Programmation par contraintes
TP 2

Vous devez déposez sur le Moodle le code source d'un programme Prolog répondant aux questions ci-après le **mardi 25 mai 2021 à minuit au plus tard**. Votre nom et les réponses en français doivent être insérés dans votre fichier sous forme de commentaires.

Problème de recouvrement

Soit un « grand » carré de taille T et n « petits » carrés de tailles $T_1, T_2, \ldots, T_n$. Placer les n petits carrés à l'intérieur du grand, sans qu'ils se chevauchent ni ne dépassent du grand carré, afin qu'il soit complètement couvert.

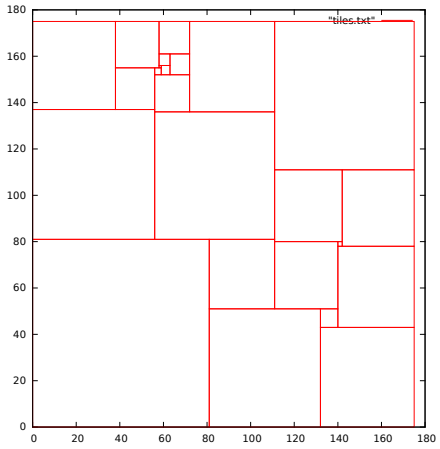

Figure 1 – Une solution pour le quatrième problème (affichée avec `gnuplot`).

Quatre instances de ce problème et d'autres prédicats sont définies dans le fichier source `libtp2.pl`. On peut utiliser le contenu d'un fichier source `src.pl` dans un autre grâce à la directive :

```
:- include(src).
```

Les instances sont décrites par le prédicat `data(Num, T, Ts)` où `Num` est le numéro de l'instance (de 1 à 4), `T` le côté du grand carré et `Ts` les côtés des petits carrés.

T	$T_1, T_2 \ldots$
3	2,1,1,1,1,1
19	10,9,7,6,4,4,3,3,3,3,3,2,2,2,1,1,1,1,1,1
112	50,42,37,35,33,29,27,25,24,19,18,17,16,15,11,9,8,7,6,4,2
175	81,64,56,55,51,43,39,38,35,33,31,30,29,20,18,16,14,9,8,5,4,3,2,1

Modèle basique

1. Écrire un prédicat `solve(Num, Xs, Ys)` qui résolve l'instance `Num` et affecte les listes de variables de décisions `Xs` et `Ys` correspondant aux coins inférieurs gauches des petits carrés :
   — Pour résoudre la première instance, il est suffisant de contraindre **toute paire unique de petits carrés à ne pas se chevaucher**, en veillant à définir correctement les domaines des variables de décision pour que **les petits carrés ne puissent pas dépasser du grand**.
   — Pour ce problème, il est plus efficace d'utiliser deux buts d'affectation `fd_labeling` (un pour les `Xs` et l'autre pour les `Ys`) plutôt qu'un seul portant sur toutes les variables.

2. Vérifier la solution obtenue à l'aide du prédicat `printsol(File, Xs, Ys, Ts)` qui écrit les coordonnées des coins des petits carrés de la solution dans le fichier `File` (chaîne de caractères entre simples quotes). On peut ensuite visualiser cette solution grâce au logiciel `gnuplot` en tapant `plot "tiles.txt" w l` à l'invite de commande.

Contraintes redondantes

1. Pour résoudre des instances de plus grande taille, on peut améliorer l'efficacité de la propagation de contraintes en ajoutant les **contraintes redondantes** suivantes : la somme des tailles des petits carrés coupés par une verticale (ou une horizontale) est égale à la taille du grand carré. Spécifier cette contrainte pour toute verticale et horizontale (entre les coordonnées 0 et $T-1$) à l'aide de **contraintes réifiées**, e.g. pour les verticales :

$$\forall v \in [0, T[,\ \ \Sigma_{i=1}^{n} T_i \times (X_i \leq v \wedge v < X_i + T_i) = T$$

2. On peut obtenir le nombre de backtracks depuis la dernière solution en ajoutant l'option `backtracks(B)` à la liste d'options du prédicat `fd_labeling`. Ajouter ce paramètre au prédicat `solve` pour que le nombre de backtracks soit affiché. Comparer et commenter les nombres de backtracks avec et sans les contraintes redondantes pour la première instance.

Stratégie de recherche

1. Pour résoudre les instances les plus grandes, il faut également utiliser une stratégie de sélection des variables qui choisit **la variable de plus petite valeur minimale**, mais de manière **plus dynamique** qu'avec le prédicat `fd_labeling`. Les stratégies « classiques » choisissent la prochaine variable à affecter en optimisant un critère de sélection, mais une fois la variable choisie, elle n'est pas remise en cause. On peut obtenir une stratégie « plus dynamique » en réévaluant le critère après chaque point de choix car le retrait de la valeur d'affectation peut modifier le critère et la prochaine variable sélectionnée (notamment pour la stratégie « min-min », moins fréquemment pour « min-size »).
   Utiliser le prédicat `labeling(Xs, Ys, Goal, Criterion, B, NbSol)` (défini dans `libtp2`) avec le but `assign` (au lieu de `indomain` qui correspond à la stratégie de `fd_labeling`) et le critère `minmin`. Ce prédicat permet également d'afficher le nombre de backtracks `B` depuis le début de la recherche et le nombre de solutions trouvées `NbSol` (qu'on ajoutera en argument du prédicat `solve`).
   Attention : pour pouvoir résoudre les plus grandes instances, il peut être nécessaire sur certains systèmes d'augmenter la taille de la pile des contraintes en définissant la variable d'environnement `CSTRSZ` *:*

   ```
   export CSTRSZ=16384
   ```

2. Sur la deuxième instance, comparer et commenter le nombre de backtracks obtenu en utilisant les buts `assign` et `indomain`.

Symétries

1. Afficher toutes les solutions pour la première instance (en tapant `'a'` après avoir obtenu la première solution). Rompre la symétrie de permutation entre petits carrés de même taille en ordonnant leurs coordonnées lexicographiquement, puis afficher le nombre de solutions restantes.
2. Rompre (partiellement) les symétries géométriques en restreignant la position du centre du plus grand carré dans le quart inférieur gauche. Combien reste-t-il de solutions pour la première instance ? Pour la seconde ?