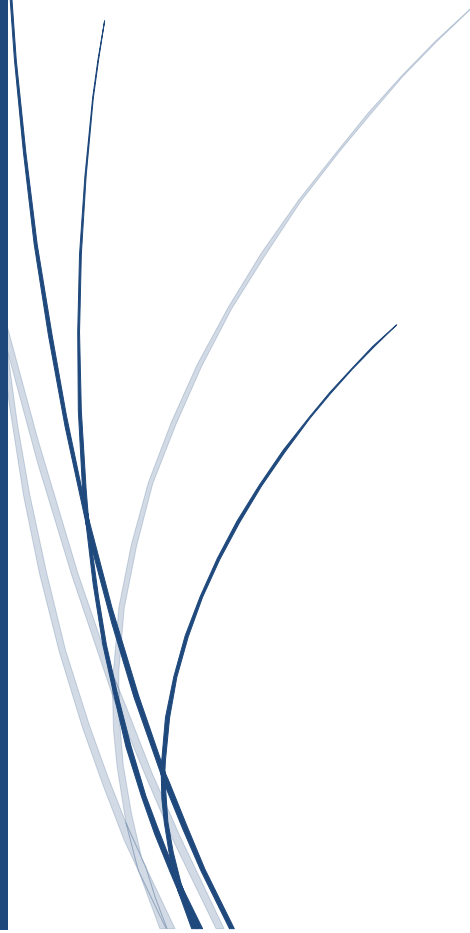




Rapport de projet

Mise en œuvre et analyse  
expérimentale d'un algorithme  
de résolution du problème de  
satisfaction de contraintes

*Encadré par M. Philippe JEGOU*



Amadou KANE – Hamza ZOUDANI  
MASTER 1 INFORMATIQUE – 2014/2015

## Sommaire

Sommaire.....	1
1. Problème à satisfaction de contraintes.....	5
2. Algorithmes de résolution .....	6
2.1. Backtrack .....	6
2.2. Foward Checking .....	7
2.3. Arc consistance et filtrage.....	8
2.3.1. <i>RFL</i> .....	9
2.3.2. <i>MAC</i> .....	9
3. Méthode de travail .....	9
3.1. Gestion de projet .....	9
3.2. Langage et outils de développement.....	10
3.2.1. Langage de programmation .....	10
3.2.2. Travail collaboratif .....	11
4. Travail réalisé .....	11
4.1. Description du projet.....	11
4.2. Description des structures créés .....	11
4.3. Génération d'un graphe aléatoire.....	12
4.3.1. Procédures pour générer le graphe .....	12
4.3.2. Modélisation du problème de 3-colorabilité avec la structure.....	13
4.4. Implémentation des algorithmes .....	14
4.4.1. Backtrack .....	14
4.4.2. Foward Checking.....	14
4.4.3. <i>RFL</i> .....	15
4.5. Intégration des heuristiques .....	16

4.6. Graphes de comparaison .....	17
4.6.1. Foward Checking.....	17
4.6.2. RFL.....	18

# Remerciements

Nous tenons à remercier tout particulièrement et à témoigner notre reconnaissance à M. Philippe Jégou, notre tuteur de projet, pour l'aide qu'il nous a apporté durant la période du projet. Sa disponibilité, sa pédagogie et ses conseils nous ont été précieux dans nos choix de conception et d'élaboration de nos programmes.

# Introduction

Dans le cadre de notre Master 1 Informatique, les étudiants sont amenés à choisir entre un projet et un stage. Si les étudiants choisissent un projet, ils doivent en choisir un parmi ceux proposés par les enseignants de l'Université d'Aix-Marseille. Nous avons choisi le projet « Mise en œuvre et analyse expérimentale d'un algorithme de résolution du problème de satisfaction de contraintes » tuteuré par M Philippe Jégou, enseignant-chercheur d'Aix-Marseille.

## 1. Problème à satisfaction de contraintes

Un problème à satisfaction de contraintes a trois composantes :  $X$ ,  $D$  et  $C$  :

- $X$  est un ensemble de variables,  $\{x_1, \dots, x_n\}$ .
- $D$  est un ensemble de domaines,  $\{d_1, \dots, d_n\}$ , un pour chaque variable.
- $C$  est un ensemble de contraintes qui spécifie les combinaisons admissibles de valeurs.

Chaque domaine  $d_i$  est constitué d'un ensemble de valeurs admissibles  $\{v_1, \dots, v_k\}$  pour la variable  $x_i$ . Chaque contrainte  $c_i$  est constituée d'une paire de (*portée*, *rel*), où *portée* est un n-uplet de variables qui participent à la contrainte et *rel* est une relation qui définit les valeurs que ces variables peuvent prendre. Une relation peut être représentée comme une liste explicite de tous les n-uplets de valeurs qui satisfont la contrainte, ou comme une relation abstraite sur laquelle s'appliquent deux opérations:

- Tester si un n-uplet est un membre de la relation.
- Enumérer les membres de la relation.

Par exemple, si  $x_1$  et  $x_2$  ont tous les deux le domaine  $\{A, B\}$ , la contrainte qui dicte que les variables doivent avoir des valeurs différentes peut s'écrire  $\langle(x_1, x_2), [(A, B), (B, A)]\rangle$  ou  $\langle(x_1, x_2), x_1 \neq x_2\rangle$ .

L'objectif consiste simplement à trouver un ensemble de valeurs à affecter aux variables, de sorte que toutes les contraintes soient satisfaites.

Nous nous intéressons dans le cadre de ce projet aux problèmes de satisfaction de contraintes binaires en domaines finis.

Un CSP<sup>1</sup> binaire est un CSP  $P = (V, D, C)$  dont toutes les contraintes  $c_i$  appartenant à  $C$  ont une arité à 2, c'est-à-dire que chaque contrainte a exactement 2 variables pertinentes.

Exemple d'un CSP binaire : le coloriage de graphes

Le problème de coloriage de graphe avec 3 couleurs consiste à colorier un graphe non-orienté comprenant  $n$  nœuds. Chaque nœud doit être colorié avec une des trois couleurs disponibles de telle sorte que deux nœuds voisins n'aient pas la même couleur.

