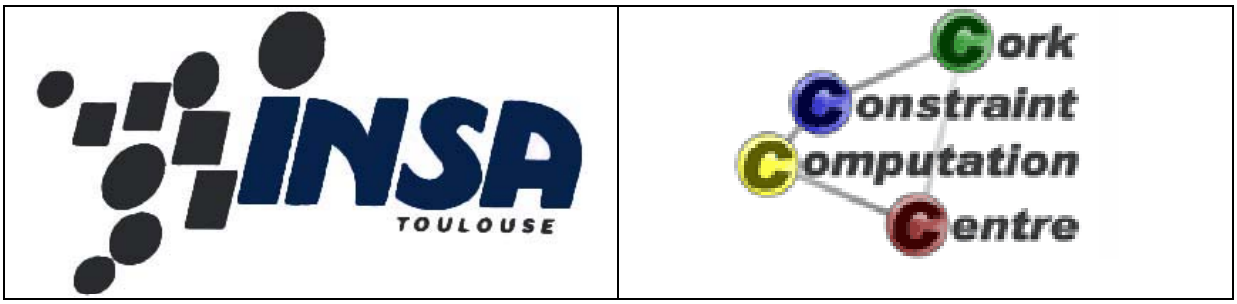
Rapport de stage
Fin de 4e année



Bailly-Salins Iver

Introduction

Le Cork Constraint Computation Center, ou 4C est un laboratoire rattaché au département de 'computer science' de l'université de Cork, Irlande. Ce laboratoire a été fondé lors de la fusion du laboratoire de recherche du professeur E. Freuder de l'université du New Hampshire avec le 'Département of Computer Science Constraint Processing Group' de l'université de Cork, dirigé par le professeur James Bowen. Une quarantaine de personnes y travaillent, en majorité des chercheurs, ainsi que quelques élèves non diplômés de l'université et quelques stagiaires venant d'horizons internationaux différents. En parallèle d'un travail de recherche dans le champ du traitement des problèmes de satisfaction de contraintes (CSP), le centre met également ses savoir-faire au service de l'industrie et poursuit une activité importante de promotion des méthodes de traitement de problèmes concrets par CSP.

J'avais avant de venir travailler 3 mois au 4C, déjà effectue un stage de 2 mois au LAAS dans le domaine des CSP sous la tutelle de M Lopez, M Esquirol et Mm Huguet. Ce premier contact avec le domaine des CSP s'est révèle être un atout important dans mon travail au 4C. J'avais au LAAS, en binôme avec Mehmet Celikbas, participé à la programmation d'un paquettage de fonctions utiles à la définition de CSP et à leur résolution par des algorithmes. Nous n'avions pas eut le temps d'approfondir les méthodes d'optimisation par ordonnancement dynamiques de variables et de valeurs, bien que leur utilité nous avait été signalée par nos tuteurs. C'est justement à l'étude de l'heuristique d'ordonnancement que s'est consacré mon travail à Cork, reprenant ainsi mon travail préliminaire du LAAS là où je l'avais laissé.

I/ Quelques notions théoriques sur les CSP.

J'introduis ici les notions théoriques utiles a la compréhension de la présentation de mon travail durant ce stage. C'est lors du stage de fin de 3e année au LAAS que j'ai acquis ces bases, que j'ai ensuite complétées par la lecture de quelque papier plus spécifiquement lié à mes intérêts dans le domaine du traitement des CSP.
Notez que si j'utilise la dénomination CSP, abréviation issue de l'anglais, c'est parce que c'est la dénomination généralement utilisée aussi bien dans les documentations francophone que dans les documentations anglophones auxquelles j'ai eus accès.

1/ Définitions

Un CSP ou 'Constraint satisfaction problem', est défini par la donnée :
d'un ensemble de variables.
d'un ensemble de domaines.
d'un ensemble de contraintes.

Les CSP traités comprennent habituellement un nombre fini de variables.
A chaque variable est associé un domaine. Le domaine d'une variable est l'ensemble de valeurs que peut prendre cette variable. Ces ensembles sont habituellement des énumérations de valeurs ordonnées ou non, ou des unions d'intervalles d'un ensemble ordonné dénombrable ou non, fini ou non, définies par la donnée des]borne inf , borne sup[de chaque intervalle (intervalles de l'axe des réels ou de l'ensemble des entiers par exemple). Nous nous restreindrons aux CSP dont les domaines sont des ensembles finis de valeurs que l'on énumère.

On appelle instanciation la décision d'attribuer à une variable une des valeurs de son domaine.

Une contrainte est une relation entre les valeurs de n variables du CSP. On dit qu'une contrainte est satisfaite si les variables impliquées dans cette contrainte sont instanciées et que la relation définie par la contrainte est vérifiée pour les valeurs auxquelles sont instanciées ces variables.
Une manière de définir une contrainte entre n variables est d'énumérer tous les n-uplets de valeurs qui constituent une instanciation des n variables en jeu satisfaisant ladite contrainte. Cela n'est pas généralement possible, mais convient au type de CSP avec lesquels j'ais travailler. Je choisirais donc cette représentation dans la suite du rapport.

Nous nous restreindrons aux contraintes binaires (n=2), qui établissent une relation entre les valeurs de 2 variables. Notons en passant que tout CSP défini a partir de contraintes n-aires peut être ramenée a un CSP binaire équivalent.
Une solution à un CSP est une instanciation de toutes les variables à des valeurs telles que toute les contraintes soit satisfaites.

2/ un exemple simple

Un exemple va clarifier ces concepts :
Soit les variables :

{colour, letter, number}

et les domaines associés :

colour : {red,green,blue,yellow}
letter : {A,D,G}
number : {1,2,3,5,7}

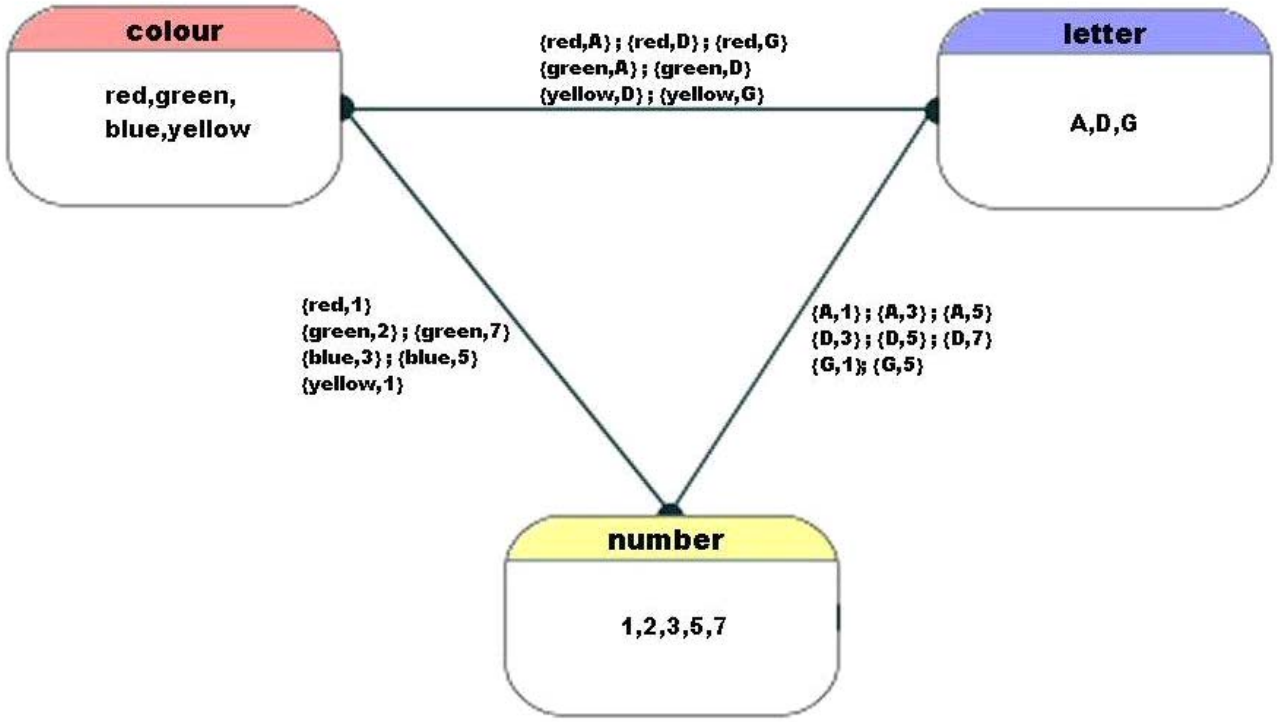
Définissons les contraintes :

C1 lie les variables 'colour' et 'letter'. La liste des couples de valeurs satisfaisant cette contrainte est:
{ {red,A} , {red,D} , {red,G} , {green,A} , {green,D} , {yellow,D} , {yellow,G} }

C2 lie les variables 'colour' et 'number'. La liste des couples de valeurs satisfaisant cette contrainte est:
{ {red,1} , {green,2} , {green,7} , {blue,3} , {blue,5} , {yellow,1} }

C3 lie les variables 'number' et 'letter'. La liste des couples de valeurs satisfaisant cette contrainte est:
{ {A,1} , {A,3} , {A,5} , {D,3} , {D,5} , {D,7} , {G,1} , {G,5} }

Ce problème peut être représenté sous forme d'un graphe :
Problème colour-letter-number :



Nous reviendrons plus en détail sur la représentation graphique des problèmes plus tard.

L'instanciation : {colour,red}, {number,1},{letter,A} est une solution valide à ce CSP car toute les contraintes de ce CSP y sont satisfaite et que toute les variables y sont instanciées.

