RLBS: Une stratégie de retour arrière adaptative basée sur l'apprentissage par renforcement pour l'optimisation combinatoire

Ilyess Bachiri[1,2] **Jonathan Gaudreault**[1,2] **Brahim Chaib-draa**[2] **Claude-Guy Quimper**[1,2]

[1] FORAC Research Consortium, Québec, Canada
[2] Université Laval, Québec, Canada
ilyess.bachiri@cirrelt.ca jonathan.gaudreault@forac.ulaval.ca
{brahim.chaib-draa,claude-guy.quimper}@ift.ulaval.ca

Résumé

Les problèmes d'optimisation combinatoire sont souvent très difficiles à résoudre et le choix d'une stratégie de recherche joue un rôle crucial dans la performance du solveur. Une stratégie de recherche est dite *adaptative* quand elle s'adapte dynamiquement à la structure du problème et est capable d'identifier les zones de l'espace de recherche qui contiennent les bonnes solutions. Nous proposons un algorithme (RLBS) qui apprend à faire des retours arrière de manière efficace lors de l'exploration d'arbres non-binaires. Le branchement est effectué en utilisant une stratégie de choix de variable/valeur quelconque. Cependant, quand un retour arrière est nécessaire, la sélection du nœud cible s'appuie sur l'apprentissage automatique. Comme les arbres sont non-binaires, on a l'occasion de faire des retours arrière à plusieurs reprises vers chaque nœud durant l'exploration, ce qui permet d'apprendre quels nœuds mènent vers les meilleures récompenses en général (c'est-à-dire, vers les feuilles les plus intéressantes). RLBS est évalué sur un problème de planification en utilisant des données industrielles. Il surpasse aussi bien des stratégies d'exploration classiques (non-adaptative) (DFS, LDS) qu'une stratégie adaptative (IBS).

1 Introduction

Les problèmes d'optimisation combinatoire (Constraint Optimization Problem – COP) sont souvent très difficiles à résoudre et le choix de la stratégie de recherche a une influence importante sur la performance du solveur. Afin de résoudre un problème d'optimisation combinatoire en explorant un arbre de recherche, il faut choisir une heuristique de choix de variable (qui définie l'ordre dans lequel les variables vont être instanciées), une heuristique de choix de valeurs (qui définie l'ordre dans lequel les valeurs seront essayées), et une stratégie de retour arrière (qui détermine vers quel nœud effectuer les retours arrière lorsqu'une feuille de l'arbre est rencontrée). Quelques politiques de retours arrière sont totalement déterministes (e.g. Depth-First Search, DFS) pendant que d'autres s'appuient sur des mécanismes d'évaluation de nœuds plus dynamiques (e.g. Best-First Search). Certaines (e.g. Limited Discrepancy Search [9]) peuvent être implémentées soit comme un algorithme itératif déterministe ou un évaluateur de nœud [3].

Une stratégie est dite *adaptative* quand elle s'adapte dynamiquement à la structure du problème et identifie les zones de l'espace de recherche qui contiennent les bonnes solutions. Des stratégies de branchement adaptatives ont été proposées (e.g. Impact-Based Search (IBS) [17]) ainsi qu'une stratégie de retour arrière adaptative (e.g. Adaptive Discrepancy Search [7], proposée pour les problèmes d'optimisation distribués).

Dans cet article, nous proposons une stratégie de retour arrière qui se base sur l'apprentissage automatique pour améliorer la performance du solveur. Plus particulièrement, nous utilisons l'apprentissage par renforcement pour identifier les zones de l'espace de recherche qui contiennent les bonnes solutions. Cette approche a été développée pour les problèmes d'optimisation combinatoire dont l'espace de recherche est encodé dans un arbre non-binaire. Comme les arbres sont non-binaires, on a l'occasion d'effectuer plusieurs retours arrière vers chaque nœud durant l'exploration. Ceci permet d'apprendre quels nœuds mènent vers les

meilleures récompenses en général (c'est-à-dire, vers les feuilles les plus intéressantes).

