Accélération de l'algorithme de séparation et évaluation pour les diagrammes de décision grâce à la mémoïsation

Vianney Coppé*, Xavier Gillard, Pierre Schaus

UCLouvain

{vianney.coppe, xavier.gillard, pierre.schaus}@uclouvain.be

Résumé

L'optimisation discrète avec diagrammes de décision permet de résoudre des problèmes complexes sur base d'un modèle de programmation dynamique. L'approche consiste en un algorithme de séparation et évaluation dans l'espace des états du modèle, où les bornes dérivent de diagrammes de décision. Cet article explique comment accélérer l'algorithme en mémorisant certains résultats intermédiaires afin d'éviter au maximum l'exploration répétée de certaines parties de l'espace de recherche. Les résultats expérimentaux montrent que cette accélération est significative.

Mots-clés

Optimisation discrète, diagrammes de décision, programmation dynamique, séparation et évaluation.

Abstract

Discrete optimization with decision diagrams is a framework for solving complex problems based on a dynamic programming model. It consists in a branch-and-bound algorithm applied on the state space of the model, where the bounds are derived from decision diagrams. This paper explains how storing some intermediate results can help avoid the repeated exploration of some parts of the search space and speed up the algorithm. The experiments show that it results in a significant speed up.

Keywords

Discrete optimization, decision diagrams, dynamic programming, branch-and-bound.

1 Introduction

L'optimisation discrète avec *diagrammes de décision* (DDs) est une méthode générique d'optimisation introduite en 2016 par Bergman et al. [3]. Cette approche est basée sur les modèles de *programmation dynamique* (PD) pour résoudre des problèmes complexes. Lorsque leur résolution directe par PD est hors de portée en termes à la fois de temps d'exécution et de mémoire, un algorithme de *séparation et évaluation* (S&E) est proposé. Celui-ci repose sur des DDs *approximés – restreints* ou *relaxés* – pour explorer l'espace des états tout en élaguant certains sous-problèmes sur base des bornes fournies par les DDs. L'approche permet donc de profiter de la compacité des modèles PD tout en offrant la possibilité de résoudre des problèmes de grande taille.
Cependant, l'algorithme tel que présenté originellement [3] ne possède aucun mécanisme qui permette d'éviter le traitement répété et inutile d'un même sous-problème. En effet, hormis la file constituée de l'ensemble des sous-problèmes restant à explorer, seules des bornes sont gardées en mémoire. Cela n'empêche un sous-problème ni d'être traité à différents stades de l'exécution de la S&E ni d'être traversé à de multiples reprises dans les DDs approximés. Or, le propre des modèles PD est de décomposer récursivement un problème difficile en un ensemble de sous-problèmes plus faciles à résoudre, avec un fort recoupement entre ceux-ci. Dans cet article, nous introduisons un mécanisme de mise en cache permettant de stocker un *seuil d'exploration* pour chaque sous-problème, indiquant à partir de quelle valeur il est nécessaire de traiter à nouveau le sous-problème en question, évitant ainsi de nombreux calculs redondants.

2 Notions préliminaires

Soit un problème d'optimisation discrète $\mathcal{P}$ défini comme $\max\{f(x) \mid x \in D, x \in C\}$ avec f la fonction objective à maximiser, $x = \langle x_0, \dots, x_{n-1} \rangle$ un vecteur de variables appartenant à $D = D_0 \times \dots D_{n-1}$ avec $x_i \in D_i$ et l'ensemble C représentant les contraintes imposées à x. Cette section présente comment modéliser et résoudre $\mathcal{P}$ grâce aux DDs.

2.1 Programmation dynamique

Un modèle PD pour $\mathcal{P}$ est constitué d'un *espace d'états* S partitionné en $n+1$ ensembles $S_0, \dots, S_n$ correspondant aux étapes successives du modèle. On distingue plusieurs états particuliers : l'état *racine* $\hat{r}$, *terminal* $\hat{t}$ et *non-admissible* $\hat{0}$. Les fonctions $t_i : S_i \times D_i \to S_{i+1}$ et $h_i : S_i \times D_i \to \mathbb{R}$ pour $i = 0, \dots, n-1$ déterminent respectivement les connexions entre les états d'étapes successives et la valeur associée à celles-ci. La *valeur racine* v_r permet de modéliser les constantes de la fonction objective. Sur base d'un tel modèle, la solution optimale est obtenue en résolvant :

$$\begin{aligned} \max\ & f(x) = v_r + \sum_{i=0}^{n-1} h_i(s^i, x_i) \\ \text{tel que } & s^{i+1} = t_i(s^i, x_i), i = 0, \dots, n-1 \\ & s^i \in S_i, i = 0, \dots, n \text{ et } x \in D \cap C. \end{aligned} \tag{1}$$

*L'auteur principal est doctorant.