3/ critères descriptifs

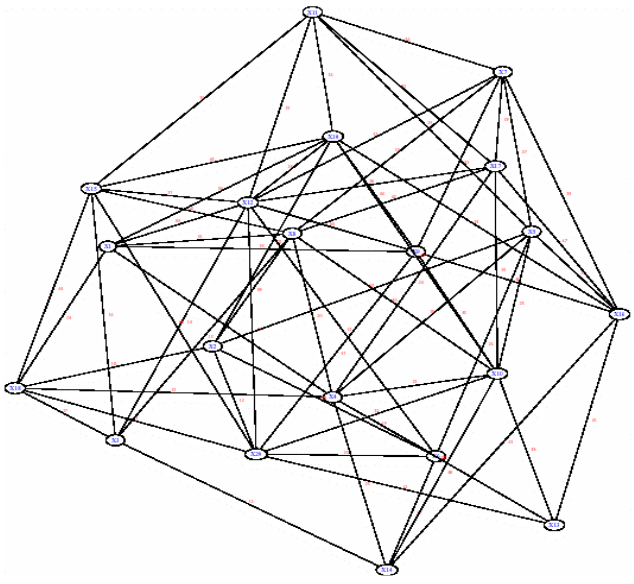
Il est évident que les CSP traites ne sont pas aussi simples que l'exemple cite au dessus. Il n'est presque jamais utile de décrire dans leurs moindres détails les CSP traités qui sont souvent complexes et traités en grand nombre. Pour décrire de manière satisfaisante un CSP plus complexe, nous faisons appel aux critères suivants :
- Nombre de variables.
- Taille des domaines, si ceux-ci sont de taille constante.
- Taille moyenne des domaines et profil de la répartition des tailles de domaines si ceux-ci sont de taille variable.
- Densité qui est définie par : (nombre de contraintes) / (nombre maximum de contraintes).
- Satisfiabilité des contraintes qui est définie par : (nombre de couples de valeurs satisfaisant la contrainte)/(nombre maximum de couples).
- Nombre de solutions (si celui ci est connu).

Notons que pour un CSP à taille de domaines constant et satisfiabilité constante, il existe une relation donnant l'espérance en probabilité du nombre de solution du CSP:

$$\text{NbSol} = (\text{Satisfiabilité}^{\text{nombre de contraintes}}) \cdot (\text{Taille domaines}^{\text{nombre de variables}})$$

4/ Représentation graphique et vocabulaire associé

Les CSP sont souvent représentés sou forme de graphes dont les sommets sont des variables et les arcs sont des contraintes.
Ci-dessous est représenté un problème typique de ceux traités lors de mes recherches.
Ses caractéristiques appréciables sur le graphe sont les suivantes :
20 Variables
Densité de contraintes: 0.3



Nous utiliserons souvent les notions de degré statique, degré avant et degré arrière d'une variable.
Le degré statique d'une variable est le nombre d'arcs dont une des extrémités abouti a cette variable dans le graphe correspondant au problème initial, le degré statique d'une variable est constant tout au long de la résolution. Le sommet le plus bas dans le graph représenté a ici un degré statique égal a 5.
Le degré avant d'une variable est le nombre d'arcs dont une extrémité abouti a cette variable et l'autre extrémité abouti a une variable non instanciée quelconque, ce degré évolue au fur et a mesure de la

résolution. Apres instanciation de deux de ses variables adjacentes, le sommet le plus bas du graphe aura un degré avant de 3.
Le degré arrière d'une variable est le nombre d'arcs dont une extrémité abouti a cette variable et l'autre extrémité abouti a une variable instanciée quelconque, ce degré évolue au fur et a mesure de la résolution. Apres instanciation de deux de ses variables adjacentes, le sommet le plus bas du graphe aura un degré arrière de 2.

De manière générale et tout au long de la résolution nous avons la relation :
degré-avant + degré-arriere = degré statique.

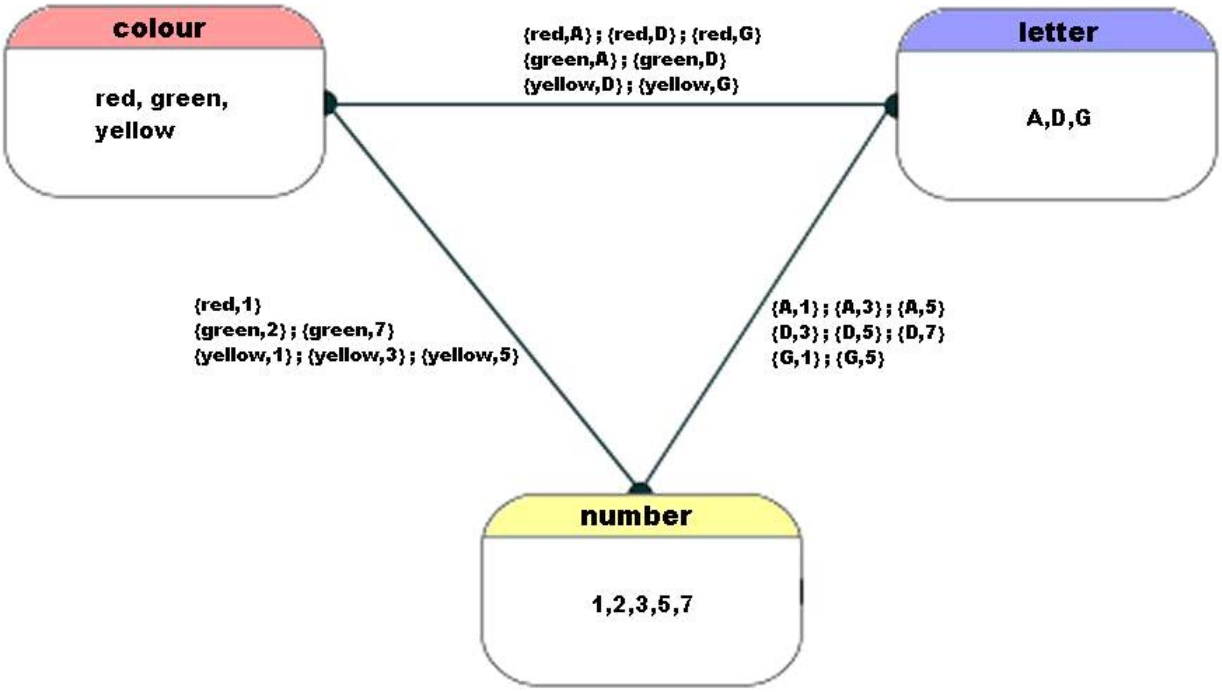
Deux variables sont dites adjacentes dans le graphe si il existe une contrainte qui les lie.

5/ Consistances

Les consistances sont des caractéristiques des CSP, il existe différents types de consistances définies pour des CSP.

Consistance d'arc
Nous appelons un problème arc-consistant quand toute ses contraintes liant 2 variables sont elles même arc-consistantes.
Nous appelons une contrainte liant 2 variables X et Y arc-consistante quand la décision d'instancier X (respectivement Y) à une des valeurs de son domaine ne rend pas impossible l'instanciation de Y (resp X) à une des valeurs de son domaine qui satisfasse la contrainte liant X et Y.
Le problème colour-letter-number exposé plus haut n'est par exemple pas arc-consistant :
La décision d'instancier 'colour' à 'blue' rend impossible l'instanciation de 'letter' à une valeur de son domaine satisfaisant la contrainte C1.
La contrainte C3 n'esp pas arc-consistante non plus : la valeur 2 n'a aucune lettre à laquelle être associée.
Il n'y a pas d'implications valides entre arc-consistance d'un problème et existence d'une solution. Le problème cité plus haut n'est pas arc-consistant mais admet une solution, certains problèmes arc-consistants peuvent ne pas admettre de solution.

Consistance de chemin
Nous appelons un problème chemin-consistant quand quelque soit la triplette de variables prise dans ce problème, cette triplette est chemin-consistante.
Pour qu'une triplette de variable soit chemin-consistante, elles doivent vérifier la propriété suivante :
Quelque soit les 2 variables prisent parmi les 3, la contrainte existant entre ces 2 variables est plus difficilement satisfiable que la combinaison des contraintes qui forment un chemin entre ces 2 variable en passant par la troisième variable. Pour 3 variables X, Y, Z c'est-à-dire qu'il n'existe pas d'instanciation {X=Val1 , Y=Val2} qui satisfasse la contrainte entre X et Y mais qui ne satisfasse pas les contraintes entre X et Z d'une part et entre Z et Y d'autre part. (idem pour toute les permutation possible entre X,Y et Z dans la phrase précédente).
Ce problème est arc-consistant mais n'est pas chemin-consistant :



En effet la contrainte colour-letter nous autorise l'instanciation {'colour'='red' , 'letter'='D'}
Mais le choix de 'red' et la contrainte colour-number nous forcent à choisir la valeur '1' pour 'number'. Le choix de '1' pour 'number' et la contrainte 'number-letter' nous interdissent de prendre la valeur 'D' pour la variable 'letter'. La chaine de contrainte de 'colour' à 'letter' en passant par 'number' est donc plus contraignante que la contrainte directe colour-letter.
Si une telle situation n'arrive jamais quelque soit les 3 variables et les valeurs prisent dans un CSP, alors ce CSP est chemin-consistant.
Un problème chemin-consistant est arc-consistant. Un problème peut être chemin-consistant et ne pas admettre de solution, tout comme il peut ne pas être chemin-consistant et admettre des solutions.

