Programmation Par Contraintes - TP Choco
Projet 2: Traveling Salesman Problem with Time Windows

Charles Prud'homme & Sophie Demassey

9 novembre 2010

A Présentation du projet

A.1 Organisation

- séance 1 (TD – 2h30) : modèle contrainte du TSP
- pour la séance suivante : implémentation Choco du modèle
- séance 2 (TD/TP – 2h30) : heuristiques de branchement pour l'optimisation
- pour la séance suivante : terminer l'implémentation
- séance 3 (TD/TP – 2h30) : prise en compte des fenêtres de temps
- 2 semaines après : livraison du prototype + rapport d'évaluation

A.2 Objectifs de formation

Programmation par contraintes
- conception d'un modèle d'optimisation de contraintes à partir de spécifications concrètes
- application de contraintes globales à la modélisation
- conception d'heuristiques de branchement
- conception de contraintes globales

Implémentation / solveur Choco
- manipulation des éléments de Choco, lecture et recherche de documentation
- développement d'heuristiques de branchement
- développement de contraintes globales

Ingénierie
- analyse des spécifications
- développement logiciel (programmation objet, qualité logicielle, documentation)
- mise en place de tests expérimentaux sur des instances benchmark
- expérimentation et analyse des résultats

B Présentation du problème

B.1 (TSP) : Traveling Salesman Problem

Étant donnés un ensemble de villes et la matrice des distances minimales entre toutes paires de villes, il s'agit de construire une tournée de distance minimale visitant chaque ville en partant et terminant en une ville préalablement définie, appelée le dépôt. Comme la distance satisfait l'inégalité triangulaire, on sait alors que chaque ville, exceptée la ville de départ, est visitée exactement une fois dans toute solution optimale. On peut donc restreindre la résolution à une recherche parmi l'ensemble de ces solutions, qu'on appellera *tours*, à savoir les cycles hamiltoniens dans le graphe complet des villes.

B.2 (TSPTW) : Traveling Salesman Problem with Time Windows

À chaque ville sont maintenant associées une durée de service et une fenêtre de temps pendant laquelle le service de la ville doit nécessairement débuter. La tournée peut donc arriver dans une ville avant l'ouverture de la fenêtre de temps, mais elle doit attendre ce moment pour débuter le service. Elle repart dès la fin de son service, au plus tard donc, à la date de fermeture de la fenêtre de temps plus la durée du service. Le coût d'un trajet entre deux villes est défini comme la somme de la durée du trajet et de la durée du service dans la ville de départ. L'objectif est maintenant de construire une tournée *(i)* valide au sens des fenêtres de temps, *(ii)* servant chaque ville une fois et *(iii)* minimisant le coût total de la tournée.

Le TSPTW est aussi utile à la modélisation de problèmes de tournées de véhicules qu'à la modélisation de problèmes d'ordonnancement disjonctif de tâches avec temps de set-up dépendant de la tâche précédemment exécutée.

B.3 Benchmark pour le TSPTW

Le site suivant propose une collection d'instances pour le TSPTW :

`http://iridia.ulb.ac.be/~manuel/tsptw-instances`

Les instances sont décrites par des fichiers texte suivant un format commun :

1. le nombre de villes (incluant le dépôt)
   – les villes sont naturellement numérotées, la première étant le dépôt.
2. la matrice des coûts de trajet entre toutes paires de villes :
   – la i-ème ligne décrit les coûts de trajet depuis la i-ème ville vers toutes les autres villes, en colonnes dans le même ordre, séparés par des espaces.
   – les coûts sont soit des entiers, soit des flottants positifs en notation anglaise (ex : `40.3521`).
3. les fenêtres de temps à chaque ville :
   – la i-ème ligne décrit la date de début et la date de fin de la fenêtre de temps de la i-ème ville, séparées par des espaces.
   – les dates sont représentées par des entiers positifs.

Dans un premier temps, on limitera les tests aux seules instances Dumas et Solomon/Pesant. Pour ce second groupe d'instances, on tronquera les décimales des valeurs de coûts (arrondi à l'entier inférieur).

