TP IA-Résolution de problèmes
2ème année SRI

M.C. Lagasquie

12 février 2020

1 Contexte

R2D2 est un robot placé dans un monde 2D représenté par un graphe non orienté : les arêtes représentent les routes que R2D2 peut suivre, alors que les sommets représentent les lieux où R2D2 a des choses à faire. Les arêtes seront pondérées pour représenter la longueur du chemin que doit parcourir R2D2 pour aller du sommet origine de l'arête au sommet arrivée de l'arête. Et chaque sommet est pondéré par sa position dans le plan 2D du monde (coordonnées euclidiennes).
On prendra comme hypothèse que R2D2 connaît le monde dans lequel il est placé.
Le travail que doit faire R2D2 : déposer 1 cube de couleur à chaque lieu de manière à ce qu'il y ait dans deux lieux voisins (c-à-d liés par une arête) des cubes de couleur différente.
On considérera que R2D2 dispose de suffisamment de cubes.
Le travail de R2D2 va se décomposer en plusieurs tâches :

- tâche 1 : Au début, R2D2 décide de n'utiliser que 3 couleurs. Il va donc devoir "raisonner" pour savoir si ces 3 couleurs lui suffisent ou pas pour réaliser son travail.
- tâche 2 : Ensuite, R2D2 cherche à savoir comment il va aller déposer les cubes le plus rapidement possible. Pour cela, il cherche des chemins les plus courts en terme de distance parcourue.
- tâche 3 : Finalement, R2D2 cherche à savoir combien il lui faut de couleurs a minima pour réaliser son travail.

2 Planning

12 séances de TP sont prévues correspondant à 5 étapes. La répartition du travail à faire est la suivante :

Etape 1 = TP 1 et 2 Réalisation de la tâche 1 : Vous devrez identifier le nom et le type du problème, proposer un *encodage en logique propositionnelle* et le résoudre en utilisant le solver SAT fourni.

Pour cela vous compléterez la classe `Etape1` du package `etape1` et vous l'exécuterez. Il peut être souhaitable de compléter les tests déjà proposés.

Etape 2 = TP 3 à 6 Réalisation de la tâche 2. Trois cas sont prévus :

- Cas 1 : R2D2 commence par calculer le *plus court chemin entre deux lieux* de son monde. Pour cela, il utilise les distances entre les lieux, ainsi que les coordonnées cartésiennes de chaque lieu.
- Cas 2 : Puis il cherche le *chemin le plus court permettant de passer dans chaque lieu une seule fois puis de revenir ensuite à son point de départ.*

— Cas 3 : R2D2 vient d'être "upgradé" : son concepteur l'a équipé de la capacité à voler. Il peut désormais *relier en ligne droite* chacun des lieux de son monde sans être obligé de suivre les routes. Il peut donc trouver un autre chemin plus court permettant de passer par chaque lieu et de revenir ensuite à son point de départ.

Dans les trois cas, vous devrez identifier le nom et le type du problème, proposer un mode de représentation et le résoudre en utilisant un des algorithmes fournis.

Pour cela vous compléterez les classes `EtatCas1`, `EtatCas2` et `EtatCas3` du package `etape2` et vous les utiliserez pour compléter et exécuter la classe `Etape2` du package `etape2`. Il peut être souhaitable de compléter les tests déjà proposés.

Etape 3 = TP 7 et 8 Réalisation de la tâche 2 (suite). R2D2 se rend compte que sa méthode précédente met trop de temps à s'exécuter ! Du coup, il renonce à trouver le chemin le plus court et est *prêt à tenter des chemins un peu moins bons pourvu qu'il arrive à les calculer plus vite.* Et comme il est curieux, il va essayer deux méthodes différentes pour voir celle qui est la plus efficace. Vous devrez proposer un mode de représentation et résoudre le problème en utilisant au moins deux des algorithmes fournis.

Pour cela vous compléterez la classe `UneSolution` du package `etape3` et vous l'utiliserez pour compléter et exécuter la classe `Etape3` du package `etape3`. Il peut être souhaitable de compléter les tests déjà proposés.

Etape 4 = TP 9 et 10 Réalisation de la tâche 3. Vous devrez identifier le nom et le type du problème, proposer un *encodage sous la forme d'un graphe de contraintes* et le résoudre en utilisant un des algorithmes fournis.

Pour cela vous compléterez et exécuterez la classe `Etape4` du package `etape4`. Il peut être souhaitable de compléter les tests déjà proposés.

Etape 5 = TP 11 et 12 Réalisation de la tâche 2 (suite et fin). Comme R2D2 aime aussi beaucoup les maths et qu'il veut épater ses concepteurs, il reprend le *cas 3 de la tâche 2*, et cherche à le résoudre en l'exprimant à l'aide de formules mathématiques. Vous devrez proposer un encodage approprié utilisant le langage ZIMPL et résoudre le problème en utilisant le solver SCIP.

Une analyse comparative des résultats obtenus lors des étapes 2, 3 et 5 sera un plus.

3 Travail en bonus

Pour les étudiants les plus rapides et les plus motivés, il y a la possibilité d'écrire et de tester ensuite son propre solver pour l'étape 4 en utilisant la librairie Choco-Solver écrite en Java.
Cette librairie est décrite sur le site web : `https://choco-solver.org/`
Et la JavaDoc de cette librairie est accessible en suivant le lien : `https://javadoc.io/static/org.choco-solver/choco-solver/4.10.1`. Elle est aussi donnée dans l'archive jar `DocChoco.jar` récupérable sous Moodle.
Suggestion : il pourrait être intéressant de tester ce solver sur l'exemple de Lewis Caroll vu en cours en plus des cas cités dans le fichier `Etape4.java` du package `etape4`.