Atteindre la consistance
A partir d'un problème donné qui est inconsistant (arc ou chemin) il est possible d'obtenir un problème consistant en retirant certaines valeurs des domaines des variables. Ces valeurs génératrices d'inconsistances ne font partie d'aucune solution au problème. Il est donc souvent avantageux de chercher à rendre un problème consistant avant de chercher à la résoudre, de manière à retirer un ensemble de valeurs inutiles avant le début de la résolution. Il existe des algorithmes spécialisés dans ce travail.
Nous reparlerons des consistances dans les méthodes de résolution des CSP.

6/ Méthodes de résolution

Il existe des algorithmes permettant la résolution efficace et rapide des CSP. Ces algorithmes reposent souvent sur l'instanciation successive des variables, le filtrage de domaines, la propagation de contraintes après instanciation, et le retour en arrière si les instanciations successives nous entraînent dans une situation ou il est prouvé qu'on ne trouvera pas de solution en continuant l'instanciation des variables non instanciées restantes.
Plusieurs aspects de l'algorithme ont une influence sur la performance de la résolution :

La manière dont est choisie la prochaine variable à instancier
La manière dont est choisie la valeur à laquelle instancier cette variable
La propagation des conséquences de cette instanciation dans la suite de la résolution
(Cette propagation définie la rapidité avec laquelle on détectera l'impossibilité de trouver une solution en continuant d'instancier les variables libres restantes).
La manière dont on revient sur ses pas dans les instanciations successives lorsque une telle impossibilité a été montrée.

C'est l'étape de propagation qui différencie les principaux algorithmes de résolution de CSPs.
L'algorithme le plus basique de recherche de solution est le backtracking, pour lequel aucune propagation n'est effectuée.
Voila un algorithme de backtracking en pseudo-code, constitue d'une procédure appelée récursivement :

6.1 Algorithmes

Le backtrack

On note :
VNI la liste des variables non instanciées du problème.
VI la liste des variables instanciées du problème.
Les variables sont repérées sous forme d'entiers, VNI et VI sont des listes de ces entiers.
VNI(i) ou VI(i) l'accès au i^e élément de ces listes.
D la liste des domaines des variables du problème.
D(i) l'accès au i^e élément de cette liste. D(V(i)) est le domaine de la variable V(i).
D(i) est une liste de valeurs.
Les valeurs sont repérées sous forme d'entiers, dont D(i) est une liste.
D(i)(j) l'accès au je élément de cette liste.
|L| le nombre d 'éléments d'une liste L
Vérifier_contrainte(Var1, Val1, Var2, Val2) la procédure qui vérifie si la contrainte entre les variables Var1 et Var2 est vérifiée par l'instanciation {Var1=Val1 , Var2 = Val2 }.

Algorithme du Backtrack.

```
Backtrack(D,V,{},C)
Procédure Backtrack (D, VNI, VI,C)
        Var = VNI(1)
        Pour IndVal = 1.. |D(Var)|
                Val = D(Var)(IndVal)
                Cont_sont_verifiées = vrai
                IndVar2 = 1 ;
                Tant-que : IndVar2 (< ou =) |VI| et Cont_sont_verifiées
                        Var2 = VI(IndVar2)
                        Cont_sont_verifiées = Vérifier_contrainte(Var, Val, Var2,D(Var2))
                        IndVar2 = IndVar2+1
                Fin Tant-que
                Si Cont_sont_verifiées
                        NouvD = D
                        NouvD(Var) = {Val}
                        NouvVNI = VNI (2..|VNI|)
                        NouvVI = {VI, Var }
                        Si |NouvVNI| ~= 0
                                Trouv_sol = Backtrack(NouvD, NouvVNI, NouvVI,C)
                        Sinon
                                Trouv_sol =vrai ;
                        Fin Si
                Fin Si
                Si Trouv_sol
                        Retourne Vrai
                Fin Si
        Fin Pour IndVal
        Retourne Faux
```

Pour documentation, une implémentation MATLAB du Backtrack est fournie en annexe. Cette implémentation a été réalisée dans le but de clarifier cet algorithme, mon travail au 4C n'a jamais inclus l'utilisation de MATLAB, qui est un langage très mal adapté au type de problèmes traités.

On peur résumer le principe de l'algorithme du backtrack écrit ci dessus de cette manière :
L'algorithme fait appel récursivement à une procédure Backtrack.
Cette procédure choisit une variable Var parmi les variable non encore instanciées.
Puis on choisi une valeur Val dans le domaine de cette variable.
On regarde si l'instanciation de Var à Val satisfait les contraintes existant entre Var et ses variables adjacentes déjà instanciées. On parcours les valeurs du domaine de Var jusqu'à trouver une valeur satisfaisant les contraintes existant entre Var et ses variables adjacentes déjà instanciées. Si aucune des valeurs de Var n'est satisfaisante, alors il va être nécessaire de revenir sur la dernière instanciation effectuée dans les appels récursif de la procédure Backtrack.
Si une valeur Val du domaine de Var satisfait les contraintes entre Var et les variables déjà instanciées, alors Var est instanciée à la valeur Val, et la procédure Backtrack est appelée de nouveau jusqu'à avoir instancié toute les variables et trouvé une solution, ou jusqu'à avoir testé toutes les combinaisons d'instanciation possibles et n'en avoir trouvé aucune satisfaisant les contraintes.
Il est important de noter que les choix successifs des variables et valeurs ne sont ici dictés que par un classement arbitraire de ces objets dans des listes. Or nous verrons plus tard que l'ordre de choix des variables et valeurs joue un rôle capital dans la rapidité avec laquelle une solution est trouvée. Nous y reviendrons plus tard dans le rapport.

6.2 Les Algorithmes avec propagation

Nous avons vu que l'algorithme du backtrack vérifie lors de l'instanciation de la variable Var que les contraintes qui lient la variable à instancier Var et les variables déjà instanciées ne sont pas brisées. Une fois une valeur satisfaisante trouvée et l'instanciation réalisée, aucune conséquence de cette instanciation n'est directement observée. L'instanciation effectuée n'aura de conséquences que plus tard dans la recherche d'une solution, lors de l'instanciation d'une variable Var2 adjacente à Var, au moment où il faudra vérifier que la valeur choisie pour Var2 est compatible avec l'instanciation déjà réalisée de Var.
Le principe de 'propagation des instanciations' sur les contraintes consiste, lors de l'instanciation de Var, à retirer certaines des valeurs des domaines des variables adjacentes à Var, de manière à atteindre une consistance (plus ou moins forte) du graphe constitué de Var et de ses variables adjacentes.

L'algorithme FC (forward checking)

Dans cet algorithme l'instanciation se fait de la manière suivante :
On fait le choix de la variable Var à instancier, puis le choix de la valeur Val d'instanciation dans le domaine de cette variable. Aucune observation n'est effectuée pour vérifier que l'instanciation {Var=Val} en cours est en accord avec les instanciation déjà réalisées précédemment, cela est connu d'avance. En revanche, après réduction du domaine de la variable Var à l'unique valeur Val, les domaines des variables adjacentes à Var sont filtrés de manière à en retirer toutes les valeurs qui ne supportent pas cette instanciation. On effectue donc une recherche de consistance d'arc sur toute les contraintes liant Var à une de ses variables adjacentes (après instanciation de Var, c'est-à-dire que le domaine de Var est {Val} lors de cette recherche de consistance).
On définit le support d'une valeur Val du domaine de Var dans le domaine d'une variable Var2, l'ensemble des valeurs du domaine de Var2 qui supportent l'instanciation de Var à Val.
L'algorithme FC consiste, lors de l'instanciation {Var=Val} à retirer des domaines des variables adjacentes à Var, les valeurs qui ne font pas partis du support de Val.
Le filtrage des domaines pose quelques problèmes lors du retour en arrière sur une instanciation qui a mené à un cul-de-sac dans l'arbre de recherche, en effet, il est nécessaire lors de l'abandon de l'instanciation {Var=Val} de revenir à l'état des domaines adjacents à Var tel qu'il était avant l'instanciation, c'est-à-dire être capable de faire l'opération inverse du filtrage des domaines. En fait il est nécessaire de garder en mémoire l'état des domaines de l'ensemble des variables à chaque étape de la recherche de manière à pouvoir revenir en arrière sur les instanciations et filtrages de domaines réalisés précédemment.

Un exemple simple de résolution par FC :
Soit le problème suivant :
3 variables A,B,C, au domaines respectifs D_A = {1,2,3} , D_B = {1,2} , D_C = {1,2,3}.

