Convergence accélérée de la Décomposition Lagrangienne pour la Planification Automatique

Hugo Barral[1,2], Quentin Cappart[1], Marie-José Huguet[2], Sylvie Thiébaux[2,3]

[1] Polytechnique Montréal, Montreal, Canada
[2] LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
[3] Australian National University, Canberra, Australie

{hugo.barral,quentin.cappart}@polymtl.ca
huguet@laas.fr
sylvie.thiebaux@anu.edu.au

Résumé

La décomposition lagrangienne est une méthode de programmation linéaire en nombres entiers qui exploite la structure d'un problème pour construire des sous-problèmes indépendants. La résolution de ces sous-problèmes procure une borne à l'optimalité sur le problème original. La qualité de cette borne peut être optimisée par des itérations de sous-gradients sur un vecteur de multiplicateurs. On compte plusieurs contributions qui transposent ces principes en programmation par contraintes. Une approche récente, introduite par Bessa et al. (2025) permet d'inférer une meilleure borne grâce à un modèle d'apprentissage. Notre projet de recherche vise à trouver des transpositions de cette méthode vers d'autres domaines de résolution de problèmes combinatoires et d'en évaluer l'apport. Une première discipline identifiée est la planification automatique. Ce papier montre nos avancées préliminaires pour améliorer une méthode existante de décomposition lagrangienne en planification. Nos premiers résultats témoignent d'une avancée des performances avant d'avoir incorporé l'apprentissage dans notre méthodologie.

Mots-clés

Décomposition lagrangienne, Apprentissage automatique, Planification automatique.

Abstract

Lagrangian decomposition is an integer programming method that exploits the structure of a problem to construct independent sub-problems. Solving these sub-problems provides an optimality bound on the original problem. The quality of the bound can be optimized by subgradient iterations on a vector of multipliers. Many contributions transpose these principles to constraint programming. A recent approach, introduced by Bessa et al. (2025), allows a better bound to be inferred using a learning model. Our research aims to find transpositions of this method to other domains of combinatorial optimization, and to evaluate its contribution. A first discipline identified is AI planning. This paper shows our preliminary advances in improving an existing Lagrangian decomposition method for planning. Our first results show an improvement in performance before incorporating learning into our methodology.

Keywords

Lagrangian decomposition, Machine learning, AI planning.

1 Introduction

Les méthodes complètes en recherche opérationnelle sont utilisées pour résoudre des problèmes combinatoires et garantir l'optimalité des solutions trouvées. Cependant, lorsqu'elles sont confrontées à des problèmes de grande taille, il devient nécessaire de s'appuyer sur des mécanismes permettant de réduire efficacement l'espace de recherche et de guider la recherche d'une solution. Dans les méthodes de recherche opérationnelle classique, comme la programmation linéaire en nombres entiers (PLNE), une famille de techniques permet de tirer parti de la structure globale du problème et de ses contraintes en le décomposant [28, 23]. La résolution de ces décompositions fournit une borne à l'optimalité sur laquelle la résolution peut s'appuyer pour éliminer une partie des solutions candidates de l'espace de recherche. En programmation par contraintes (PPC), il existe déjà des approches inspirées par la décomposition en PLNE [17] pour utiliser ces méthodes sur un ensemble plus varié de problèmes. L'une d'entre elles, la décomposition lagrangienne [12], a été l'objet de contributions récentes [13, 2] proposant de borner un modèle de PPC ou d'améliorer la propagation d'un groupe de contraintes afin d'accélérer la convergence. La difficulté de cette méthode reste la recherche de multiplicateurs lagrangiens permettant d'obtenir une borne la plus stricte possible. Plusieurs contributions ont essayé de répondre à des problématiques similaires en PLNE avec des modèles d'apprentissage automatique [1, 5, 8] mais ce n'est que plus récemment qu'une méthodologie a été considérée en PPC [3].