---

<sup>1</sup> CSP : Problème à satisfaction de contraintes

## 2. Algorithmes de résolution

Les algorithmes de résolution ont pour but de vérifier que toutes les variables peuvent prendre des valeurs qui appartiennent à leur domaine, tout en respectant les différentes contraintes.

Cet algorithme devra donc générer au moins une solution qui vérifie le système. Il devra pour cela, affecter des valeurs aux variables tout en vérifiant que ces valeurs ne s'opposent pas aux contraintes.

Il existe plusieurs types d'algorithmes. Certains fonctionnent de manière systématique en testant toutes les solutions possibles jusqu'à ce qu'ils trouvent une bonne solution. D'autres, plus intuitifs, construisent la réponse à partir d'une solution non satisfaisante quelconque en la modifiant selon les contraintes.

### 2.1. Backtrack

Cet algorithme trouve systématiquement une solution s'il en existe une. Pour cela, il affecte au fur et à mesure une valeur de  $D$  à chaque variable correspondante dans  $V$ . Il vérifie évidemment que la valeur est correcte par rapport aux contraintes.

A chaque affectation, les domaines des variables restantes diminuent. Il arrive qu'à l'affectation d'une variable  $x_{n-1}$ , l'algorithme ne puisse trouver de valeur pour  $x_n$  car  $D(x_n)$  est vide.

A ce stade, l'algorithme revient en arrière, à la dernière variable affectée  $x_{n-1}$ . Il modifie la valeur de  $x_{n-1}$  en espérant que le domaine de la variable  $x_n$  suivante ne sera plus nul. Si après avoir essayé toutes les valeurs du domaine  $D(x_{n-1})$ , il n'a pas toujours de solutions pour  $x_n$ , il recule et modifie la variable précédente  $x_{n-2}$ . Si il n'existe pas de solution, il reculera jusqu'à  $x_0$ , sinon il trouve obligatoirement la solution.

Le défaut de cet algorithme est qu'il peut tester toutes les valeurs possibles avant de trouver une solution. Pour le problème de la coloration de graphe, si on lance la procédure avec 100 sommets et 3 couleurs possibles, il existe  $3^{100}$  solutions. Si jamais la première bonne solution se trouve dans les dernières possibilités ou si jamais il n'y a pas de solutions, le temps de calcul sera très long.

Exemple :

On va exécuter l'algorithme du backtrack avec les contraintes suivantes :

$$\left\{ \begin{array}{l} x_1 < x_2 \\ x_1 < x_3 \\ x_1 < x_4 \\ x_1 < x_5 \\ x_3 < x_6 \\ x_4 > x_6 \\ x_5 \in \{1,2\} \\ x_i \in \{1,2,3\}, i \neq 5 \end{array} \right.$$

Le graphe suivant montre le nombre de tests qu'on aura effectué :

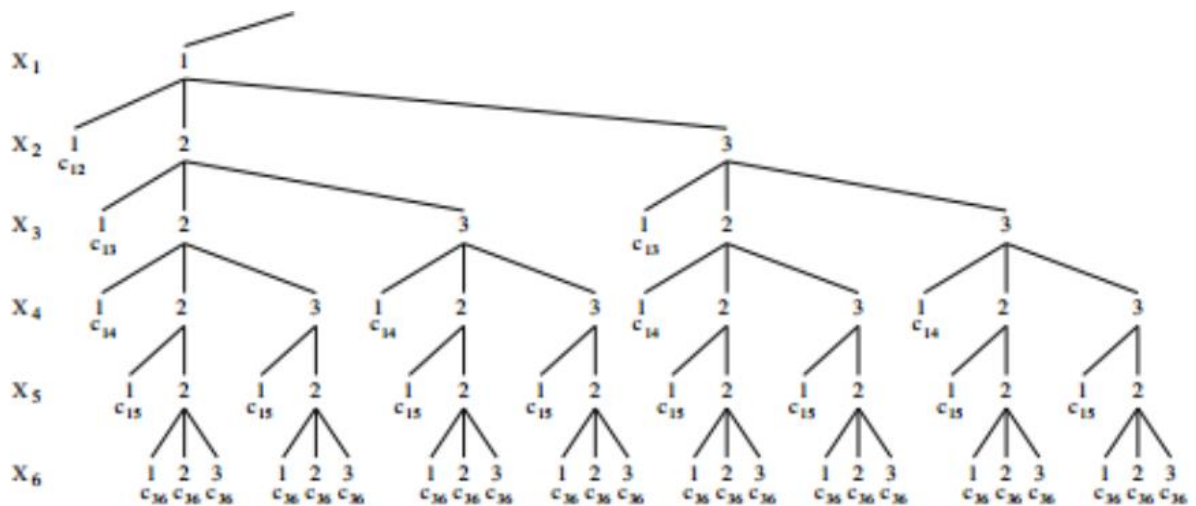


Figure 1 : Graphe résumant les tests du backtrack

## 2.2. Forward Checking

Cet algorithme est assez proche du backtrack, il affecte des valeurs aux variables au fur et à mesure. La différence 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 que les autres variables  $x_i$  ( $i > n$ ) pourront être affectées. Si l'affectation de  $x_i$  ne peut être faite, l'algorithme choisit une autre valeur pour  $x_n$ .

Si aucune solution n'est trouvée pour  $x_n$ , on ne pourra pas faire d'affectation aux 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  $x_0$ .