Le reste de cet article est organisé comme suit : La section 2 passe en revue quelques concepts préliminaires relatifs à la recherche adaptative et l'apprentissage par renforcement. La section 3 explique comment le retour arrière peut être formulé en tant que tâche d'apprentissage par renforcement et introduit l'algorithme proposé (*Reinforcement Learning Backtracking Search*, or RLBS). La section 4 présente les résultats pour un problème industriel complexe qui combine planification et ordonnancement. RLBS est comparé à des stratégies de recherche classiques (non-adaptatives) (DFS, LDS) ainsi qu'à une stratégie de branchement adaptative (IBS). La section 5 conclue l'article.

2 Concepts préliminaires

La résolution des problèmes d'optimisation combinatoire en utilisant l'exploration globale se réduit à définir trois éléments-clés : une stratégie de choix de variable, une stratégie de choix de valeur, et une stratégie de retour-arrière [20]. L'espace de recherche est structuré sous forme d'un arbre dont la topologie est définie par les stratégies de choix de variable/valeur. Peu importe les stratégies de choix de variable/valeur employées, l'arbre de recherche couvre entièrement l'espace de recherche. La stratégie de retour-arrière détermine l'ordre dans lequel les nœuds vont être visités. Les stratégies de retour-arrière peuvent être implémentées sous forme d'algorithmes itératifs, ou de mécanismes d'évaluation de nœud [3].

2.1 Heuristiques de choix de variable/valeur et apprentissage

Quelques algorithmes apprennent durant l'exploration quelles variables sont les plus difficiles à instancier, dans le but de changer l'ordre d'instanciation des variables dynamiquement (e.g. YIELDS [10]). Dans [4] et [8], à chaque fois qu'une contrainte entraîne un échec, la priorité des variables impliquées dans cette contrainte est augmentée.

Dans Impact Based Search (IBS) [17], l'impact des variables est mesuré en observant à quel point leurs instanciations réduit la taille de l'espace de recherche. Comme IBS choisit en même temps la variable à instancier et la valeur à lui attribuer, on peut dire qu'il apprend une combinaison de stratégie de choix de variable et de stratégie de choix de valeur.

2.2 Apprendre à effectuer les retours arrière

Les approches où le système apprend à évaluer la qualité des nœuds sont d'un grand intérêt quand il s'agit de stratégies de retour arrière. Ruml [18] propose une approche intéressante à cet égard. Pendant que LDS basique attribue le même degré d'importance à toutes les déviations, Best Leaf First Search (BLFS) donne différents poids aux déviations selon leur profondeur. BLFS utilise une régression linéaire afin de déterminer la valeur des poids. Le modèle n'a pas vraiment été utilisé pour définir une stratégie de retour arrière. Au lieu de ceci, l'algorithme d'exploration effectue des descentes successives dans l'arbre. L'algorithme de Ruml a abouti à de très bons résultats (voir [19]), et a été l'inspiration derrière le prochain algorithme.

Adaptive Discrepancy Search (ADS) [7] est un algorithme qui a été proposé pour l'optimisation distribuée mais qui peut être employé dans un contexte de COP classique. Durant la recherche, il apprend dynamiquement vers quels nœuds il est le plus profitable d'effectuer des retours arrière (afin de concentrer les efforts sur ces zones de l'arbre en premier). Pour chaque nœud, il tente d'apprendre une fonction $Improvement(i)$ qui prédit à quel point la première feuille rencontrée après avoir effectué un retour arrière vers ce nœud pour la i-ème fois sera de bonne qualité, en comparaison avec les retours arrière précédents effectués vers le même nœud. L'inconvénient de cette méthode est qu'une fonction doit être apprise pour chacun des nœuds ouverts. Ces fonctions doivent aussi être mises à jour à chaque fois qu'un nœud mène vers une nouvelle solution [13] (même si une approximation peut être calculée en utilisant la régression).

2.3 Apprentissage par renforcement

L'idée fondamentale derrière l'apprentissage par renforcement est de trouver une façon d'associer les actions aux situations afin de maximiser la récompense totale. L'apprenti ne sait pas quelles actions prendre, il doit découvrir quelles actions mènent vers les plus grandes récompenses (à long terme). Les actions peuvent affecter non seulement la récompense immédiate mais aussi la situation prochaine et, au final, toutes les récompenses qui suivent [2]. De plus, les actions peuvent ne pas mener vers les résultats espérés à cause de l'incertitude de l'environnement.