2.2 Diagrammes de décision

Les DDs permettent de représenter un ensemble de solutions au modèle PD sous forme d'un graphe acyclique dirigé $\mathcal{B} = (U, A, \sigma, l, v)$ composé d'un ensemble de nœuds U organisé en couches $L_0, \ldots, L_n$ et d'un ensemble d'arcs A. À chaque nœud $u \in U$, la fonction σ fait correspondre un état du modèle. De manière similaire, chaque arc $a = (u_i \xrightarrow{d} u_{i+1})$ reliant deux nœuds de couches successives $u_i \in L_i, u_{i+1} \in L_{i+1}$ matérialise une transition, associée à une décision $l(a) = d \in D_i$ assignée à la variable x_i et à une valeur $v(a)$. L'algorithme 1, dont les lignes grisées seront expliquées en Section 3, décrit la compilation d'un DD enraciné en un nœud u_r tel que $\sigma(u_r) \in S_i$, commençant par un première couche L_i contenant seulement u_r. Ensuite, à partir d'une couche L_j, on génère la couche L_{j+1} en appliquant toutes les transitions possibles du modèle PD aux nœuds de la couche L_j, jusqu'à atteindre L_n constituée du seul *nœud terminal* t. Si l'on dispose d'un chemin $p_1 : r \rightsquigarrow u_r$, sa combinaison avec chaque chemin $p_2 : u_r \rightsquigarrow t$ forme une solution du problème $\mathcal{P}$ dont la valeur est donnée par $v(p_1 \cup p_2) = v_r + \sum_{i=0}^{n-1} v(a_i)$ où a_i est l'arc en i-ième position du chemin.
Pour des problèmes complexes, compiler un DD exact est aussi coûteux que de résoudre directement le modèle PD. Deux variantes de DDs sont donc utilisées pour calculer des bornes, ensuite employées dans un algorithme de S&E.

DD restreints Afin de trouver des solutions admissibles et donc des bornes inférieures les DDs *restreints* [4] sont compilés à la manière d'une recherche en faisceau. Lorsqu'une couche L_i contient plus de W nœuds, on retire les $|L_i| - W$ nœuds les moins prometteurs, sur base d'une heuristique, avant de générer la couche suivante. On obtient donc un DD qui contient un sous-ensemble des solutions de $\mathcal{P}$, fournissant ainsi des bornes inférieures.

DD relaxés Dans la même veine, il est possible d'obtenir des bornes supérieures grâce aux DDs *relaxés* [1, 2] en fusionnant les nœuds excédentaires plutôt qu'en les supprimant. Pour ce faire, un opérateur de fusion d'états doit être spécifié pour le modèle PD étudié, dont le résultat conserve toutes les transitions comprises dans le DD exact correspondant. En général, la fusion peut introduire des solutions non-admissibles et fournit donc des bornes supérieures.

2.3 Séparation et évaluation

Sur base des DDs approximés, l'algorithme 2 explore l'espace des états du modèle PD par S&E [3]. Une file de sous-problèmes – c'est-à-dire des nœuds de DDs – à traiter est maintenue, et pour chacun de ceux-ci, un DD restreint est compilé afin d'obtenir des solutions admissibles et éventuellement améliorer la meilleure solution courante. Ensuite, la construction d'un DD relaxé joue un rôle double : celui d'identifier l'ensemble des sous-problèmes restant à explorer – appelé *ensemble-coupe exact* – mais également de calculer une borne supérieure pour chacun d'eux.

Définition 1 (Ensemble-coupe)**.** *Soit un DD relaxé $\overline{\mathcal{B}}$ enraciné en u_r, un ensemble-coupe $EC(\overline{\mathcal{B}})$ est un sous-ensemble des nœuds de $\overline{\mathcal{B}}$ tel que chaque chemin $u_r \rightsquigarrow t$ dans $\overline{\mathcal{B}}$ traverse au moins un nœud de $EC(\overline{\mathcal{B}})$.*

Algorithme 1 Compilation d'un DD, restreint ou relaxé, enraciné au nœud u_r et d'une largeur maximale W.

