Projet CSP – Génération de benchmark et évaluation de méthode

L'objectif de ce projet est de comprendre la méthodologie d'expérimentation et d'évaluation d'algorithmes de résolution de problèmes. Le travail demandé comporte 3 parties :

1. Se construire un jeu d'essai pertinent permettant une réelle évaluation des performances des algorithmes de résolution.
2. Évaluer sur ce jeu les performances d'un algorithme de résolution.
3. Éventuellement, proposer quelques modifications de l'algorithme par défaut et évaluer comparativement les bénéfices/pertes de ces modifications.

Vous rendrez au plus tard le 14/11 à minuit sur le devoir Moodle prévu pour ce projet une archive au format zip contenant :

- un **rapport correctement rédigé au format PDF** détaillant très clairement le travail réalisé et donnant tous les éléments permettant de refaire vos évaluations (paramètres de génération des réseaux, les exécutions à faire…)
- l'ensemble de vos **sources Java** (de ce projet et du TP Modélisation)
- les **scripts bash de génération** des jeux d'essais (NE PAS RENDRE LES JEUX D'ESSAIS qui peuvent prendre beaucoup de place).

Ce travail peut être fait en binôme mais chaque membre du binôme pourra être interrogé individuellement sur le travail réalisé !

A. Matériel fourni

Vous trouverez sur Moodle :

- un fichier `bench.txt` contenant dans un format texte ad-hoc un jeu d'essai de 3 réseaux de contraintes binaires en extension ;
- une classe java `Expe.java` contenant une méthode de lecture de tels fichiers texte de réseaux de contraintes ;
- un programme C `urbcsp.c` implémentant un générateur aléatoire de réseaux de contraintes binaires en extension.

Le générateur urbcsp va vous permettre de tester vos algorithmes sur des instances plus ou moins difficiles de réseaux de contraintes binaires. C'est un programme en C adapté du uniform random binary CSP generator (by D. Frost, C. Bessiere, R. Dechter, and J.C. Régin). Tous les domaines sont de même taille et toutes les contraintes ont le même nombre de tuples.

Pour l'utiliser, il vous faudra d'abord compiler le source : `gcc -o urbcsp urbcsp.c`
Puis lancer la génération de réseaux en spécifiant les 5 paramètres de génération et en ajoutant une redirection vers un fichier : `./urbcsp nbVariables tailleDomaine nbConstraints nbTuples nbRes > fic.txt`

Exemple : `./urbcsp 10 15 10 30 3 > bench.txt` est la commande qui a permis de générer le fichier `bench.txt` de 3 réseaux de 10 variables et 10 contraintes. Il y a 15 valeurs dans le domaine de chaque variable et 30 tuples dans chaque contrainte.

Le fichier java Expe.java doit être importer dans votre projet. Il contient une méthode `lireReseau` qui lit un réseau au format de sortie de urbcsp et le charge comme un Model Choco. Un exemple de `main` lisant les 3 réseaux du fichier `bench.txt` et affichant les réseaux lus est également fourni. Attention le fichier `bench.txt` doit être mis à la racine de votre projet Java `tp-ia-choco`.

Travail à faire :

1. *Récupérer les différents fichiers*
2. *Importer le fichier `Expe.java` dans votre projet*
3. *Mettre le fichiers `bench.txt` à la racine de votre projet*
4. *Exécuter le programme `Expe.java` et observer l'affichage produit*
5. *Compiler le programme `urbcsp.c`*

6. *Générer un fichier `benchSatisf.txt` de 3 réseaux de 10 variables et 10 contraintes. Les domaines ont 15 valeurs et le nombre de tuples par contraintes est de 80.*
7. *Générer un fichier `benchInsat.txt` de 3 réseaux de 10 variables et 15 contraintes. Les domaines ont 15 valeurs et le nombre de tuples par contraintes est de 20.*
8. *Modifier le programme `Expe.java` pour qu'il calcule pour chacun des 2 nouveaux jeux d'essai le nombre de réseaux qui ont une solution. Qu'observez-vous ?*

B. Construction d'un jeu d'essais conséquent et identification de la transition de phase

L'objectif de cette partie est de se doter d'un jeu d'essais suffisamment gros pour « voir quelque chose » mais pas trop gros pour ne pas avoir à attendre trop longtemps la résolution. On s'attend à ce que vous fixiez un nombre de variables, un nombre de contraintes (cela fixera la densité du réseau), et une taille de domaines. Puis que vous fassiez varier le nombre de tuples pour balayer la zone des réseaux très satisfiables aux réseaux très insatisfiables. Pour chaque dureté retenu, vous générerez au moins 10 réseaux (mais si possible plus) et calculerez le % de réseaux ayant au moins une solution.

