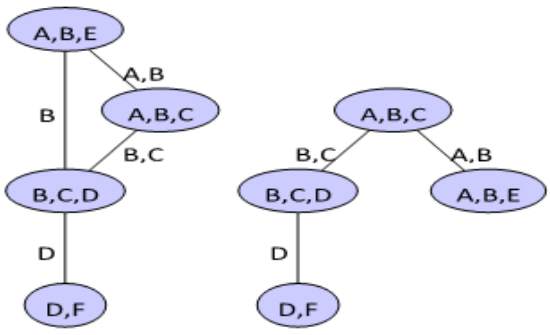


Fig 2.5 – Exemple de calcul d'arbre couvrant de poids maximum[22].

Ainsi on obtient une décomposition arborescente du graphe de contraintes, puis on passe à la résolution de notre CSP.

2.2.7 Méthode de décomposition arborescente (Tree décomposition)

Définition 20 (Arbre (Tree)) **[17]**

Soit G = (V, E) un hypergraphe. Un arbre (Tree) de l'hypergraphe G est une paire <T, χ>, ou T = (V (T), E(T)) est un arbre enraciné, et χ est une fonction qui associe à chaque noeud p ∈ V (T) l'ensemble de variables $\chi(p) \subseteq V$.

Définition 21 (Tree décomposition) **[18]**

Une tree décomposition d'un hypergraphe G = (V, E) est un cuple couple TD =< T,χ > de G qui respecte les conditions suivante :

1. ∀h ∈ E, il existe t ∈ T tel que h ⊆ χ(t).
2. ∀x ∈ V, l'ensemble {t ∈ T/x ∈ χ(t)} induit un sous arbre connecté de T.

La première condition assure que chacune des contraintes du CSP original doit apparaitre dans, au minimum, un sous problème du nouveau CSP acyclique, et la deuxième condition, garantit que toute variable doit avoir la même valeur affectée dans chaque sous problème dans lequel elle apparait (propriété de connectivité).

Le concept de Tree décomposition a été présenté par Robertson et Seymour **[19]** et n'a été, initialement, défini que pour les graphes.

Etant donné un CSP, son graphe de contrainte, et sa Tree décomposition de largeur k, une solution de ce CSP peut être calculée en un temps $O(nd^{k+1})$, avec n est le nombre de variables du CSP et d le nombre maximum de valeurs par domaine **[18]**.

La Tree décomposition est calculée par différentes méthodes la plus importante est la méthode Tree clustering.

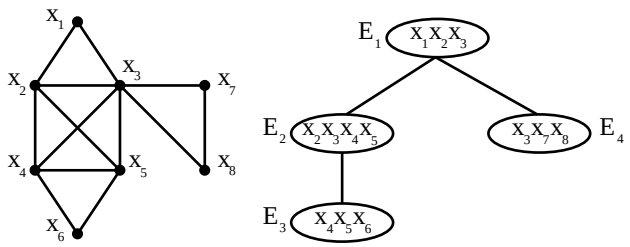


Fig. 2.6 – Un graphe (a) et sa Tree décomposition (b) [16]

2.2.8 Méthode Tree-Clustering

La tree clustering s'appuie sur l'algorithme de triangulation qui transforme un graphe quelconque en un graphe triangulé. Les cliques maximales résultant de cet algorithme formeront les contraintes du CSP acyclique équivalent. L'algorithme de triangulation consiste en deux étapes :

1. Ordonner les sommets du graphe primal en utilisant l'heuristique MCS (Maximum Cardinality Search) [20].
2. ajouter récursivement des arêtes entre chaque couple de nœuds non adjacents qui sont connectés par des nœuds supérieurs dans l'ordre calculé précédemment.

L'heuristique MCS numérote les sommets du graphe primal de 1 à n (n : nombre de variable de CSP) de telle sorte que le prochain numéro à attribuer est assigné au sommet ayant le plus de voisins déjà numérotés.

Le joint tree généré lors de la troisième étape est un arbre dont les sommets sont les mêmes que ceux du graphe dual et son ensemble d'arêtes est un sous ensemble d'arêtes du graphe dual. Les arêtes sont choisies de telle façon que pour chaque variable du CSP, l'ensemble des sommets constitués de contraintes contenant cette variable soit connectée

La TC début par le calcul d'une décomposition arborescente qui utilise l'algorithme MCS (Maximum Cardinality Search). Dans la seconde étape, les clusters sont résolus indépendamment, en considérant chaque cluster comme un sous-problème et donc, en énumérant toutes ses solutions. Après cela, une solution globale au CSP, s'il en existe une, peut alors être efficacement calculée en exploitant la structure arborescente de la décomposition.

Exemple : Soit le CSP P = (X, D, C, R) tel que X = {A, B, C, D, E} et C = {{A,D},{A,C},{C,E},{D,E},{B,D}}.

Le graphe de contraintes de P ainsi que son graphe triangulé, le graphe dual du CSP acyclique généré et son joint tree sont illustrés dans la Fig2.7 suivante

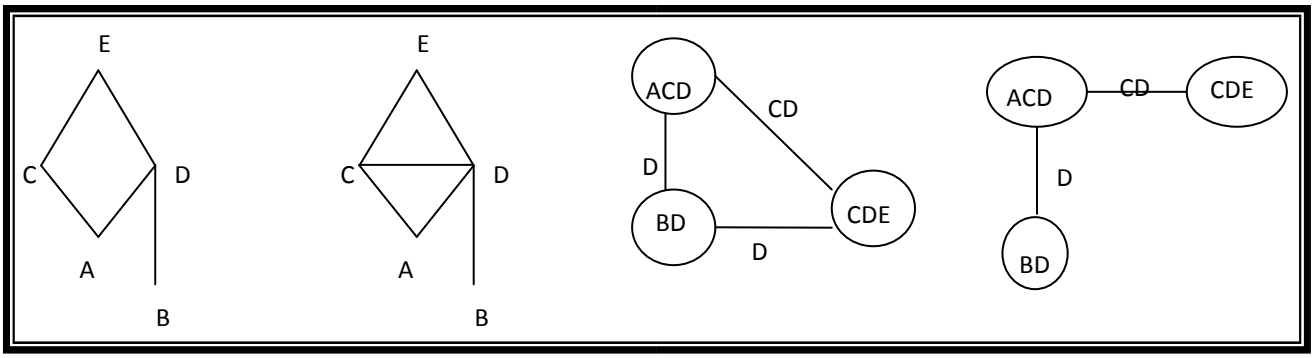


Fig - 2.7 Etape de décomposition de P. [1]

La largeur de décomposition de la tree clustering est égale au nombre maximal de variables des cliques maximales.