B.4 Notations

On note n le nombre de villes, incluant le dépôt. On numérote les villes de 0 à $n-1$ avec 0 désignant le dépôt. On note $[a..b]$ l'ensemble des entiers compris entre a et b, (avec $a \geqslant b$). On note d_{ij} le coût (ou la distance pour le TSP) du trajet de la ville i à la ville j. On note enfin $[e_i, l_i]$ la fenêtre de temps de début de service de la ville i.

C Séance 1 : Modèle mathématique et modèle Choco du TSP.

Dans un premier temps, nous considérons la planification de la tournée d'un véhicule, minimisant la somme des distances (qu'on appellera également coûts) des trajets.

C.1 Modèle successeur du problème du cycle hamiltonien.

De nombreux problèmes de graphes, y compris le TSP, consistent à identifier des sous-graphes (i.e. à sélectionner un sous-ensemble d'arcs) vérifiant des propriétés particulières. Modéliser un tel problème au moyen de variables discrètes se fait naturellement de deux manières :

- au moyen de variables booléennes $x_e \in \{0, 1\}$ indicées sur les arcs $e \in E$: $x_e = 1$ si l'arc e est sélectionné ;
- au moyen de variables ensemblistes discrètes $S_i \subseteq V$ indicées sur les sommets $i \in V$: $j \in S_i$ si l'arc $(i, j) \in E$ est sélectionné.

Le premier encodage est privilégié en Programmation Linéaire, notamment, car il offre la possibilité de linéariser des propriétés plus complexes que le second. Le second modèle, appelé le *modèle successeur* est lui privilégié en Programmation Par Contraintes comme il est plus compact (et comme il n'est pas nécessaire de linéariser les contraintes). De plus, des variables entières $S_i \in V$ (et non plus ensemblistes) sont suffisantes quand il s'agit de modéliser des problèmes de chemins. Par exemple, le graphe de la

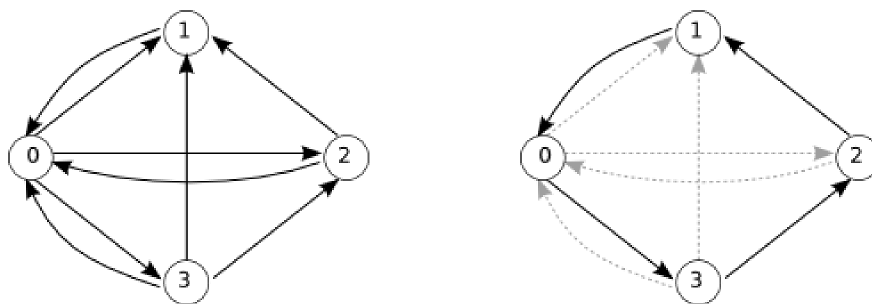

FIGURE 1 – Un graphe orienté à 4 sommets (gauche) et un chemin comme sous-graphe solution (droite)

figure 1 peut être représenté par quatre variables entières $S = (S_0, S_1, S_2, S_3)$ de domaines respectifs : $D_0 = \{1, 2, 3\}$, $D_1 = \{0\}$, $D_2 = \{0, 1\}$, $D_3 = \{0, 1, 2\}$. Le chemin-solution, illustré à droite, correspond à l'instanciation complète $S = (3, 0, 1, 2)$.

Question 1 *On considère le problème du cycle hamiltonien (orienté) dans un graphe orienté quelconque (i.e. potentiellement non-complet)* $G = (V, E)$ *avec* $|V| = n$ *:*

1. *adaptez le modèle successeur* S *à ce problème et précisez les conditions qu'une instanciation complète de* S *doit vérifier pour modéliser un cycle hamiltonien orienté ;*
2. *soit* $n = 6$ *et le graphe définit par l'ensemble* E *des arcs suivants :*

   $$\{(0,1),(0,2),(0,3),(0,4),(0,5),(1,0),(2,0),(2,1),(2,3),(2,4),(3,0),(3,2),(4,1),(4,3),(4,5),(5,0),(5,1),(5,2)\},$$

   représentez le graphe et déterminez le modèle successeur associé (domaine initial des variables). Quels sont les arcs que l'on peut naturellement éliminer ? Faîtes tourner l'algorithme de filtrage à la main *jusqu'à déduire l'unique solution de cette instance.*