Il vous faudra donc lancer la génération de plusieurs fichiers de dureté croissante. Vous pourrez vous aider d'un script `bash` du type :

```
for i in 10 30 50 80 110
do
     ./urbcsp 10 15 10 $i 3 > csp$i.txt
done
```

Par ailleurs, il vous faudra modifier le programme `Expe` pour qu'il puisse calculer pour chaque niveau de dureté le % de réseaux ayant au moins une solution. Vous tracerez alors la courbe % de réseau ayant au moins une solution en fonction de la dureté. Le mieux est sans doute que votre programme `Expe` génére un fichier `CSV` *dureté;%* que vous exploiterez avec Libre Office pour dessiner la courbe (cf. annexe). Votre objectif ici est de mettre en évidence une transition de phase.

Travail à faire :

1. *Modifier son programme `Expe.java` pour qu'il puisse calculer la fonction % de réseau ayant au moins une solution pour un benchmark dont les paramètres sont spécifiés.*
2. *Réaliser un script de construction d'un benchmark*
3. *Procéder à plusieurs essais pour identifier des paramètres intéressants de génération permettant de bien mettre en évidence une transition de phase. On attend de vous que vous produisiez différents jeux d'essais en particulier pour différentes densités de réseau.*
4. *Expliquer clairement les paramètres des jeux d'essais produits.*
5. *Dessiner les courbes % résolus/dureté.*

C. Évaluation de la méthode de résolution par défaut de Choco sur vos jeux d'essais

L'objectif de cette partie est de mesurer expérimentalement le comportement de Choco sur vos jeux d'essais. On s'attachera à produire différentes mesures d'évaluation de ce comportement : temps de calcul, taille de l'arbre développé… Référez-vous au document « Méthodologie d'évaluation des méthodes ».

Vous aurez certainement besoin de mettre en place un mécanisme de Time-Out. Pour cela, vous pourrez vous définir une limite de temps grâce à la fonction `solver.limitTime("10s")` et tester en sortie si le solver s'est arrêté par qu'il a trouvé une solution, parce qu'il a atteint la limite imparti de temps ou parce qu'il a trouvé qu'il n'y avait pas de solution à l'aide de la séquence :

```
if(solver.solve()){
    // do something, e.g. print out variable values
}else if(solver.hasReachedLimit()){
    System.out.println("The solver could not find a solution
                        nor prove that none exists in the given limits");
}else {
    System.out.println("The solver has proved the problem has no solution");
}
```

Vous expliquerez bien votre méthodologie d'évaluation (nombre d'exécutions, gestion des time-out…) et tracerez des courbes mesure d'évaluation/dureté et ferez une analyse des résultats obtenus.

Travail à faire :

1. *Ajouter à votre programme `Expe.java` un mécanisme de Time-Out (pour ne pas attendre trop longtemps les exécutions).*
2. *Modifier son programme `Expe.java` pour qu'il puisse calculer vos mesures d'évaluation/dureté.*
3. *Tracer les courbes de ces mesures*
4. *Expliquer la méthodologie d'évaluation mise en place*
5. *Analyser les résultats obtenus.*

D. Modification du solveur par défaut

Dans cette dernière partie, optionnelle, il vous est demandé de modifier le comportement par défaut du solver, par exemple en jouant sur l'heuristique de choix des variables. En exploitant la documentation Choco vous verrez qu'il est assez facile de programmer (ou simplement de sélectionner) une heuristique spécifique.

L'objectif est d'évaluer l'influence du choix d'une heuristique sur les performances de la résolution. Pour ça, il faut relancer votre évaluation sur les même jeux d'essais avec votre solver modifié.

Puis mettre sur un même graphique les deux résultats d'évaluation afin d'observer si l'une des heuristiques est meilleure qu'une autre.

Travail à faire :

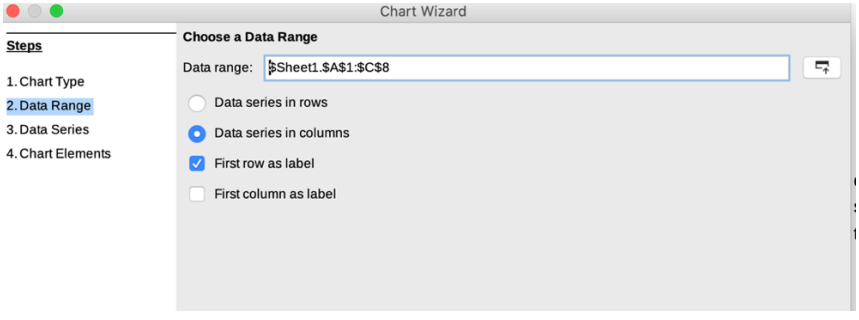
1. *Prévoir un système de choix permettant de lancer votre évaluation sur un solveur paramétré.*
2. *Lancer les évaluations avec les différents choix de solver.*
3. *Tracer les courbes.*
4. *Faire une analyse comparative des résultats.*

