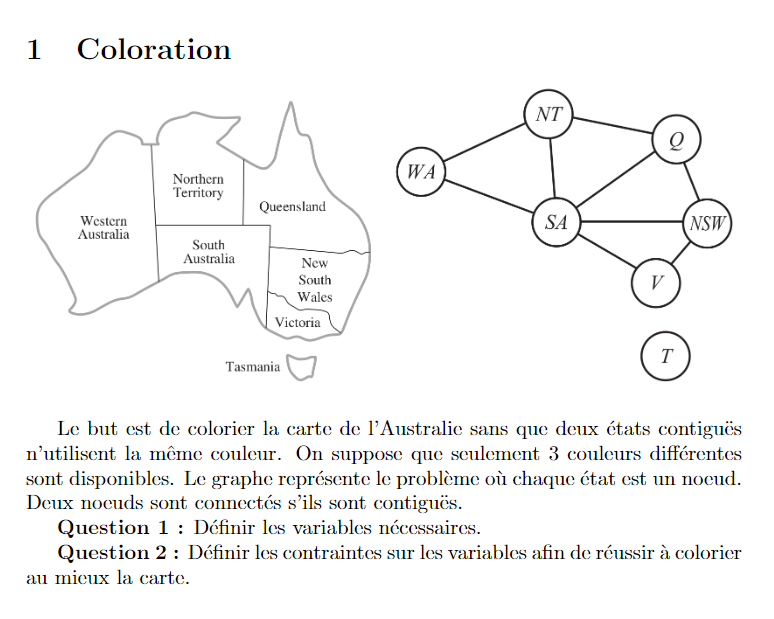
Introduction à l’IA

Programmation par contraintes

**Question 1 :**

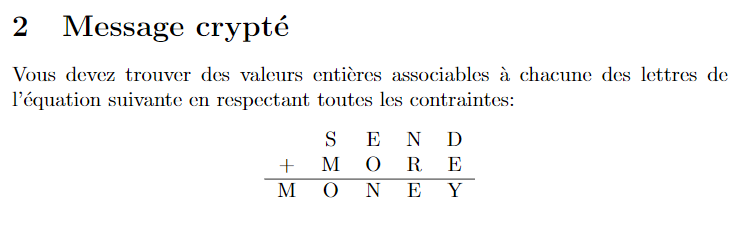
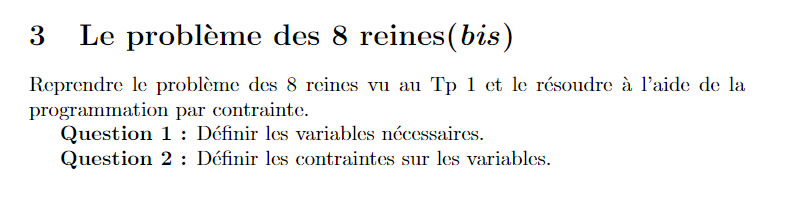
Ici les variables du problème sont les états de l’Autralie :

Le domaine de chacune des variables est

Les variables seront ensuite converties en entier pour le solveur

**Question 2 :**

Ici les contraintes consistent à avoir des couleurs différentes pour chaque état contigu : du problème sont les états de l’Autralie :



**Question 1 :**

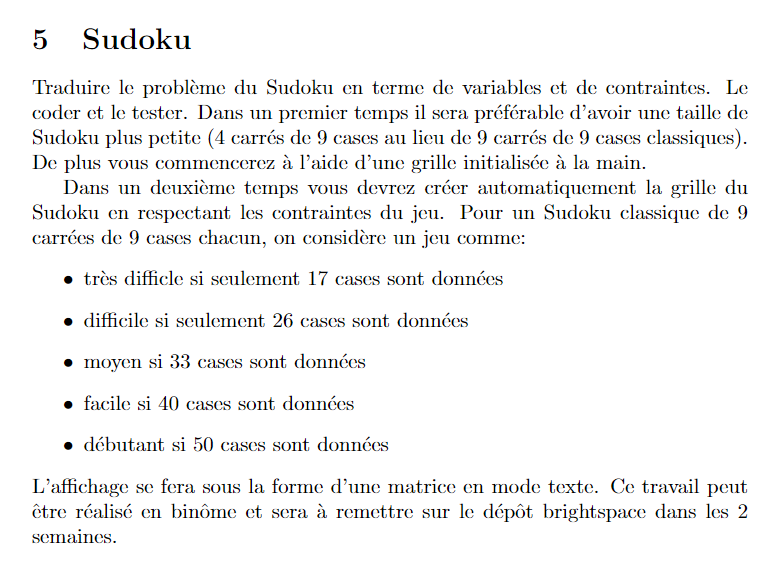
Ici les variables du problème sont les lettres distinctes du message :

Etant donné qu’il y a moins de 10 lettres, nous allons travailler en base **10**.