Question 2 *On souhaite utiliser une contrainte globale d'élimination des sous-cycles. On propose la spécification suivante :* $\texttt{forest}(S_0, \ldots, S_{n-1})$ *est satisfaite si et seulement si il n'existe pas de séquence d'entiers* $i_1, \ldots, i_p$ *de longueur* $1 < p <= n$ *telle que* $S_{i_k} = i_{k+1}$ *pour tout* $k = 1, \ldots, p-1$ *et* $S_{i_p} = i_1$.

1. *pourquoi une telle contrainte n'est-elle pas directement applicable au problème du cycle hamiltonien ?*
2. *proposez un modèle* complet minimal *de satisfaction de contraintes pour le problème du cycle hamiltonien, basé sur cette contrainte.*

C.2 Modèle Choco du cycle hamiltonien

La contrainte `forest` n'est pas implémentée dans Choco. En revanche, on dispose de la contrainte `path` dont la spécification est la suivante :

$$\texttt{path}(S_0, S_1, \ldots, S_n) \iff \begin{array}{ll} S_i \in [1..n] & \forall i \in [0..n-1] \\ S_i \neq S_j & \forall i \neq j \in [0..n-1] \\ \texttt{forest}(S_0, \ldots, S_{n-1}) & \end{array}$$

Le filtrage de cette contrainte n'assure pas la cohérence d'arc.

Question 3 *Proposez un modèle Choco du problème de cycle hamiltonien dans le graphe* $G = (V, E)$.

Question 4 algorithme de filtrage pour la contrainte `path`.

1. *montrez qu'une instanciation partielle* $(S_0, \ldots, S_n)$*, cohérente avec* `path`*, définit une partition du graphe associé à S en chemins sommet-disjoints, telle que chaque sommet appartient à un unique chemin (éventuellement réduit à ce sommet) ;*
2. *pour tout sommet* $i \in [0..n]$*, on note* γ_i *le chemin passant par* i*, et* $e_i, f_i \in [0..n]$ *les extrémités initiale et finale de ce chemin ; proposez un algorithme de calcul des* e_i *et* f_i *;*
3. *traduisez, en terme de ces chemins, l'opération d'instanciation d'une variable* S_i *à une valeur* j.
4. *proposez un algorithme de filtrage, basée sur les* e_i, f_i*, et qui se déclencherait à chaque nouvelle instanciation* $S_i = j$*, avec* $i \in [0..n-1]$.

C.3 Modèle Choco du TSP

Question 5 *Quelles sont les API disponibles dans Choco pour modéliser la contrainte globale* `element` *?*

Question 6 *On considère maintenant un graphe complet avec* n *sommets/villes, dont le dépôt numéroté* 0*, et* $d_{ij} \in Z_+$ *le coût de trajet de* i *vers* j*, pour toute paire de sommets* (i, j)*. Proposez un modèle Choco du TSP.*

C.4 Implémentation.

Un début d'implémentation vous est proposé dans le package `emn.fr`. Il comprend les 6 classes suivantes :

- **solomon/Instance** : lit un fichier d'instance du TSPTW et enregistre les données *(à implémenter)* ;
- **constraint/PathConstraint** : définit la contrainte globale `path` ; accessible par la méthode statique `Constraint build(IntegerVariable[] s);`
- **TSPCmd** lance les opérations de parsing/modélisation/résolution en ligne de commande :
  `$> java -cp` *<liste des jar>* `emn.fr.TSPCmd -f` *path/dir/benchs/* `-tl 10 -rp 0 -se 0`
- **TSPModeler** : crée le modèle Choco du TSP (`buildModel()` *à implémenter*), paramétrise le solveur (`buildSolver()` *à personnaliser*) et lance la résolution (`solve()` *à implémenter*)
- **TSPParser** définit une instance et un fichier et appelle la fonction de lecture de `Instance`.
- **TSPSettings** : définit des paramètres de résolution.