Notre projet de recherche vise à prolonger ce travail de transposition en explorant de nouvelles possibilités d'hy-

bridation entre la décomposition lagrangienne et l'apprentissage automatique vers d'autres méthodes de résolution de problèmes combinatoires. Ce papier propose d'analyser une méthode de décomposition existante en planification automatique. Nous détaillons les travaux préliminaires à l'incorporation d'un mécanisme d'apprentissage et les premiers résultats sur les performances. Enfin, nous expliquerons nos futurs travaux pour compléter notre contribution.

2 Méthode initiale

Le papier de Pommerening et al. [22] présente une méthode appliquant la décomposition lagrangienne à la planification automatique. Dans cette section, nous caractérisons les notions de base de la planification et les apports du papier sur lesquels s'appuient nos travaux.

2.1 Notions de base

Une tâche de planification classique peut adopter plusieurs formalismes [11], nous définissons d'abord le système de transitions d'une tâche pour ensuite évoquer SAS$^+$ [4].

On définit une tâche de planification par un tuple $\Theta = \langle S, L, c, T, s_0, S_* \rangle$ avec S l'ensemble des états, L l'ensemble des labels des transitions, $c : L \to \mathbb{R}_0^+$ une fonction de coût, $T \subseteq S \times L \times S$ l'ensemble des relations de transition, s_0 l'état initial et $S_* \subseteq S$ l'ensemble des états terminaux (*goal state*). L'objectif est de trouver un plan correspondant à une séquence de labels $\langle l_1, \ldots, l_n \rangle$ permettant de transitionner de s_0 vers un état $s_* \in S_*$. Transitionner d'un état s vers un état s' est possible si $\exists l \in L$ tel que $\langle s, l, s' \rangle \in T$. Chaque état du plan doit pouvoir transitionner vers l'état suivant, tel que $\{\langle s_0, l_1, s_1 \rangle, \langle s_1, l_2, s_2 \rangle, \ldots, \langle s_{n-1}, l_n, s_* \rangle\} \subseteq T$. On souhaitera minimiser le coût du plan obtenu par $\sum_{i=1}^{n} c(l_i)$ pour une recherche optimale.

Le formalisme SAS$^+$ permet de représenter ce système de transition de façon plus compacte, via des variables d'état et des actions de changement d'état. On définit une tâche SAS$^+$ par un tuple $\Pi = \langle V, O, s_0, s_*, cost \rangle$, avec V l'ensemble des variables, O l'ensemble des actions, s_0 et s_* les assignations partielles de variables de l'état initial et des états terminaux. Finalement, $cost : O \to \mathbb{R}_0^+$ est une fonction de coût similaire à c. Chaque variable $v \in V$ est caractérisée par un domaine D_v comprenant la valeur indéfinie $\emptyset$. Dans cette représentation, un état s est formalisé par un ensemble de faits. On peut représenter un fait, ou une assignation, par une paire $\langle v, D_v \rangle$. On dénote $vars(s) \subseteq V$ l'ensemble des variables assignées dans s et $s[v]$ la valeur de v dans s. Chaque action $o \in O$ possède des préconditions $\texttt{pre}(o)$ et des effets $\texttt{eff}(o)$, deux ensembles de faits. Une action o est applicable à un état s si $\texttt{pre}(o) \subseteq s$, i.e. tout fait dans $\texttt{pre}(o)$ l'est aussi dans s. La fonction $o(s)$ est l'application de o sur s, pour toutes variables $v \in vars(\texttt{eff}(o))$, $o(s)[v] = \texttt{eff}(o)[v]$, les autres variables restant inchangées. De façon équivalente, l'objectif est de trouver une séquence d'actions $\pi = \langle o_1, \ldots, o_n \rangle$ applicable à l'état s_0 et aux états intermédiaires vers un état s_*.