Les contraintes sont :
C1 lie les variables {A,B} : {{1,2},{2,1},{3,2}}
C2 lie les variables {B,C} : {{1,1},{2,2}}
C3 lie les variables {A,C} : {{1,1},{1,2},{2,2},{3,1}}

L'algorithme FC recherche une solution :

Instanciation {A=1}
    D_B = {2}
    D_C = {1,2}
    Instanciation {C=1}
        D_B = {}
        Retour sur la dernière instanciation
        D_B = {2}

    Instanciation {C=2}

D_B = {2}
Instanciation {B=2}
Solution trouvée

L'algorithme MAC (Maintained Arc Consistency)

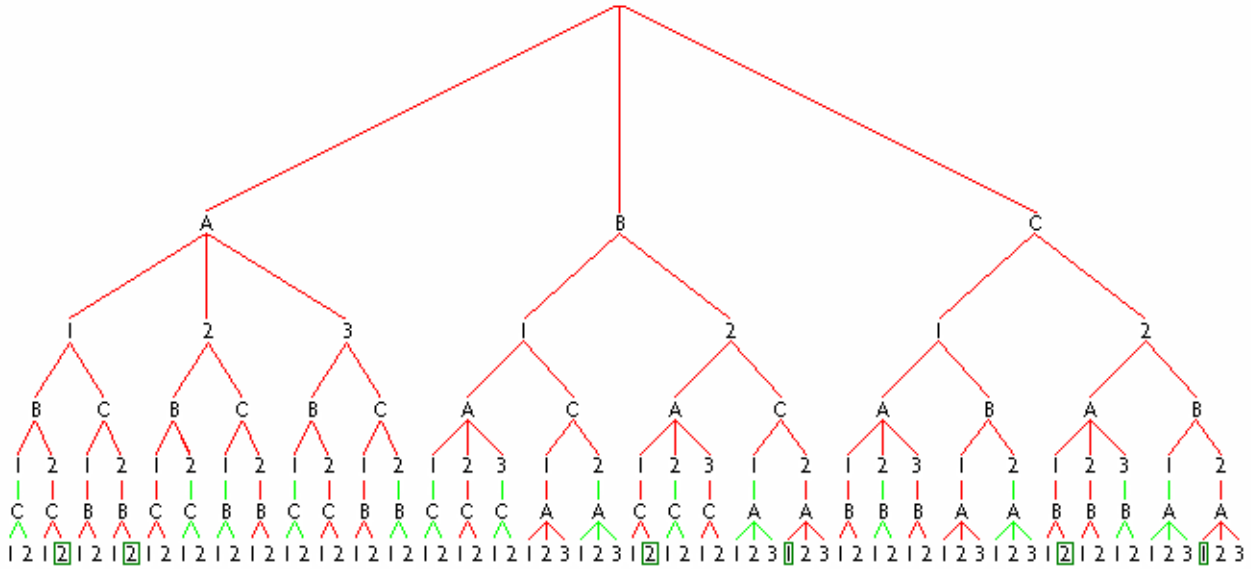
Cet algorithme repose sur le même principe que l'algorithme FC, mais utilise une consistance plus forte que ce dernier.
Après l'instanciation de la variable Var à la valeur Val, le problème résultant (c'est-à-dire le même graphe que celui précédant l'instanciation avec la modification : domaine(Var) = {Val}) est rendu Arc consistant.
Alors que pour FC, seules les contraintes liant Var à une variable adjacente étaient rendues arc-consistantes, pour MAC, le problème en entier est rendu arc-consistant.
Pour rendre plus claire cette différence, exposons là dans un exemple :

On instancie Var1 à la valeur Val1, cette instanciation rend impossible l'instanciation {Var2=Val2}, les 2 algorithmes FC et MAC vont retirer la valeur Val2 du domaine de Var2. Ce que ne va pas faire FC et que va faire MAC, c'est prendre en compte cette modification du domaine de Var2 pour voir si il n'est pas possible de retirer de nouvelles valeurs au domaine de Var3 variable adjacente à Var2, et si tel est le cas, répercuter cette nouvelle modification du domaine de Var3 à ses variables adjacentes, et ainsi de suite jusqu'à obtenir la consistance d'arc sur l'ensemble du problème après instanciation. FC atteint la consistance d'arc uniquement sur les contraintes dans lesquelles intervient la variable en cours d'instanciation, alors que MAC recherche cette consistance sur l'ensemble du problème avant de passer à la prochaine instanciation.

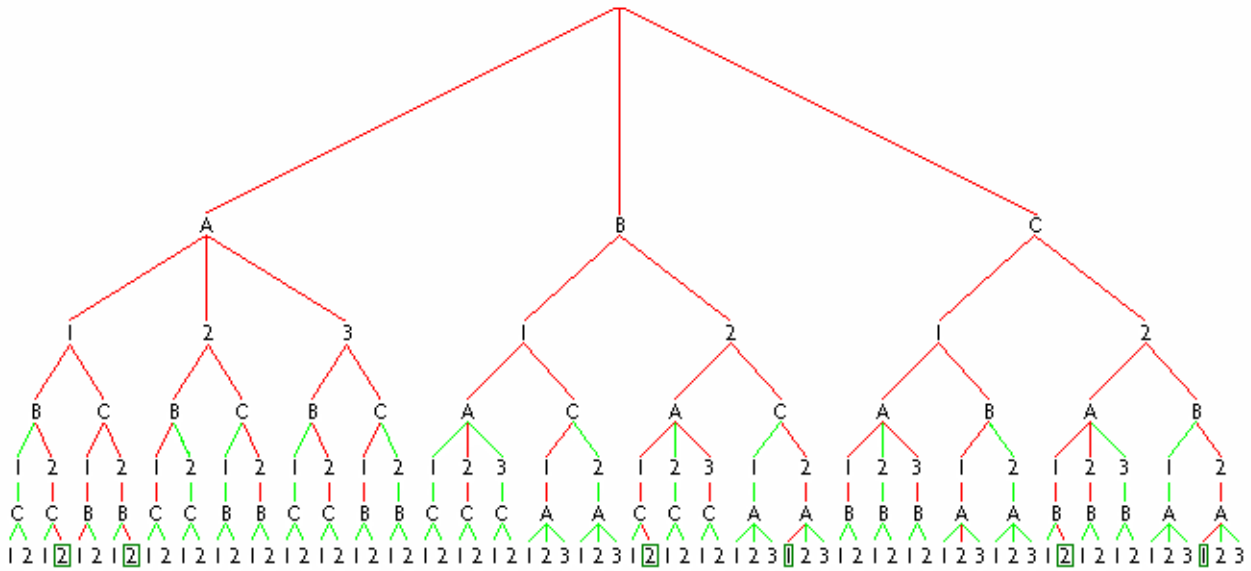
Différences de performance entre algorithmes

La recherche de consistance e un coût, et il n'est pas toujours plus avantageux de faire cette recherche lors de la recherche d'une solution. Cela dépend des caractéristiques du CSP traité. En règle général, il est souvent plus intéressant de ne pas rechercher de consistance dans les CSP très faciles, c'est-à-dire comportant une grande quantité de solutions, puisque sur ces problèmes la recherche de consistance ne retire que peu de valeurs des domaines, et peu de retours sur instanciations sont nécessaires avant de trouver une solution. En revanche pour les problèmes difficiles, très contraints et comportant peu de solutions, il est très intéressant de détecter le plus tôt possible un mauvaise instanciation par recherche de consistance, cela évide un grand nombre de recherches et retours sur instanciation (ce nombre est d'autant plus grand que la mauvaise instanciation est repérée haute dans l'arbre de recherche).

Mettons en relief les différences d'exploration d'arbre de recherche pour les algorithmes backtrack et FC, pour un petit problème a 3 variables A, B, C dont les contraintes s'écrivent :
C1 lie les variables {A,B} : {{1,2},{2,1},{3,2}}
C2 lie les variables {B,C} : {{1,1},{2,2}}
C3 lie les variables {A,C} : {{1,1},{1,2},{2,2},{3,1}}



L'arbre de recherche entier est représenté ci dessus. Chaque noeud correspond au choix d'une variable puis d'une valeur pour établir une instanciation. Les branches vertes sont celles que l'algorithme du backtrack n'explore pas (car elles interviennent en aval d'un noeud qui rompe une des 3 contraintes).
La même représentation de l'arbre de recherche pour l'algorithme FC donne :



L'algorithme MAC aurait réduit directement les domaines à l'unique solution. Plus la recherche de consistance est profonde, moins il est nécessaire de réaliser de retours sur instanciations, car les branches infructueuses de l'arbre de recherche sont détectées et éliminées plus tôt.
Néanmoins la recherche de consistance a elle aussi un coût, et il n'est pas garanti de trouver une solution plus rapidement en utilisant un algorithme qui vérifie la consistance plus profondément. Le choix de la

profondeur a laquelle effectuer la recherche de consistance dépend des caractéristiques du problème (plus le problème est jugé facilement solvable moins il est nécessaire de rechercher une consistance profonde).
La caractérisation ‘facile’ ou ‘difficile’ d’un problème n’est pas subjective, il existe des relations entre les paramètres de densité, de nombre de variables, de taille des domaines et de satisfiabilité des contraintes qui situent un problème traité par rapport à une zone dite ‘de transition de phase’ autour de laquelle les problèmes sont dits difficiles. Cette transition de phase localise la transition entre les problèmes facilement solvables car comportant une forte proportion de solutions et les problèmes fortement contraints pour lesquels il est facilement démontrable qu’aucune solution n’existe. Certains aspects de ce problème sont développés dans le papier ‘An empirical study of phase transitions in binary constraint satisfaction problems’, par Patrick Prosser, et ‘Locating the phase transition in binary constraint satisfaction problems’ par B. M. Smith et M. E. Dyer.

7/ Politique et heuristique.

On distingue plusieurs niveaux d’identification d’objectifs utilises pour résoudre les CSP de manière plus performante, que l’on appelle politiques et heuristiques.
Une politique identifie les buts ou résultats finaux désirables. Une heuristique identifie les caractéristiques d’une situation présente permettant de faire les choix qui augmentent la probabilité d’atteindre ces buts ou ces résultats finaux. Cette distinction est présente dans de nombreux domaines de recherche, comme le montre cette traduction d’un article de biologie du comportement :
Une politique prescrit les règles générales pour maximiser le gain, sans toutefois identifier de procédure spécifique qui permettrait à l’animal de suivre la règle. Une politique pour la maximisation de la population d’une espèce pourrait être : ‘traverser les routes en évitant de se faire écraser’, une procédure pour remplir cette politique serait ‘regarder dans les 2 directions avant de traverser’.

Politique d’échec rapide, heuristiques de choix de variables et de valeurs.