L'apprentissage par renforcement s'appuie sur un cadre formel qui définit l'interaction entre l'apprenti et l'environnement en termes d'états, d'actions, et de récompenses. L'environnement qui supporte l'apprentissage par renforcement est typiquement formulé en tant qu'un processus décisionnel de Markov (Markov

Decision Process - MDP) à états finis. Dans chaque état $s \in S$, un ensemble d'actions $a \in A$ sont disponibles, parmi lesquelles l'apprenti doit choisir celle qui maximise la récompense cumulative. L'évaluation des actions est entièrement basée sur l'expérience de l'apprenti, cumulée à travers ses interactions avec l'environnement. Le but de l'apprenti est de trouver une politique optimale $\pi : S \to A$ qui maximise la récompense cumulative. La récompense cumulative est soit définie en tant que la somme de toutes les récompenses $R = r_0 + r_1 + \cdots + r_n$, soit exprimée en tant que la somme des récompenses dévaluées $R = \sum_t \gamma^t r_t$. Le facteur de dévaluation $0 \leq \gamma \leq 1$ est appliqué pour privilégier les récompenses récentes. La représentation sous forme de somme dévaluée de la récompense est utilisée pour un MDP sans état final.

L'intuition centrale derrière l'apprentissage par renforcement est que les actions qui mènent vers les grandes récompenses doivent être choisie plus souvent.

Dans une tâche d'apprentissage par renforcement, chaque action a, dans chaque état s, est associé à une valeur numérique $Q(s, a)$ qui représente la désirabilité de choisir l'action a dans l'état s. Ces valeurs sont appelées Q-Values. Plus grande est la Q-Value, plus c'est probable que l'action associée mène vers une bonne solution, selon le jugement de l'apprenti. À chaque fois qu'une récompense est retournée à l'apprenti, il doit mettre à jour la Q-Value de l'action qui a mené à cette récompense. Cependant, l'ancienne Q-Value ne devrait pas être complètement oubliée, sinon l'apprenti se baserait uniquement sur la toute dernière expérience à chaque fois qu'il doit prendre une décision. Pour ce faire, on garde une partie de l'ancienne Q-Value et on la met à jour avec une partie de la nouvelle expérience. Aussi, on suppose que l'apprenti va agir de manière optimale par la suite. De plus, les récompense futures espérées doivent être dévaluées pour privilégier les récompenses les plus récentes.

Soit s l'état courant, s' l'état prochain, a une action, r la récompense retournée après avoir pris l'action a, α le taux d'apprentissage, et γ le facteur de dévaluation. La formule de mise à jour des Q-Values est la suivante :

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')] \quad (1)$$

2.4 Apprentissage automatique et exploration

L'idée d'utiliser l'apprentissage par renforcement pour résoudre les problèmes d'optimisation combinatoires est exploitée par différents travaux de recherche [14, 16, 21]. Des chercheurs ont essayé d'appliquer l'apprentissage par renforcement pour résoudre des problèmes d'optimisation alors que d'autres l'ont employé dans le contexte de problèmes de satisfaction de contraintes (Constraint Satisfaction Problems – CSP).

À titre d'exemple, Xu et al. [21] ont proposé une formulation de la résolution du CSP en tant que tâche d'apprentissage par renforcement. Un ensemble d'heuristiques de choix de variable est fourni à l'algorithme qui apprend laquelle utiliser, et quand l'utiliser, afin de résoudre un CSP dans le plus bref délai. Le processus d'apprentissage prend place de manière *offline* et est mené sur différentes instances du même CSP. Les états sont les instances ou sous-instances du CSP. Les différentes heuristiques de choix de variable définissent les actions. Une récompense est retournée à chaque fois qu'une instance est résolue. Cette approche se base sur le Q-Learning pour apprendre l'heuristique de choix de variable optimale à chaque point de décision dans l'arbre de recherche, pour une (sous)-instance du CSP. En d'autres termes, le processus d'apprentissage détermine, dans chaque contexte, quelle heuristique de choix de variable est le plus efficace.

