Raffiner l'heuristique CHS à l'aide de bandits[*][†]

Mohamed Sami Cherif **Djamal Habet** **Cyril Terrioux**

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

{mohamed-sami.cherif, djamal.habet, cyril.terrioux}@univ-amu.fr

Résumé

Récemment, une heuristique efficace, appelée Conflict History Search (CHS), a été introduite pour la résolution du problème de satisfaction de contraintes (CSP). Elle repose sur une technique d'apprentissage par renforcement appelée *Exponential Recency Weighted Average* (ERWA) pour estimer la dureté des contraintes. CHS favorise les variables qui apparaissent souvent dans les échecs récents. Le paramètre de pas utilisé dans CHS est important car il contrôle l'estimation de la dureté des contraintes. Dans cet article, nous envisageons un raffinement de ce paramètre à l'aide d'un bandit manchot. Le bandit sélectionne une valeur appropriée de ce paramètre lors des redémarrages effectués par l'algorithme de recherche. Chaque bras correspond à l'heuristique CHS avec une valeur donnée pour le paramètre de pas et est récompensé selon sa capacité à mener une recherche efficace. Une phase d'entraînement est introduite en amont de la recherche pour aider le bandit à choisir un bras pertinent. L'évaluation expérimentale montre que cette approche conduit à des améliorations significatives.

1 Contexte

Une instance CSP est définie par la donnée d'un triplet (X, D, C), où $X = \{x_1, \ldots, x_n\}$ est un ensemble de n variables, $D = \{d_{x_1}, \ldots, d_{x_n}\}$ est un ensemble de domaines finis de taille au plus d, et $C = \{c_1, \ldots, c_e\}$ est un ensemble de e contraintes. Chaque contrainte c_i est un couple $(S(c_i), R(c_i))$, où $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ définit la *portée* de c_i, et $R(c_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ est une *relation de compatibilité*. Une affectation d'un sous-ensemble de X est dite *localement cohérente* si toutes les contraintes couvertes par ce sous-ensemble sont satisfaites. Une *solution* est une affectation localement cohérente de toutes les variables. Déterminer si une instance CSP possède une solution est NP-complet.

Dans ce contexte, l'heuristique de choix de variables permet de désigner la prochaine variable à instancier. Elle joue un rôle important dans la résolution des instances CSP. Son influence sur l'efficacité de la recherche est souvent considérable. Dans la littérature, de nombreuses heuristiques ont été proposées. Les heuristiques dynamiques et adaptatives (par exemple [2, 6, 8]), conduisent généralement aux meilleurs résultats.

Dans ce registre, l'heuristique CHS [4] maintient pour chaque contrainte c_i un score $q(c_i)$ (initialement nul). Lorsque c_i conduit à un échec, $q(c_i)$ est mis à jour avec la formule $q(c_i) = (1 - \alpha) \times q(c_i) + \alpha \times r(c_i)$, dérivée d'ERWA. Le paramètre $0 < \alpha < 1$ (dit *paramètre de pas*) définit l'importance donnée à l'ancienne valeur de $q(c_i)$ par rapport à la valeur $r(c_i)$ de la récompense. Sa valeur, initialisée à une valeur α_0 donnée, décroit au cours du temps par pas de 10^{-6} jusqu'à un seuil fixé à $0,06$. La valeur de $r(c_i)$ dépend des échecs rencontrés récemment au niveau de c_i et est définie par $r(c_i) = \frac{1}{\#Conflicts - Conflict(c_i) + 1}$. Initialisé à 0, $\#Conflicts$ est le nombre d'échecs rencontrés depuis le début de la recherche tandis que $Conflict(c_i)$ (initialisé à 0) mémorise, pour chaque contrainte c_i, la valeur qu'avait $\#Conflicts$ lors du dernier échec rencontré grâce à c_i. À chaque conflit, $\#Conflicts$ est incrémenté de 1. CHS choisit, comme prochaine variable, celle qui a le plus grand score chv avec $chv(x_j) = \frac{\sum_{c_i \in C \ | \ x_j \in S(c_i) \wedge |Uvars(c_i)| > 1} (q(c_i) + \delta)}{|D_j|}$ où $Uvars(c_i)$ désigne l'ensemble des variables non instanciées de $S(c_i)$. Le paramètre δ permet de débuter la recherche avec des scores initiaux reflétant le nombre de contraintes dans lesquelles apparaît chaque variable. CHS privilégie donc les variables ayant un petit domaine et intervenant dans des contraintes qui sont à l'origine d'échecs récents et récurrents. Enfin, lors de chaque redémarrage, la valeur de α est réinitialisée à α_0 et les valeurs des $q(c_i)$ sont lissées suivant la formule $q(c_i) = q(c_i) \times 0,995^{\#Conflicts - Conflict(c_i)}$.