La recherche d'une séquence à travers l'espace d'état est complexe, d'autant plus lors d'une recherche optimale. Elle est donc nécessairement guidée par une fonction heuristique. L'algorithme de recherche guidée A* [14] priorise l'expansion d'états selon l'évaluation d'une fonction heuristique. On définit $h \in \mathbb{R}_0^+ \cup \{\infty\}$ une fonction heuristique sur un état s et estimant le coût de transition de cet état vers s_*. La valeur ∞ est retournée si l'objectif n'est pas atteignable depuis s. Une heuristique h admissible, i.e., qui ne surestime pas le coût optimal de s vers s_*, permet de guider la recherche tout en préservant son optimalité. On dénote f^* la fonction optimale donnant le coût minimal de s vers s_*, alors h est admissible si $\forall s \in S, h(s) \leq f^*(s, cost)$.

2.2 Heuristiques d'abstraction

En planification, les heuristiques d'abstractions imposent une équivalence entre des états d'un système de transition en les réduisant à un état abstrait. Cette méthode permet de construire un nouveau système de transition dont la taille de l'espace de recherche est réduite et d'en obtenir une valeur heuristique grâce à un algorithme de recherche, comme Dijkstra [9], ou par un solveur grâce à une modélisation en programme de flots. Helmert et al. [16] donnent une définition générale de la représentation d'une abstraction pour un système de transition.

Une abstraction d'un système Θ est une surjection ϵ qui applique S vers un ensemble d'état abstraits S^ϵ. On dénote ensuite le système de transition abstrait Θ_ϵ un tuple $\langle S^\epsilon, L, T^\epsilon, s_0^\epsilon, S_*^\epsilon \rangle$ avec $T^\epsilon = \{(\epsilon(s), l, \epsilon(s')) | (s, l, s') \in T\}$, $s_0^\epsilon = \epsilon(s_0)$ et $S_*^\epsilon = \{\epsilon(s_*) | s_* \in S_*\}$. Le calcul d'une heuristique d'abstraction h_ϵ se fait en deux étapes : l'état courant est d'abord projeté dans l'espace de l'abstraction grâce à la fonction ϵ et on résout le système abstrait en commençant par l'état projeté afin de retourner le coût du plan π_ϵ comme valeur de l'heuristique.

2.3 Partitionnement de coûts

Utiliser plusieurs abstractions différentes d'une tâche est souhaitable pour obtenir une meilleure heuristique par l'addition des résultats des abstractions. Seulement, ces heuristiques sont rarement additives [29]. C'est à dire que la somme de leur valeur peut ne pas être admissible, à cause de la possibilité d'une utilisation redondante d'actions entres plusieurs abstractions. Des contributions proposent de distribuer les coûts de chaque action entre l'ensemble des abstractions pour dépasser cette limite lors de la recherche d'un plan optimal [18]. On note la part des coûts distribués à une abstraction ϵ par un vecteur $cost_\epsilon$. Pour que la distribution assure l'additivité d'un ensemble d'abstractions E, la contrainte suivante doit être satisfaite : $\sum_{\epsilon \in E} cost_{\epsilon,o} \leq cost(o), \forall o \in O$. Les heuristiques de partitionnement de coûts font partie de la famille des heuristiques de comptage d'opérations [21] dont chaque abstraction peut se représenter par un programme linéaire avec des variables $count_{\epsilon,o}$ le nombre d'utilisations de l'action o dans ϵ et des variables auxiliaires y_ϵ. On dénote A_ϵ et B_ϵ des matrices de poids sur le nombre d'utilisation des actions et b_ϵ un vecteur contraignant un nombre d'utilisation minimum sur les ensembles d'actions. La forme générale de ces programmes est la sui-

vante :

$$\begin{aligned} \min \quad & cost_\epsilon^\top count_\epsilon \\ \text{s.t} \quad & A_\epsilon count_\epsilon + B_\epsilon y_\epsilon \geq b_\epsilon \\ & count_\epsilon, y_\epsilon \geq 0 \end{aligned} \tag{1}$$

Pour qu'une heuristique de comptage soit admissible, il faut que chaque contrainte ait une solution pour chaque plan π_ϵ où $count_{\epsilon,o}$ est équivalent au nombre de o dans le plan.