Exemple :

On va exécuter l'algorithme du foward checking avec les contraintes suivantes :

$$\left\{ \begin{array}{l} x_1 < x_2 \\ x_1 < x_3 \\ x_1 < x_4 \\ x_2 < x_5 \\ x_3 > x_5 \\ x_3 \in \{1,2\} \\ x_i \in \{1,2,3\}, i \neq 3 \end{array} \right.$$

Le graphe suivant montre le nombre de tests qu'on aura effectué :

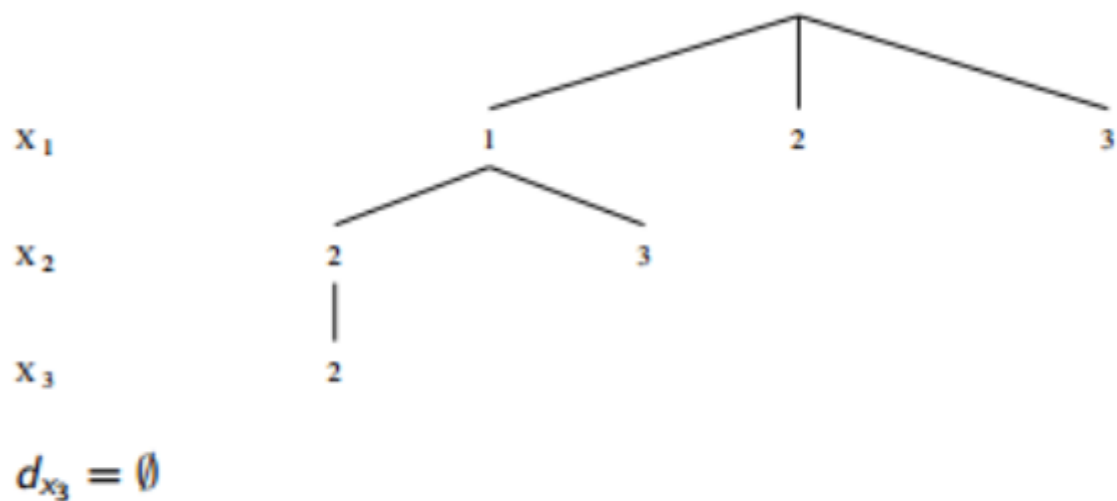


Figure 2 : Graphe résumant les tests du foward checking

### 2.3. Arc consistance et filtrage

Un CSP  $(X, D, C)$  est consistant d'arc, si pour tout couple de variables  $(x_i, x_j)$  de  $X$ , et pour toute valeur  $v_i$  appartenant à  $D(x_i)$ , il existe une valeur  $v_j$  appartenant à  $D(x_j)$  telle que l'affectation partielle  $\{(x_i, v_i), (x_j, v_j)\}$  satisfasse toutes les contraintes binaires de  $C$ .

L'algorithme qui enlève les valeurs des domaines des variables d'un CSP jusqu'à ce qu'il soit consistant d'arc -on dit que l'algorithme filtre les domaines des variables- s'appelle  $AC^2$ . Il existe différentes versions de cet algorithme:  $AC1$ ,  $AC2$ ,  $AC3$ , ..., chaque version étant plus efficace (en général) que la précédente.

<sup>2</sup> Arc Consistency : (fr) Consistance d'arc

### 2.3.1.RFL<sup>3</sup>

L'algorithme a pour objectif de résoudre des instances de *CSP* et réalise une recherche en profondeur d'abord avec les retours en arrière. A chaque étape de la recherche, une assignation de variable est effectuée suivie par un processus de filtrage qui correspond à établir la consistance d'arc.

### 2.3.2.MAC<sup>4</sup>

L'algorithme consiste à instancier une variable par valeur de son domaine qui est considérée comme la réduction de ce dernier à la valeur choisie ; cette réduction est donc propagée pour obtenir la consistance d'arc. Ce processus est répété lors de chaque instanciation.

Cette technique permet d'élaguer considérablement l'espace de recherche et semble (en l'état actuel des connaissances) présenter le meilleur compromis entre surcoût occasionné par le filtrage et réduction de l'espace de recherche.

## 3. Méthode de travail

### 3.1. Gestion de projet

Nous avons utilisé une méthode adaptative, la méthode « Scrum » vu en cours de génie logiciel. Celle-ci consiste à planifier le travail en constituant un « product backlog ». Les items seront évalués selon leurs valeurs, charges et risques.

Plus les items sont prioritaires, plus ils seront :

- Sous-divisés en sous-tâches pour une meilleure évaluation.
- Décrits sous diverses formes.

Les plannings effectués doivent être développés pendant une itération de durée fixe, nommée « sprint backlog ».

A la fin de chaque itération, un bilan est fait sur le travail effectué afin de le comparer à ce qui était planifié. Le bilan permet également de voir avec le client si le travail correspond à ses attentes. Le planning de la prochaine itération sera fait en fonction du résultat du bilan.

Malgré cette organisation, nous avons dû faire face à des contraintes et des obstacles qui ont ralenti l'avancement du projet. Nous avons mis beaucoup de temps à analyser et comprendre

---

<sup>3</sup> *RFL : Real-full look-ahead*

<sup>4</sup> *MAC : Maintaining Arc-Consistency*

les objectifs. Il nous a fallu une grande période afin de se documenter sur le sujet, et de comprendre le fonctionnement des algorithmes.

Durant tout le projet, la difficulté principale était que l'on se mette d'accord sur la façon d'implémenter les algorithmes, les différences entre eux n'étaient pas claires pour nous au départ.

De plus, les périodes de révisions des examens du deuxième semestre ne nous ont pas permis de nous focaliser totalement sur le projet.

## 3.2. Langage et outils de développement

### 3.2.1. Langage de programmation

Nous avons utilisé le langage C pour la programmation ainsi que la génération des graphes. Ce langage était imposé dans le sujet du TER.

Il a fallu utiliser plusieurs bibliothèques propres au C :

Time.h et Math.h : Bibliothèques nécessaires pour pouvoir générer un nombre aléatoire.

Pour pouvoir tester les performances de chaque algorithme, on a dû utiliser plusieurs outils liés avec la plateforme eclipse :

- *Valgrind* :  
Il s'agit d'un cadre d'instrumentation pour construire des outils d'analyse dynamique qui peuvent être utilisés pour profiler les applications en détail. Ces outils sont généralement utilisés pour détecter automatiquement de nombreux problèmes de gestion de mémoire et de filtrage. La suite *Valgrind* comprend également des fonctionnalités permettant de créer de nouveaux outils de profilage.
- *Oprofile* :  
Cet outil est un profileur statistique pour les systèmes Linux. Ce logiciel est capable de profiler tout le code en cours d'exécution à faible surcharge.
- *Gprof* :  
C'est un outil qui permet de savoir où un programme a dépensé son temps et quelles sont les fonctions qui ont fait un appel de méthodes durant leurs exécutions. Ces informations sont nécessaires afin de savoir quels sont les éléments les plus lents du programme.

