Enseignant : David LESAINT

Résolution de problèmes

-

Mickaël FARDILHA – Raphaël PILLIE

université angers

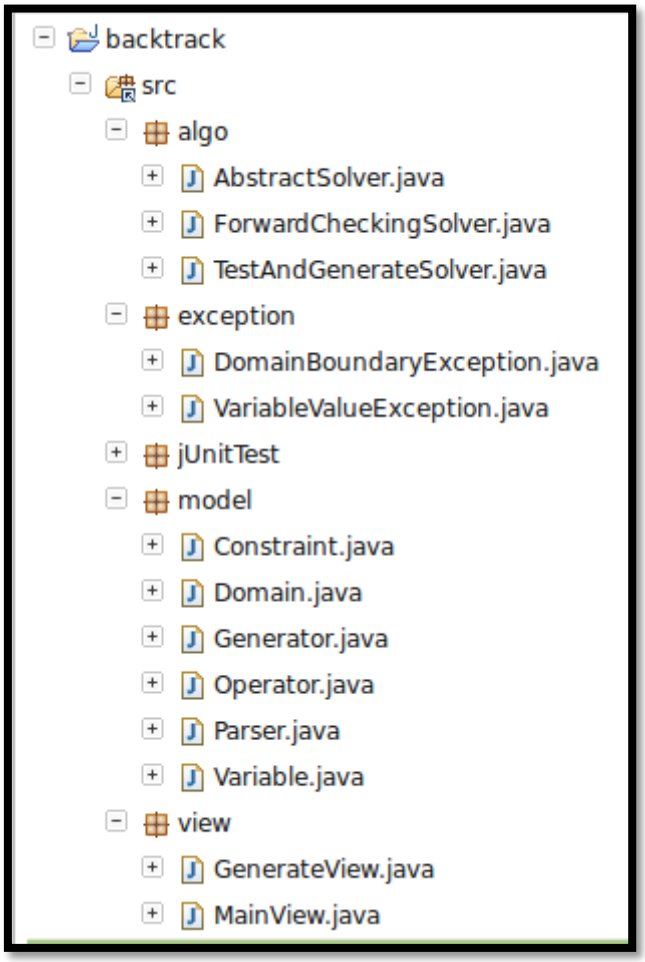
Sommaire

Introduction

Le master 1 informatique d'Angers propose deux options à choisir parmi quatre propositions. Pour cette première option, nous avons choisi la résolution de problèmes.
Ainsi, suite au cours qui nous a été donné, nous avons mis en place des algorithmes permettant de résoudre des problèmes : nous avons créé un solveur.

I – Structure du projet

Afin de répondre le mieux possible au sujet, nous sommes passés par une phase d'analyse afin de définir une structure propre pour ce projet. En effet, nous avons découpé l'ensemble du code pour séparer la partie algorithme de la partie modèle de donnée, interface graphique, exceptions ou encore tests unitaires :

