Projet IAP1 - Groupe 4.2

Développement d'un solveur de programmes linéaires

Octobre 2007

1 Les programmes linéaires en nombres entiers

De nombreux problèmes d'optimisation NP-difficiles (c'est-à-dire dont le temps de résolution est une fonction exponentielle des caractéristiques du problème) peuvent être résolus par *programmation mathématique*. Cette méthode consiste à modéliser le problème comme la minimisation d'une fonction (appelée fonction économique) sous certaines contraintes (égalités et inégalités). Nous nous plaçons dans le cas de la *programmation linéaire en nombres entiers*, c'est-à-dire la programmation mathématique où les variables sont à valeur dans l'ensemble {0,1} et la fonction économique ainsi que les contraintes sont linéaires en fonction des variables. Nous nous limitons également à des fonctions économiques et des contraintes dont les coefficients sont des entiers relatifs. La forme générale d'un programme linéaire en nombres entiers est :

$$(P)\begin{cases} Min & f(x) \\ s.c. & \\ & x \in X \subseteq \{0,1\}^n \end{cases}$$

où f est la fonction à minimiser, n désigne le nombre de variables du problème, x est un vecteur de taille n et X désigne l'ensemble des solutions admissibles (c'est-à-dire qui vérifient les contraintes) du problème.

Plus concrètement, le programme mathématique (Ex) est un exemple de programme linéaire en nombre entiers

$$(Ex)\begin{cases} Min & x_1 + 3x_2 + x_3 - 2x_4 & & \\ s.c. & & & \\ & x_1 + x_2 - x_3 & \geq & 1 \\ & -x_1 + x_3 + x_4 & \leq & 1 \\ & x_1 + x_4 & \leq & x_2 + x_3 \\ & x_1 + x_3 + x_4 & = & 2 \\ & x \in \{0,1\}^4 & & \end{cases}$$

Le projet se décompose en deux parties. La première consiste à transformer le problème afin de le mettre sous forme standard et la seconde à le résoudre.

Première partie : mise sous forme standard

Structures de données
Il y a quatre types de structures de données à manipuler dans ce projet. Les litteraux sont construits à partir d'une chaîne de caractères (qui représente le nom d'une variable) et d'une coefficient ou sont des constantes. Les expressions sont des listes de litteraux. Les contraintes sont construites par comparaison de deux expressions avec les opérateurs de comparaison classiques ($\leq, =, \geq$). Enfin un programme linéaire est composé d'une fonction à minimiser et d'une liste de contraintes.

Question 1
Écrire une fonction qui change le signe d'une expression. Par exemple l'expression $x_1 + x_2 - x_3$ est transformée en $-x_1 - x_2 + x_3$.

Question 2
Une expression est sous forme standard si elle ne contient pas plus d'une occurence de chaque variable et d'une constante. Écrire une fonction qui met une expression sous forme standard.

Question 3
On souhaite que dans, la forme standard du programme, le membre droit de chaque contrainte soit une constante et le membre gauche une expression sous forme standard. De plus, les contraintes ne doivent être que des contraintes d'infériorité et d'égalité et aucune variable ne doit posséder un coefficient nul. Écrire une fonction qui transforme le programme pour le mettre sous forme standard.

Seconde partie : résolution des programmes linéaires

Nous proposons maintenant de résoudre les programmes linéaires sous forme standard. Pour ce faire il est nécessaire de pouvoir associer aux variables une valeur numérique. Nous allons utiliser pour cela un environnement, c'est-à-dire une liste qui associe à chaque variable sa valeur numérique.

La recherche de solution correspond à un arbre binaire de recherche. On fixe la valeur de la première variable à 0 dans le sous arbre gauche, à 1 dans le sous arbre droit et on itère le procédé sur le noeud le plus à gauche jusqu'à une condition d'arrêt (par exemple toutes les feuilles de l'arbre ont été développées). La figure 1 représente l'ordre dans lequel les sommets doivent être développés. On associe donc à chaque sommet de l'arbre de recherche un environnement courant.

