Rapport recherche opérationnelle TP
Sujet 2

CORBELLARI Nolan, CANON Ayoub

Département Sciences du Numérique — Deuxième année
2023–2024

Contents

Questions préliminaires

1. À chaque noeud, le solveur va résoudre un problème d'optimisation sous contraintes. Chaque valeur de $x(i)$ est à priori binaire et indique si oui ou non l'objet est inclus dans le sac. Cependant le solveur peut trouver une solution comportant un ou plusieurs $x(i)$ dont la valeur n'est pas binaire mais flottante. C'est sur ce critère que l'algorithme se base pour créer les branches à partir d'un noeud. Lorsqu'un résultat obtenu par un solveur sur un noeud de profondeur i comporte une ou plusieurs valeurs non booléenne (notons les $x_{\lambda 1}, x_{\lambda 2}, ..., x_{\lambda m}$) dans l'ordre lexicographique), l'algorithme selectionne x_{λ_1} car c'est le premier dans l'ordre lexicographique et il créée deux noeuds de profondeur $i+1$ et une arrête possède la condition $x_{\lambda_1} = 1$ et l'autre possède la condition $x_{\lambda_1} = 0$

2. L'algorithme calcule la borne supérieur et inférieure de la manière suivante :

   - borne supérieur :

   $$\forall i \in [1..n], BorneSup(x[i]) = min(x[i], 1)$$

   - borne inférieure :

   $$\forall i \in [1..n], BorneInf(x[i]) = max(x[i], 0)$$

3. L'algorithme calcule la TA, la TO et la TR de la manière suivante :

   - TA : Le problème d'optimisation sous contraintes ne possède pas de solution.
   - TO : Le problème d'optimisation sous contraintes possède une solution moins intéressante que celle déjà enregistrée.
   - TR : Le problème d'optimisation sous contraintes possède une solution plus intéressante que celle déjà enregistrée.

4. La stratégie de parcours est un parcours en profondeur de l'arbre crée en prenant d'abord le fils gauche plutot que le fils droit.

Code et analyse

Détail de l'implémentation

Borne supérieur

```
        function calculBorne(model::Model,mode::String="1")
        if mode == "1"
            % Creation des ratios
            r = map((x,y) -> x./y,model.price,model.weight)

            #Obtention du max des ratios
            max_ri,index = findmax(r)

            #Calcul borne sup
            borneSup = (model.capacity)*max_ri

            #Renvoie fonction
            return borneSup,index
        else
            # Trier les objets par ratio decroissant
            r = map((x, y) -> x / y, model.price, model.weight)
            sorted_indices = sortperm(r, rev=true)

            # Initialiser la capacite restante et la borne superieure
            remaining_capacity = model.capacity
            borneSup = 0.0

            # Parcourir les objets tries
            for i in sorted_indices
                if model.weight[i] <= remaining_capacity
                    # Ajouter l'objet entier si possible
                    borneSup += model.price[i]
                    remaining_capacity -= model.weight[i]
                else
                    # Ajouter la fraction de l'objet
                    fraction = remaining_capacity / model.weight[i]
                    borneSup += fraction * model.price[i]
                    break
                end
            end

            return borneSup, sorted_indices[1]
        end
    end
```

Pour le calcul de la borne 1, nous avons simplement calculé le ratio de chaque objet et nous avons pris le maximum de ces ratios pour le multiplier par la capacité du sac. Nous avons ensuite retourné la borne supérieure ainsi que l'indice de l'objet

de ratio maximum. Pour la borne 2, nous avons trié les objets par ratio décroissant et nous avons ajouté les objets tant que la capacité du sac le permettait. Si la capacité du sac ne permettait pas d'ajouter un objet entier, nous avons ajouté une fraction de cet objet. Nous avons ensuite retourné la borne supérieure ainsi que l'indice de l'objet qui a été ajouté en dernier.

Règle de séparation

Dans notre implémentation, la règle de séparation est basée sur l'indice retourné par la fonction calculant la borne supérieure. Cet indice représente l'objet avec le meilleur ratio (ou le dernier ajouté pour la borne 2) et sert à déterminer le point de branchement. Deux branches sont générées : dans la première, l'objet est inclus dans le sac à dos (fils gauche), tandis que dans la seconde, l'objet n'est pas inclus (fils droit). Cette approche permet une exploration efficace de l'espace des solutions en se focalisant sur les objets les plus prometteurs.

Tests de sondabilité (TA, TO, TR)

```
function testTA(model::Model)
    return model.capacity < 0
end

function testTO(upperbound::Float64,bestSol::Float64)
    return upperbound < bestSol
end

function testTR(xi::Float64)
    return xi != 0 && xi != 1
end
```

- TA : Si le poids total des objets sélectionnés est supérieur à la capacité du sac, la branche est non sondable.
- TO : Si la borne supérieure est inférieure à la meilleure solution trouvée, la branche est également non sondable.
- TR : Si la décision de prendre ou non l'objet est déjà prise, la branche est redondante et donc non sondable.

Stratégie d'exploration

Notre stratégie d'exploration utilise un approche gloutonne basée sur les bornes supérieures calculées. Après la génération des branches, nous sélectionnons d'abord la branche avec la borne supérieure la plus élevée pour l'exploration. Cette méthode permet de prioriser les chemins les plus prometteurs, augmentant ainsi les chances de trouver rapidement une solution optimale.

Choix de structure de données

```
mutable struct Model
     price::Vector{Int64}
     weight::Vector{Int64}
     capacity::Int64
     x::Vector{Float64}
     value::Int64
 end
```

La structure de données clé de notre implémentation est la structure 'Model'. Elle encapsule toutes les informations nécessaires pour représenter un nœud dans l'arbre de recherche du Branch-and-Bound :

- *price* et *weight* sont des vecteurs d'entiers représentant respectivement les valeurs et les poids des objets. Ces vecteurs sont essentiels pour le calcul des bornes et la détermination des objets à inclure ou à exclure lors de la création des branches.
- *capacity* est un entier représentant la capacité restante du sac à dos à un nœud donné. Elle est cruciale pour les tests de sondabilité, en particulier le test TA.
- x est un vecteur de nombres flottants représentant la solution partielle à un nœud donné, où chaque élément indique si un objet spécifique est inclus (1.0) ou exclu (0.0) du sac.
- *value* est un entier indiquant la valeur totale des objets inclus dans le sac à un nœud donné. C'est un indicateur clé de la performance d'une solution partielle.

Comparaison des calculs de bornes

En moyenne, l'écart relatif entre le temps de calcul utilisant la borne 1 et le temps de calcul utilisant la borne 2 est d'environ 0.001%. Cet écart s'explique par le fait que le calcul de la borne 2 trie directement les objets par ratio de prix sur poids décroissant, ce qui permet de limiter les branchement aux objets les plus pertinents plutôt que de brancher sur celui qui a le ratio maximal.

Améliorations possibles