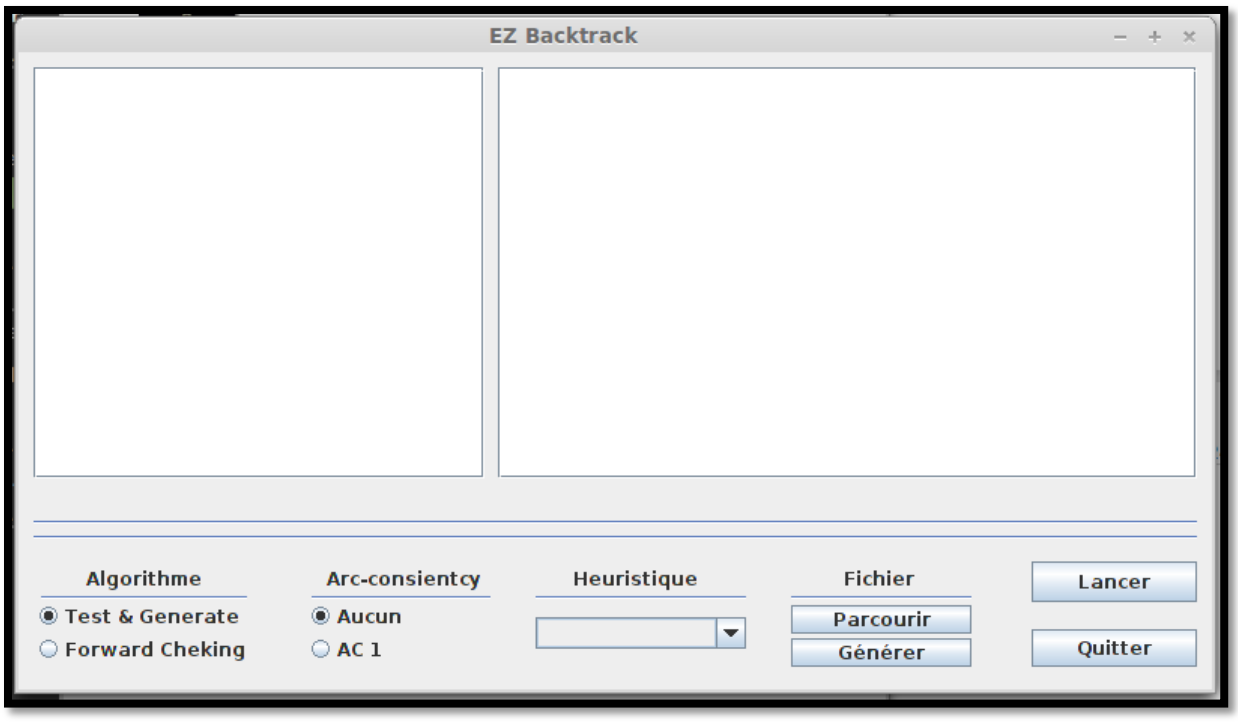


Suite à notre analyse préliminaire, nous avons décidé de mettre en place un certain nombre d'objets nous permettant de mettre en œuvre le plus efficacement possible notre projet. Tout d'abord, nous avons créé l'objet domaine, disposant d'une borne inférieur et supérieur.

Puis, nous avons mis en place la classe variable, possédant un nom, une valeur, une liste de domaine et un booléen nous permettant de savoir si cet objet a été instancié.
Ensuite, nous avons défini un opérateur, qui représente un opérateur booléen et qui ne contient que sa représentation graphique.
Enfin, nous avons créé l'objet contrainte (qui correspond à une contrainte binaire). Celui-ci contient deux variables et l'opérateur qui lui correspond.
Nous avons mis en place cela de façon à ce que seul un opérateur sache comment réduire les domaines des variables (car la réduction se fait en fonction de l'opérateur). C'est pourquoi la classe opérateur est une énumération, et que chaque élément de l'énumération surcharge une fonction permettant de réduire les domaines d'une variable. De plus, chaque élément de l'énumération implémente également une méthode permettant de tester une contrainte binaire. De cette manière, la classe contrainte a seulement besoin d'appeler les méthodes de l'objet opérateur pour fonctionner.