Question 7 *D'ici la prochaine séance :*

- *téléchargez le code sur Campus et importez le projet sous Idea IntelliJ ou sous Eclipse ;*
- *complétez le code des classes* **solomon/Instance** *et* **TSPModeler**
- *mettez en place une procédure de test benchmarks qui lance la résolution sur plusieurs instances : vous pouvez, au choix, utiliser un langage de script, implémentez en java une classe de test, ou utilisez la classe* **TSPCmd** *;*
- *expérimentez votre modèle sur* toutes *les instances Dumas et Solomon/Pesant*

D Séance 2 : Implémentation de stratégies de recherche pour le TSP

D.1 Stratégies de recherche avec Choco

La stratégie de recherche détermine comment construire l'arbre de recherche et dans quel ordre parcourir ses nœuds. Elle se divise donc en deux :
- stratégie de choix du prochain nœud à évaluer
- stratégie de choix de la méthode de séparation du nœud choisi (branchement)

En programmation par contraintes, les algorithmes classiques de backtracking (satisfaction) ou de branch-and-bound (optimisation) sont basées, pour la première stratégie, sur la règle de profondeur d'abord (Depht-First Search ou DFS) : les nœuds sont empilés et sélectionnés dans un ordre LIFO : premier noeud-fils puis prochain noeud-frère. Il existe également une **stratégie de branchement** standard. Elle consiste à choisir une variable x non encore instanciée, et une valeur v dans son domaine, puis à séparer l'espace de recherche en deux : dans le premier nœud-fils, on prend la décision (i.e. on ajoute la contrainte) $x = v$, tandis que dans le second nœud-fils, on ajoute la contrainte opposée $x \neq v$. Cette stratégie construit un arbre binaire de recherche. Une variante consiste à construire un arbre n-aire en créant une branche i pour chaque valeur v_i du domaine de x, associée à la décision $x = v_i$. Ces stratégies de branchement standard reposent donc sur deux algorithmes de choix (heuristiques) qu'il est possible de personnaliser pour son problème :
- l'**heuristique de choix de variable** détermine la variable x à instancier ;
- l'**heuristique de choix de valeur** : détermine la valeur v à laquelle instancier x dans un branchement binaire, ou bien l'ordre $v_1, \ldots, v_n$ dans lequel évaluer les valeurs de x dans un branchement n-aire.

Choco dispose d'un ensemble d'heuristiques de recherche pré-implémentées (voir annexe à la documentation), basées sur les interfaces suivantes :
- **BranchingStrategy** définit une stratégie de branchement : composée d'une heuristique de choix de variable et d'une heuristique de choix de valeur
- **VarSelector** définit une heuristique de choix de variable : retourne la prochaine variable à instancier
- **ValSelector** définit une heuristique de choix de valeur pour un branchement binaire ou n-aire : retourne la prochaine valeur à laquelle instancier la variable préalablement choisie
- **ValIterator** définit une heuristique de choix de valeur pour un branchement n-aire : ordonne les valeurs de la variable préalablement choisie et retourne successivement ces valeurs dans cet ordre

Question 8 *Soit* `IntegerVariable[] x` *un tableau de variables entières d'un modèle Choco. Décrire de 3 manières différentes l'appel à la stratégie de recherche suivante :*
on crée un arbre n-aire basé sur l'affectation des variables x à chacune de leurs valeurs : à chaque nœud, on choisit la première variable non instanciée parmi $x_1, \ldots, x_n$, prises dans cet ordre, et on y associe chacune des valeurs de son domaine, sélectionnéees par ordre croissant de valeur.

Adapter la stratégie de recherche à un problème particulier, plus encore s'il s'agit d'un problème d'optimisation, est aussi primordial que d'améliorer son modèle. Choco offre donc la possiblité d'implémenter les interfaces ci-dessus pour définir ses propres heuristiques de recherche.