1: $i \leftarrow$ indice de la couche contenant u_r
2: $L_i \leftarrow \{u_r\}$
3: **for** $j = i$ **to** $n - 1$ **do**
4: $\quad \acute{e}lagu\acute{e}s \leftarrow \emptyset$
5: $\quad$ **if** $j > i$ **then** // racines élaguées dans l'algorithme 2
6: $\quad\quad$ **for all** $u \in L_j$ **do**
7: $\quad\quad\quad$ **if** $Cache.contains(\sigma(u))$ **and** $v(u) \leq \theta(Cache.get(\sigma(u)))$ **then**
8: $\quad\quad\quad\quad \acute{e}lagu\acute{e}s \leftarrow \acute{e}lagu\acute{e}s \cup \{u\}$
9: $\quad L'_j \leftarrow L_j \setminus \acute{e}lagu\acute{e}s$
10: $\quad$ **if** $|L'_j| > W$ **then**
11: $\quad\quad$ restriction ou relaxation de L'_j jusque W nœuds
12: $\quad L_{j+1} \leftarrow \emptyset$
13: $\quad$ **for all** $u \in L'_j$ **do**
14: $\quad\quad$ **if** $v(u) + \overline{v}_{gr.}(u) \leq \underline{v}$ **then**
15: $\quad\quad\quad$ **continue**
16: $\quad\quad$ **for all** $d \in D_j$ **do**
17: $\quad\quad\quad$ créer nœud u' avec $\sigma(u') = t_j(\sigma(u), d)$
18: $\quad\quad\quad$ créer arc $a = (u \xrightarrow{d} u')$ avec $v(a) = h_j(\sigma(u), d)$ et $l(a) = d$
19: $\quad\quad\quad$ $L_{j+1} \leftarrow L_{j+1} \cup \{u'\}$, $A \leftarrow A \cup \{a\}$

Par ailleurs, un nœud u de $\overline{\mathcal{B}}$ est dit *exact* si on obtient le même état PD en suivant tous les chemins $u_r \rightsquigarrow u$ dans $\overline{\mathcal{B}}$. Un ensemble-coupe *exact* (ECE) est un ensemble-coupe contenant uniquement des nœuds exacts. Il existe plusieurs manières de choisir l'ensemble-coupe exact, la plus populaire étant de sélectionner tous les nœuds appartenant à la couche la plus profonde dont tous les nœuds sont exacts.

2.4 Bornes supérieures

Étant donné un DD relaxé $\overline{\mathcal{B}}$, la *borne locale* [5] peut être calculée pour chaque nœud de $ECE(\overline{\mathcal{B}})$.

Définition 2 (Borne locale)**.** *Soit un DD relaxé $\overline{\mathcal{B}}$ et un nœud $u \in \overline{\mathcal{B}}$, la borne locale de u dans $\overline{\mathcal{B}}$ est définie par :*

$$\overline{v}_{loc.}(u \mid \overline{\mathcal{B}}) = \begin{cases} v^*(u \rightsquigarrow t \mid \overline{\mathcal{B}}), & \textit{si } (u \rightsquigarrow t) \in \overline{\mathcal{B}}, \\ -\infty, & \textit{sinon.} \end{cases} \quad (2)$$

Avec $v^*(u \rightsquigarrow t \mid \overline{\mathcal{B}})$ la valeur du meilleur chemin $u \rightsquigarrow t$ dans $\overline{\mathcal{B}}$.

D'autre part, une *borne supérieure grossière* $\overline{v}_{gr.}(u) \geq v^*(u \rightsquigarrow t \mid \mathcal{B})$ est calculée pour chaque nœud u avant de considérer les transitions qui en découlent [5], voir ligne 14 de l'algorithme 1. Soit $v^*(u \mid \mathcal{B})$ la plus haute valeur obtenue par un chemin terminant en u dans $\mathcal{B}$ et $\underline{v}$ une borne inférieure, si $v^*(u \mid \mathcal{B}) + \overline{v}_{gr.}(u) \leq \underline{v}$ alors le nœud u peut être élagué, c'est-à-dire que toutes les transitions qui en sont originaires peuvent être ignorées. Cette technique s'applique aux DDs restreints comme relaxés.
Pour décider si un nœud de $ECE(\overline{\mathcal{B}})$ doit être ajouté à la file de l'algorithme 2, la borne la plus stricte est utilisée à la ligne 20 : $\overline{v}(u' \mid \overline{\mathcal{B}}) = \min\left\{\overline{v}_{gr.}(u'), \overline{v}_{loc.}(u' \mid \overline{\mathcal{B}})\right\}$.