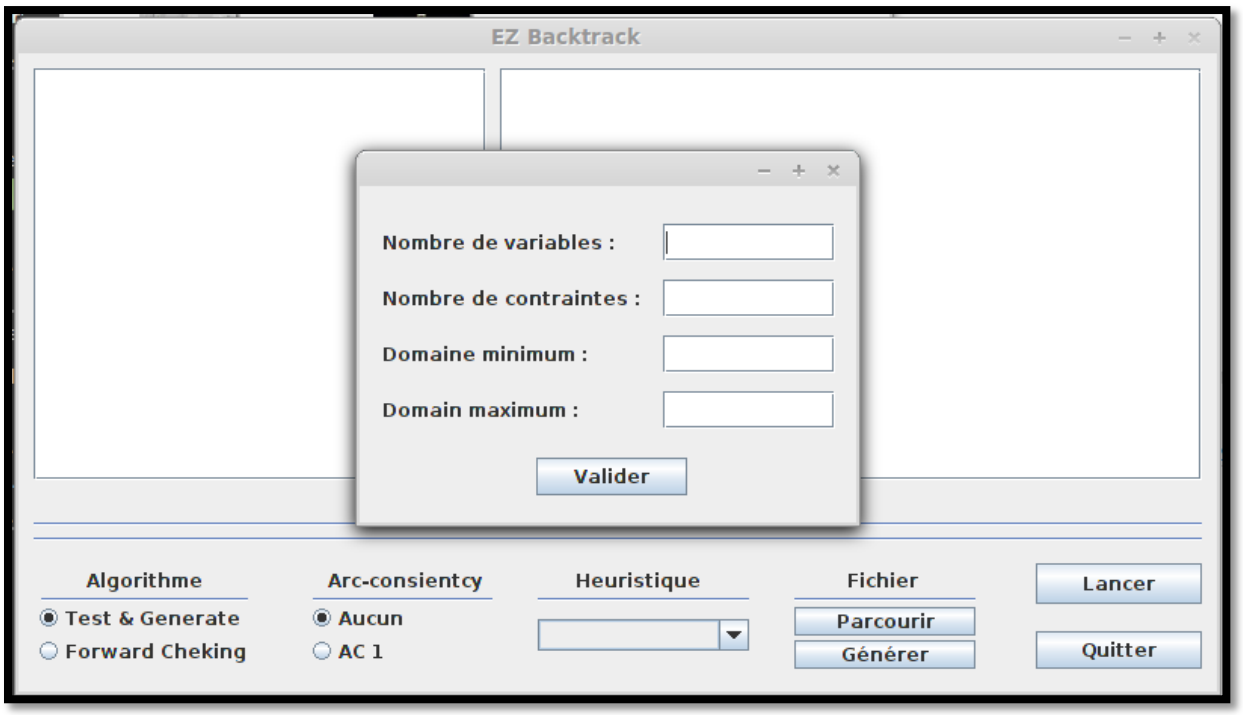
II – Présentation de l'application

L'application que nous avons mise en place une interface graphique permettant d'utiliser toutes les fonctions que nous avons développées. En effet, la fenêtre principale est présentée de la façon suivante :