### 3.2.2. Travail collaboratif

Nous avons utilisé un gestionnaire de versions appelé Git, avec un dépôt central gratuit appelé GitHub. Notre choix s'est porté sur ce gestionnaire plutôt que *Hg*<sup>5</sup> ou *SVN*<sup>6</sup>, car il offrait un meilleur compromis entre la puissance des fonctionnalités disponibles et la simplicité d'utilisation de l'outil. De plus, nous avons dû en avoir recours lors du projet de Génie Logiciel.

Lien du projet : <https://github.com/AmadouK/TER.git>

Nous avons dû également communiquer via des mails entre nous et avec notre encadrant, afin de tester et de valider les tâches du sprint backlog.

## 4. Travail réalisé

### 4.1. Description du projet

Ce projet rentre dans le cadre de recherche national appelé TUPLES soutenu par l'Agence Nationale de la Recherche.

L'objectif est de :

- Générer un graphe aléatoire représentant les variables ainsi que les contraintes sur leurs valeurs du domaine.
- Implémenter et comparer l'algorithme du Forward Checking avec RFL. Les comparaisons seront sur le temps d'exécution des algorithmes, le pourcentage de CPU utilisé et le nombre d'affectations effectuées.
- Utiliser des heuristiques afin d'optimiser le choix des variables dans les algorithmes précédents.
- Etudier les comportements des algorithmes et comparer les résultats obtenus avec *FC* et *RFL* naïfs. En déduire si les heuristiques utilisées sont efficaces ou non.
- Proposer des améliorations pour optimiser le fonctionnement de chaque algorithme.

### 4.2. Description des structures créées

Pour pouvoir représenter les graphes d'arc consistance, il a fallu créer deux structures :

- La première, nommée *var\_domaine* représente les valeurs qui doivent correspondre entre deux variables selon les contraintes. Elle est représentée par

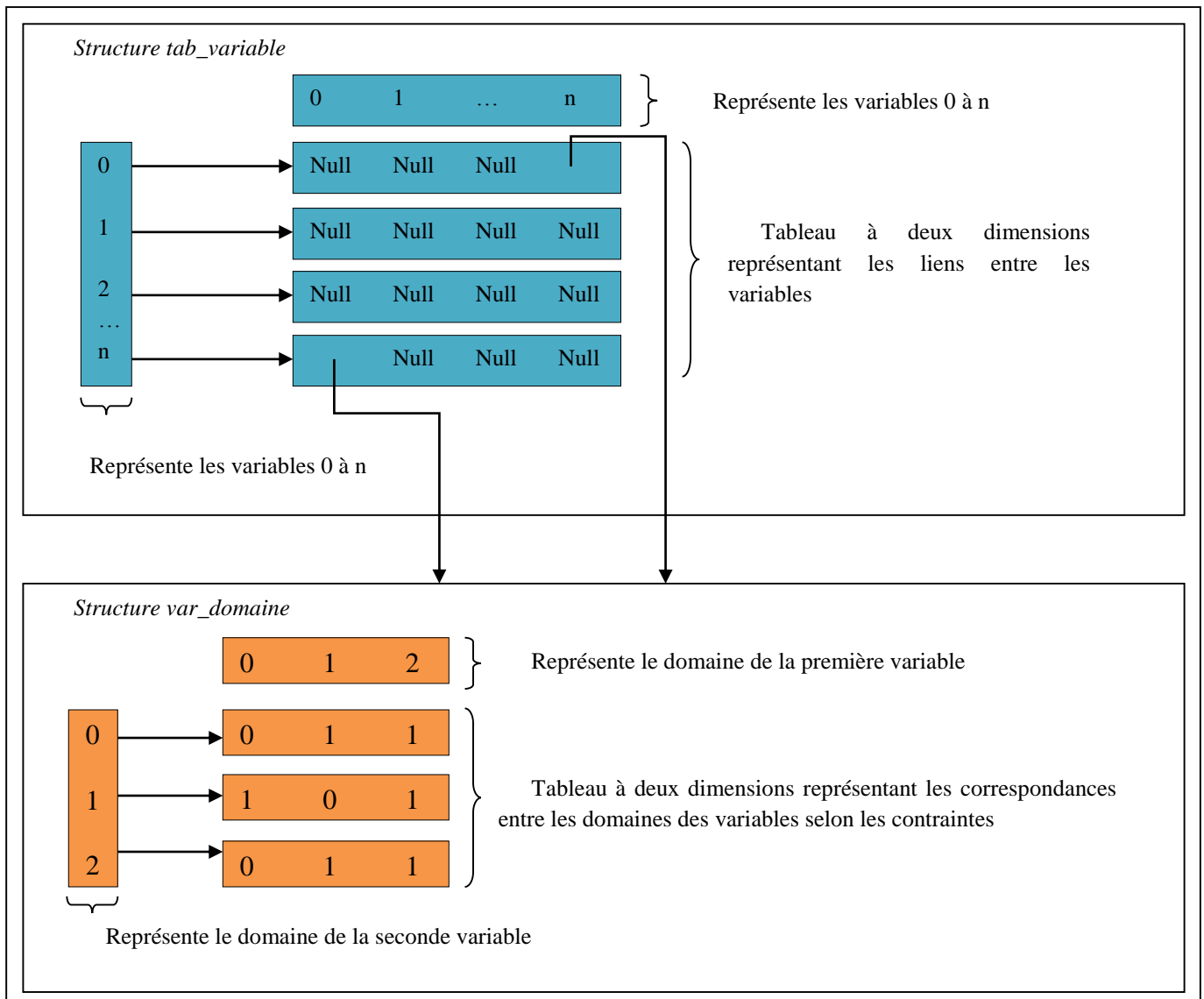
---

<sup>5</sup> *SVN* : Apache Subversion, logiciel de gestion de versions

<sup>6</sup> *Hg* : Mercurial, logiciel de gestion de version décentralisé

un tableau à deux dimensions de pointeurs d'entier. Si la case 0, 1 de l'instance de la structure est à 1, cela veut dire que la variable  $x_1$  avec la valeur 0 peut être associée avec la variable  $x_2$  de valeur 1.