[*]Ce papier est un résumé de [3].

[†]Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet DEMOGRAPH (ANR-16-CE40-0028).

2 Raffiner CHS via un bandit manchot

Les résultats expérimentaux de CHS montrent que certaines instances ne sont résolues que pour certaines valeurs de α_0. Afin de gagner en efficacité, nous exploitons un bandit manchot dont chacun des K bras correspond à l'heuristique CHS pour une valeur donnée de α_0. À chaque redémarrage de l'algorithme de résolution, le bandit choisit un de ses bras et donc désigne la variante de CHS à utiliser jusqu'au prochain redémarrage. Ce choix est fait via la politique Upper Confidence Bound (UCB1) [1]. Il repose sur les récompenses attribuées à la fin de chaque relance pour estimer la performance de l'algorithme de résolution par rapport à l'heuristique de recherche employée.

Soit nc_t le nombre de conflits rencontrés durant la relance t. On note p_j le ratio de variables non instanciées lors du j-ème conflit. La fonction de récompense est calculée, à la fin de la relance, selon la formule $R_t(i) = \frac{\sum_{j=1}^{nc_t} p_j}{nc_t}$. Ainsi, cette fonction de récompense consiste en une moyenne des ratios de variables non instanciées rencontrées lors des conflits, évaluant la capacité de l'heuristique à identifier rapidement les instanciations incohérentes. En d'autres termes, moins il y a de variables instanciées, plus la valeur de la récompense sera élevée pour l'heuristique correspondante.

Le nombre de redémarrage semble souvent insuffisant pour évaluer pleinement le potentiel de chaque heuristique candidate, ce qui rend difficile pour le bandit de converger rapidement vers les bras les plus adéquats. Pour cette raison, une phase d'entraînement a été ajoutée. Elle consiste à choisir successivement chaque bras un par un, comme le ferait un algorithme de type tourniquet. Afin de ne pas favoriser un bras au détriment d'un autre, la durée entre deux redémarrages est constante dans cette phase et le nombre de redémarrage est un multiple de K. Cette phase d'entraînement peut être considérée comme une étape d'initialisation des paramètres utilisés par le bandit, notamment des valeurs des récompenses et de leurs moyennes. Une fois l'entraînement terminé, UCB1 prend le relais pour sélectionner un bras parmi les heuristiques candidates. Par ailleurs, durant la phase d'entraînement, les poids des CHS sont mis à jour à chaque exécution lorsqu'un conflit survient ou lors d'un redémarrage par lissage. Enfin, la phase d'entraînement seule peut suffire à résoudre certaines instances.

3 Résultats expérimentaux

Nous comparons, sur 12 901 instances du dépôt XCSP3 [1], notre approche avec entraînement (MAB-CHS+Train) et sans (MAB-CHS) à des heuristiques de l'état de l'art (à savoir *dom/wdeg* [2], ABS [6], wdeg$^{ca.cd}$ [8] et CHS) ainsi qu'à une approche basée sur un bandit multi-heuristiques (MAB-VH [7]). La résolution s'effectue via l'algorithme MAC avec redémarrages et enregistrement de nogoods [5] avec un temps limite de 30 minutes. MAB-CHS(+Train) exploite un bandit à 9 bras. Chaque bras correspond à CHS avec α_0 qui varie de 0,1 à 0,9 par pas de 0,1.