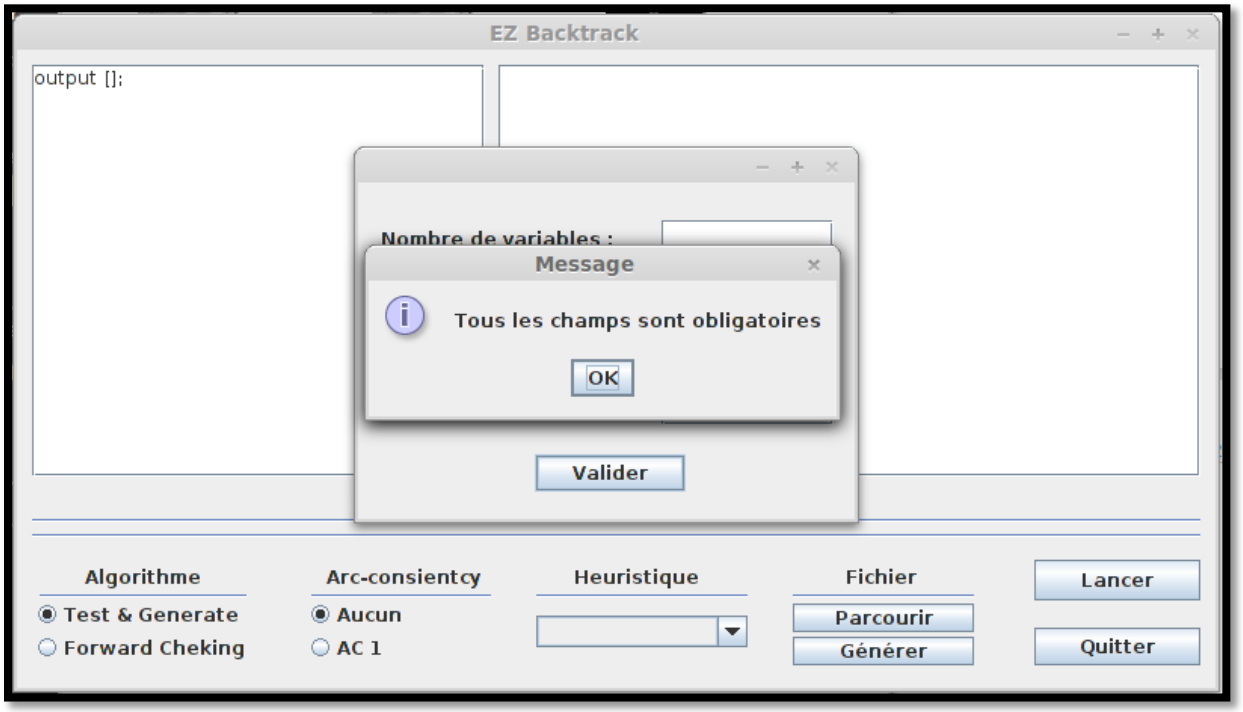


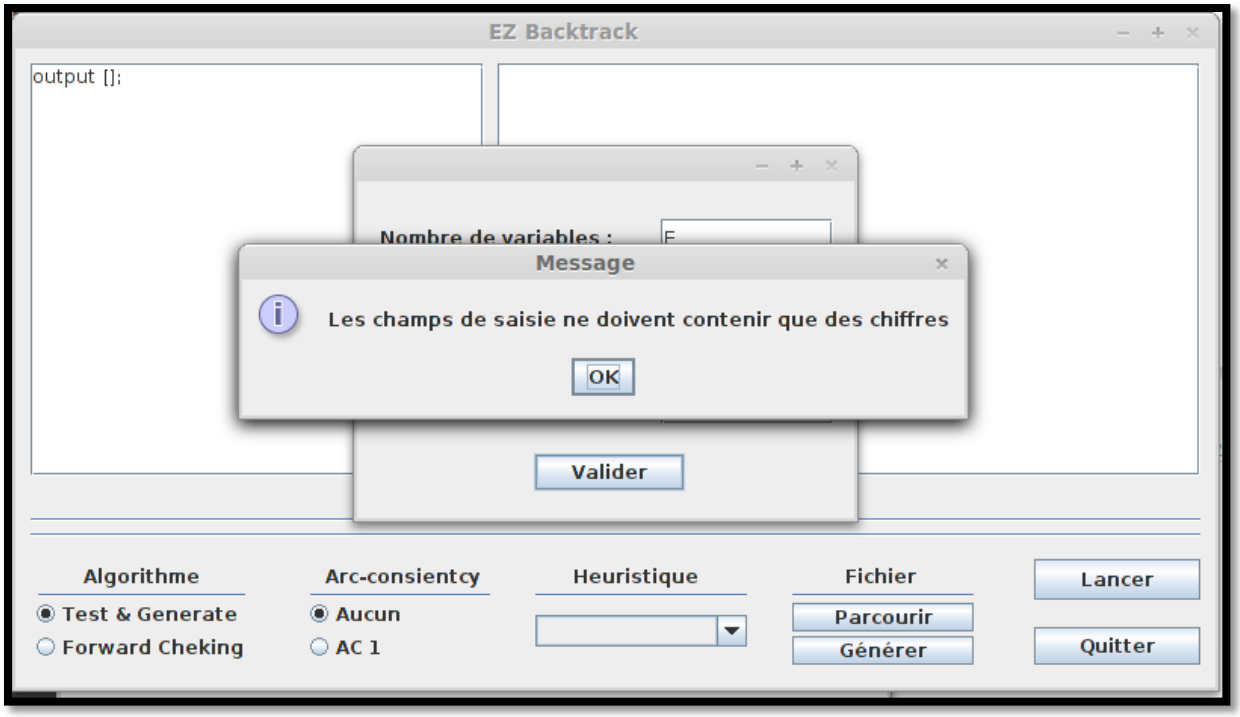
Ainsi, il est possible de choisir quel algorithme nous souhaitons utiliser, avec ou sans l'arc-consistency, avec ou sans un heuristique. Nous avons également mis à disposition un bouton parcourir permettant de charger un fichier de type MZN simple.
Une fois chargé, le contenu de ce fichier est disponible dans la partie gauche de l'interface, et il est possible de modifier le contenu avant l'exécution via le bouton « Lancer ».