Il est important de noter que le choix des variables et des valeurs lors des instanciations successives a des conséquences majeures sur la difficulté de résolution d'un problème. On évalue cette difficulté de résolution au nombre total de contraintes qu'il a fallu observer avant de trouver une solution. Une partie essentielle de mon travail au 4C a été l’étude des heuristiques appliquées lors du choix de ces variables et valeurs. Il est important d'introduire les notions de promesse et d’échec rapide, qui sont les critères que l'on cherche à maximiser lors des choix respectif des valeurs et des variables lors de la résolution.
Observons la résolution d'un problème par FC, avec des choix successifs de variables et de valeurs différentes sur l’arbre dessine ci-dessus. Le choix successifs des variables et valeurs lors de la résolution est dicte dans les algorithmes présentés précédemment par le classement des indices de variables et valeurs respectivement dans les listes VNI et D(i), i=1..nombre_de_variables.

Il est évident une fois la résolution terminée que certaines séquences sont meilleurs que d'autre, mais est il possible avant la résolution et au cours de la résolution de distinguer les meilleurs choix de variables et de valeurs ?

La réponse à cette question est oui. Les classement judicieux effectues de manière dynamiques des variables et des valeurs au cours d’une résolution permettent de diviser de manière considérable les efforts nécessaires a l’obtention d’une solution. Je présente plus tard quelques uns des résultats obtenus pour différentes heuristiques de classement de variables et de valeurs. Essayons dans un premier temps de comprendre les mécanismes qui font que l’ordonnancement de variables et de valeurs est si important.

Politique d’échec rapide :

Les choix précédents un instant donne de la résolution du CSP nous ont conduit a une situation où un certain nombre de variables sont instanciées (Variables contenues dans VI), et un certain nombre de variable restent a être instanciées (variables contenues dans VNI). Pour limiter les efforts de résolution il est capital de savoir le plus rapidement possible (ou au moindre coût) si il est possible de compléter les instanciations déjà effectuées par une instanciation des variables de VNI qui satisfasse toute les contraintes. Dans le cas ou cela

ne serait pas possible, cela voudrait dire que parmi toute les solutions au problème d'origine, aucune ne contient la sous instanciation des variables de VI aux valeurs auxquelles elles sont couramment instanciées. Dans le but de repérer l'impossibilité de compléter instanciation partielle courante en une solution générale le plus rapidement possible, on applique le principe d'échec rapide aux choix successifs des variables de VNI a instancier. Cela signifie que l'on choisi la variable parmi VNI que l'on estime a le plus de chances de révéler une impossibilité de compléter la situation courante à une solution de notre CSP d'origine. On désire recouvrer de notre potentielle erreur sur les instanciations déjà effectuée le plus rapidement possible. C'est cette politique que l'on appelle la notion d'échec rapide.
Le choix parmi les VNI de la variable que l'on estime a le plus de chances de révéler impossibilité de trouver une solution dans le sous arbre de recherche dans lequel nous a amené les instanciations successives des variables de VI se fait en considérant les paramètres suivants :

La taille des domaines des variables de VNI
Le degré de ces variables dans le graphe du problème initial
Le degré de ces variables dans le sous graphe des variables instanciées (degré arrière)
Le degré de ces variables dans le sous graphe des variables non instanciées (degré avant)

Politique de promesse :
Le mot 'promesse' fait a la fois référence à une politique de résolution et a une mesure. La politique de promesse est simplement de faire les choix qui sont les plus prometteurs, c'est à dire qui ont la plus grande chance d'aboutir a une solution.
La mesure de la promesse d'une méthode de résolution est égale à la probabilité d'aboutir directement à une solution du problème, en appliquant cette méthode de résolution. Aboutir directement à une solution signifie que l'instanciation successive des variables amène a une solution sans qu'il ne soit jamais nécessaire de revenir sur une des instanciations effectuées.
Pour un problème sans solution, la mesure de la promesse est nulle quelque soit la méthode de résolution appliquée.
Pour un problème où toute instanciation possible des variables est une solution, la mesure de la promesse est 1 quelque soit la méthode de résolution appliquée.
La promesse d'une méthode de résolution dépend de l'ordonnancement des variables et des valeurs, elle dépend également de l'algorithme utilise. (plus on regarde en profondeur les conséquence d'une instanciation avant de la faire, plus la promesse à une chance d'être élevée).
La promesse n'est pas généralement équivalente à la densité de solutions. (elle est équivalente pour un algorithme sans recherche de consistance, c'est a dire un algorithme qui instancie les variables une par une sans réduction de domaines ni observation de contraintes avant de constater si oui ou non l'instanciation complète effectuée constitue une solution)
Des expériences ont été effectuées pour mesurer la promesse sur des problèmes différents et pour des heuristiques de choix de variables différentes pour l'algorithme FC. Les résultats sont publies dans le papier 'Toward understanding variable ordering heuristics for CSPs' (Christopher Beck, Patrick Prosser, Richard Wallace).

On pourrait croire que les politiques de promesse et d'échec rapide sont contradictoires. Ces deux politiques sont liées et ne sont pas contradictoire. Le fait de chercher à rebrousser chemin le plus rapidement possible si les instanciations précédentes ne peuvent pas aboutir a une solution diminue le nombre de retours sur instanciations (backtracks), car les retours sur instanciations sont effectués plus tôt dans l'arbre de recherche. La notion de promesse correspond a l'effort effectué pour augmenter ses chances de trouver une solution sans prendre en compte l'utilité du retour sur instanciation. Il a été observe que les heuristiques de choix de variable utilisées pour satisfaire la politique d'échec rapide peuvent être les mêmes que celles utilisées pour satisfaire la politique de promesse. Cela est dû au fait qu'une bonne heuristique de choix de variable pour la politique d'échec rapide est une qui a pour effet de diminuer le plus possible la taille du sous-arbre de recherche résultant de l'instanciation de la variable choisie. Parallèlement une bonne heuristique pour la politique de promesse est une qui a pour effet d'augmenter le plus possible la densité estimée de solutions

dans le sous-arbre de recherche résultant de l'instanciation de la variable choisie. Or il n'est pas contradictoire d'avoir des heuristiques de choix de variables qui tout en diminuant rapidement la taille du sous arbre de recherche résultant d'une instanciation, augmente également la densité de solution dans ce sous arbre.

Une fois la prochaine variable à instancier choisie, les algorithmes utilisés essayent successivement toutes les valeurs du domaine de cette variable avant de revenir en arrière sur une instanciation précédente. L'ordre dans lequel on choisi les valeurs n'a donc pas d'importance dans le cas ou les instanciations précédentes ne peuvent pas conduire a une solution, puisqu'il faudra les parcourir toute quoi qu'il arrive. L'ordonnancement des valeurs n'a d'effet que lorsqu'on est dans une branche de l'arbre de recherche qui peut aboutir à une solution. Dans tous les cas on choisit donc de prendre les valeurs qui sont les plus prometteuses lors de la recherche d'une solution, et la politique d'échec rapide ne s'applique pas au choix des valeurs.

Différentes heuristiques basiques
Le nom des heuristiques est donné sous leur forme abrégée, la même forme que celle utilisée dans les feuilles de présentation des résultats.

Pour le choix de valeurs :
Le terme 'promesse' n'est ici plus utilisé pour définir la politique de promesse mais désigne la mesure de la promesse.
Promesse : la promesse d'une valeur Val du domaine de la variable Var est estimée par :

$$\prod_{Var2 \in A} |\text{support}(Val, Var2)|$$

Où A est l'ensemble des variables adjacentes à Var, et support(Val, Var2) le support de l'instanciation {Var=Val} dans le domaine de Var2.
La promesse de la valeur Val du domaine de Var est donc estimée par le produit des tailles des supports de la valeur Val dans les domaines des variables adjacentes à Var.
Rappelons ce que nous avons écrit précédemment :
On définit le support d'une valeur Val du domaine de Var dans le domaine d'une variable Var2, l'ensemble des valeurs Val2 du domaine de Var2 qui supportent l'instanciation de Var à Val (ie : la contrainte entre Var et Var2 permet l'instanciation {Var=Val, Var2=Val2}).
En fait, plus la promesse d'une valeur est grande, moins l'instanciation à cette valeur est contraignante sur les futurs instanciations des variables adjacentes.
L'heuristique de choix de valeur correspondant à cette mesure, est celle qui consiste à choisir dans un domaine la valeur qui à la plus haute mesure de promesse.
C'est la seule heuristique utilisée pour le choix des valeurs.

Pour le choix de variables :
Un modèle probabiliste a été établi pour mesurer la probabilité que le choix d'une variable nous conduise à un échec immédiatement après instanciation et propagation des contraintes. Les heuristiques de choix de variables reposent sur ce modèle.

Exposons brièvement la philosophie de la chose :
La variable que nous allons instancier va nous révéler directement que nous sommes dans une branche infructueuse de l'arbre de recherche si quelque soit la valeur à laquelle nous instanciions cette variable, le fait d'instancier la variable choisie a cette valeur révèle une inconsistance (vide le domaine d'une autre variable par propagation des contraintes). Nous désirons estimer la probabilité que l'instanciation d'une variable non instanciée nous révèle directement que nous sommes dans une branche infructueuse de l'arbre

de recherche. Cette estimation doit se faire à un coût moindre que celui d'instancier effectivement et de vérifier.

Notons :

$$P\left[\text{instantiation de Vi échoue}\right]$$

La probabilité que l'instanciation de la variable Vi non encore instanciée nous révèle directement que nous sommes dans une branche infructueuse de l'arbre de recherche. 'Directement' signifie ici qu'il ne sera pas nécessaire de réaliser d'autres instanciations pour montrer cela.