Par ailleurs, Loth et al. [11] ont proposé l'algorithme BASCOP (*Bandit Search for Constraint Programming*) qui guide l'exploration dans le voisinage de la dernière meilleure solution trouvée. Cet algorithme se base sur des estimations calculées à travers plusieurs redémarrage. BASCOP a été testé sur un problème de *job shop* [12] et a donné des résultats similaires à ce qui existait dans l'état de l'art de la programmation par contrainte.

Une technique de recherche locale qui utilise l'apprentissage par renforcement est aussi proposée dans [14]. Cette approche consiste en la résolution des COPs en se basant sur une population d'agents d'apprentissage par renforcement. Les paires $\langle$variable, valeur$\rangle$ sont considérés comme les états de la tâche d'apprentissage par renforcement, et les stratégies de branchement définissent les actions. Chaque agent se voit assigné une zone spécifique de l'espace de recherche où il est censé apprendre et trouver de bonnes solutions locales. L'expertise de toute la population d'agents est ensuite exploitée. Une nouvelle solution globale est produite en gardant une partie de la meilleure solution locale trouvée par un agent, et complétée en utilisant l'expertise d'un autre agent de la population.

Selon [16], la recherche locale peut être formulée comme une politique de résolution dans un processus décisionnel de Markov (MDP) où les états représentent les solutions, et les actions définissent les solutions voisines. Les techniques d'apprentissage par renforcement peuvent, dès lors, être utilisées pour apprendre une fonction de coût afin d'améliorer la recherche locale. Une manière de faire serait d'apprendre une nouvelle fonction de coût sur de multiples trajectoires de

la même instance. L'algorithme STAGE de Boyan et Moore [5] procède ainsi et alterne entre l'utilisation de la fonction de coût originale et la fonction de coût nouvellement apprise. En améliorant la précision de prédiction de la fonction de coût apprise, la capacité des heuristiques à guider la recherche s'améliore.

Une autre approche qui utilise l'apprentissage par renforcement pour améliorer la recherche locale dans le contexte de l'optimisation combinatoire est d'apprendre une fonction de coût de manière *offline*, et ensuite l'utiliser sur de nouvelles instances du même problème. Les travaux de Zhang et Dietterich [22] se situent dans cette catégorie.

3 RLBS : Le retour arrière en tant que tâche d'apprentissage par renforcement

Cette section introduit *Reinforcement Learning Backtracking Search* (RLBS). Le branchement est effectué selon n'importe quelle heuristique de choix de variable/valeur. À chaque fois qu'une feuille/solution est atteinte, un nœud vers lequel effectuer un retour arrière doit être sélectionné. À chaque candidat au retour arrière (c'est-à-dire un nœud avec au moins un enfant non-visité) correspond une *action* ("effectuer un retour arrière vers ce nœud"). Une fois un nœud est sélectionné, l'exploration de l'arbre de recherche poursuit à partir de ce nœud jusqu'à ce qu'une feuille/solution est rencontrée. La différence entre la qualité de cette nouvelle solution et la meilleure solution trouvée jusqu'à date constitue la *récompense* retournée pour avoir choisi l'action précédente. Comme les arbres de recherche sont non-binaires, on est amené à effectuer plusieurs retours arrière vers chaque nœud durant l'exploration. Ceci permet d'identifier les actions qui payent le plus (à savoir, les nœuds qui ont le plus de chance de mener vers des feuilles/solutions intéressantes).

Cette situation est similaire au problème du *k-armed bandit* [1]. Il s'agit d'un problème d'apprentissage par renforcement mono-état. Plusieurs actions sont possibles (tirer sur l'un des bras de la machine à sous). Chaque action peut mener vers une récompense (de manière stochastique) et un équilibre doit être maintenu entre l'exploration et l'exploitation. Dans notre situation de retour arrière, choisir une action mène à la découverte de nouveaux nœuds (nouvelles actions), en plus de retourner une récompense (ce qui est stochastique et non-stationnaire).