Nous avons également développé un générateur permettant, selon certains critères de générer un jeu de test :

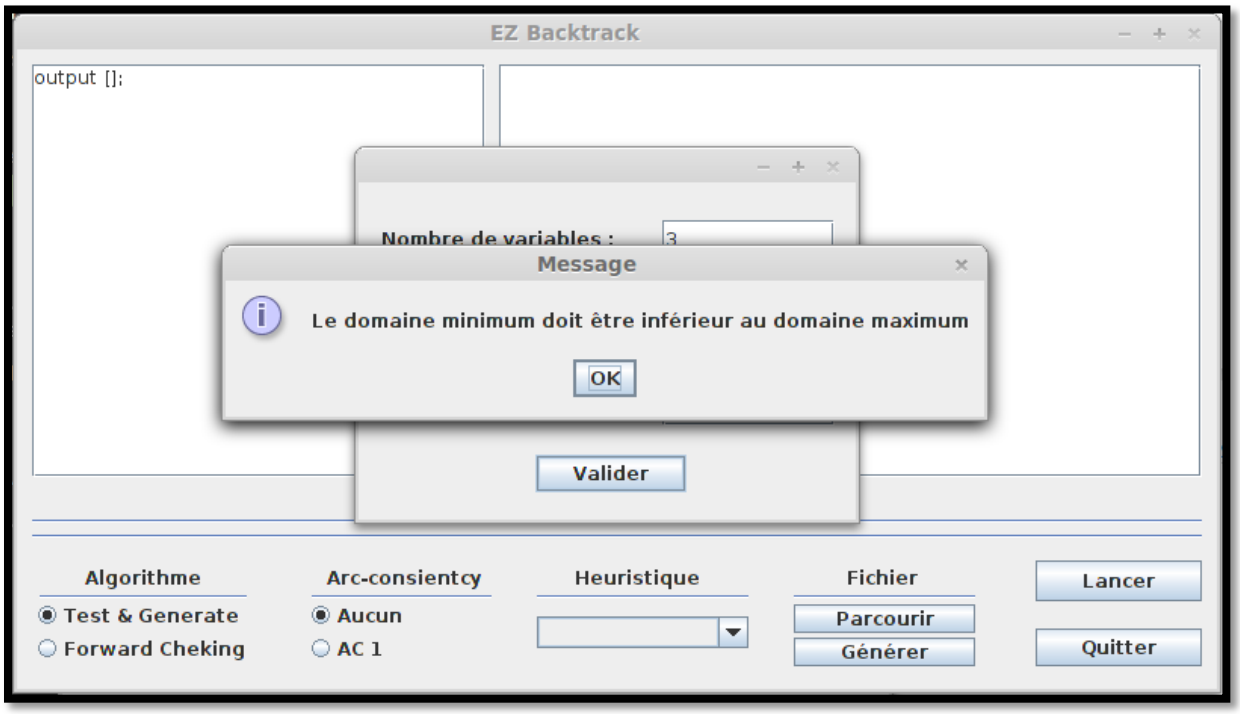


Bien évidemment, nous avons mis en place des contrôles de surface sur la fenêtre du générateur afin d'empêcher la saisie de données erronées, ou encore la non saisie des champs obligatoires :