2.3 Méthodes de décomposition sans la triangulation

2.3.1 Algorithme Least-T D

Pour un graphe G = (X, C), l'algorithme Least-TD calcule une décomposition arborescente en temps polynomial, bien sûr sans aucune garantie quant à son optimalité, mais sans recours à la triangulation et en prenant en compte la topologie du graphe **[16]**.

La première étape de l'algorithme calcule un premier cluster, noté $E_{0.\ X'}$ qui notera par la suite l'ensemble des sommets déjà traités est donc initialisé à E_0. Cette première étape peut se faire facilement, en utilisant une méthode heuristique.

Notons $X_1, X_2, \ldots X_k$ les composantes connexes du sous-graphe $G[X \setminus E_0]$ induit par la suppression dans G des sommets de E_0.

Chacun de ces ensembles X_i est inséré dans une file F.

pour chaque élément X_i supprimé de F, on notera V_i l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i.

On peut noter que V_i est un séparateur du graphe G puisque la suppression de V_i dans G rend G non connexe (X_i étant déconnecte du reste de G).

Nous considérons alors le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$.

L'étape suivante va consister à déterminer un sommet v de V_i qui possède un minimum de voisins dans Xi.

Soit $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$ cet ensemble. On construit alors un nouveau cluster $E_i = N(v, X_i) \cup V_i$.

La Fig2.8 présente le calcul de E_1, le second cluster (après E_0), lors du premier passage dans la boucle.

Parmi les sommets de V_1, celui qui possède un minimum de voisins dans X_1 est le sommet v puisque l'on a $N(v, X_1) = \{a\}$ et que pour les autres sommets, cet ensemble est $N(w, X_1) = N(x, X_1) = \{b, c\}$.

Le cluster E1 aura ainsi pour sommets $V1 \cup \{a\}$.

À partir de là, on va obtenir deux nouvelles composantes connexes : $X_{11} = \{d, e, h\}$ et $X_{12} = \{b, c, f, g, i, j, k\}$qui vont alors être ajoutées à la file F. Quand X_{11} en sera retiré, on aura $V_i = \{a\}$, un nouveau cluster sera alors construit avec $V_i \cup \{d, e\} = \{a, d, e\}$ et {h}sera ajouté à F. Quand X12 sera retiré de la file, on considérera l'ensemble Vi = {a, w, x} pour lequel le Sommet choisi sera a qui ne possède qu'un voisin dans X_{12} pour former un nouveau cluster {a, f, w, x}. Deux nouveaux ensembles seront alors insérés dans F : {i} et {b, c, g, j, k}. On remarque

qu'ainsi, l'algorithme exploite explicitement la topologie du graphe par le biais de séparateurs et de composantes connexes.

La file F va être exploitée jusqu'à la suppression de l'ensemble des sommets du graphe. La complexité en temps de l'algorithme Least-TD est O (n(n + e)).

Algorithme10 : Least-TD

Entrées : Un graphe G = (X, C)

Sorties : Un ensemble de clusters $E_0 \ldots\ldots E_m$ d'une décomposition arborescente de G

1 : Choix d'un premier cluster E_0 dans G

2 : X' ← E_0

3 : Soient $X_1 \ldots\ldots X_k$ les composantes connexes de $G[X \setminus E_0]$

4 : F ← $\{X_1 \ldots\ldots X_k\}$

5 : tant que F = ∅ faire /* calcul d'un nouveau cluster Ei */

6 : Enlever Xi de F

7 : Soit Vi ⊆ X' le voisinage de Xi dans G

8 : Déterminer un sommet v ∈ Vi qui possède un minimum de voisins N (v, Xi) dans Xi

9 : Ei ← N (v, Xi) ∪ Vi

10 : X' ← X' ∪ N (v, Xi)

11 : Soient $X_{i1}, X_{i2}, \ldots X_{iki}$ les composantes connexes de G [Xi \Ei]

12 : F ← F ∪ $\{X_{i1}, X_{i2}, \ldots X_{iki}\}$

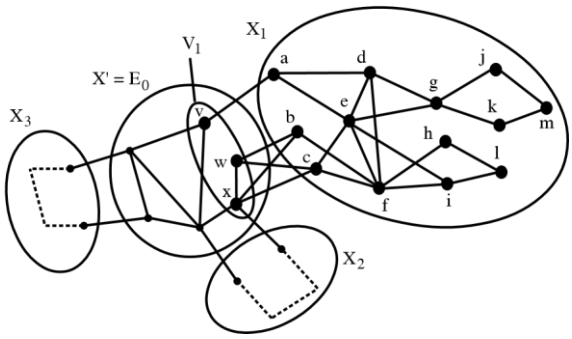


Fig. 2.8 - Illustration de Least TD [16]

2.3.2 Heuristique *H-TD-WT*

H-TDWT (*Heuristic Tree-Decomposition Without Triangulation*) calcule une décomposition arborescente du graphe $G = (X,C)$ sans triangulation en temps polynomial (à savoir en $O(n(n + e))$).

La première étape de *H-TD-WT* (ligne 1 dans l'algorithme 1) calcule un premier cluster, note E_0, à l'aide d'une heuristique. X^0 qui représente l'ensemble des sommets déjà considères est initialisé a E_0 (ligne 2).