3.1 Apprentissage

Comme dans le contexte d'apprentissage par renforcement classique, l'évaluation (Q-value) d'une action a est mise à jour à chaque fois qu'une récompense est retournée après avoir choisi une action. Comme il s'agit d'un environnement mono-état, le facteur de dévaluation γ est égal à 0 et l'équation (1) se réduit à l'équation (2) :

$$Q(a) \leftarrow Q(a) + \alpha[r(a) - Q(a)] \qquad (2)$$

où $r(a)$ est la récompense et α est le taux d'apprentissage.

La prochaine action à effectuer est sélectionnée en se basant sur ces évaluations. Un nœud qui a généré une grande récompense au début mais qui n'a jamais mené vers de bonnes solutions par la suite verra sa Q-Value décroître au fil du temps, jusqu'à ce qu'il devienne moins intéressant que les autres nœuds/actions.

3.2 Initialisation de l'algorithme

Au début de l'exploration, on descend jusqu'à la première feuille/solution de l'arbre en utilisant un DFS. Puis, on effectue un retour arrière une fois vers chacun des nœuds ouverts (ceci est similaire à réaliser les 2 premières itérations de LDS), ce qui permet de calculer leurs Q-Values. Ensuite, on commence à se baser sur les Q-Values pour choisir le prochain nœud pour effectuer le retour arrière. Quand un nœud est visité pour la première fois, sa Q-Value est initialisée par celle de son parent.

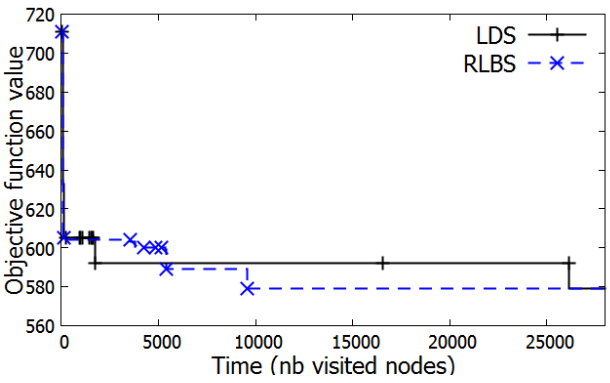

FIGURE 5 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #5

4 Expérimentations en utilisant des données industrielles

L'objectif principal de ces travaux de recherche était de trouver une façon plus intelligente de sélectionner le nœud vers lequel effectuer le retour-arrière durant l'exploration globale d'un arbre de recherche. Et ce pour (1) des problèmes d'optimisation combinatoires