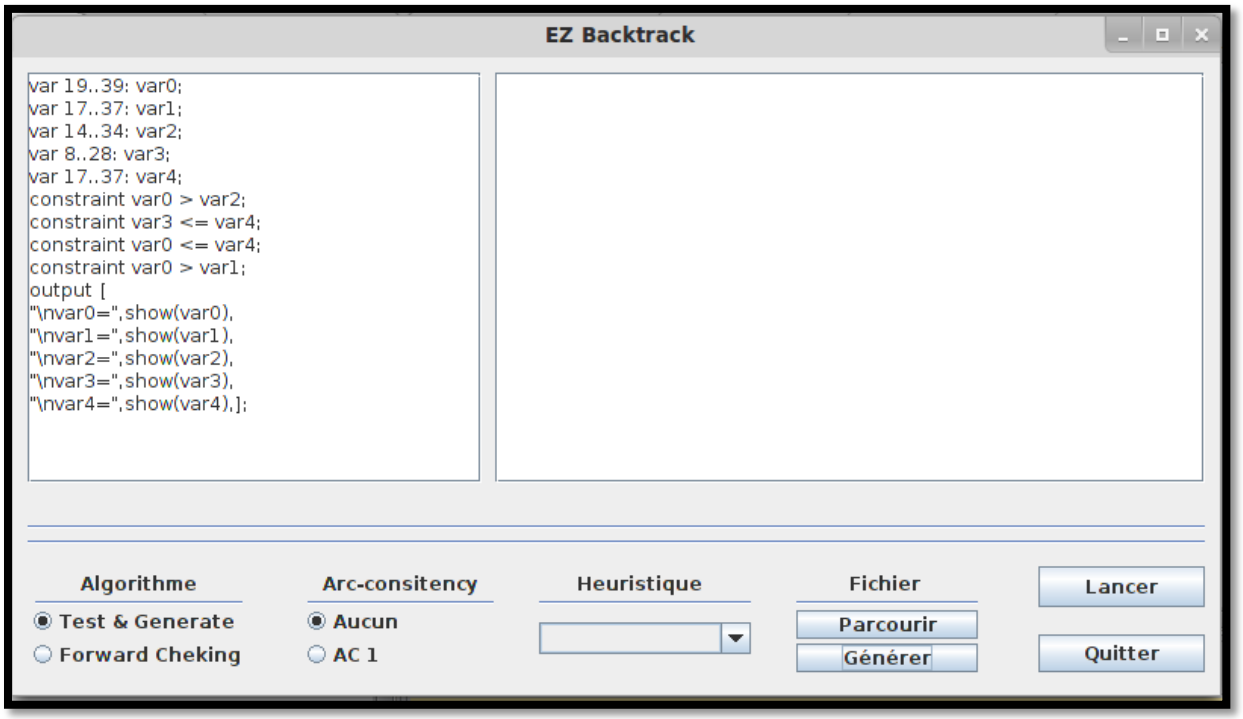




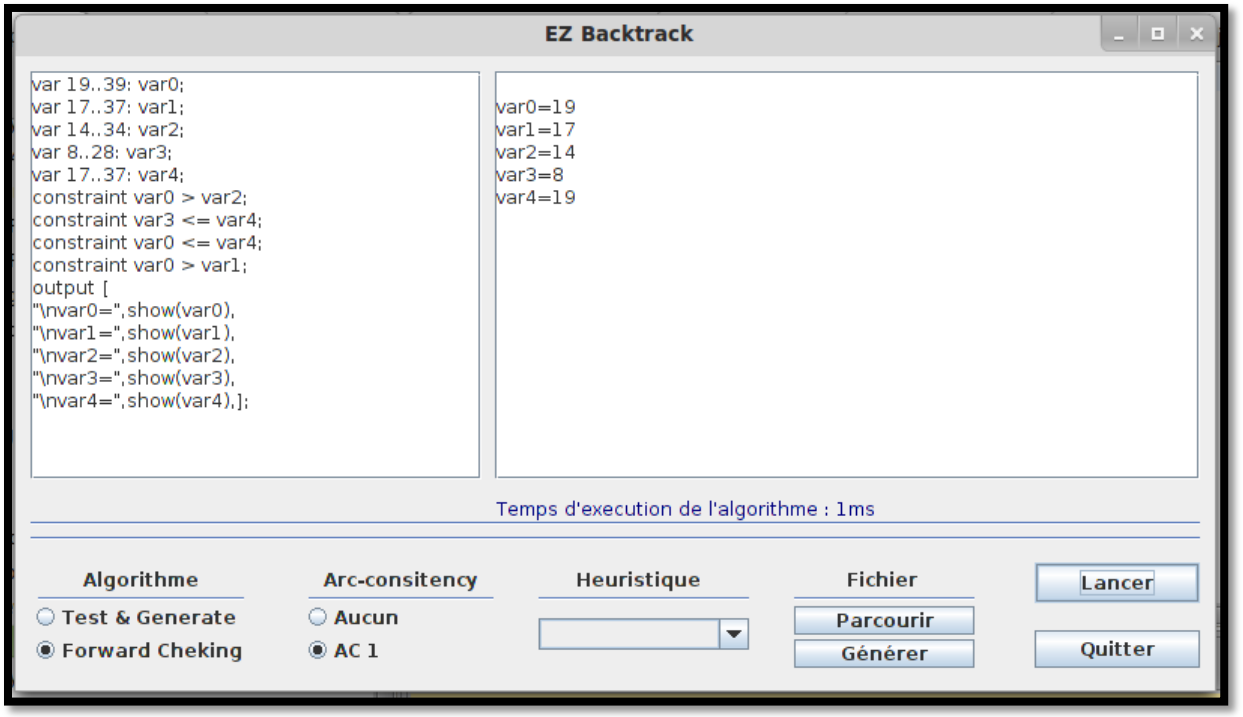
Enfin, nous avons mis en place un contrôle sur la saisie des champs liées au domaine afin d'éviter une saisie incohérente tel qu'une valeur pour le minimum du domaine supérieur au maximum de ce même domaine :



Une fois généré, nous pouvons observer le résultat dans la fenêtre de gauche :



Puis, en cliquant sur le bouton « Lancer », nous pouvons observer un résultat :



Comme explicité précédemment, il est également possible de lancer les algorithmes avec un fichier importé.