2.4 Décomposition lagrangienne

Pommerening et al. [22] montrent un parallèle entre le partitionnement de coûts et la décomposition lagrangienne en s'appuyant sur la formalisation en programme linéaire de ces heuristiques. La décomposition lagrangienne exploite la structure des problèmes d'optimisation par l'utilisation des variables liantes entre des groupes de contraintes. Elle introduit tout d'abord des contraintes d'équivalence entre les variables liantes et de nouvelles variables indépendantes qui les remplaceront dans les contraintes du problème original. Par exemple pour un programme générique avec des variables x et une série de contraintes $A_i x + B_i y_i \geq b_i$, avec $1 \leq i \leq k$. On peut créer de nouvelles variables x_i, de nouvelles contraintes $x = x_i$ puis remplacer les contraintes sous la forme $A_i x_i + B_i y_i \geq b_i$. La relaxation lagrangienne est alors appliquée aux contraintes d'équivalences en les déplaçant dans la fonction objectif avec un vecteur $\lambda \in \mathbb{R}^k_{\geq 0}$ appelé multiplicateurs lagrangiens. Par exemple, une contrainte générique $Ax \geq b$ relaxée par cette méthode est ajoutée à la fonction sous la forme $\lambda(b - Ax)$. Le choix des multiplicateurs λ est important puisque ces derniers agissent directement sur la qualité de la borne. Dans la littérature, les multiplicateurs peuvent être optimisés grâce à leur dual et par plusieurs méthodes, tels que l'algorithme du sous-gradient qui sera abordé plus loin dans ce papier. L'objectif est d'ensuite pouvoir décomposer le problème initial en séparant les contraintes originales dans ϕ_k en sous-problèmes que l'on peut ensuite formaliser comme suit :

$$\begin{aligned} \phi_i(\lambda_i) = \min \quad & \lambda_i^T x_i \\ \text{s.t} \quad & A_i x_i + B_i y_i \geq b_i \\ & x_i, y_i \geq 0 \end{aligned} \tag{2}$$

Un sous problème supplémentaire ϕ_0 est formalisé pour définir la contrainte $\sum_{i=1}^k \lambda_i \leq c$ sur les multiplicateurs :

$$\begin{aligned} \phi_0(\lambda) = \min \quad & (c - \sum_{i=1}^{k} \lambda_i)^T x \\ \text{s.t} \quad & x \geq 0 \end{aligned} \tag{3}$$

Les k sous-problèmes montrent une similitude avec les heuristiques d'abstractions partitionnées et leurs représentations mathématiques. Les multiplicateurs lagrangiens sont en fait un équivalent au coût partitionné dans chaque abstraction et les variables x sont similaires aux variables $count$. La contrainte d'admissibilité sur le partitionnement du coût est assurée par le sous problème supplémentaire. On peut donc employer les mêmes méthodes d'optimisation des multiplicateurs lagrangiens aux coûts partitionnés pour resserrer l'heuristique vers le coût optimal de la tâche de planification. Pommerening et al. [22] itère sur des méthodes de sous-gradient [26] pour améliorer un partitionnement initial. Chaque abstraction est résolue lors d'une itération t et retourne un plan $\pi_\epsilon^{(t)}$ afin de compter le nombre d'occurrences des actions. Le partitionnement des coûts pour l'itération suivante est calculée par sous-gradient, i.e., $cost_\epsilon^{(t+1)} = cost_\epsilon^{(t)} + \eta(t) occurences(\pi_\epsilon^{(t)})$, avec η une fonction de pas et $occurences$ une fonction retournant le vecteur d'occurrences des actions dans un plan. Une projection du partitionnement dans un espace admissible est requise en fin d'itération pour satisfaire la contrainte d'additivité. Enfin, la valeur heuristique est retournée par l'addition du coût des plans de chaque abstraction à la dernière itération.

3 Implémentation actuelle

Dans cette section nous décrivons les changements apportés au planificateur *Fast Downward* [15] pour l'implémentation des mécanismes de partitionnement et de décomposition lagrangienne. L'implémentation résultante est ensuite comparée avec celle initialement proposée par Pommerening et al. [22] disponible en ligne [1].