Algorithme 2 Algorithme de S&E avec DDs.

$File \leftarrow \{r\}$
$Cache \leftarrow \emptyset$
$\underline{v} \leftarrow -\infty$
while $File$ n'est pas vide **do**
  $u \leftarrow$ meilleur nœud de $File$, retiré de $File$
  if $v(u) + \overline{v}(u) \leq \underline{v}$ **then**
    continue
  if $Cache.contains(\sigma(u))$ **then**
    $\langle \theta, exploré \rangle \leftarrow Cache.read(\sigma(u))$
    if $v(u) < \theta$ **or** $(v(u) = \theta \wedge exploré)$ **then**
      continue
  $\underline{\mathcal{B}} \leftarrow Restreint(u)$
  if $v^*(\underline{\mathcal{B}}) > \underline{v}$ **then**
    $\underline{v} \leftarrow v^*(\underline{\mathcal{B}})$
    $\underline{x} \leftarrow x^*(\underline{\mathcal{B}})$
  if $\underline{\mathcal{B}}$ n'est pas exact **then**
    $\overline{\mathcal{B}} \leftarrow Relaxé(u)$
    mise à jour du $Cache$ avec l'algorithme 3
    for all $u' \in ECE(\overline{\mathcal{B}})$ **do**
      if $v^*(u' \mid \overline{\mathcal{B}}) + \overline{v}(u' \mid \overline{\mathcal{B}}) > \underline{v}$ **then**
        $v(u') \leftarrow v^*(u' \mid \overline{\mathcal{B}}), \overline{v}(u') \leftarrow \overline{v}(u' \mid \overline{\mathcal{B}})$
        ajouter u' à $File$
return $(\underline{x}, \underline{v})$

3 Mémoïsation

Comme mentionné plus tôt, l'algorithme de S&E pour les DDs [3] n'exploite pas tout le potentiel des modèles PD puisque certains sous-problèmes sont explorés plusieurs fois sans pour autant qu'un meilleur chemin y menant ait été trouvé. L'idée générale des techniques introduites dans cet article est donc de mémoriser un *seuil d'exploration* pour chacun des sous-problèmes traités ou ajoutés à la file de S&E. Avant d'appliquer toutes les transitions possibles à chacun des nœuds lors de la compilation des DDs approximés, une comparaison est d'abord effectuée avec le potentiel seuil d'exploration en mémoire. L'expansion d'un nœud est seulement poursuivie si le meilleur chemin y menant dans le DD courant a une valeur supérieure au seuil d'exploration. Celui-ci combine deux seuils distincts, le premier relatif aux relations de dominance découvertes dans les DDs et le second à l'élagage effectué dans ceux-ci.

3.1 Seuils de dominance

Avec la compilation de DDs relaxés, on obtient une représentation partiellement exacte d'une partie de l'espace de recherche. Au sein de celle-ci, il est possible de déduire des relations de *dominance* entre solutions partielles.

Définition 3 (Dominance)**.** *Soient deux chemins $p_1 : r \rightsquigarrow u$ et $p_2 : r \rightsquigarrow u$ appartenant à un DD $\mathcal{B}$. On dit que p_1 domine p_2 – formellement noté $p_1 \succ p_2$ – si et seulement si $v(p_1) > v(p_2)$. Si $v(p_1) \geq v(p_2)$, on parle de dominance faible, notée $p_1 \succeq p_2$.*

Dans un DD relaxé $\overline{\mathcal{B}}$, le sous-problème correspondant à chaque nœud exact est décomposé de manière unique en plusieurs sous-problèmes appartenant à l'ECE de $\overline{\mathcal{B}}$, ceux-ci étant ajoutés dans la file de la S&E. Le premier scénario lors duquel un nœud u_1 doit être ré-exploré se produit lorsqu'un nouveau chemin terminant en u_1 permet de dominer le meilleur chemin $p^*(u_2 \mid \overline{\mathcal{B}})$ menant à un nœud $u_2 \in ECE(\overline{\mathcal{B}})$. Pour détecter ce scénario, on définit un *seuil de dominance individuel* de u_1 par rapport à u_2. Afin de simplifier les définitions suivantes, on pose $Succ.(u \mid \overline{\mathcal{B}}) = \{u\} \cup \{u' \mid (u \rightsquigarrow u') \in \overline{\mathcal{B}}\}$ l'ensemble des successeurs de u, y compris u lui-même.