De ce fait, le domaine de chacune des variables (hormis S et M) est . Pour S le domaine est le même que le précédent sans la valeur 0 car ils sont en début de chiffre (0923 ne peut pas exister). M vaut forcément 1 car on ne peut pas aller plus loin que 2\*10^4 avec une addition de deux valeurs < 9999.

**Question 2 :**

Ici les contraintes consistent à avoir des valeurs différentes pour chacune des lettres (sinon le problème n’a plus de sens car si les lettres sont strictement distinctes alors leurs valeurs aussi). La contrainte indiquée dans le sujet s’exprime par :



**Question 1 :**

Ici les variables du problème sont les index (q1,…,q8) des lignes de 8 reines :

En effet, la manière de stocker une solution au problème impose une coordonnée colonne différente automatiquement, cela nous permet de réduire les contraintes pour le solveur.

Ici le domaine des variables est donc .

**Question 2 :**

Ici les contraintes consistent à avoir des valeurs différentes pour chacun des index ligne des reines et également que aucune reine ne soit sur les mêmes diagonales. On a donc deux contraintes :

Pour la deuxième contrainte, nous avons remarqué que toutes les valeurs de X(i) – i (diagonale descandante) et de X(i) + i (diagonale ascendante) étaient différentes dans le cas d’un jeu valide.

Alors il vient ***:***

Le module à réaliser doit être donc capable de générer des grilles de tous niveaux et aussi de les résoudre. Notre approche au problème va se dérouler en deux temps :

**Résolution de grille :**

Nous allons modéliser le problème de la résolution avec la programmation par contraintes.

Nous avons choisi de stocker le sudoku dans une liste de ligne (liste). De ce fait, les variables du problème sont les valeurs contenues dans les lignes du Sudoku :

Ici le domaine des variables est donc . Les valeurs déjà présentes dans le Sudoku sont en revanche, constantes de domaine.

Il y a 3 contraintes, l’unicité des valeurs sur une ligne fixée, une colonne fixée ou une région fixée.

Au niveau du code, cela est représentée par une classe Sudoku initialisée avec une grille rentrée en dur. La grille est ensuite convertie en variables du solveur OR-tools, et les contraintes ajoutées en fonctions de X. Etant donné que X contient les lignes du Sudoku, afin d’obtenir la contrainte sur les colonnes, il suffit de transposer notre matrice X et d’ajouter la contrainte au modèle, et pour les régions de restructurer notre liste de ligne en liste de regions (détails dans le code) et d’ajouter la contrainte au modèle. Le solveur affiche donc en sortie les solutions trouvées pour une grille d’entrées.

**Création de grilles solvables :**

Afin de créer des grilles solvables en fonction d’une difficulté en entrée, le procédé suivant a été utilisé :

* Créer une grille avec la diagonale descendante principale remplie de chiffres entre 1 et 9, de manière aléatoire. Cette opération délivre 1 grille possible parmi et assure une initialisation presque différente pour chaque exécution du programme.
* Résoudre cette grille pour 1 solution avec notre solveur. En effet, si nous n’initialisions pas la grille précédente de manière aléatoire, le solveur retournerait tous la temps la même première solution.
* En fonction de la difficulté on détermine le nombre de cases a retirer de cette solution valide.
* On retire une à une les cases de la grille, en **imposant une solution unique pour chaque case retirée.** Cela vérifie que le sudoku est dit strictement valide.
* Enfin, on affiche la grille générée et on résout le Sudoku.

Après plusieurs tests pour la difficulté très difficile (hell), il est impossible de garantir une solution unique au solveur à chaque itération étant donné que les contraintes deviennent de plus en plus « souples ». Une option max\_solutions a donc été implémentée afin de ne pas obstruer la console avec le nombre important de solutions possibles.

Pour la deuxième contrainte, nous avons remarqué que toutes les valeurs de X(i) – i (diagonale descandante) et de X(i) + i (diagonale ascendante) étaient différentes dans le cas d’un jeu valide.

Alors il vient :