Estimons cette probabilité :

Nous allons utiliser la compacité des contraintes qui lient Vi à ses variables adjacentes Vj.
La compacité de la contrainte liant la variable Vi a la variable Vj est la proportion de couples de valeurs (Va,Vb) appartenant aux domaines courants de Vi et Vj respectivement et qui ne satisfont pas la contrainte Cij. La compacité d'une contrainte est une caractéristique liée à la satisfiabilité déjà définie précédemment (compacité=1-satisfiabilité). Nous appelons la compacité de la contrainte Cij pij.

$$p_{ij} = \frac{\sum_{Va \in Domaine(Vi)} \sum_{Vb \in Domaine(Vj)} \{Va,Vb\} \notin C_{ij}}{\sum_{Va \in Domaine(Vi)} \sum_{Vb \in Domaine(Vj)} \{Va,Vb\}}$$

L'instanciation de la variable Vi à la valeur particulière Va révélera un échec si pour au moins une variable Vj adjacente à Vi, aucune valeurs Vb du domaine de Vj ne satisfait la contrainte Cij.

$$\exists \text{ Vj variable adjacente à Vi tq } \forall \text{ Vb} \in \text{ Domaine(Vj)}, \{\text{Va,Vb}\} \notin C_{ij}$$

Nous avons dis que la probabilité qu'un couple (va,vb) ne satisfasse pas la contrainte Cij est pij (la compacité).
Il y a mj valeurs dans le domaine de Vj.

$$\left|\text{Domaine(Vj))}\right| = m_j$$

La probabilité que l'instanciation {Vi = Va} se révèle être un échec direct lors de l'observation de la contrainte Cij est donc :

$$p_{ij}^{\ m_j}$$

(Aucune des mj valeurs du domaine de Vj ne satisfait la contrainte Cij).
La probabilité que l'instanciation {Vi = Va} ne se révèle pas être un échec direct lors de l'observation de la contrainte Cij est donc :

$$1 - p_{ij}^{\ m_j}$$

La probabilité que l'instanciation (Vi = Va) ne se révèle pas être un échec direct quelque soit la variable adjacente Vj considérée est donc :

$$\prod_{\text{Vj Adjacente à Vi}} 1 - p_{ij}^{\ m_j}$$

La probabilité que l'instanciation {Vi = Va} se révèle être un échec directement lors de l'observation d'une des contraintes quelconque Cij est donc :

$$1 - \prod_{\text{Vj Adjacente à Vi}} 1 - p_{ij}^{\ m_j}$$

Nous cherchons la probabilité que quelque soit la valeur va prise dans le domaine de Vi, l'instanciation

{Vi = Va} se révèle être un échec directement. En effet si quelque soit la valeur va prise dans le domaine de Vi cette valeur se révèle être un échec direct, on pourra dire que le choix d'instancier Vi nous a révélé directement que nous étions dans une branche infructueuse de l'arbre de recherche.
La probabilité que quelque soit la valeur Va prise dans le domaine de Vi, l'instanciation {Vi = va)} se révèle être un échec directement est :

$$\left(1 - \prod_{\text{Vj Adjacente à Vi}} 1 - {p_{ij}}^{m_j}\right)^{m_i}$$

C'est la probabilité que nous cherchions.

$$\mathrm{P}\left[\text{instantiation de Vi échoue}\right] = \left(1 - \prod_{\text{Vj Adjacente à Vi}} 1 - {p_{ij}}^{m_j}\right)^{m_i}$$

Les 4 heuristiques de Smith et Grant qui découlent de ces calculs simples sont (FF signifie ici Fail-First) : FF (que nous appelons également min-domain car elle revient à prendre la variable au domaine courant le plus petit), FF2, FF3 et FF4.
Nous n'avons utilise que FF et FF4 lors de nos études.

D'autres heuristiques de choix de variables ayant pour but de satisfaire la politique échec rapide sont basées sur l'utilisation des degrés des variables dans la représentation graphique du CSP.
Explications :
Max-fwd-degree : choix dynamique de la variable au degré-avant le plus grand.
Max-bkw-degree : choix dynamique de la variable au degré-arriere le plus grand.
Max-static-degree : choix statique de la variable au degré le plus grand dans le graphe du problème initial.

Une heuristique mixte et, on le verra, très efficace est :

Min domain over degree : choix dynamique de la variable dont la caractéristique $\frac{\left(\text{taille du domaine}\right)}{\left(\text{degré}\right)}$

est le minimum.

II/ Travail expérimental.

Mon travail expérimental au 4C s'est effectue sous la tutelle de R.Wallace. Rick m'a fourni son code écrit en LISP contenant les fonctions de générations de problèmes, de résolution par les algorithmes FC et MAC. Ce code comprenait l'implémentation de l'ordonnancement de variables et de valeur pour les heuristiques décrites plus haut. Mon travail dans un premier temps s'est résumé à lire et comprendre le code dans ses aspects généraux. N'ayant jamais eut de contact avec le langage LISP, cette première approche s'est révélée longue et fastidieuse.
LISP est un des plus anciens langages de programmation encore utilisé de nos jours. Créer dans les années 50, il est base sur le traitement de listes (LISP signifie LISt Processing). Ce langage a essentiellement trouvé son utilité dans le domaine de l'intelligence artificielle, même si le langage LISP a évoluer dans de nombreuses directions, et l'on parle plus désormais de famille de langages LISP. Il se révèle très performant pour traiter les CSP, les algorithmes existants dans ce domaine étant bien adaptés pour être utilisé sur des structures de données reposant essentiellement sur des listes.
Le code existant sur lequel j'ai travaille mesurait plusieurs milliers de lignes de code, et une dizaine de jours me furent nécessaire avant de pouvoir l'éditer et ajouter y des modestes fonctions utiles aux recherches qui me furent déléguées.
Les tests des programmes étaient effectués sur des serveurs dédiés aux expériences.

Ci-dessous sont présentés un échantillon représentatif des recherches expérimentales que j'ais effectué.

Problèmes utilisés

De manière à étudier les caractéristiques des différentes heuristiques, il a été nécessaire de générer des ensembles de problèmes sur lesquels appliquer les algorithmes de résolution.
Nous nous sommes généralement placés dans la zone de densité et satisfiabilité des problèmes qui garantissait des problèmes difficiles à résoudre, mais solvables néanmoins. C'est-à-dire que l'espérance du nombre de solution était :
$\text{NbSol} = (\text{Satisfiabilité}^{\text{nombre de contraintes}}) \cdot (\text{Taille domaines}^{\text{nombre de variables}})$ proche de 1.
Cette zone était la plus intéressante pour nous car elle implique un résolution longue, donc où les heuristiques appliquées ont le temps de prouver leur efficacité au cours de la résolution.

Nous présentons ici des résultats sur la résolution de 3 ensembles de 500 problèmes :
. 500 problèmes à 20 variables, chacune ayant une taille de domaine de 10, avec une densité de contrainte 0.3, et une satisfiabilité des contraintes de 0.52.

. 500 problèmes à 20 variables, chacune ayant une taille de domaine variant entre 1 et 20 et de moyenne 10, avec une densité de contrainte 0.3, et une satisfiabilité des contraintes de 0.52.

. 500 problèmes à 20 variables, chacune ayant une taille de domaine de 10, avec une densité de contrainte 0.7, et une satisfiabilité des contraintes de 0.72.

Performances de résolution

Ci-dessous est présenté un tableau indiquant le nombre de contraintes observées au cours de la résolution avant l'obtention d'une solution. Cette quantité est révélatrice de la difficulté éprouvée à résoudre le problème. Nous appelons ce paramètre fc-check-count. Nous mettons également dans la 3^{e} colonne l'écart type de ce paramètre sur un ensemble de 100 problèmes que nous appelons ici std (standard deviation).

Pour 100 problèmes à 20 variables, chacune ayant une taille de domaine de 10, avec une densité de contrainte 0.3, et une satisfiabilité des contraintes de 0.52. Observons les performances de résolution des différentes heuristiques et leurs anti-heuristiques correspondantes :

Heuristique de choix de variable	fc-check-count	std
Max-fwd-degree	47059	51248
Min-fwd-degree	2422793	3837037
Max-bkw-degree	33309	32924
Min-bkw-degree	2169846	2279622
Maxdom	3027055	3751804
Mindom	10927	7524

Nous voyons bien ici que l'application d'une heuristique simple peut diminuer le temps de recherche par 100.

Evolution de caractéristiques des problèmes

Il est également possible de changer dynamiquement d'heuristique de choix de variable. C'est vers cette direction de recherche que s'orientait mon travail avec R. Wallace. De manière à faire un choix judicieux de l'heuristique à un moment de la résolution donnée, il est nécessaire d'être capable de dégager certaines caractéristiques de l'évolution de la morphologie du problème pertinentes. Une partie du travail effectué constituait à ajouter au code existant les fonctions enregistrant les évolutions au cours de la résolution des caractéristiques des variables choisies et du problème en général telles que :
. Le nombre de composantes connexes du graphe courant (cad le graphe du problème original auquel on a retiré les variables déjà instanciées) et la connectivité moyenne de ces composantes.
. Les degrés (statique, avant et arrière) des variables choisies
. La taille des domaines des variables choisies
Nous évaluons ces paramètres au cours de la résolution et nous les regroupons dans des tableaux pour obtenir leurs valeurs moyennes dans chaque portion de l'arbre de recherche que l'on parcourt.
Ainsi, à chaque instanciation d'une variable, ces caractéristiques sont relevées et reportées à des moyennes calculées. On dissocie plusieurs niveaux dans l'arbre de recherche, par exemple pour un problème à 20 variables que l'on souhaite diviser en 4 niveaux, le niveau de l'instanciation d'une variable est égal au nombre de variables couramment instanciées au moment de la nouvelle instanciation divisé par 20, multiplié par 4 et ajouté à 1. On fait des allés retours dans ces niveaux de l'arbre de recherche au cours de la recherche de la solution au fur et à mesure des instanciations et des retours sur instanciations.