3.1 Améliorations apportées

Notre première amélioration a consisté à transposer les algorithmes de décomposition lagrangienne vers une version plus récente de Fast Downward (24.06) car plusieurs changements majeurs ont été opérés depuis la version utilisée par Pommerening et al. [22]. Nous avons aussi simplifié le code et amélioré sa consommation mémoire en ne retranscrivant que la partie nécessaire à nos travaux. Par exemple, nous avons fait le choix de ne pas transposer les structures et méthodes qui résolvent les abstractions avec un programme linéaire et de réduire la taille des systèmes de transitions abstraits. Cette démarche nous permet d'économiser de la mémoire vive lorsque nous générons de nombreuses abstractions de nos tâches. L'amélioration n'a pas d'impact sur la structure des algorithmes de recherche, par conséquent, le nombre de nœuds étendus et le nombre de nœuds évalués sont identiques lorsque les paramètres du planificateur sont les mêmes.

Notre deuxième amélioration a été de changer la fonction de pas dans le calcul du sous-gradient. Pommerening et al. [22] implémentent un pas tel que $\eta(t) = \frac{1}{t}$. Cette méthode possède plusieurs désavantages, notamment, elle ne prend pas compte de la possibilité que les coûts entre les actions soient hétérogènes. Un effet de zigzag peut également apparaitre sur le gradient, car la méthode le pousse dans une mauvaise direction, e.g., la direction inverse de l'itération précédente. Un effet de "sur place" peut aussi être constaté pendant l'étape de projection. La littérature sur les méthodes de sous-gradients répertorie des approches qui ré-

1. https://zenodo.org/records/5500499

duisent ces phénomènes [10]. Nous avons implémenté un nouveau pas inspiré du pas de Polyak [19]. On remplace alors $\eta(t)$ par $\beta^{(t)}(ub_\epsilon^{(t)} - h_\epsilon^{(t)})/\|d^{(t)}\|^2$. $\beta^{(t)} \in [0,2]$ est un paramètre arbitraire, $ub_\epsilon^{(t)}$ est une estimation de la borne haute de notre heuristique sur l'abstraction ϵ, $h_\epsilon^{(t)}$ est la valeur de notre heuristique d'abstraction ϵ et $d^{(t)}$ le vecteur de la règle de déviation. Nous avons fait le choix de diminuer $\beta^{(t)}$ de moitié lorsque la somme des abstractions n'augmente pas pendant 10 itérations de sous-gradients. La borne $ub_\epsilon^{(t)}$ est calculée sur le plan $\pi_\epsilon^{(t)}$ avec le coût original des actions utilisées. Ainsi, l'aspect hétérogène des actions et de leur répartition à travers les abstractions est pris en compte dans la méthode de sous-gradients. La valeur $d^{(t)}$ est calculée par $d^{(t)} = \alpha occurences(\pi_\epsilon^{(t)}) + (1-\alpha)d^{(t-1)}$. $\alpha \in [0,1]$ est un paramètre arbitraire et $d^0 = \vec{0}$. Le but de $d^{(t)}$ est de "dévier" le gradient avec le calcul de déviation précédent pour diminuer l'effet de zigzag.