III – Algorithme de résolution

1) Arc-consistency

L'Arc-consistency est simplement une boucle parcourant l'ensemble des contraintes et demandant à chaque contrainte de réduire l'ensemble de domaine de chaque variable qui la compose, tant qu'un changement de détecté. C'est donc la classe opérateur qui est appelé pour réduire l'ensemble des domaines. L'avantage de cette implémentation est de permettre l'utilisation des méthodes d'instance et n'ont pas de test sur le type d'opérateur pour ensuite agir en fonction de ce dernier. Ce procédé évite de faire trop de tests et permet d'augmenter la vitesse d'exécution de l'algorithme de résolution qui le suivra.

2) solver:

Les algorithmes du « Forward checking » et du « Test and Generate » sont relativement semblables : seule la fonction de vérification effectuée sur chaque nœud est différente.
Plus précisément, le solveur parcourt l'ensemble des variables jusqu'à en trouver une qui ne soit pas instancié. Une fois repérée, l'algorithme parcourt chaque valeur possible sur ses ensembles de domaines et exécute la fonction de vérification. Si la fonction de vérification ne remonte pas d'erreurs, le solveur continue sur une autre variable. De plus, si toutes les variables sont instanciées et qu'il n'y a pas d'erreurs, l'algorithme a alors trouvé une solution.

3) forward checking:

Comme il a été explicité dans la partie précédente, seule la fonction de vérification est différente.
En effet, pour l'algorithme du « Forward checking », pour chaque nœud, nous réduisons les domaines de chaque variable et nous vérifions s'il reste des possibilités. Si ce n'est pas le cas, nous ne cherchons pas plus loin.

4) Test & Generate :

On parcourt toutes les contraintes, et on ne test que celles qui ont leurs deux variables instancié.

IV – Statistiques des résultats & des performances

Les statistiques sur les problèmes que nous avons générés aléatoirement ne sont pas suffisamment dispersées pour justifier l'utilité de l'utilisation d'un graphe. En effet, lorsque nous comptons le nombre de nœud parcouru pendant l'utilisation des algorithmes de « forward checking » avec ou sans « arc consistency » et de « test and generate » avec « arc consistency », le nombre de nœud parcouru est sensiblement égale aux nombres de variable que comporte le problème. En effet suivant le problème généré avec une même configuration, le nombre de nœud parcouru varie rarement au-dessus du nombre de variable que comporte le problème. Et ceci jusqu'a un grand nombre de variable. Seul l'algorithme « test and generate » sans « arc consistency » nous donnes des résultats totalement différent, mais le nombre de nœud parcours dépend totalement de la génération du problème et n'est donc pas très représentative.
En termes de temps d'exécutions, nous avons observé qu'il augmente en fonction du nombre du nombre de variable.

V – Extensions

Au cours de la réalisation de ce projet, nous avons également mis en place deux classes permettant de gérer les exceptions spécifiques à notre modèle de données.
De plus, nous nous sommes attelé à mettre en place des tests unitaires afin de vérifier la véracité des résultats obtenus.
D'autre part, en suivant les conseils avisés de l'enseignant dirigeant cette option, nous avons décidé de créer un générateur de données afin de pouvoir pousser nos tests toujours plus loin.
Enfin, nous avons mis en place une interface graphique afin de rendre plus facile l'utilisation de notre application.

Conclusion

Ce projet nous a permis de mettre en pratique les notions importantes vues en cours afin de nous permettre de mieux comprendre les problèmes et leurs résolutions, qui peuvent vraisemblablement devenir un enjeu majeur dans certains domaines de l'informatique.

Ainsi nous avons pu apercevoir la complexité de ce domaine, et commencer à apercevoir les techniques de résolution utilisés à ce jour.