D.2 Heuristiques orientées optimisation pour le TSP

Soit un modèle d'optimisation de contraintes du TSP basé sur les variables successeurs $S_i = j$: j est le sommet successeur du sommet i dans le tour, et les variables de coût $c_i \in \mathbb{Z}$ est la distance de l'arc sortant du sommet i dans le tour. On considère les stratégies de branchement suivantes pour le TSP :

1. on crée un branchement binaire basé sur l'instanciation $S_i = j$ où les sommets i et j sont sélectionnés dans l'ordre lexicographique ;
2. on crée un branchement binaire basé sur l'instanciation $S_i = j$ où les sommets i sont choisis aléatoirement et les sommets j sont sélectionnés dans l'ordre lexicographique ;
3. on crée un branchement binaire basé sur l'instanciation $S_i = j$ où les sommets i sont choisis aléatoirement et j est un plus proche voisin de i ;
4. on crée un branchement n-aires basé sur l'instanciation de S_i à chacune de ses valeurs j possibles, où les sommets i sont choisis aléatoirement et les sommets j sont sélectionnés par ordre de distance croissante de i.

Question 9 *Pour chacune de ces stratégies, déterminez :*

- *si la stratégie est statique ou dynamique*
- *les variables (successeurs S ou coût c) sur lesquelles portent les décisions*
- *si la stratégie peut être implémentée entièrement au moyen de heuristiques pré-définies de Choco, et sinon, quelles sont les interfaces à implémenter*

Implémentez.

On considère la stratégie suivante :
on crée un branchement binaire basé sur l'instanciation $S_i = j$ où le sommet i choisi minimise la distance à son voisin le plus proche, et j est un de ces plus proches voisins ;

Question 10
- *Proposez deux implémentations pour cette stratégie et comparez ;*
- *Modifiez de manière à ce que en cas d'égalité dans le choix de la variable, celle de plus petit domaine soit choisie.*

On considère la stratégie suivante :
on crée un branchement binaire basé sur l'instanciation $c_i = d_{ij}$ où le sommet i choisi minimise la distance à son voisin le plus proche (et minimise la taille du domaine de c_i en cas d'égalité), et j est un de ces plus proches voisins ;

Question 11 *Comparez avec la stratégie précédente*

- *cette stratégie est-elle complète ?*
- *implémentez*

Question 12 *D'ici la prochaine séance :*

- *terminez l'implémentation ;*
- *réfléchissez/implémentez à d'autres heuristiques orientés optimisation pour le TSP ;*
- *mettez en place une procédure de test benchmarks qui lance la résolution sur plusieurs instances au moyen des diverses heurstiques implémentées.*
- *expérimentez votre modèle sur* toutes *les instances Dumas et Solomon/Pesant*

E Séance 3 : Implémentation d'une contrainte pour le TSPTW

E.1 Modèle du TSPTW en Choco

On considère maintenant, pour chaque ville i, une fenêtre de temps $[e_i, l_i]$ durant laquelle doit débuter le service de i par un véhicule partant du dépôt au temps 0. Dans les instances considérées, le temps de service est réduit à 0, et le temps de trajet est égale au coût d_{ij} entre deux villes i et j. Le véhicule est ainsi autorisé à arriver dans la ville i avant le moment e_i, mais il ne peut en repartir avant ce moment. Sinon, le véhicule doit nécessairement arriver entre les moments e_i et l_i (compris) et il peut en repartir aussitôt. La date au plus tard du retour au dépôt l_0 est également connue. Le problème consiste à trouver un tour de durée minimale pour le véhicule visitant chaque ville une et une seule fois, dans les fenêtres de temps imposées.