3.2 Résultats comparatifs

Nous avons effectué plusieurs expérimentations comparatives entre notre version de la décomposition lagrangienne dans Fast Downward et celle de Pommerening et al. [22]. Nous utilisons les instances de tâches de la piste optimale de l'*International Planning Competition* (IPC) de 1998 à 2018 et nous limitons le temps de recherche à 1800s et la mémoire à 2GB pour chaque instance. La limite de temps correspond au standard d'IPC et la limite de mémoire rejoint celle de Pommerening et al. pour leurs expérimentations. Nous avons paramétré les deux approches afin qu'elles utilisent toutes deux des abstractions de 2 variables au plus et nous nous limitons aux patrons intéressants pour du partitionnement non négatif [20]. Le partitionnement initial est le partitionnement saturé [24] qui était celui avec les meilleures performances initiales dans le papier initial. Le processus de partitionnement est appelé uniquement sur le premier état évalué, les itérations de sous-gradients sont exécutées à chaque état et sont limitées à 200. Les paramètres de notre pas sont $\alpha = 0.5$ et $\beta_1 = 0.001$.
Premièrement, nous avons constaté que les longueurs des plans conjoints aux deux approches ne sont pas différentes, ce qui veut dire que notre approche reste optimale. Une première différence est que notre approche résout 453 tâches sur les 1827 instances sélectionnées contre 427 pour la méthode initiale. La valeur évaluée par notre heuristique sur l'état initial est plus élevée dans 522 tâches. Cet écart montre que notre changement de pas rend notre heuristique plus informative, puisque les deux méthodes utilisent un partitionnement ainsi qu'un nombre d'itérations similaire. Ce changement s'explique par la prise en compte des coûts hétérogènes qui permet des pas plus grands et par mitigation des effets de stagnation lors des itérations. Nous pouvons également constater une baisse de l'empreinte mémoire de notre implémentation, ce qui pourrait s'avérer utile pour des instances ou des configurations générant des abstractions en plus grand nombre. La figure 1 présente une comparaison entre les deux méthodes en termes de nombre de nœuds évalués et de nombre de nœuds étendus. Un point

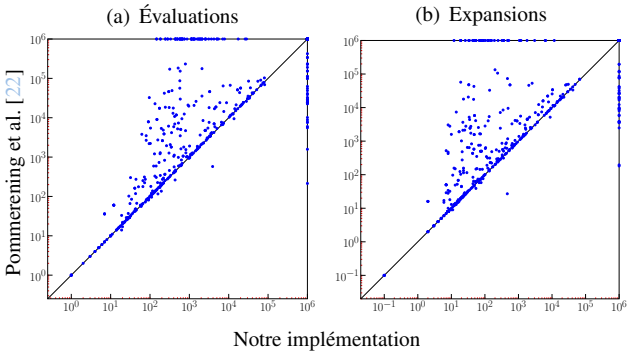

FIGURE 1 – Distribution comparative du nombre de nœuds évalués a et étendus b entre les deux implémentations sur les instances références.

s'écartant d'un côté ou de l'autre de la diagonale indique un nombre supérieur d'évaluations ou d'expansions pour la méthode sur l'axe associé. Le nombre de nœuds évalués est inférieur dans 208 tâches contre 26 et le nombre de nœuds étendus est inférieur dans 218 tâches contre 30 pour notre méthode. Nous pouvons donc étendre et évaluer moins d'états, parfois d'un ordre supérieur à 10^3, tout en résolvant plus de tâches.

4 Futurs travaux

A terme, notre objectif de recherche est d'obtenir directement des multiplicateurs lagrangiens donnant une borne strict et d'ainsi éviter les étapes de sous-gradients qui sont prohibitives en temps de calcul. A cette fin, notre prochaine étape est de quantifier le potentiel du gain de temps que peut apporter un modèle d'apprentissage des multiplicateurs lagrangiens. Pour cela, nous allons mettre en place une heuristique *oracle* en encodant les évaluations d'état de la décomposition lagrangienne sur nos instances de référence. Une comparaison de cette méthode avec notre implémentation nous permettra d'obtenir la quantité de temps gagné par l'économie des itérations du mécanisme du sous-gradient et du partitionnement des coûts afin de valider notre hypothèse de recherche.
Ensuite, il conviendra de définir notre modèle d'apprentissage. Celui-ci pourra s'inspirer des dernières contributions sur des représentations en graphes dans le domaine de la planification automatique [7, 6] ainsi que des mécanismes d'apprentissage des multiplicateurs lagrangiens dans le domaine de la programmation par contraintes [3]. Il nous faudra également étendre les expérimentations à la comparaison avec d'autres planificateurs de l'état-de-l'art. Un point intéressant sera d'analyser les performances par rapport au planificateur *Scorpion* [25] sur un jeu d'instances communes. Il s'agit d'un planificateur employant également le partitionnement de coûts comme heuristique principale et démontrant d'excellents résultats sur la dernière compétition IPC [27]. Nous voulons enfin pouvoir étendre notre recherche à d'autres problèmes combinatoires en dehors de la planification.