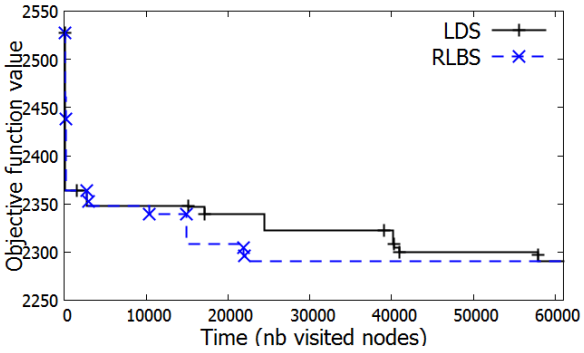

FIGURE 1 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #1

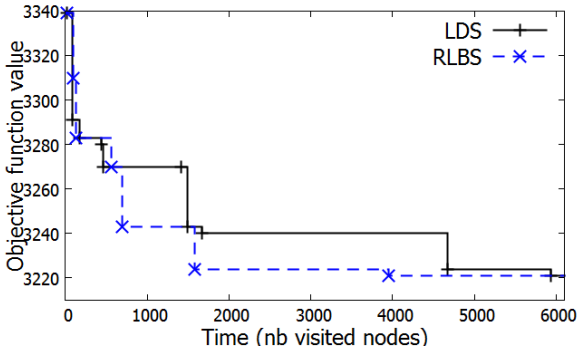

FIGURE 2 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #2

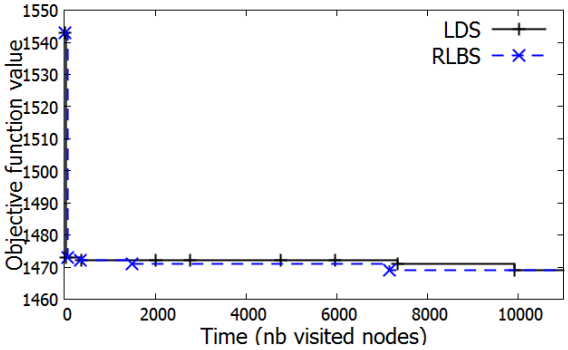

FIGURE 3 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #3

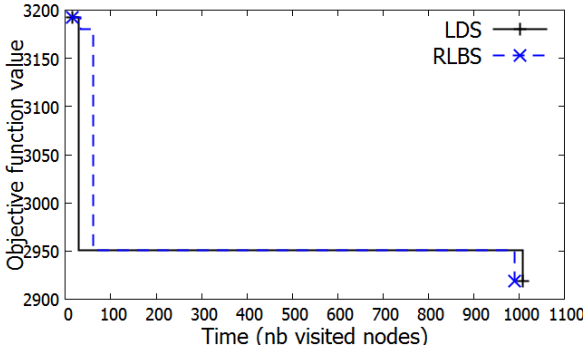

FIGURE 4 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #4

(2) pour lesquels on connaît déjà de bonnes stratégies de choix de variable/valeur. Des expérimentations ont été menées sur un problème de planification et ordonnancement issu de l'industrie du bois (le problème de planification et ordonnancement des opérations de rabotage du bois) car il répond parfaitement aux critères mentionnés précédemment.

Ce problème est difficile car il traite des processus *divergents* avec de la *coproduction* : un processus génère plusieurs produits différents en même temps, à partir d'un même produit de matière première. De plus, plusieurs processus alternatifs peuvent produire le même produit. Finalement, il implique des règles de mis en course complexes. Le but est de minimiser les retards de livraison.

Le problème est décrit plus en détails dans [6] qui fournit de bonnes heuristiques de choix de variable/valeur spécifiques à ce problème. Dans [7], ces heuristiques ont été employées pour guider la recherche dans un modèle de programmation par contraintes. Muni de ces heuristiques, LDS surpasse DFS ainsi qu'une autre approche de programmation mathématique. Dans [15], de la parallélisation a été utilisée pour améliorer la performance. Cela dit, l'ordre de visite des nœuds est le même que pour la version centralisée, donc elle implémente la même stratégie.

Les mêmes heuristiques de choix de variable/valeur du travail précédent ont été employées. Aussi, les mêmes données industrielles fournies par une compagnie canadienne de l'industrie du bois ont été utilisées. Cependant, afin de comparer les algorithmes en termes de temps de calcul nécessaire pour trouver les solutions optimales, on a dû réduire la taille des problèmes (5 périodes au lieu de 44).

RLBS a été testé avec un taux d'apprentissage $\alpha = 0.5$. Il a été comparé à LDS (qui privilégie les nœuds avec le moins de déviations).

Les figures 1 à 5 présentent les résultats pour cinq cas différents. Le tableau 1 montre la réduction du temps de calcul (mesuré en tant que le nombre de nœuds visités) nécessaire pour obtenir une solution optimale. RLBS réduit le temps de calcul dans chacun des cinq cas (de 37.66% en moyenne).