Question 13 *Proposez un modèle Choco du TSPTW en identifiant les contraintes redondantes.*

Question 14 *On appelle contraintes de temps les relations entre variables successeurs et variables de temps. Chacune de ces contraintes est une expression logique complexe (voir Documentation > The model > Reified Constraints) qui est, traité de manière particulière par le Solver de Choco :*

- *par défaut, quelle est l'implémentation utilisée dans le Solver pour cette contrainte ? quel est le degré de filtrage obtenu ?*
- *quelles sont les limites d'application d'une telle implémentation ? donnez un ordre de grandeur de la taille de cette implémentation, et du temps de filtrage, en fonction de n et de l la taille maximale des fenêtres de temps.*
- *l'implémentation alternative consiste à décomposer la contraintes en plusieurs contraintes réifiées ; explicitez cette décomposition et le filtrage associé. Exhibez un exemple de défaut de filtrage par rapport à la première implémentation.*

Question 15 *Une contrainte d'inégalité. Soient deux variables discrètes bornées x, y et la contrainte $x \leqslant y$:*

- *sous quelle condition (sur les bornes de x et y) la contrainte est nécessairement vérifiée ?*
- *sous quelle condition (sur les bornes de x et y) la contrainte possède-t'elle au moins une solution ?*
- *quelles sont les valeurs de x (resp. y) qui n'apparaissent dans aucune solution de la contrainte ?*
- *décrire un algorithme de filtrage aux bornes et les évènements fins de réveil correspondants pour cette contrainte.*

E.2 Projet à rendre et concours.

Il vous est demandé de développer un solveur de contraintes pour le TSPTW, basé sur la librairie Choco et le code d'entrée `TSPCmd`, et de l'expérimenter, en satisfaction et en optimisation, a minima sur les instances de Dumas ($n = 20$, $n = 40$) et les instances de Solomon-Pesant rc201 et rc202, avec une durée limite d'exécution égale à 10 minutes. Vous consignerez dans un rapport écrit les détails d'implémentation et les résultats d'expérimentation (requis ou complémentaires) de votre solveur. Vous fournirez également le jar exécutable, en modes satisfaction et optimisation, sur ces instances. Nous exécuterons tous les projets dans des environnements identiques et établirons un classement des meilleures méthodes développées, fonction du nombre de preuves d'optimalité, ou des meilleures solutions, et/ou des meilleurs temps.

Évaluation. Il s'agit d'un **projet d'étude et de développement individuel noté** : les similarités avérées entre codes seront sanctionnées dans la note. Toute référence empruntée devra être citée. Seront sanctionnés également, les projets qui ne répondent pas aux consignes ci-dessous.

La note globale du projet sera établie selon les critères suivants :

- justesse des méthodes implémentées, pertinence (à justifier dans le rapport), difficulté, originalité
- résultats du classement
- justesse et qualité du rapport : description et justification des méthodes, description et analyse des résultats expérimentaux

Par ailleurs, la qualité de l'implémentation donnera lieu à une évaluation dans le cadre de l'UV Ingénierie du Logiciel.

Consignes de rendu du projet. Vous déposerez, le **vendredi 17 décembre 2010** au plus tard, une archive complète respectant obligatoirement [1] le contenu et le format suivant, présenté ici pour un élève nommé Toto :

- l'archive `tsptwToto.zip` (format `zip` uniquement) d'un répertoire `tsptwToto` comprenant, à l'exception de toute autre chose :
- le fichier `tsptwToto.jar` exécutable
- deux fichiers texte `optionsat.txt` et `optionopt.txt` contenant les options en ligne de commande pour l'exécution du jar respectivement, en mode satisfaction, et en mode optimisation. L'exécution du solver sera lancée, depuis le répertoire `tsptwToto` par la commande bash suivante :

  ```
  java  -cp ../choco-2.1.1-xx.jar:tsptwToto.jar
        -f ../benchsrep
        -tl ''10'' -v ''VERBOSE.QUIET''
        -o ''outToto.txt''
        `cat optionsat.txt`
  ```

  `../benchrep` désigne un répertoire contenant l'ensemble des fichiers d'instance à résoudre, `-tl ''10''` indique que la résolution dure au plus 10 minutes, `-o ''outToto.txt''` spécifie le fichier de sortie des résultats.
- le répertoire `src/` du code source compilable, exécutable et documenté (et uniquement les sources `.java` !)
- un rapport (format `pdf` uniquement) décrivant les méthodes de résolution implémentées et testées, la table des résultats sur les instances testées, une analyse de performance des différentes méthodes de résolution, des perspectives d'évolution du logiciel.

1. afin de permettre le traitement automatique de l'ensemble des projets.