Références

[1] Ahmed Abbas and Paul Swoboda. DOGE-train : discrete optimization on GPU with end-to-end training. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024.

[2] David Bergman, Andre A. Cire, and Willem-Jan van Hoeve. Improved constraint propagation via lagrangian decomposition. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 30–38, Cham, 2015. Springer International Publishing.

[3] Swann Bessa, Darius Dabert, Max Bourgeat, Louis-Martin Rousseau, and Quentin Cappart. Learning Valid Dual Bounds in Constraint Programming : Boosted Lagrangian Decomposition with Self-Supervised Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 11113–11121, 2025.

[4] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4) :625–655, 1995.

[5] Quentin Cappart, David Bergman, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Augustin Parjadis. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 34(5) :2552–2570, 2022.

[6] Dillon Z Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20078–20086, 2024.

[7] Dillon Z. Chen, Felipe Trevizan, and Sylvie Thiébaux. Return to Tradition : Learning Reliable Heuristics with Classical Machine Learning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1) :68–76, May 2024.

[8] Yanchen Deng, Shufeng Kong, Caihua Liu, and Bo An. Deep attentive belief propagation : Integrating reasoning and learning for solving constraint optimization problems. *Advances in Neural Information Processing Systems*, 35 :25436–25449, 2022.

[9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1) :269–271, December 1959.

[10] Antonio Frangioni, Bernard Gendron, and Enrico Gorgone. On the computational efficiency of subgradient methods : a case study with lagrangian bounds. *Mathematical Programming Computation*, 9 :573–604, 2017.

[11] Hector Geffner and Blai Bonet. *Classical Planning : Full Information and Deterministic Actions*, pages 15–36. Springer International Publishing, Cham, 2013.

[12] Monique Guignard and Siwhan Kim. Lagrangean decomposition : A model yielding stronger lagrangean bounds. *Mathematical Programming*, 39 :215–228, 1987.

[13] Minh Hoàng Hà, Claude-Guy Quimper, and Louis-Martin Rousseau. General bounding mechanism for constraint programs. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 158–172, Cham, 2015. Springer International Publishing.

[14] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2) :100–107, 1968.

[15] Malte Helmert. The Fast Downward planning system. *J. Artif. Int. Res.*, 26(1) :191–246, July 2006.

[16] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-Shrink Abstraction : A Method for Generating Lower Bounds in Factored State Spaces. *J. ACM*, 61(3), June 2014.

[17] John N. Hooker. Chapter 15 - operations research methods in constraint programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 527–570. Elsevier, 2006.

[18] Michael Katz and Carmel Domshlak. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12) :767–798, 2010.

[19] B.T. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3) :14–29, 1969.

[20] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From Non-Negative to General Operator Cost Partitioning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Mar. 2015.

[21] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-Based Heuristics for Cost-Optimal Planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 24(1) :226–234, May 2014.

[22] Florian Pommerening, Gabriele Röger, Malte Helmert, Hadrien Cambazard, Louis-Martin Rousseau, and Domenico Salvagnin. Lagrangian Decomposition for Optimal Cost Partitioning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1) :338–347, Jul. 2019.

[23] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The Benders decomposition algorithm : A literature review. *European Journal of Operational Research*, 259(3) :801–817, 2017.

[24] Jendrik Seipp, Thomas Keller, and Malte Helmert. Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.

[25] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67 :129–167, 2020.

[26] Naum Zuselevich Shor. *Minimization Methods for Non-Differentiable Functions*. Springer Berlin, Heidelberg, 1985.

[27] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2) :280–296, 2024.

[28] François Vanderbeck and Laurence A. Wolsey. *Reformulation and Decomposition of Integer Programs*, pages 431–502. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[29] Fan Yang, Joseph Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32 :631–662, 2008.