Question 4
(a) Après avoir défini la structure d'environnement, écrire une fonction qui évalue une expression selon un environnement. Si une des variables de l'expression n'a pas de valeur la fonction échoue. Écrire ensuite une fonction qui évalue une borne inférieure d'une expression selon un environnement. Cette borne se calcule en fixant les variables qui n'ont pas de valeur de façon à minimiser la valeur de l'expression.

(b) Une contrainte d'égalité ou d'inégalité (infériorité) est satisfiable dans un environnement si la borne inférieure calculée à la question précédente est inférieure ou égale à la constante du membre droit de

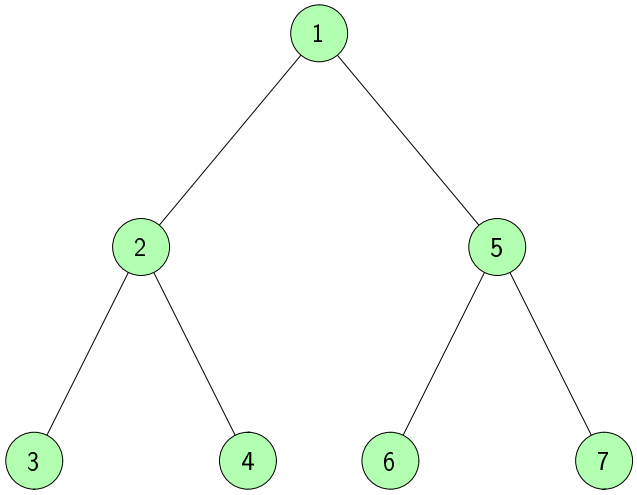

FIG. 1 – Ordre de développement de l'arbre pour un programme à deux variables.

la contrainte dans cet environnement. Un programme est satisfiable si toutes les contraintes sont satisfiables. Écrire une fonction qui vérifie qu'un programme est satisfiable dans un environnement.

Question 5
Après avoir défini le type des arbres de recherche, implanter l'algorithme de recherche de solutions du problème sur l'arbre de recherche. Attention : un noeud de l'arbre ne doit être développé que si toutes les contraintes sont satisfiables dans l'environnement courant. De plus, les variables doivent être fixées par ordre lexicographique (ordre de comparaison classique des chaînes de caractères). L'algorithme doit renvoyer la liste des solutions et le nombre de noeuds développés.

Question 6
Améliorer l'implantation de l'algorithme afin que le noeud courant ne soit pas développé si la borne inférieure de la fonction économique dans l'environnement courant est supérieure ou égale à la valeur d'une solution connue. L'algorithme doit renvoyer une solution optimale et le nombre de noeuds développés.

Question 7 (A ne faire que si toutes les autres questions ont été traitées)
Reprendre l'algorithme pour que le noeud développé soit celui qui possède la plus petite borne inférieure. Si la borne inférieure de la fonction économique dans l'environnement de ce noeud est supérieure ou égale à la valeur d'une solution connue alors l'algorithme doit terminer. L'algorithme doit renvoyer une solution optimale et le nombre de noeuds développés.

2 Travail à rendre

Le projet est à réaliser en OCaml **individuellement**. Il sera accompagné d'un **dossier** contenant impérativement la description des choix faits, la description des types et des fonctions. Pour chaque fonction, on donnera impérativement l'interface complète (dans le code en commentaire et dans le rapport pour les fonctions présentées). Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés. Sur le site du cours figure un petit document sur ce que l'on attend dans un rapport. Consultez le !

Le projet (dossier + copie du listing de code) est à rendre au secrétariat **le 7 novembre à 16h30 au plus tard**. Le numéro du groupe, ainsi que le nom du chargé de TP, devront figurer en gros, et en rouge sur la page de garde. Le fichier .ml contenant votre code devra être déposé electroniquement au plus tard **le 7 novembre à minuit** (la procédure sera envoyée par email).

Les soutenances seront organisées une dizaine de jours après le retour des projets. Il vous faudra consulter les panneaux d'affichage et votre courrier electronique pour obtenir l'ordre de passage.

Enfin n'attendez pas pour vous mettre au travail ! Un projet se travaille dès la remise du sujet afin d'avoir le temps de laisser murir la solution et de poser des questions au client (dans votre cas, votre chargé de TP).