Nous obtenons des résultats variant d'une heuristique de résolution à l'autre, par exemple :
Pour les 500 problèmes à 20 variables, chacune ayant une taille de domaine de 10, avec une densité de contrainte 0.7, et une satisfiabilité des contraintes de 0.72.

Résolution avec l'heuristique mindom :

	Taille domaines	Degre avant	Degre arrière	Degre statique	Nb composantes connexes	Connectivite moyenne
Niveau 1	1.37	9.32	5.02	14.35	1.00	6.77
Niveau 2	1.61	10.39	4.05	14.45	1.00	7.57
Niveau 3	1.09	8.10	6.13	14.24	1.00	5.86
Niveau 4	1.10	2.48	11.00	13.48	1.05	1.90

Résolution avec l'heuristique max-fwd-deg :

	Taille domaines	Degre avant	Degre arrière	Degre statique	Nb composantes connexes	Connectivite moyenne
Niveau 1	2.71	9.97	4.59	14.56	1.00	5.85
Niveau 2	3.08	10.85	3.95	14.81	1.00	6.44
Niveau 3	1.93	8.17	5.90	14.07	1.00	4.65
Niveau 4	1.12	1.08	11.36	12.44	2.24	0.43

Même tableaux pour l'ensemble de problèmes :
500 problèmes à 20 variables, chacune ayant une taille de domaine variant entre 1 et 20 et de moyenne 10, avec une densité de contrainte 0.3, et une satisfiabilité des contraintes de 0.52.

Résolution avec l'heuristique mindom :

	Taille domaines	Degre avant	Degre arrière	Degre statique	Nb composantes connexes	Connectivite moyenne
Niveau 1	1.43	4.77	2.93	7.70	1.03	2.42
Niveau 2	1.65	5.45	2.40	7.86	1.00	2.76
Niveau 3	1.18	4.20	3.42	7.62	1.03	2.12
Niveau 4	1.27	1.39	4.91	6.30	1.44	0.84

Résolution avec l'heuristique max-fwd-deg :

	Taille domaines	Degre avant	Degre arrière	Degre statique	Nb composantes connexes	Connectivite moyenne
Niveau 1	3.69	5.75	2.02	7.76	1.15	1.71
Niveau 2	4.04	6.17	1.75	7.92	1.04	1.86
Niveau 3	2.15	3.96	3.13	7.09	1.57	1.05
Niveau 4	1.17	0.03	5.74	5.77	3.48	0.01

Ce genre de données observées pour de nombreuses heuristiques et sur des ensembles de problèmes aux caractéristiques différentes nous a permis de mieux discerner les mécanisme de la résolution de CSP.

Nous nous sommes alors intéressés à des techniques de résolutions mélangeant 2 heuristiques, suivant le niveau de l'instanciation dans l'arbre de recherche. Je ne mets pas tous les résultats dans ce rapport.

Corrélations

Les difficultés de résolutions étant acquises pour la résolution des échantillons de 500 problèmes avec différentes heuristiques et différents mélanges d'heuristiques, nous nous sommes intéressés aux corrélations entre chaque vecteur de 500 valeurs correspondant aux difficultés de résolution d'une heuristique sur un échantillon de problèmes. Ceci de manière à voir si les heuristiques différentes butaient sur les mêmes problèmes à l'intérieur des échantillons de 500 problèmes. Par exemple pour l'ensemble de problèmes aux caractéristiques :
. 500 problèmes à 20 variables, chacune ayant une taille de domaine de 10, avec une densité de contrainte 0.7, et une satisfiabilité des contraintes de 0.72.
Et pour des mélanges d'heuristiques différents nous obtenons ce type de résultats :

FWDDEG ABOVE 20 % (fwddeg=maxfwddeg)

	fwddeg below	**mindom below**	**mindomovdeg below**	**random below**
fwddeg below	1,0000			
mindom below	0,9285	1,0000		
mindomovdeg below	0,9282	0,9997	1,0000	
random below	0,9457	0,9867	0,9863	1,0000

FWDDEG ABOVE 20 %

fwddeg below	281337
mindom below	115745
mindomovdeg below	114127
random below	308206

FWDDEG ABOVE 15 % (fwddeg=maxfwddeg)

	fwddeg below	**mindom below**	**mindomovdeg below**	**random below**
fwddeg below	1,0000			
mindom below	0,8988	1,0000		
mindomovdeg below	0,8990	0,9988	1,0000	
random below	0,9082	0,9874	0,9865	1,0000

FWDDEG ABOVE 15 %

fwddeg below	281337
mindom below	72351
mindomovdeg below	69757
random below	358688

FWDDEG ABOVE 10 % (fwddeg=maxfwddeg)

	fwddeg below	**mindom below**	**mindomovdeg below**	**random below**
fwddeg below	1,0000			
mindom below	0,8721	1,0000		
mindomovdeg below	0,8801	0,9957	1,0000	
random below	0,8642	0,9828	0,9795	1,0000

FWDDEG ABOVE 10 %

fwddeg below	281337
mindom below	54378
mindomovdeg below	50213
random below	444261

FWDDEG ABOVE 5 % (fwddeg=maxfwddeg)

	fwddeg below	**mindom below**	**mindomovdeg below**	**random below**
fwddeg below	1,0000			
mindom below	0,8292	1,0000		
mindomovdeg below	0,8561	0,9766	1,0000	
random below	0,7945	0,9382	0,9370	1,0000

FWDDEG ABOVE 5 %

fwddeg below	281337
mindom below	51953
mindomovdeg below	44727
random below	591009

MINDOM ABOVE 20 %

	Mindom below	**fwddeg below**	**min domovdeg below**	**random below**
Mindom below	1,0000			
fwddeg below	0,9476	1,0000		
min domovdeg below	0,9992	0,9466	1,0000	
random below	0,9920	0,9458	0,9911	1,0000

MINDOM ABOVE 20 %

Mindom below	67385
fwddeg below	204741
min domovdeg below	65225
random below	294675

MINDOM ABOVE 15 %	Mindom below	fwddeg below	min domovdeg below	random below
Mindom below	1,0000			
fwddeg below	0,9251	1,0000		
min domovdeg below	0,9981	0,9264	1,0000	
random below	0,9876	0,9073	0,9868	1,0000

MINDOM ABOVE 15 %	
Mindom below	67385
fwddeg below	265894
min domovdeg below	63650
random below	445576

MINDOM ABOVE 10 %	Mindom below	fwddeg below	min domovdeg below	random below
Mindom below	1,0000			
fwddeg below	0,8960	1,0000		
min domovdeg below	0,9947	0,9039	1,0000	
random below	0,9740	0,8587	0,9740	1,0000

MINDOM ABOVE 10 %	
Mindom below	67385
fwddeg below	316651
min domovdeg below	61188
random below	632038

MINDOM ABOVE 5 %	Mindom below	fwddeg below	min domovdeg below	random below
Mindom below	1,0000			
fwddeg below	0,8336	1,0000		
min domovdeg below	0,9794	0,8662	1,0000	
random below	0,9375	0,7925	0,9388	1,0000

MINDOM ABOVE 5 %	
Mindom below	67385
fwddeg below	344820
min domovdeg below	57530
random below	808677

MINFWDDEG ABOVE 20 %	Minfwddeg below	mindom below	min domovdeg below	random below
Minfwddeg below	1,0000			
mindom below	0,8440	1,0000		
min domovdeg below	0,8430	0,9997	1,0000	
random below	0,8661	0,9865	0,9855	1,0000
fwddeg below	0,7462	0,9008	0,9017	0,8978

MINFWDDEG ABOVE 20 %	
Minfwddeg below	2104805
mindom below	398218
min domovdeg below	389698
random below	1956858
fwddeg below	1568715

MINFWDDEG ABOVE 15 %	Minfwddeg below	mindom below	min domovdeg below	random below
Minfwddeg below	1,0000			
mindom below	0,8002	1,0000		
min domovdeg below	0,8006	0,9991	1,0000	
random below	0,8421	0,9833	0,9834	1,0000
fwddeg below	0,6903	0,8748	0,8763	0,8711

MINFWDDEG ABOVE 15 %	
Minfwddeg below	2104805
mindom below	231413
min domovdeg below	222215
random below	1708669
fwddeg below	1189925

MINFWDDEG ABOVE 10 %	Minfwddeg below	mindom below	min domovdeg below	random below
Minfwddeg below	1,0000			
mindom below	0,7374	1,0000		
min domovdeg below	0,7389	0,9976	1,0000	
random below	0,8098	0,9735	0,9742	1,0000
fwddeg below	0,6014	0,8440	0,8505	0,8250

MINFWDDEG ABOVE 10 %	
Minfwddeg below	2104805
mindom below	139936
min domovdeg below	129867
random below	1390869
fwddeg below	794584

MINFWDDEG ABOVE 5 %				
	Minfwddeg below	mindom below	min domovdeg below	random below
Minfwddeg below	1,0000			
mindom below	0,6807	1,0000		
min domovdeg below	0,6697	0,9845	1,0000	
random below	0,7473	0,9374	0,9432	1,0000
fwddeg below	0,5301	0,8397	0,8589	0,8250

MINFWDDEG ABOVE 5 %	
Minfwddeg below	2104805
mindom below	86930
min domovdeg below	75830
random below	1064880
fwddeg below	484765

Conclusion

Aucune conclusion définitive n'est ressortie de mon travail expérimental au 4C, qui s'inscrivait plus dans un travail visant à mieux comprendre certains mécanismes de résolution des CSP. Certaines réflexions sont ressorties de la mise en parallèle de ces expériences avec des travaux déjà existants sur le sujet.
L'importance du choix des variables pour satisfaire les politiques décrites plus haut a été montrée, mais la relation entre heuristiques et satisfaction des politiques a due être nuancée et rapportée à d'autres facteurs entrants en compte dans la résolution et modifiant ces relations tels que des caractéristiques du problème traiter.