- La seconde structure, appelée `tab_variable` regroupe les liens entre les variables. Elle est représentée par un tableau à deux dimensions de pointeurs sur des `var_domaine`. Ces pointeurs sont tous initialisés à Null à départ, et prennent l'adresse d'une instance de la structure `var_domaine` s'il y a un arc entre deux variables.



*Figure 3 : Relations entre les deux structures*

### 4.3. Generation d'un graphe aleatoire

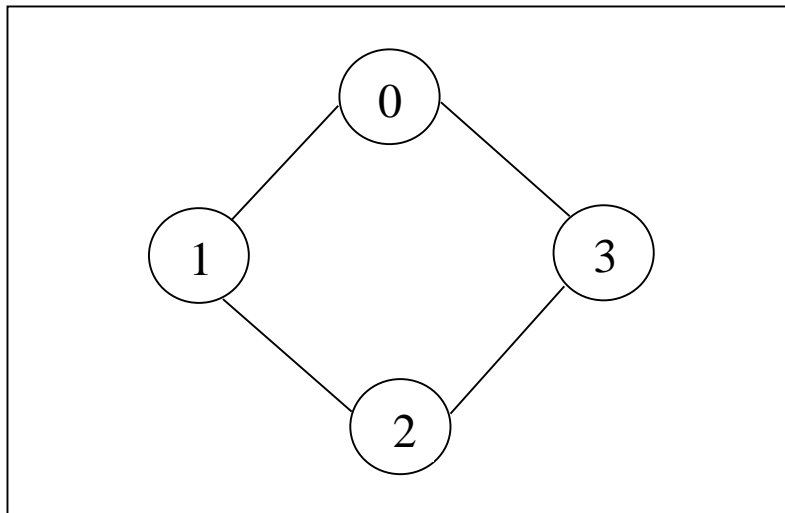
#### 4.3.1.Procédures pour générer le graphe

La génération d'un graphe aléatoire passe par deux fonctions :

- `void genereGraphe(int nbSommet,int nbArrete,int nbDomaine,float sat)`  
 Cette fonction aura pour but de créer une instance de la structure `tab_variable` et de la configurer en fonction des paramètres `nbSommet` et `nbArrete`, qui sont le nombre de sommets et le nombre d'arrêtes dans le graphe. Les arrêtes seront générées aléatoirement, et les correspondances seront effectuées à l'aide de la fonction `genereDomaine` décrite ci-dessous.
- `var_domaine* genereDomaine(float s,int nbDomaine,int var1,int var2)`  
 Cette fonction a pour but de créer une instance de la structure `var_domaine` et de la configurer en fonction des paramètres de la fonction. On aura donc un tableau à deux dimensions d'entiers de taille `nbDomaine`. Les correspondances des domaines seront générées aléatoirement, en ne dépassant pas le coefficient de satisfiabilité `s`.

#### 4.3.2. Modélisation du problème de 3-colorabilité avec la structure

On va représenter le graphe suivant dans la structure :



*Figure 4 : Exemple de graphe pour la 3-colorabilité*

La figure ci-dessous représenter l'état de la structure après avoir implémenté le graphe avec les contraintes de la 3-colorabilité.

	0	1	2	3
0	NULL	Contrainte Sommet 0 Sommet 1	NULL	Contrainte Sommet 0 Sommet 3
1	Contrainte Sommet 0 Sommet 1	NULL	Contrainte Sommet 1 Sommet 2	NULL
2	NULL	Contrainte Sommet 1 Sommet 2	NULL	Contrainte Sommet 2 Sommet 3
3	Contrainte Sommet 0 Sommet 3	NULL	Contrainte Sommet 2 Sommet 3	NULL

*Liens entre les variables*

	Rouge	Vert	Bleu
Rouge	0	1	1
Vert	1	0	1
Bleu	1	1	0

*Contrainte Sommet 0 - Sommet 1*

	Rouge	Vert	Bleu
Rouge	0	1	1
Vert	1	0	1
Bleu	1	1	0

*Contrainte Sommet 2 - Sommet 3*

	Rouge	Vert	Bleu
Rouge	0	1	1
Vert	1	0	1
Bleu	1	1	0

*Contrainte Sommet 1 - Sommet 2*

	Rouge	Vert	Bleu
Rouge	0	1	1
Vert	1	0	1
Bleu	1	1	0

*Contrainte Sommet 2 - Sommet 3*

*Figure 5 : Génération du graphe obtenu avec les contraintes de la 3-colorabilité*

## 4.4. Implémentation des algorithmes

### 4.4.1. Backtrack

Il a fallu coder l'algorithme vu dans la partie 2.1.

Pseudo code de la fonction principale :

```

Fonction BT(variable : entier)
Début
    Pour i de 0 à taille_domaine faire
        Si (verif_Contrainte(variable,i))
            Affectation de la variable à la valeur i
            ok = BT(variable+1) ;
            si ok
                retourner 1
            FinSi
        FinSi
    FinPour
    On fait un backtrack
Fin

```

### 4.4.2. Forward Checking

Il a fallu coder l'algorithme vu dans la partie 2.2.