Comme dans le contexte industriel on n'a souvent

TABLE 1 – Temps de calcul nécessaire pour trouver la meilleure solution (RLBS vs. LDS)

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Average**
LDS	57926	5940	9922	1008	26166	20192.4
RLBS	22164	3949	7172	990	9545	8764
Time reduction	↓ 61.73%	↓ 33.52%	↓ 27.72%	↓ 1.79%	↓ 63.52%	↓ 37.66%

TABLE 2 – Moyenne de temps de calcul pour trouver une solution d'une qualité quelconque (RLBS vs. LDS)

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Average**
LDS	8777.82	1243.86	386.5	127.34	2776.05	2662.31
RLBS	4254.81	606.79	264.24	152.17	1320.15	1319.63
Time reduction	↓ 51.53%	↓ 51.22%	↓ 31.63%	↑ 19.5%	↓ 52.44%	↓ 33.46%

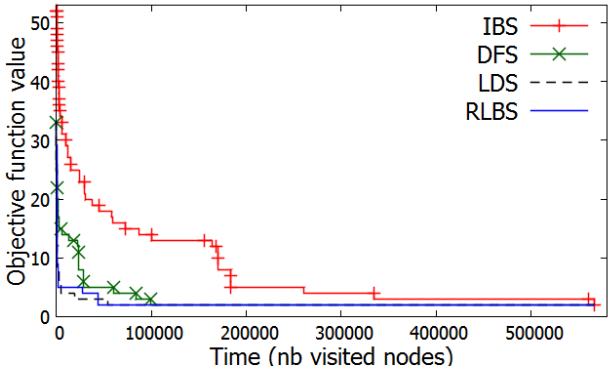

FIGURE 6 – La valeur de la fonction objectif en fonction du temps de calcul de RLBS, LDS, IBS et DFS sur un problème jouet

pas suffisamment de temps pour attendre la solution optimale, on voulait aussi considérer le temps nécessaire pour obtenir les solutions de qualités intermédiaires. Le tableau 2 montre, pour chaque cas, le temps de calcul moyen nécessaire pour obtenir une solution d'une qualité quelconque. La dernière colonne montre qu'en moyenne, pour tous les problèmes et toutes les qualités de solutions, on peut s'attendre à une amélioration de temps de calcul de 33.46%.

Enfin, on a généré des problèmes jouets (Fig. 6) afin de comparer RLBS à un autre algorithme pour lequel on n'a pas été capable de résoudre le problème original dans une durée de temps raisonnable (plus de 150 heures de traitement avec Choco v2.1.5). RLBS est comparé à d'autres approches qui ne modifient pas les heuristiques de branchement (DFS, LDS). On a choisi d'inclure les résultats de Impact-Based Search (IBS) [17] (qui ne nous permet pas d'employer nos heuristiques de choix de variable/valeur spécifiques) afin d'illustrer le fait que tenter d'apprendre à brancher au lieu d'utiliser de bonnes heuristiques de branchement déjà connues ne semble pas être une bonne idée (du moins pas dans le cas du problème étudié et IBS). IBS montre le pire résultat, probablement car il ne peut pas tirer profit des stratégies de branchement spécifiques connues d'être très efficaces pour ce problème. DFS est aussi surpassé par LDS, comme rapporté dans la littérature pour ce problème précédemment.

5 Conclusion

Nous avons proposé un simple mécanisme d'apprentissage basé sur l'apprentissage par renforcement qui permet au solveur d'apprendre dynamiquement à effectuer des retours arrière de manière efficace. Ceci a été évalué pour un problème industriel difficile de planification et ordonnancement, qui est uniquement résolu en utilisant des heuristiques de branchement spécifiques. La stratégie de retour arrière proposée améliore grandement la performance de l'exploration en comparaison avec LDS. Ceci grâce au mécanisme d'apprentissage qui permet d'identifier les nœuds vers lesquels il est le plus profitable d'effectuer des retours arrière.

L'utilisation de données réelles issues de l'industrie a montré la grande valeur de cette approche. Cependant, des questions demeurent concernant la performance de l'algorithme sur des problèmes pour lesquels on ne connaît pas de bonnes heuristiques de branchement. Dans cette situation, est-ce que ceci vaut la peine d'essayer d'identifier un meilleur nœud candidat au retour arrière ?

D'autres chercheurs [21] ont déjà utilisé l'apprentissage par renforcement pour apprendre à brancher. À notre connaissance, ceci est la première utilisation de l'apprentissage par renforcement pour apprendre une stratégie de retour arrière. Notre approche offre l'avantage qu'elle est utilisable peu importe la stratégie de branchement employée. Ceci la rend très efficace pour les problèmes pour lesquels une bonne straté-

gie de branchement est connue. Cependant, rien n'empêche de combiner les deux (apprendre une stratégie de branchement et une stratégie de retour arrière), ce qui pourra être fait dans le cadre de travaux futurs.