III/ Travail auxiliaire.

En parallèle du travail que me donnait R. Wallace qui s'apparentait à de la recherche, j'ai participé avec le groupe d'étudiants irlandais et étrangers au développement du site Internet du laboratoire. Le projet de E. Freuder est de 'populariser' le travail du laboratoire, et de manière générale de promouvoir les techniques de résolution de CSP aux yeux du grand public, et des élèves de lycée plus particulièrement. Un des moyen à mettre en œuvre pour atteindre ce but était de mettre en ligne, lié au site officiel du laboratoire, une série de jeux . Pour démontrer les principes et l'utilité des techniques de résolution de CSP, ces jeux étaient sensés proposer, en plus d'un mode de jeux 'normal', sans assistance, une aide issue d'algorithmes des techniques de CSP, et quelques information facilement compréhensibles sur la manière dont est 'calculée' cette aide.
Nous avons choisi entre les étudiants liés au projet, de programmer chacun un jeu sous forme d'applet JAVA. Le premier travail fut donc de trouver des jeux qui permettraient d'illustrer les différents aspects des techniques de CSP. Le travail en groupe permettait de tester entre nous les jeux et de les améliorer en prenant en compte les réflexions des autres étudiants. Le fait d'avoir chacun un jeu a développer nous incitait a travailler pour améliorer nos jeux, dans un esprit de compétition amicale.
Le jeu que j'ai choisi de développer est celui de MasterMind. Dans ce jeu le joueur doit trouver une combinaison de couleurs qui lui est cachée par un autre joueur (ici l'ordinateur). Le joueur offre des propositions de combinaisons et l'ordinateur lui renvoi pour chaque proposition, la réponse aux 2 questions suivante :
Combien de couleurs dans la combinaison proposée se retrouve dans la solution à des positions différentes dans la séquence proposée et dans la séquence solution ?
Combien de couleurs dans la combinaison proposée se retrouve dans la solution à la même position dans la séquence proposée et dans la séquence solution ?
Par exemple si la solution cachée est (rouge vert bleu jaune) et la solution proposée (rouge rose jaune bleu), le joueur recevra l'évaluation a cette proposition sous la forme : 1 couleur bien placée et 2 mal placées. Quelles couleurs sont bien placées et mal placées reste une inconnue.
Parmi les différentes versions du jeu, j'ai choisi celle qui propose 8 couleurs possibles pour remplir chaque composante d'une proposition, avec l'information supplémentaire que la même couleur n'est pas répétée 2 fois dans la solution cherchée (1680 possibilités de solution). La version la plus répandue propose 6 couleurs et permet à la solution recherchée de contenir des répétitions de couleurs (1296 possibilités).

Ce jeu m'as semblé adapté à l'exposition des techniques et des concepts des CSPs pour les raisons suivantes :
Le jeu est relativement connu.
Les concepts de variables, domaines, contraintes peuvent être introduits assez naturellement :
Chaque évaluation peut être considérée comme une contrainte que la solution recherchée doit satisfaire.
Chacune des 4 positions dans la séquence des propositions peut être considérée comme une variable, dont le domaine est l'ensemble des couleurs disponibles pour remplir ces positions.
Les algorithmes de filtrage des domaines peuvent alors être appliques au fur et a mesure que des contraintes additionnelles sont ajoutées après chaque proposition.
En plus de ces techniques propres aux CSP, quelques techniques propres au MasterMind peuvent être ajoutés à l'assistance par ordinateur. Nous exposerons ces techniques plus bas.
Quelque applets JAVA de ce jeu existaient déjà sur Internet, mais je n'en ai trouvé aucune qui présente le degré de développement que présente maintenant l'applet issue de notre travail. Ben Schneiderman, spécialiste en interface homme machine de l'université de Maryland nous a aidé lors de son passage au laboratoire à rendre nos jeux plaisant et stimulant, en utilisant quelques principes simples :
Des graphismes attractifs, stimulant l'imagination mais ne nuisant pas a la lisibilité et a la clarté du jeu.
Plusieurs niveaux d'aides d'utilité croissante, invitant le joueur à trouver le niveau d'aide qui correspond a la fois au degré d'éducation qu'il souhaite retirer du jeu et au degré de défi qu'il souhaite relever.
L'invitation à découvrir seul les techniques les mieux adaptées a la résolution du problème, et à les proposer aux concepteurs du jeu.
Suivant ces principes mon applet propose un menu qui ouvre l'accès à 2 types d'information :
Les aides à la résolution du problème
Un court guide explicatif sur la manière dont sont obtenues ces aides.
Etant réaliste sur la proportion d'utilisateurs ouvrant effectivement et délibérément le guide explicatif, il fut nécessaire d'introduire dans les pages d'obtention d'aide des éléments dévoilant quelques aspects les techniques de résolution de manière a inciter le joueur a se documenter dans la partie explicative du programme dans un premier temps puis dans des pages web spécialisées dans un second temps.
Les aides se divisent en 4 niveaux offrant une aide de plus en plus importante jusqu'à donner la proposition optimale.

L'applet du jeu de MasterMind ainsi que d'autres applets programmées par des étudiants du laboratoire seront bientôt accessibles à partir du site internet du 4C :

http://4c.ucc.ie/web/index.jsp

IV/ Aspects généraux du stage

Je suis arrive a Cork le 17 Juin pour y rester jusqu'au 19 Septembre. Plusieurs stagiaires étrangers travaillaient déjà dans les locaux du 4C. Ces stagiaires ainsi que les stagiaires de l'université de Cork qui eux travaillent a temps partiel dans le laboratoire durant toute l'année, rendirent mon intégration rapide et efficace. La langue ne fut une barrière en aucun cas, étant donne le fait que j'ai habite en Angleterre pendant mon enfance et que je maîtrise l'anglais de manière satisfaisante.

L'organisation du travail des stagiaires au sein du laboratoire est divise en 2 aspects parallèles :
A chaque stagiaire est assigne un travail de recherche en collaboration avec un chercheur du laboratoire, et l'ensemble des stagiaires constituent un groupe de travail auquel est confie le développement du site web du laboratoire. Le degré de confiance que les responsables du laboratoire exprimaient implicitement vis a vis des stagiaires fut une grande surprise pour moi. Le fait que les professeurs veuillent bien volontiers abandonner le charabia technique pour réellement discuter en termes clairs de mes idées, et me donner le temps et la confiance nécessaire pour formuler ces idées était une sensation qui m'était étrangère, et même déroutante aux premiers abords. Je n'étais évidement pas spécialiste dans les matières étudiée dans le laboratoire, mais cela n'altérait en rien le respect que l'on pouvait avoir pour mon travail. Il est évident que

ce respect va de paire avec une certaine responsabilité que les étudiants stagiaires doivent accepter. Bien que très peu de pression soit explicitement mis sur leurs épaules, on attend d'eux un travail sérieux, pour lequel ils sont bien payés.
Le laboratoire en général dispose de fonds importants, venant en grande partie d'investissements irlandais dans quelques domaines particuliers de la recherche. Cela se traduit par la fourniture de matériel informatique récent et performant. Chaque étudiant et chercheur a sa disposition un PC récent (2.4 Ghz), et un accès sur 2 machines dédiées aux expériences.
Le laboratoire est installe au centre ville, dans un ancien bâtiment industriel, il est compose de 2 étages sur lesquels se repartissent une dizaine de chercheurs et thésards, une dizaine étudiants et quelques personnels administratifs. Les bureaux sont délimités par des cloisons amovibles. L'atmosphère générale est plutôt relâchée, et le moindre papier publié est l'occasion de sortir du bâtiment, de traverser la rue et de partager une pinte de bière au pub. L'accès au bâtiment se fait à n'importe quelle heure du jour ou de la nuit, par l'intermédiaire de cartes magnétique et du code de l'alarme. Le laboratoire met également a la disposition des chercheurs et étudiants frigo, micro ondes, machine a café, et une salle de conférence, qui sert parfois de salle de projection de films.
Tout cela contribue au fait que le travail des étudiants est peu productif, mais se poursuit pendant des journées et des nuits très longues, et finalement tout le monde est satisfait de la manière dont le laboratoire est organisé.
Les chercheurs et thésards exposent régulièrement leurs travaux à l'ensemble du personnel. Ces exposés ainsi que les exposés tout aussi réguliers de chercheurs extérieurs au 4C entretiennent une dynamique qui ravive régulièrement l'intérêt de chacun dans son propre travail. Ces exposés, accompagnés du travail de documentation que j'ai effectué, et de la simple absorption de connaissances aux contacts de multiples discussions sur des sujets variés, ont réellement enrichie mes connaissances dans le domaine des CSP. Le fait que le laboratoire soit constitué d'une multitude de personnes venant d'horizons très différents (Irlande, Etats Unis, Tunisie, Espagne, France, Thaïlande, et je dois en oublier) est aussi une incitation a la curiosité et a l'échange d'idées. Un simple parcours des origines des chercheurs et visiteurs du 4C sur la page web qui leur est consacrée donne un aperçu assez nette de la diversité des personnalité qu'on peut rencontrer dans le laboratoire.
La ville de Cork en elle même n'est pas très attrayante mais très touristique et très pluvieuse.