### Pseudo code des fonctions principales:

```
Fonction FC(variable : entier, values : entier**)
Début
    Pour i de 0 à taille_domaine faire
        x = values[variable][i]
        Si (CF(values,variable,x))
            Affectation de la variable à la valeur i
            ok = FC(variable+1,values) ;
            si ok
                retourner 1
            sinon
                Récupération des valeurs du domaine avant affectation
            FinSi
        FinSi
    FinPour
On fait un backtrack
Fin
```

```
Fonction CF(variable, x : entier,values : entier**)
Début
    Pour i de variable+1 à nb_sommet faire
        Si il y a un lien entre i et variable
            Lien_var = 1
            Si variable peut avoir la valeur x en respectant les contraintes avec j
                Ok = 1
            FinSi
        FinSi
        Si ok = 0 et lien_var = 0
            Retourner 0
        Sinon
            Ok = 0
            Lien_var = 0
        FinSi
    FinPour
    Retourner 1
Fin
```

#### 4.4.3.RFL

Il a fallu coder l'algorithme vu dans la partie 2.3.2.

### Pseudo code de la fonction principale :

```
Fonction RFL(variable : integer)
Début
    Récupération du domaine de toutes les variables
    Pour i de 0 à taille_domaine faire
        Si variable peut avoir la valeur i
            Affectation de la valeur i à la variable
            AC3()
            ok = RFL(variable+1)
            Si ok
                retourner 1
            Sinon
                Récupération des valeurs des valeurs de la structure avant affectation
            finSi
        finSi
    finPour
    backtrack
    retourner 0
Fin
```



Pour que RFL puisse fonctionner, il a fallu implémenter un des algorithmes de filtrage. Nous en avons implémenté un.

Pseudo code des fonctions de « AC3 » :

```
Procédure AC3()
Début
    L ← {(xi, xj), i ≠ j liées par une contrainte}
    TantQue L ≠ ∅
        Choisir et supprimer dans L un couple (xi, xj)
        Si Revise(xi, xj) alors
            L ← L ∪ {(xk, xi) / ∃ contrainte liant xk et xi}
        finSi
    finTantQue
Fin
```

```
Procédure revise(xi, xj)
Début
    modification ← false
    Pour chaque v ∈ Di faire
        Si ∄ v' ∈ Dj | {xi → v, xj → v'} est consistante alors
            Di ← Di \ {v}
            modification ← vrai
        finSi
    finPour
    retourner modification
Fin
```

## 4.5. Intégration des heuristiques

Les algorithmes que nous venons d'étudier choisissent, à chaque étape, la prochaine variable à instancier parmi l'ensemble des variables qui ne sont pas encore instanciées. Ensuite, une fois la variable choisie, ils essaient de l'instancier avec les différentes valeurs de son domaine. Ces algorithmes ne disent rien sur l'ordre dans lequel on doit instancier les variables, ni sur l'ordre dans lequel on doit affecter les valeurs aux variables. Ces deux ordres peuvent changer considérablement l'efficacité de ces algorithmes.

Les heuristiques concernant l'ordre d'instanciation des valeurs sont généralement dépendantes de l'application considérée et difficilement généralisables. En revanche, il existe de nombreuses heuristiques d'ordre d'instanciation des variables qui permettent bien souvent d'accélérer considérablement la recherche. L'idée générale consiste à instancier en premier les variables les plus "critiques", c'est-à-dire celles qui interviennent dans beaucoup de contraintes et/ou qui ne peuvent prendre que très peu de valeurs.

L'ordre d'instanciation des valeurs peut être :

- Domaine le plus petit  
On choisit les variables dont le domaine est le plus petit en premier dans l'algorithme. Si cette variable ne peut pas être instanciée, on se rendra compte plus rapidement que les contraintes ne sont pas satisfaisables.
- « First fail »

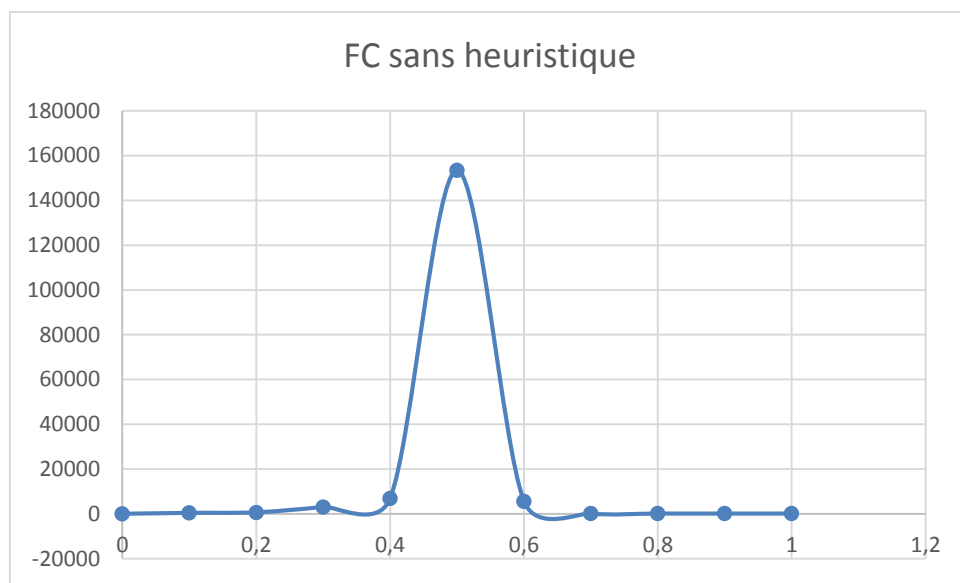
Les variables de l'algorithme sont choisies selon le calcul suivant :  $\min\left(\frac{|D_i|}{d^+(x_i)}\right)$

$D_i$  représente le domaine de la valeur  $i$ , et  $d^+(x_i)$  le nombre de liens avec les autres variables non affectées. On cherche avec cette heuristique à avoir un échec le plus rapidement possible.