Définition 4 (Seuil de dominance individuel)**.** *Soient deux nœuds exacts $u_1 \in \overline{\mathcal{B}}$ et $u_2 \in ECE(\overline{\mathcal{B}})$. Si $u_2 \in Succ.(u_1 \mid \overline{\mathcal{B}})$ alors le seuil de dominance individuel de u_1 par rapport à u_2 est défini par :*

$$\theta_{dom.i.}(u_1 \mid u_2, \overline{\mathcal{B}}) = v^*(u_2 \mid \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) \quad (3)$$

et donne la valeur qu'un nouveau chemin $p_1 : r \rightsquigarrow u_1$ doit dépasser pour que $p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) \succ p^*(u_2 \mid \overline{\mathcal{B}})$.

Démonstration. Un nouveau chemin $p_1 : r \rightsquigarrow u_1$ permet de dominer $p^*(u_2 \mid \overline{\mathcal{B}})$ si $\overline{\mathcal{B}}$ contient un chemin $p_2 : u_1 \rightsquigarrow u_2$ tel que $p_1 \cdot p_2 \succ p^*(u_2 \mid \overline{\mathcal{B}})$, autrement dit $v(p_1) + v(p_2) > v^*(u_2 \mid \overline{\mathcal{B}})$. Le chemin $p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$ étant le meilleur chemin entre u_1 et u_2 dans $\overline{\mathcal{B}}$, la condition devient $v(p_1) + v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) > v^*(u_2 \mid \overline{\mathcal{B}})$ et de manière équivalente : $v(p_1) > v^*(u_2 \mid \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$. □

De manière plus générale, on définit le *seuil de dominance* d'un nœud $u_1 \in \overline{\mathcal{B}}$ comme le seuil de dominance individuel minimal parmi tous les nœuds successeurs $u_2 \in ECE(\overline{\mathcal{B}})$.

Définition 5 (Seuil de dominance)**.** *Soit un nœud exact $u_1 \in \overline{\mathcal{B}}$, le seuil de dominance de u_1 dans $\overline{\mathcal{B}}$ est donné par :*

$$\theta_{dom.}(u_1 \mid \overline{\mathcal{B}}) = \min_{\substack{u_2 \in ECE(\overline{\mathcal{B}}) \\ u_2 \in Succ.(u_1 \mid \overline{\mathcal{B}})}} \theta_{dom.i.}(u_1 \mid u_2, \overline{\mathcal{B}}). \quad (4)$$

3.2 Seuils d'élagage

Le second cas de figure qui nécessite la ré-exploration d'un nœud se présente lorsqu'un nouveau chemin terminant en ce nœud permet d'éviter son élagage ou celui de l'un de ses successeurs.

Définition 6 (Seuil d'élagage individuel)**.** *Soient une borne inférieure $\underline{v}$, un nœud exact $u_1 \in \overline{\mathcal{B}}$ et un nœud élagué $u_2 \in \overline{\mathcal{B}}$ avec $\overline{v}(u_2 \mid \overline{\mathcal{B}})$ la borne grossière ou locale ayant permis de l'élaguer : $v^*(u_2 \mid \overline{\mathcal{B}}) + \overline{v}(u_2) \leq \underline{v}$. Si $u_2 \in Succ.(u_1 \mid \overline{\mathcal{B}})$, alors on définit le seuil d'élagage individuel de u_1 par rapport à u_2 :*

$$\theta_{él.i.}(u_1 \mid u_2, \overline{\mathcal{B}}) = \underline{v} - \overline{v}(u_2 \mid \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) \quad (5)$$

et donne la valeur qu'un nouveau chemin $p_1 : r \rightsquigarrow u_1$ doit dépasser pour que $p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$ ne soit pas élagué.