4 Organisation pratique

Vous devrez tenir à jour un "journal de bord" décrivant ce que vous faites à chaque séance, vos réponses aux questions posées, les choix que vous avez fait (en particulier en terme d'encodage) et les résultats que vous avez obtenus. Ce journal de bord sera à déposer 15 jours après la dernière séance de TP au format `pdf` (et uniquement `pdf`) dans le devoir moodle correspondant.

Vous devrez aussi rendre l'archive finale de votre projet contenant la totalité de vos répertoires `etape1` à `etape5`. Cette archive sera à déposer 15 jours après la dernière séance de TP au format `zip` (et uniquement `zip`) dans le devoir moodle correspondant.

La notation se basera sur l'état d'avancement de votre projet (évalué en séance par l'enseignant), sur votre journal de bord et sur l'archive de votre projet.
Attention, le non-respect des consignes entraînera automatiquement une note de 0 à la partie correspondante.

5 Matériel mis à disposition

Deux cas sont possibles :
- soit vous travaillez sous linux et en ligne de commande,
- soit vous travaillez avec Eclipse.

5.1 Sous linux et en ligne de commande

Vous devrez récupérer l'archive `aUtiliser.zip` disponible sous Moodle.
Elle contient :
- le dossier `Data` dans lequel vous trouverez des jeux de test ;
- le dossier `projet` dans lequel il y a :
  - le dossier `solvers` contenant les classes Java qui implémentent les algorithmes évoqués en cours ; vous y déposerez aussi le solver SCIP (pour la PLNE) que vous aurez récupéré sur internet ;
  - les dossiers `org` et `gnu` contenant des classes utilisées par les solvers ;
  - le dossier `outils` contenant les classes Java qui permettent de représenter le monde du robot ;
  - les dossiers `etape1` à `etape5` dans lesquels vous trouverez les fichiers sources java avec lesquels vous devrez travailler.
- le dossier `Doc` dans lequel vous trouverez les documentations des classes à utiliser ainsi que le mode d'emploi du solver SCIP et la documentation concernant le langage ZIMPL.

Installation : (sous linux et en ligne de commande uniquement)

1. télécharger l'archive et la décompresser avec la commande "unzip nomArchive.zip"
2. les fichiers sources sont à travailler avec un éditeur simple
3. la compilation se fait depuis le répertoire "projet" en utilisant la commande "javac nomFichier.java" (par exemple javac etape1/Etape1.java)
4. l'exécution se fait depuis le répertoire projet avec la commande "java nomFichierAvecUnMainSansExtension" (par exemple java etape1/Etape1)

5.2 Avec Eclipse

Vous devrez récupérer l'archive `aUtiliser-v2.jar` et les 5 autres fichiers `solvers.jar`, `outils.jar`, `org.jar`, `gnu.jar` et `DocChoco.jar` disponibles sous Moodle.
L'archive `aUtiliser-v2.jar` contient :
- le dossier `Data` dans lequel vous trouverez des jeux de test
- les dossiers `etape1` à `etape5` dans lesquels vous trouverez les fichiers sources java avec lesquels vous devrez travailler.
- le dossier `Doc` dans lequel vous trouverez les documentations des classes à utiliser ainsi que le mode d'emploi du solver SCIP et la documentation concernant le langage ZIMPL.

4 des autres fichiers jar correspondent aux dossiers `solvers`, `outils`, `org` et `gnu` décrits en section précédente.
Le dernier donne une partie de la JavaDoc correspondant à l'archive `org.jar`.

Installation : (sous Eclipse)

1. télécharger l'archive `aUtiliser-v2.jar` et la placer dans le workspace d'Eclipse
2. ouvrir Eclipse
3. dans le menu File, sélectionner "import"
4. dans l'item "general" choisir "Existing project into workspace", puis "Next"
5. sélectionner le fichier d'archive à utiliser (`aUtiliser-v2.jar`), puis "finish"
6. télécharger les 4 fichiers jar `solvers.jar`, `outils.jar`, `org.jar` et `gnu.jar`
7. les rajouter au projet en tant qu'archives externes (click droit sur le projet et sélection "build path" puis "add external archives")

Si vous voulez pouvoir utiliser sous Eclipse la JavaDoc des archives externes :
- click droit sur le projet et sélection "build path" puis "Configure Build Path" ;
- choisir ensuite le filtre "Java build Path", puis l'onglet "Librairies" ;
- développer l'item correspondant à l'archive externe dont vous voulez la JavaDoc ;
- cliquer sur "JavaDoc Location", puis "Edit" ;
- remplir le champ "Javadoc location path"avec le répertoire principal (ou bien "JavaDoc in Archive" avec le fichier jar) contenant cette documentation (celui qui contient le fichier `index.html`) ;
- et finir avec "OK" puis "Apply and Close" ; il peut être nécessaire de faire parfois un "refresh" du projet.

Pour info, la JavaDoc des archives `solvers.jar` et `outils.jar` est dans le dossier `Doc` et celle d'une partie de l'archive `org.jar` est dans l'archive jar `DocChoco.jar`.

5.3 Contraintes dans tous les cas

Attention, l'architecture des répertoires ne doit pas être modifiée !
Parmi les fichiers fournis, seuls les fichiers sources java peuvent être modifiés (mais sans modifier leur package d'appartenance) !

La version de Java utilisée doit être au moins la version 1.8.0_101