## 4.6. Graphes de comparaison

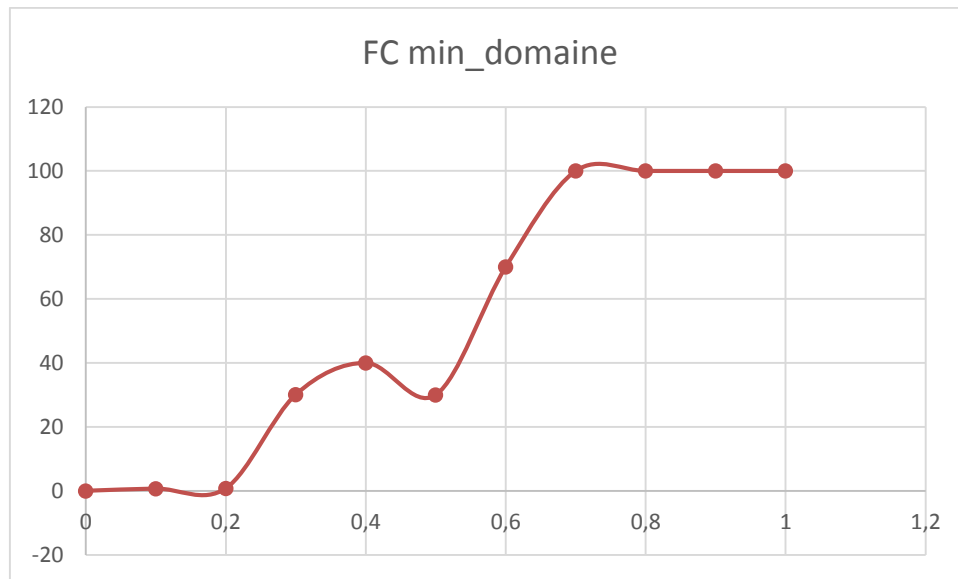
### 4.6.1. Forward Checking

Le graphe ci-dessous représente le nombre d'affectations effectués sur le forward checking sans heuristiques sur le coefficient de satisfiabilité, nous avons décidé de partir avec un graphe de 100 sommets, 10 arcs, un domaine de 5 valeurs.



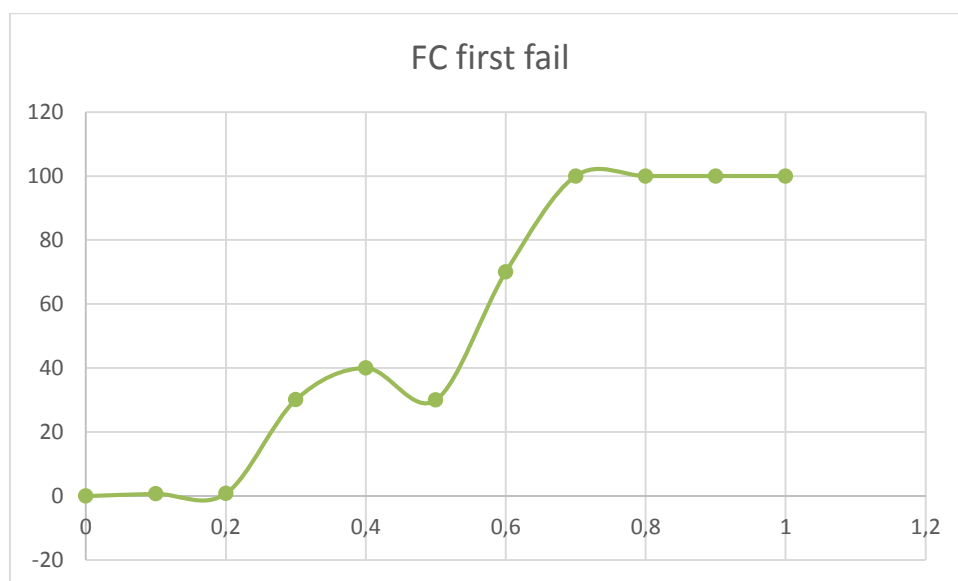
On peut remarquer que plus le coefficient se rapproche de 0,5, plus on aura de chance d'avoir plus d'affectations. On sera plus proche de 0 entre 0 et 0,5 et proche de 100 entre 0,5 et 1.

Le graphe suivant utilise est celui du forward checking avec l'heuristique du domaine le plus petit.



On remarque une différence entre les résultats du forward checking sans heuristique lorsque le nombre d'affectations, s'approche de 0,5. En effet, cette heuristique n'est efficace uniquement en cas d'échec et avec un coefficient de satisfiabilité élevé.

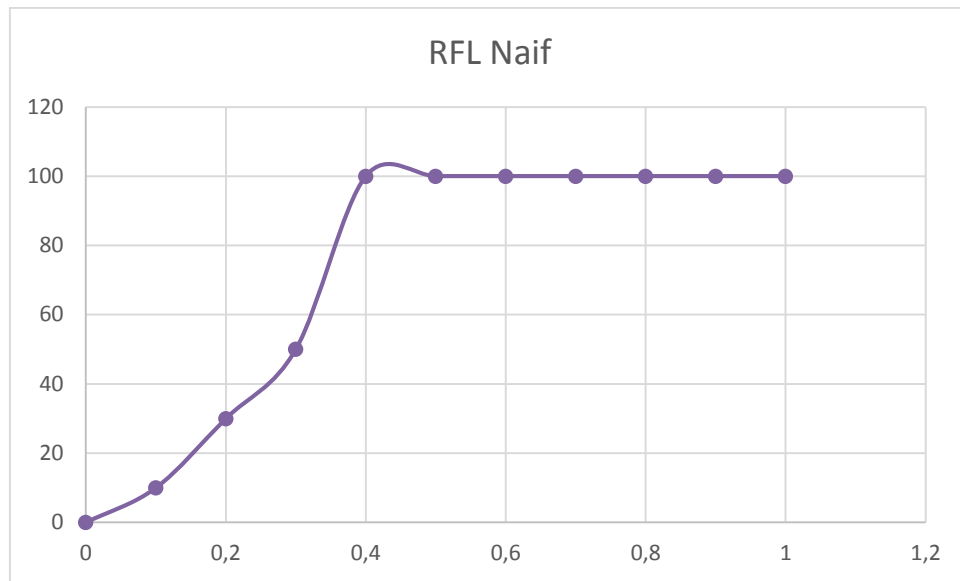
Le graphe suivant montre le nombre d'affectations effectué avec l'heuristique « *first fail* ».



Le nombre d'affectations est identique à ceux de l'heuristique du domaine le plus petit.