Remerciements

Ce travail a été soutenu par les partenaires industriels du Consortium de Recherche FORAC.

Références

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2004.

[2] Andrew G Barto. *Reinforcement learning : An introduction*. MIT press, 1998.

[3] J Christopher Beck and Laurent Perron. Discrepancy-bounded depth first search. In *Proceedings of the Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CPAIOR), Germany, Paderborn*, pages 7–17, 2000.

[4] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.

[5] Justin A Boyan and Andrew W Moore. Using prediction to improve combinatorial optimization search. In *Sixth International Workshop on Artificial Intelligence and Statistics*, 1997.

[6] Jonathan Gaudreault, Pascal Forget, Jean-Marc Frayret, Alain Rousseau, Sebastien Lemieux, and Sophie D'Amours. Distributed operations planning in the softwood lumber supply chain : models and coordination. *International Journal of Industrial Engineering : Theory Applications and Practice*, 17 :168–189, 2010.

[7] Jonathan Gaudreault, Gilles Pesant, Jean-Marc Frayret, and Sophie D'Amours. Supply chain coordination using an adaptive distributed search strategy. *Journal of Systems, Man, and Cybernetics, Part C : Applications and Reviews, IEEE Transactions on*, 42(6) :1424–1438, 2012.

[8] Diarmuid Grimes and Richard J Wallace. Learning from failure in constraint satisfaction search. In *Learning for Search : Papers from the 2006 AAAI Workshop*, pages 24–31, 2006.

[9] William D Harvey and Matthew L Ginsberg. Limited discrepancy search. In *Proceedings of International Joint Conference on Artificial Intelligence (1)*, pages 607–615, 1995.

[10] Wafa Karoui, Marie-José Huguet, Pierre Lopez, and Wady Naanaa. YIELDS : A yet improved limited discrepancy search for CSPs. In *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 99–111. Springer, 2007.

[11] Manuel Loth, Michele Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-based search for constraint programming. In *Principles and Practice of Constraint Programming*, pages 464–480. Springer, 2013.

[12] Manuel Loth, Michele Sebag, Youssef Hamadi, Marc Schoenauer, and Christian Schulte. Hybridizing constraint programming and monte-carlo tree search : Application to the job shop problem. In *Learning and Intelligent Optimization*, pages 315–320. Springer, 2013.

[13] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2) :431–441, 1963.

[14] Victor V Miagkikh and William F Punch III. Global search in combinatorial optimization using reinforcement learning algorithms. In *Proceedings of Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

[15] Thierry Moisan, Jonathan Gaudreault, and Claude-Guy Quimper. Parallel discrepancy-based search. In *Principles and Practice of Constraint Programming*, pages 30–46. Springer, 2013.

[16] Robert Moll, Andrew G Barto, Theodore J Perkins, and Richard S Sutton. Learning instance-independent value functions to enhance local search. In *Advances in Neural Information Processing Systems*. Citeseer, 1998.

[17] Philippe Refalo. Impact-based search strategies for constraint programming. In *Proceedings of Principles and Practice of Constraint Programming–CP 2004*, pages 557–571. Springer, 2004.

[18] Wheeler Ruml. *Adaptive tree search*. PhD thesis, Citeseer, 2002.

[19] Wheeler Ruml. Heuristic search in bounded-depth trees : Best-leaf-first search. In *Working Notes of the AAAI-02 Workshop on Probabilistic Approaches in Search*, 2002.

[20] Pascal Van Hentenryck. *The OPL optimization programming language*. Mit Press, 1999.

[21] Yuehua Xu, David Stern, and Horst Samulowitz. Learning adaptation to solve constraint satisfaction problems. In *Proceedings of Learning and Intelligent Optimization (LION)*, 2009.

[22] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of International Joint Conferences on Artificial Intelligence*, volume 95, pages 1114–1120. Citeseer, 1995.