Annexe : Passer d'un tableau à un graphique (de type courbes) avec LibreOffice

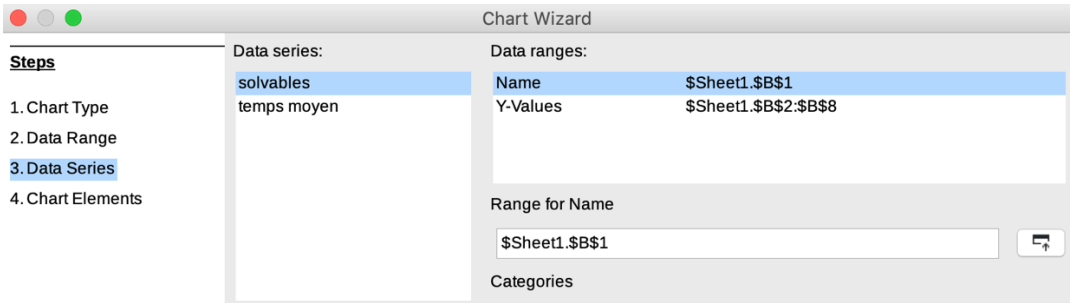
1) Créer un fichier LibreOffice de type spreadsheet (feuille de calcul) avec ce genre de contenu (les valeurs indiquées ici sont seulement à titre illustratif) :

dureté	solvables	temps moyen
1	100	10
2	95	11
3	92	12
4	50	800
5	2	80
6	0	10
7	0	8

2) Sélectionner le tableau et cliquer sur l'icône (ou le menu) Insert Chart et configurer le graphique voulu :
- Chart Type : "Line" de type "Points and lines"
- DataRange : "Data series in columns" (chaque série de données est sur une colonne) et "First row as label" (la première ligne donne le nom des courbes)



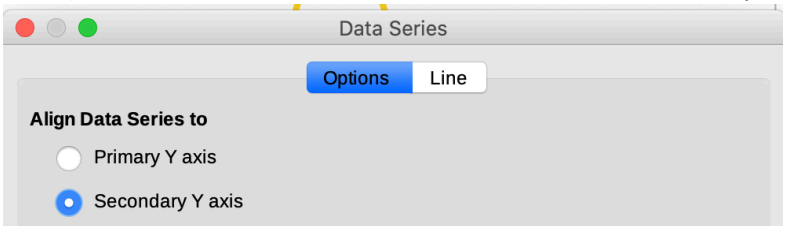
- Data series : enlever la série dureté (on ne la veut pas comme courbe, ce sera la légende de l'axe des abscisses)



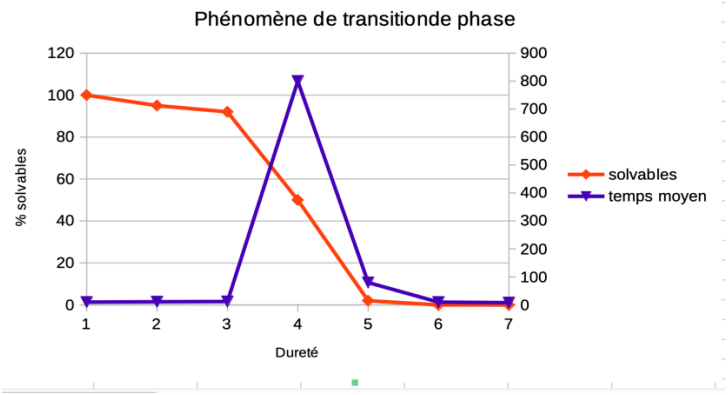
- Chart Elements: choisir le titre du graphique, le nom des axes des abscisses et des ordonnées, ...

=> Cliquer sur Finish : on obtient un premier graphique, mais avec un seul axe Y ce qui a l'inconvénient d'écraser les mesures de l'une des colonnes si les intervalles de valeurs sont très différents.

3) Effectuer un clic droit sur la deuxième courbe (celle pour laquelle on veut un axe des Y secondaire - qui sera à droite) : choisir "Format data series" dans le menu contextuel, puis dans l'onglet Option : cocher "secondary Y axis".



On obtient finalement un graphique de ce type :



Pour l'exporter dans différents formats, clic-droit sur la zone du graphique, et "Export as Image" dans le menu contextuel.