On notera par $X_1,X_2,...X_k$ les composantes connexes du sous-graphe $G[X\backslash E_0]$ induit par la suppression dans G des sommets de E_0[1] Chacun de ces ensembles X_i est insèré dans une file d'attente F (ligne 4).

Pour chaque élément X_i retiré de F (ligne 6), V_i note l'ensemble des sommets de $X^{'}$ qui sont adjacents à au moins un sommet de X_i (ligne 7). Dans l'exemple de la figure 1, pour $X_i = X_1$, on a $V_i = V_1 = \{x,y,z\}$.

On peut constater que V_i est un séparateur dans le graphe G puisque la suppression de V_i dans G déconnecte G (X_i étant déconnecte du reste de G).

Nous considérons alors le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$. L'étape suivante (ligne 8) peut être paramétrée. Elle recherche un sous-ensemble de sommets $X_i^{''} \subseteq X_i$ tel que $X_i^{''} \cup V_i$ sera un nouveau cluster E_i de la décomposition Cela peut être garanti s'il existe au moins un sommet v de V_i tel que tous ses voisins dans X_i figurent dans X_i''.

Plus précisément, si $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$, nous devons garantir l'existence d'un sommet v de V_i avec $N(v,X_i) \subseteq X_i^{00}$.

Nous définissons alors un nouveau cluster $E_i = X_i^{00} \cup V_i$ (ligne 9). Puis, nous rajoutons à X^0 les sommets de X_i'' (ligne 10), avant de calculer les composantes connexes du sous-graphe issu de la suppression des sommets de E_i dans $G[X_i]$, composantes qui sont alors rajoutées a la file F (ligne 11). Ce processus est répété jusqu'à ce que la file F soit vide.

Algorithme11 : H-TD-WT

Entrées : Un graphe G = (X, C)

Sorties : Un ensemble de Clusters E_0……E_m d'une décomposition arborescente de G.

1 : Choix d'un premier cluster E_0 dans G.

2 : $X' \leftarrow E_0$.

3 : Soient X_1……X_k les composantes connexes de $G[X \setminus E_0]$

4 : $F \leftarrow \{X_1$……$X_k\}$

5 : tant que $F = \emptyset$ faire /* calcul d'un nouveau cluster Ei */

6 : Enlever Xi de F

7 : Soit $Vi \subseteq X'$ le voisinage de Xi dans G

8 : Déterminer un sous-ensemble $X_i'' \subseteq X_i$ tel qu'il existe au moins un sommet $v \in V_i$ Tel que $N(v, X_i) \subseteq X_i''$

9 : $E_i \leftarrow X_i'' \cup V_i$

10 : $X' \leftarrow X' \cup X_i''$

11 : Soient $X_{i1}, X_{i2}, \ldots X_{iki}$ les composantes connexes de G [Xi \Ei]

12 : $F \leftarrow F \cup \{X_{i1}, X_{i2}, \ldots X_{iki}\}$

2.4 Conclusions

Dans ce chapitre, nous avons présentent les notions relatives à la décomposition arborescente et les différentes méthodes permettant de la calculer. Nous avons également calculé la complexité théorique des heuristiques de triangulation et celles sans triangulation. La première étape qui est difficile en procédant par des méthodes de décomposition arborescente sans triangulation est le choix du premier cluster qui est crucial ça nécessite des heuristiques.

La seconde est de limite la taille maximale des clusters, cela augmente le temps nécessaire pour le calcul de décomposition et leur complexité.

Il ressort que les méthodes de décomposition arborescente avec triangulation offrent le meilleur compromis entre la qualité de décomposition et le temps nécessaire pour son calcul. C'est donc parmi ces heuristiques qu'on va choisir la meilleure et l'implémenter dans le chapitre 3.

CHAPITRE -III- Résolution et Implémentation

Dans ce chapitre nous allons nous concentrer sur les méthodes de décomposition arborescente avec triangulation, on va exposer des études comparatives et choisir la meilleure pour l'implémenter.

L'heuristique Min-Fill s'avère être la plus coûteuse parmi les heuristiques de l'état de l'art, son temps est proche de l'optimum. Cependant, Min-fill souffre de plusieurs défauts. D'une part, elle procède d'une certaine façon à l'aveugle en se limitant à des décomptes d'arêtes sans se préoccuper des propriétés topologiques du graphe traité, du moins, elle est bien meilleure que les méthodes exactes, Min-fill calcule des décompositions assez proches de l'optimum. D'autre part, l'ajout d'arêtes occasionné par le principe de triangulation génère un coût temporel important.

La figure ci-dessous résume la relation entre les différentes méthodes et distingue les méthodes qui donnent une triangulation minimale.

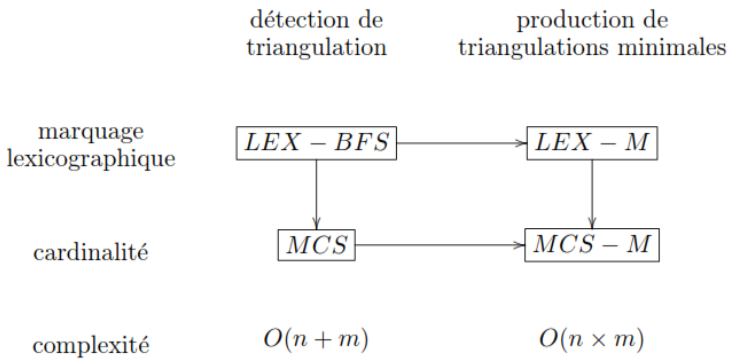


Fig 3.1 Relation entre les différents algorithmes de triangulation.[22]

Lex-BFS et MCS sont des méthodes qui donnent une triangulation mais pas minimale, alors ce qui nous intéresse c'est Lex-M et MCS-M.