Démonstration. Un nouveau chemin $p_1 : r \rightsquigarrow u_1$ combiné à un chemin $p_2 : u_1 \rightsquigarrow u_2$ dans $\overline{\mathcal{B}}$ évite l'élagage en u_2 si

Algorithme 3 Calcul du seuil d'exploration $\theta(u)$ de chaque nœud exact u d'un DD relaxé $\overline{\mathcal{B}}$ et mise à jour du cache.

1: $i \leftarrow$ indice de la couche du nœud racine de $\overline{\mathcal{B}}$
2: $(L_i, \ldots, L_n) \leftarrow Couches(\overline{\mathcal{B}})$
3: $\theta(u) \leftarrow \infty$ **for all** $u \in \overline{\mathcal{B}}$
4: **for** $j = n$ **down to** i **do**
5: **for all** $u \in L_j$ **do**
6: **if** $Cache.contains(\sigma(u))$ **and** $v^*(u \mid \overline{\mathcal{B}}) \leq \theta(Cache.read(\sigma(u)))$ **then**
7: $\theta(u) \leftarrow \theta(Cache.read(\sigma(u)))$
8: **else**
9: **if** $v^*(u \mid \overline{\mathcal{B}}) + \overline{v}_{gr.}(u) \leq \underline{v}$ **then**
10: $\theta(u) \leftarrow \underline{v} - \overline{v}_{gr.}(u)$
11: **else if** $u \in ECE(\overline{\mathcal{B}})$ **then**
12: **if** $v^*(u \mid \overline{\mathcal{B}}) + \overline{v}_{loc.}(u \mid \overline{\mathcal{B}}) \leq \underline{v}$ **then**
13: $\theta(u) \leftarrow \min\{\theta(u), \underline{v} - \overline{v}_{loc.}(u \mid \overline{\mathcal{B}})\}$
14: **else**
15: $\theta(u) \leftarrow v^*(u \mid \overline{\mathcal{B}})$
16: **if** u est exact **and** au-dessus de $ECE(\overline{\mathcal{B}})$ **then**
17: $explor\acute{e} \leftarrow$ false **if** $u \in ECE(\overline{\mathcal{B}})$ **else** true
18: $Cache.write(\sigma(u), \langle \theta(u), explor\acute{e} \rangle)$
19: **for all** arc $a = (u' \rightarrow u)$ **do**
20: $\theta(u') \leftarrow \min\{\theta(u'), \theta(u) - v(a)\}$

$v(p_1)+v(p_2)+\overline{v}(u_2 \mid \overline{\mathcal{B}}) > \underline{v}$. Le chemin $p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$ étant le meilleur chemin entre u_1 et u_2 dans $\overline{\mathcal{B}}$, la condition devient $v(p_1) + v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) > \underline{v}$, ou encore : $v(p_1) > \underline{v} - \overline{v}(u_2 \mid \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$. □

Puisque dépasser un seul seuil d'élagage individuel suffit à requérir la ré-exploration d'un nœud, on définit le seuil d'élagage de la manière suivante.

Définition 7 (Seuil d'élagage)**.** *Soient une borne inférieure $\underline{v}$ et un nœud exact $u \in \overline{\mathcal{B}}$, le seuil d'élagage de u dans $\overline{\mathcal{B}}$ est défini comme :*

$$\theta_{él.}(u_1 \mid \overline{\mathcal{B}}) = \min_{\substack{u_2 \in Succ.(u) \\ v^*(u_2|\overline{\mathcal{B}})+\overline{v}(u_2|\overline{\mathcal{B}}) \leq \underline{v}}} \theta_{él.i.}(u_1 \mid u_2, \overline{\mathcal{B}}). \quad (6)$$

3.3 Calcul et utilisation des seuils

Les deux seuils présentés dans cette section sont combinés en un seul *seuil d'exploration* indiquant à partir de quelle valeur un nœud doit être ré-exploré.