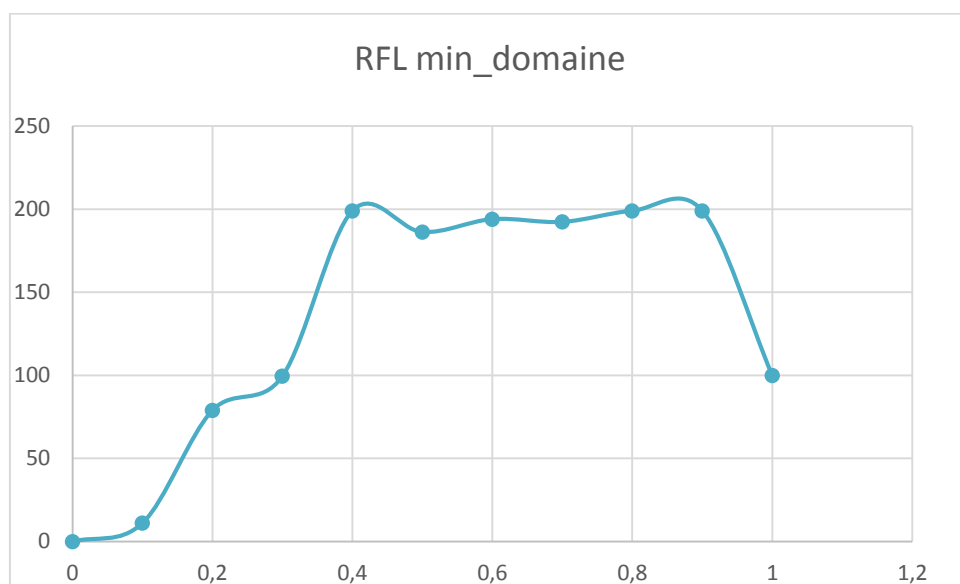
#### 4.6.2.RFL

Le graphe ci-dessous représente le nombre d'affectations effectués sur RFL sans heuristiques sur le coefficient de satisfiabilité, nous avons décidé de partir avec un graphe de 100 sommets, 10 arcs, un domaine de 5 valeurs.



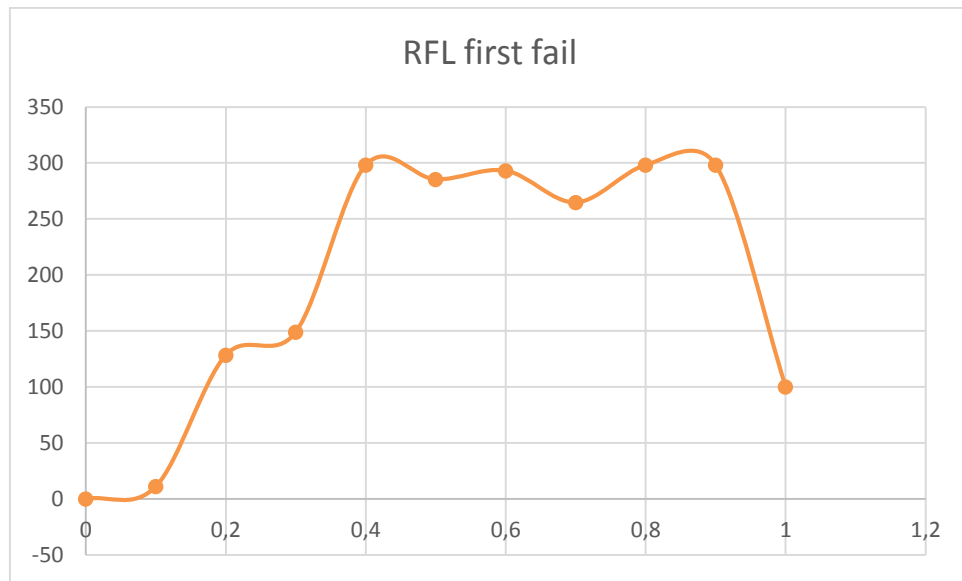
On remarque que l'algorithme RFL fait un nombre d'affectation compris entre le nombre de variable du graphe. On constate la différence entre le forward checking et RFL.

Le graphe suivant utilise est celui de RFL avec l'heuristique du domaine le plus petit.



Les résultats montrent que cette heuristique ne marche pas sur le RFL. Le nombre d'affectations obtenus sont supérieurs à ceux obtenu dans le RFL sans heuristique.

Le graphe suivant montre le nombre d'affectations effectuée avec l'heuristique « *first fail* »



Les résultats obtenus sont supérieurs à ceux de l'heuristique du domaine le plus petit. On peut en conclure que pour l'algorithme RFL, il est inutile d'utiliser des heuristiques sur le choix des variables.

# Conclusion

Ce projet nous a permis d'approfondir les connaissances que nous avons acquises lors des cours d'Intelligence Artificielle et de Complexité. Le travail en équipe a été bénéfique pour la réalisation du projet. Le temps que nous avons mis pour nous mettre d'accord sur des idées assimilées de manière différentes nous a donné un aperçu de l'organisation qu'il faudra pour réaliser des projets futurs au sein d'une équipe.

L'expérience de notre encadrant nous a beaucoup aidé pendant toute la durée du projet. Ses conseils, la documentation qu'il nous a fournis ainsi que les explications sur certains points lors des rendez-vous étaient précieux pour pouvoir coder les fonctions nécessaires.

Cette expérience a été un moyen de faire une découverte dans le monde de la recherche, et de pouvoir comparer avec les stages effectués durant notre cursus universitaire. Après réflexion, nous avons décidé de choisir un parcours professionnel après le Master 1 Informatique, vu l'effort et la patience requise dans le monde de la recherche.

# Bibliographie

<http://valgrind.org/>

<http://oprofile.sourceforge.net/>

<https://eclipse.org/linuxtools/projectPages/gprof/>

<http://liris.cnrs.fr/csolnon/Site-PPC/session1/e-miage-ppc-sess1.htm>

<http://www.cril.univ-artois.fr/~lecoutre/research/publications/2004/JNPC2004Art.pdf>