Un des premiers algorithmes pour résoudre le problème de triangulation minimale était Lex- M, qui a été présenté en 1976.Un nouvel algorithme, et une simplification de Lex- M est appelé MCS-M, a été présenté en 2002. La comparaison de ces deux algorithmes prouve qu'ils produisent la même triangulation en termes de nombre d'arcs ajoutés.

MCS-M C'est une simplification de Lex- M de sorte que des poids de cardinalité soient employés au lieu des étiquettes lexicographiques.

D'après l'étude faite par Matieu Fontain**[22]** Lex-M et **MCS-M** obtiennent des décompositions de largeur minimale.

Malgré l'apport en termes de performance de MSC-M par rapport à LEX-M, la qualité des solutions obtenues par les deux méthodes est strictement identique.

La méthode choisie pour l'implémenter est LEX-M puisque elle est la méthode de référence (présenté en 1976).

CHAPITRE -III- Résolution et Implémentation

Dans ce chapitre on réécrit l'algorithme LEX-M et on décrit la structure de données adoptée ainsi un exemple de déroulement de l'algorithme LEX-M.

Algorithme : LEX-M

Données : un graphe G = (V, E)
Résultat : un ordre minimal α et G_α^+

1 : **début**
2 : F = ø ;
3 : pour chaque sommet v de V, initialiser son étiquette e(v) = 0 ;
4 : **pour** i = n à 1 **faire**
5 : choisir le sommet v d'étiquette (e(v)) maximale dans l'ordre lexicographique ;
6 : S = ø
7 : **pour tout** sommet non numéroté u de V **faire**
8 : **si** {u, v} ∈ E ou ∃ un chemin u, x_1,…, x_k, v tel que $e(x_i) < e(u)$ pour tout $1 \leq i \leq k$
9 : **alors**
10 : S = S ∪ {u} ;
11 : **finsi**
12 : **pour tout** sommet u ∈ S **faire**
13 : e(u) = e(u) ∪ {i} ;
14 : **si** {u, v} ∉ E **alors**
15 : F = F ∪ {{u, v}} ;
16 : **finsi**
17 : **finpour**
18 : α(v) = i
19 : **finpour**
20 : G_α^+ = (V, E∪F)
21 : fin

3. Structure de Données

3.1 Déclarations des Types

TARC : On définit nos arcs de type **TARC**

Un arc est une liaison entre deux sommets, les sommets définissant l'arc sont dits origine et destination.

TSOMMET : un sommet se caractérise par les éléments suivants :

- Nom du sommet (le sommet lui-même, à définir par l'utilisateur)
- OLG : ordre lexicographique du sommet (créer par ordre de création d'arc).
- Alpha : ordre α du sommet.
- Etiquette : l'étiquette initialiser à zéro, peuvent changer de valeur au niveau de chaque itération.
- Selected : au moment de la déclaration du graphe les sommets sont non-numéroter, Selected reçoit faulse (Selected est un indicateur d'état).

TCHEMIN : Définit tout sommet ajouté au niveau d'une itération quelconque.

3.2 Déclaration des Variables

-graphe : le graphe est un ensemble d'arcs et de sommets, à l' origine on ne connait ni le nombre d'arêtes ni le nombre de sommets .C'est pour cela nous avons opté pour une structure de tableau dynamique .Autrement-dit le nombre d'arêtes peut varier d'un graphe à un autre, ainsi que le nombre de sommets.

-Arcs : est un tableau de taille variable de type Tarc. On peut saisir autant d'arcs, la seule limite à ce type de la structure est la mémoire vive de notre calculateur. De nos jours ce type de contrainte est dépassé.

-Sommets : est un tableau de type Tsommet définit précédemment.

-ListeSommetsAjouterAS : un Tableau qui contient la liste de tous les sommets ajouté à l'ensemble **S.**

-Chemin : un tableau qui contient les positions des sommets.

-ListeSommetsAjouter : le nombre de sommets ajoutés au niveau d'une itération.

-NombreArc : le nombre d'arcs que contient le graphe.

3.3 Fonctions et procédures

-Fonction SelectoioneSommet : Permit de sectionner le sommet non-numéroterd'étiquette maximale.

-Fonction SommetContenuDansChemin : Permitde déterminer si un sommet fait partis des sommets composant un tel chemin (elle évite de tourner en rond).

-Fonction DepartAdjacentArrivee : Elle détermine si deux sommets sont adjacents ou non.

-Fonction CheminExiste : Elle détermine s'il existe un chemin entre deux sommets (qui satisfaits les conditions d'existence du chemin selon Lex-M).

-Procédure DetectArcsEtSommets : Elle convertie les écritures introduites manuellement dans la grille de saisie en arcs et sommets. Elle permet aussi d'affecter aux sommets créés un ordre lexicographique suivant l'ordre de leur première apparition dans le graphe.

-Procédure UniqueOLG : du fait que notre application permet à l'utilisateur d'introduire son propre ordre lexicographique, nous nous permettons de vérifier ce dernier.

-Procédure LEX_M : elle déroule l'algorithme Lex-M.

4. Implémentation

Afin de tester la performance de l'algorithme LEX-M on doit le dérouler sur plusieurs graphes et de tailles assez considérables. Pour ce faire, nous avons choisi de réaliser notre application avec le langage orienté objet **DELPHI**.

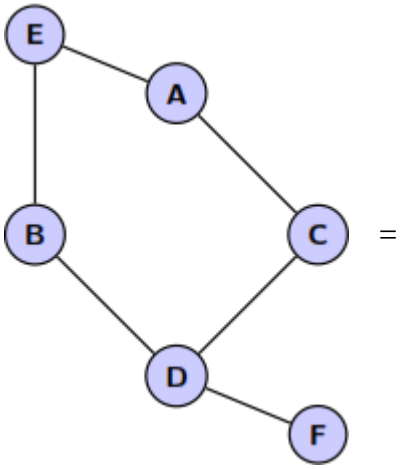
Notre application est composée de trois écrans à savoir :

CHAPITRE -III- Résolution et Implémentation

A. Écran de saisie du graphe

Dans cet écran, le graphe est introduit dans une grille de saisie sous forme d'arcs.