Définition 8 (Seuil d'exploration)**.** *Soit un DD relaxé $\overline{\mathcal{B}}$ et un nœud exact $u \in \overline{\mathcal{B}}$, le seuil d'exploration de u dans $\overline{\mathcal{B}}$ est donné par : $\theta(u \mid \overline{\mathcal{B}}) = \min\left\{\theta_{dom.}(u \mid \overline{\mathcal{B}}), \theta_{él.}(u \mid \overline{\mathcal{B}})\right\}$.*

Ces seuils d'exploration peuvent être calculés grâce à simple traversée de bas en haut du DD relaxé, comme formalisé par l'algorithme 3. Les seuils de dominance sont initialisés à la ligne 15 tandis que les seuils d'élagage le sont aux lignes 10 et 13. La ligne 20 se charge ensuite de propager les seuils vers la couche supérieure. Pour chaque nœud exact u appartenant à l'ECE ou se situant au-dessus de celui-ci, la ligne 18 met le seuil d'exploration associé $\theta(u)$ en cache. Si par ailleurs le cache contient déjà un seuil

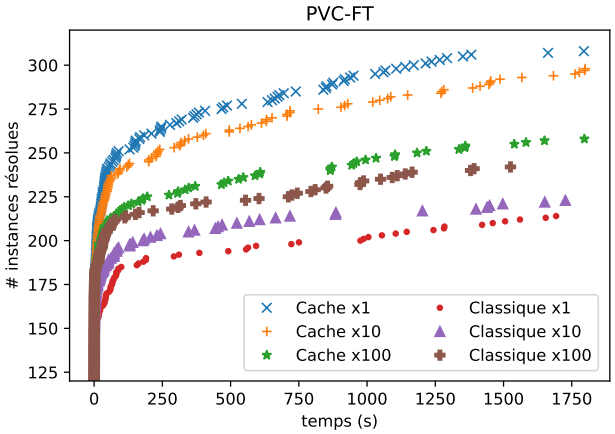

FIGURE 1 – Nombre d'instances résolues en fonction du temps pour différentes configurations de l'algorithme.

$\theta(u)$ supérieur à $v^*(u \mid \overline{\mathcal{B}})$, celui-ci peut directement être utilisé. Finalement, le cache est mis à profit dans les algorithmes 1 et 2 pour élaguer les nœuds dont l'exploration peut être évitée. Dans l'algorithme 2, un booléen $explor\acute{e}$ intervient pour décider si un nœud dont la valeur est égale au seuil d'exploration doit être traité ou non, ce qui est vrai pour les nœuds de l'ECE.

4 Validation expérimentale

Des expériences ont été conduites sur une suite de 467 instances du problème du voyageur de commerce avec fenêtres de temps, disponible à l'adresse suivante : `https://lopez-ibanez.eu/tsptw-instances`. Le modèle utilisé est détaillé dans [5]. L'algorithme *Classique* et avec *Cache* disposent de 30 minutes pour résoudre chaque instance. Une largeur maximale dynamique est imposée aux DDs suivant la formule $n \times (i+1) \times \alpha$ avec n le nombre de variables, i l'indice de la couche considérée et $\alpha \in \{1, 10, 100\}$. Comme présenté par la figure 1, la version *Cache* est supérieure à *Classique*, quelle que soit la largeur maximale utilisée, ce qui confirme l'utilité de mémoriser certains résultats intermédiaires. De plus, alors que les performances de *Classique* s'améliorent lorsque la largeur des DDs augmente, la tendance inverse est observée pour *Cache*. Il semble donc que des DDs assez étroits suffisent à identifier des parties de l'espace de recherche qui ne doivent plus être explorées. L'élagage ainsi obtenu permet aux DDs plus étroits de fournir de meilleures bornes, tout en étant très peu coûteux à compiler.

5 Conclusion

Dans cet article, un mécanisme visant à éviter au maximum l'exploration répétée de certains sous-problèmes, dans le cadre de l'optimisation discrète avec DDs, a été introduit. Pour chaque sous-problème rencontré, un seuil d'exploration est mémorisé. Celui-ci, sur base de relations de dominance et d'élagage, conditionne la nécessité de ré-explorer un sous-problème. Un algorithme permettant de calculer ef-

ficacement ces seuils a par ailleurs été présenté. Les résultats des expériences conduites sur le problème du voyageur de commerce avec fenêtres de temps démontrent l'impact significatif du mécanisme présenté.

Références

[1] Henrik Reif Andersen, Tarik Hadzic, John N Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 118–132. Springer, 2007.

[2] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2) :253–268, 2014.

[3] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1) :47–66, 2016.

[4] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and Tallys Yunes. Bdd-based heuristics for binary optimization. *Journal of Heuristics*, 20(2) :211–234, 2014.

[5] Xavier Gillard, Vianney Coppé, Pierre Schaus, and André Augusto Cire. Improving the filtering of branch-and-bound mdd solver. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 231–247. Springer, 2021.