Exemple



=

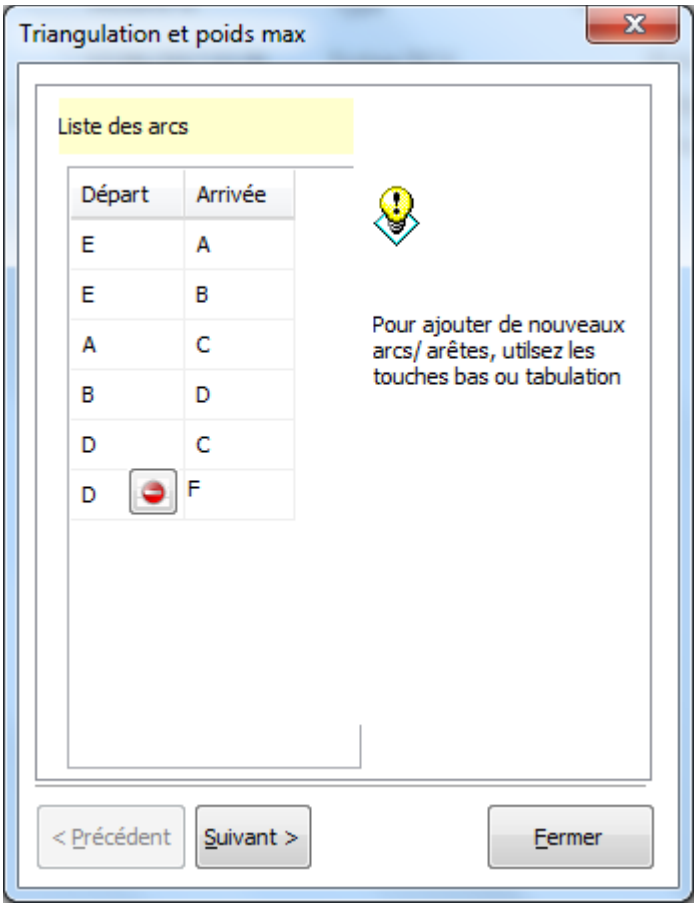


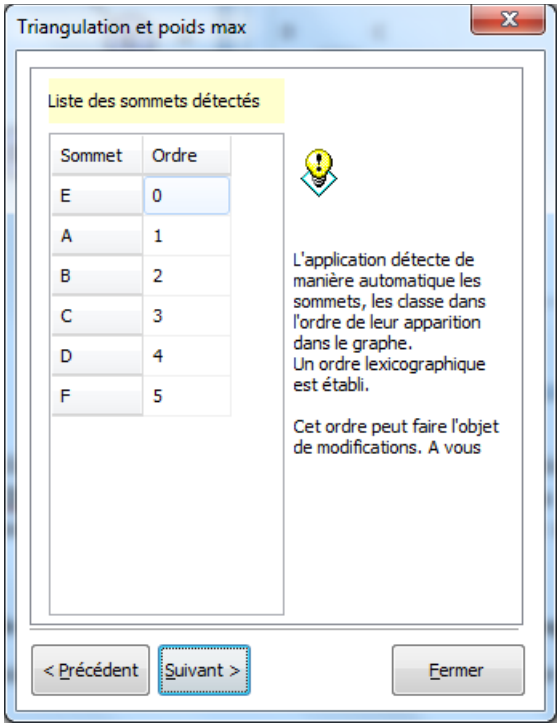
Fig. E1.

En cliquant sur la commande suivant, notre application exécute la procédure « **DetecteSommetsEtArcs** » et affiche le 2e écran contenant la liste des sommets détectés et l'ordre lexicographique qui leur a été attribué.

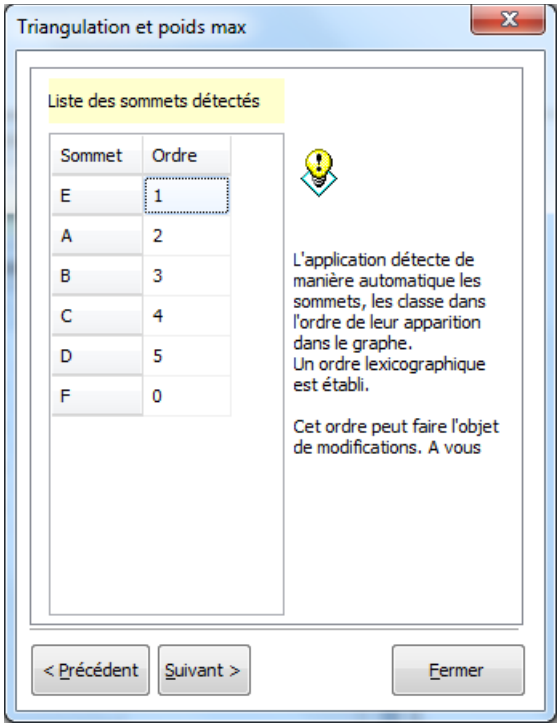
B. Ecran de modification et de validation de l'ordre lexicographique.

Cet écran affiche la liste des sommets détectés et l'ordre lexicographique qui leur est attribué.

Exemple



Ordre lexicographique obtenu par l'application (*Fig.* E2.1)



Ordre lexicographique après modification (*Fig.* E2.2)

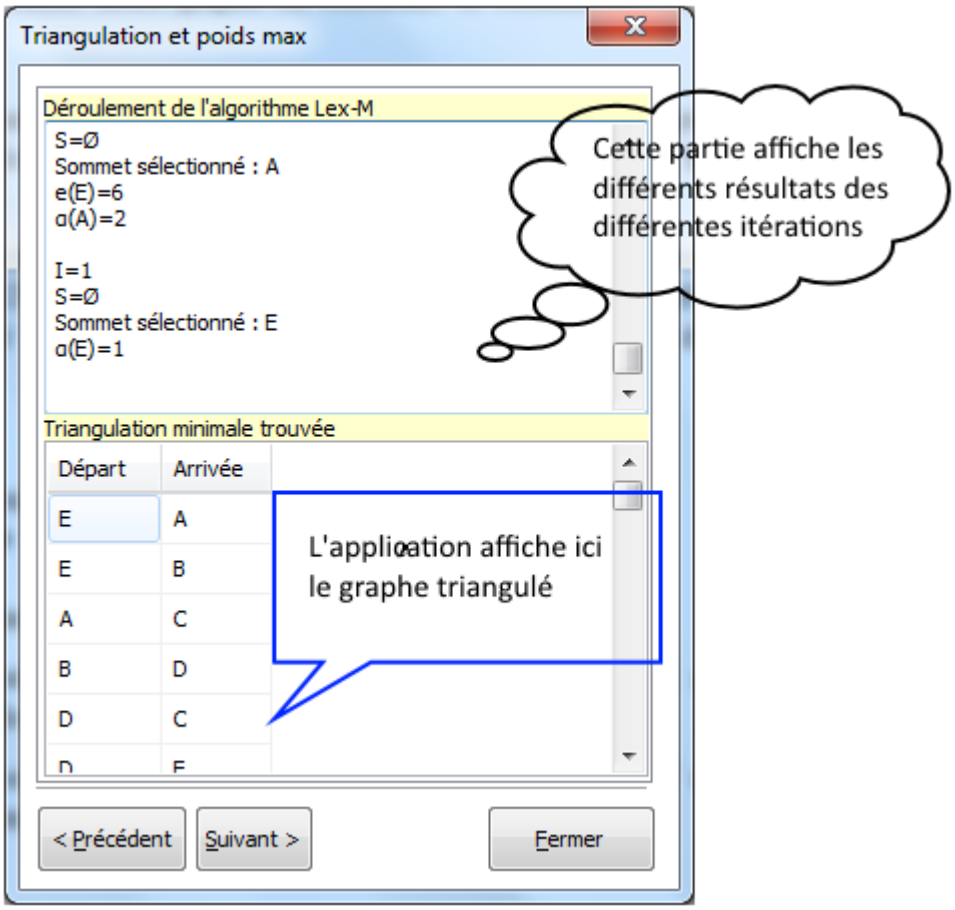
Nous avons le choix entre modifier ou ne pas modifier l'ordre lexicographique. Dans le présent exemple, nous avons choisi modifier l'ordre lexicographique (*Fig.* E2.2). Toutefois, nous pouvons revenir à l'écran précédent et modifier notre graphe en exécutant la commande "Précédent".

En exécutant la commande "Suivant", l'application procède à la vérification de l'ordre lexicographique en appelant la procédure "**UniqueOLG**".

Après validation de l'ordre lexicographique, cette dernière déroule l'algorithme LEX-M en exécutant la procédure "LEX_M" et affiche dans un 3^{e} écran appelé "Ecran Résultats" les résultats du déroulement de l'algorithme LEX-M.

C. **Ecran résultats :** (Décrit dans le paragraphe précédent).

Exemple



Les graphes triangulés ainsi obtenus peuvent varier selon l'ordre lexicographique choisi.

En choisissant l'ordre de la fig. E2.1, nous obtenons un graphe triangulé avec deux arêtes supplémentaires par contre si on choisit l'ordre de la fig. E2.2 on obtient un graphe triangulé avec cinq (5) arêtes supplémentaires.

Ceci dit, la triangulation minimale n'est pas optimale.

Exemple de déroulement

Suivant l'ordre lexicographique contenu dans Fig. E2.1	*Selon l'ordre lexicographique proposé par l'utilisateur Fig. E2.2*
Initialisation	Initialisation
F=Ø	F=Ø
e(E)=0 / Ordre=0	e(E)=0 / Ordre=1
e(A)=0 / Ordre=1	e(A)=0 / Ordre=2
e(B)=0 / Ordre=2	e(B)=0 / Ordre=3
e(C)=0 / Ordre=3	e(C)=0 / Ordre=4
e(D)=0 / Ordre=4	e(D)=0 / Ordre=5
e(F)=0 / Ordre=5	e(F)=0 / Ordre=0
Itérations LEX-M	Itérations LEX-M
I=6	I=6
S=Ø	S=Ø
Sommet sélectionné : F	Sommet sélectionné : D
e(D)=6	e(B)=6
α(F)=6	e(C)=6
	e(F)=6
I=5	α(D)=6
S=Ø	
Sommet sélectionné : D	I=5
e(B)=5	S=Ø
e(C)=5	Sommet sélectionné : C
α(D)=5	Sommet ajouté : B
	Sommet ajouté : F
I=4	Arc ajouté : (C , B)
S=Ø	Arc ajouté : (C , F)
Sommet sélectionné : C	e(A)=5
Sommet ajouté : B	e(B)=11
Arc ajouté : (C , B)	e(F)=11

e(A)=4

e(B)=9

α(C)=4

I=3

S=Ø

Sommet sélectionné : B

Sommet ajouté : A

Arc ajouté : (B , A)

e(E)=3

e(A)=7

α(B)=3

I=2

S=Ø

Sommet sélectionné : A

e(E)=5

α(A)=2

I=1

S=Ø

Sommet sélectionné : E

α(E)=1

Alpha

α(E)=1

α(A)=2

α(B)=3

α(C)=4

α(D)=5

α(F)=6

Liste des arcs ajoutés

α(C)=5

I=4

S=Ø

Sommet sélectionné : B

Sommet ajouté : A

Sommet ajouté : F

Arc ajouté : (B , A)

Arc ajouté : (B , F)

e(E)=4

e(A)=9

e(F)=15

α(B)=4

I=3

S=Ø

Sommet sélectionné : F

Sommet ajouté : A

Arc ajouté : (F , A)

e(A)=12

α(F)=3

I=2

S=Ø

Sommet sélectionné : A

e(E)=6

α(A)=2

I=1

S=Ø

Sommet sélectionné : E

α(E)=1

(C , B)	Alpha
(B , A)	α(E)=1
	α(A)=2
	α(B)=4
	α(C)=5
	α(D)=6
	α(F)=3
	Liste des arcs ajoutés
	(C , B)
	(C , F)
	(B , A)
	(B , F)
	(F , A)

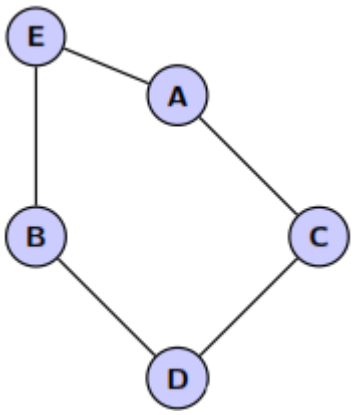
Améliorations / Propositions :

Nous savons que tout sommet isolé (feuille) contenu dans le graphe sera isolé dans l'arbre de jonction. Si nous procédons à la suppression des arêtes ajoutées contenant un sommet isolé, nous obtiendrons une triangulation minimale d'un ordre inférieur ou égal (en termes d'arêtes ajoutées).

Dans le cas de notre exemple, les arêtes (C, F) , (B, F) et (F, A) ne seront pas incluses dans le graphe G+ car le sommet F est un sommet isolé, ce qui nous donnera les mêmes arêtes ajoutées dans les deux cas de figure (*fig*. E2.1 et *fig.* E2.2)

Nous proposons de réduire la taille du graphe initial tout en éliminant tout sommet isolé.

Dans notre cas de figure, il serait préférable de trianguler le graphe suivant :

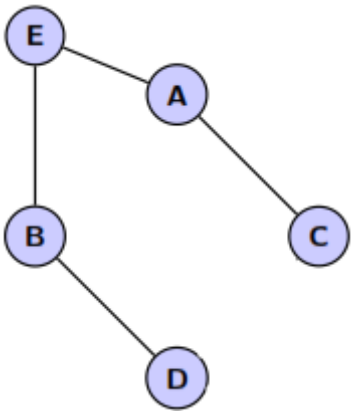


et de rajouter toute arête éliminée dans le graphe d'origine au graphe triangulé obtenu.

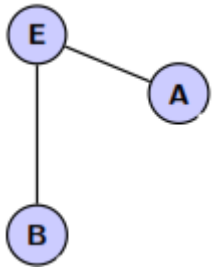
En procédant ainsi, nous réduisons le nombre d'itérations de (x) ou (x) est le nombre de sommets isolés obtenus par éliminations successives.

Nous proposons de répéter l'opération d'élimination de sommets isolés sur tout nouveau graphe obtenu jusqu'à ne plus obtenir de graphe contenant de sommets isolés.

Prenons l'exemple d'un arbre



- Graphe initial (A trianguler)



- 1e itération d'élimination des feuilles
- Feuilles **C** et **D** sont éliminées

- 2e itération d'élimination des feuilles
- Feuilles **A** et **B** sont éliminées
- Nous n'avons plus de graphe à trianguler.

- Nous proposons aussi d'éliminer les deux dernières itérations de l'algorithme LEX-M c'est-à-dire démarrer de I=n où n est le nombre de sommets obtenus après élimination des feuilles et d'arrêter à i=3.
- Nous nommons notre amélioration Algorithme LEX-M SI-2

Algorithme **LEX-M SI-2**

Données : un graphe G=(V,E)
Résultat : un ordre minimal α et G^* triangulé.
Début :
G(SI)=G(V,E)
Répéter
Eliminer tout sommet isolé
Obtenir nouveau G(SI)
Jusqu'à ce que G(SI) ne contient plus de feuilles

Dérouler l'algorithme **LEX-M** sur G(SI) tout en arrêtant ce dernier à l'itération i=3.
Affecter des ordres ***α*** aux deux sommets restants.
On obtient $G(SI)^*$
Rajouter au graphe $G(SI)^*$ toutes les arêtes précédemment éliminées pour obtenir G^*.

Fin.

Conclusion

Aucun algorithme ne précise comment choisir le premier sommet de l'ordre lexicographique alors que celui-ci s'avère d'une importance capitale.

Les méthodes de triangulation minimale s'avèrent non optimales. Cependant nous proposons donc de choisir plusieurs ordres lexicographiques tels que :

1- Aléatoire (Choix en 1^e)
2- Basé sur les valeurs des alphas obtenues de 1
3- Basé sur les valeurs des étiquettes obtenues de 1
4- Un autre ordre aléatoire

Conclusion

Conclusion et perspectives

Plusieurs techniques de résolution des problèmes de satisfaction de contraintes ont été développé depuis 1970, des techniques dites énumératives qui sont couteuses en terme de temps d'exécution, d'espace mémoire et de calculs inutiles. Leur complexité est de l'ordre exponentiel, d'autres techniques basée sur la décomposition arborescente qui sont efficaces et réduits l'espace de recherche.

Parmi ces techniques de décomposition arborescente celles basées sur la triangulation du graphe de contraintes, qui décompose le CSP à des sous-ensembles réduits en triplet qui facilite la résolution.

La plus ancienne de ces techniques de triangulation est Lex-M qui retourne à 40 ans, cette technique a subi une simplification qui a fait naissance à MCS-M qui donne pratiquement les mêmes résultats que Lex-M.

Depuis 1976 Lex-M est l'algorithme de référence dans la recherche de décomposition arborescente, les études qui ont été faites ont montré que Lex-M donne des décompositions de largeur minimale avec une complexité polynomiale et qui réduisent le temps exécution.

Au niveau des perspectives, il semble rester beaucoup à faire. Déjà pour des améliorations de la mise en œuvre de l'algorithme LEX-M SI-2 qui semble significativement optimisable au niveau du temps de calcul. Au-delà de cette approche s'ouvre un champ de recherche très vaste. En effet, le schéma de l'algorithme utilisé doit permettre la mise en œuvre d'heuristiques différentes en proposant d'autres choix pour la sélection associée au choix lexicographique et d'étiquettes e (v) qui peuvent donner une triangulation minimale optimale, même pour le choix du chemin allant du sommet **u** au sommet **v** ; déjà numéroté.

Enfin, il reste aussi à analyser cette nouvelle approche de triangulation au regard de son apport pour la résolution de CSP et étudier son impact dans le cas de la compilation.

Bibliographie

[1] - décompositions arborescentes pour résoudre les problèmes de satisfaction de contraintes Avec mise en œuvre du parallélisme [LALOU **Mohamed** pour l'obtention du Grade De Magistère en Informatique 2009 Université de Bejaia].

[2] - S. Golumb and L. Baumert, *Backtrack programming*, Journal of the ACM, pages 516-524 (1965).

[3] - R. Haralick and G. Elliot, *Increasing tree search efficiency for constraint satisfaction problems*, Artificial Intelligence, 14 :263-313, (1980).

[4] - Allocation de créneaux de décollage par décomposition arborescente de modèles CSP [Thèse docteur en science O.Gourmel 2007 toulous].

[5] - Robertson, N. ET Seymour, P. (1986). Graph minors. ii. Algorithmic aspects of tree-Width. J. Algorithms, 7(3):309–322.

[6] - hakimYannakakis, M. (1981). Computing the minimum fill-in is np-Complete. SIAM Journal on Algebraic Discrete Methods, 2(1):77–79.

[7] - Gavril, F. (1974). The intersection graphs of subtrees in trees are exactly the chordal Graphs. Journal of Combinatorial Theory, Series B, 16(1):47–56.

[8] - Koster, A., Bodlaender, H. et Van Hoesel, S. (2001). Treewidth : Computational Experiments. METEOR, Maastricht research school of Economics of Technology and Organizations; University Library, Universiteit Maastricht [Host].

[9] - Fulkerson, D. et Gross, O. (1965). Incidence matrices and Interval graphs. Pacific J. Math, 15(3) :835–855.

[10] - Gogate, V. et Dechter, R. (2004). A complete anytime Algorithm for treewidth. In Proceedings of the 20th conference on Uncertainty in artificial intelligence, pages 201–208.

[11] - Larranaga, P., Kuijpers, C., Poza, M. et Murga, R. (1997). Decomposing bayesian Networks: triangulation of the moral graph with genetic algorithms. Statistics and Computing, 7(1):19–34.

Bibliographie

[12] - Kjaerulff, U. (1992). Optimal decomposition of probabilistic networks by simulated Annealing. Statistics and Computing, 2(1):7–17.

[13] - Tarjan, R. et Yannakakis, M. (1984a). Simple linear-Time algorithms to test chordality Of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM Journal on Computing, 13:566.

[14] - Rose, D., Tarjan, R. ET Lueker, G. (1976). Algorithmic aspects of Vertex elimination On Graphs. SIAM Journal on computing, 5(2):266–283.

[15] - Berry, A., Blair, J., Heggernes, P. ET Peyton, B. (2004). Maximum Cardinality search For computing minimal triangulations of graphs. Algorithmica,

[16] - De nouvelles approches pour la décomposition de réseaux de contraintes [Philippe. Jégou, Hanan.Kanso, Cyril.Terrioux 2015].

[17] - Hypertree decompositions and tractable queries, J. Comput. Syst. Sci., 64(3), 579-627 (2002).

[18] - Constraint processing, Morgan Kaufmann Publisher (2003).

[19] - N. Robertson and P. D. Seymour, Graph minors, algorithmic aspects of tree width, J. Algorithmis, 7(3), 309-322 (1986).

[20] - R. Tarjan , M. Yannakakis, "Simple linear time algorithms to test chordality of graphs, Test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs". SIAM Journal Vol 13 N° 3, 1984.

[21] - Fulkerson, D. et Gross, O. (1965). Incidence matrices and Interval graphs. Pacific J. Math, 15(3):835–855.

[22] - Mathieu Fontaine. Thèse de doctorat soutenue le 04/07/2013 Apport de la Décomposition arborescente pour les méthodes de type VNS. Intelligence artificielle [cs.AI]. Université de Caen, 2013. Français.

[23] - Programmation par contraintes Cours 2 : Problèmes de Satisfaction de Contraintes CSP Odile PAPINI – ESIL Université de la méditerranée. http://pages-perso.esil.univmed.fr/~papini/

Abstract

It is hard NP problem to resolve a CSP. Facing a difficulty many researches have been taken and which have led to define many methods of resolving.

We can classified these methods in two approaches; the first is based in full exploration of research area. In another hand, the second method use the heuristics. Among, we can speak about the tree decomposition. From this methods there are those decomposing the graph of the constraints associated with the CSP without the triangulated as the heuristic H-TD-WT proposed by Ciril Tireoux.

The choice of the first cluster and the limiting of its maximum size that requires heuristic are the main disadvantage of this method.

The triangulation's methods make the night choice of first cluster of the begging, decreases the size and gives solutions close the optimum to resolve the CSP problem.

LEX-M is t a method of triangulation which date of 1976 which gives a minimum triangulation and minimum width, the choice of the Summit of departure is based sue the lexicographic order this helped us triangulated the graph of constraints without making use of heuristics. The fact that LEX-M has not other competing methods, in terms of the execution time or complexity theoretical, the Triangulation by LEX-M is a good choice.

The implementation of LEX-M has led us to propose the improved by eliminating the isolated peaks and to stop the progress of the algorithm to the two last summits not numbered. The search domain is open to any proposal.