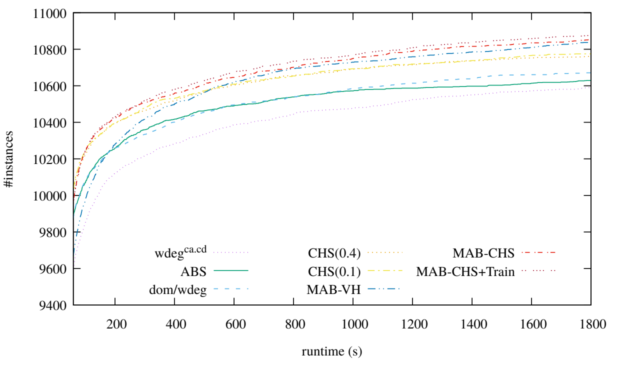


FIGURE 1 – Nombre d'instances résolues en fonction du temps écoulé (à partir de 60 s) pour chaque heuristique.

D'abord, d'après la figure 1, l'heuristique MAB-CHS+Train s'avère meilleure que MAB-CHS avec 10 875 instances résolues contre 10 853. Ensuite, comparée aux heuristiques de l'état de l'art, MAB-CHS+Train affiche un gain notable compris entre 95 et 279 instances. Enfin, MAB-CHS+Train se révèle plus efficace que MAB-VH en nombre d'instances résolues (38 instances de plus) et en temps (1,038 h contre 1,068 h).

1. http ://www.xcsp.org/series

Références

[1] Peter AUER, Nicolò CESA-BIANCHI et Paul FISCHER : Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.*, 47(2-3):235–256, 2002.

[2] F. BOUSSEMART, F. HEMERY, C. LECOUTRE et L. SAÏS : Boosting Systematic Search by Weighting Constraints. *In ECAI*, pages 146–150, 2004.

[3] M. S. CHERIF, D. HABET et C. TERRIOUX : On the Refinement of Conflict History Search Through Multi-Armed Bandit. *In ICTAI*, pages 264–271, 2020.

[4] D. HABET et C. TERRIOUX : Conflict History based Search for Constraint Satisfaction Problem. *In SAC*, pages 1117–1122, 2019.

[5] C. LECOUTRE, L. SAIS, S. TABARY et V. VIDAL : Recording and Minimizing Nogoods from Restarts. *JSAT*, 1(3-4):147–167, 2007.

[6] L. MICHEL et P. Van HENTENRYCK : Activity-based search for black-box constraint programming solvers. *In CPAIOR*, pages 228–243, 2012.

[7] H. WATTEZ, F. KORICHE, C. LECOUTRE, A. PAPARRIZOU et S. TABARY : Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts. *In ECAI*, 2020.

[8] H. WATTEZ, C. LECOUTRE, A. PAPARRIZOU et S. TABARY : Refining Constraint Weighting. *In ICTAI*, pages 71–77, 2019.

On the Refinement of Conflict History Search Through Multi-Armed Bandit

Mohamed Sami Cherif, Djamal Habet, Cyril Terrioux
Aix-Marseille Univ, Université de Toulon, CNRS, LIS, France
{mohamedsami.cherif, djamal.habet, cyril.terrioux}@lis-lab.fr

Abstract—**Reinforcement learning has shown its relevance in designing search heuristics for backtracking algorithms dedicated to solving decision problems under constraints. Recently, an efficient heuristic, called Conflict History Search (CHS), based on the history of search failures was introduced for the Constraint Satisfaction Problem (CSP). The Exponential Recency Weighted Average (ERWA) is used to estimate the hardness of constraints and CHS favors the variables that often appear in recent failures. The step parameter is important in CHS since it controls the estimation of the hardness of constraints and its refinement may lead to notable improvements. The current research aims to achieve this objective. Indeed, a Multi-Armed Bandits (MAB) framework can select an appropriate value of this parameter during the restarts performed by the search algorithm. Each arm represents a CHS with a given value for the step parameter and it is rewarded by its ability to improve the search. A training phase is introduced earlier in the search to help MAB choose a relevant arm. The experimental evaluation shows that this approach leads to significant improvements regarding CHS and other state-of-the-art heuristics.**

Index Terms—**Constraint programming, Conflict History Search, Multi-Armed Bandits**

I. Introduction

The Constraint Satisfaction Problem (CSP) is used successfully in modeling and solving a large variety of academic and real-world problems [1]. In a CSP instance, the variables are defined by their domain values and the constraints specify the relations between variables. The corresponding problem consists in finding an assignment of all the variables that satisfies all the constraints. The solvers dedicated to this NP-complete problem are often based on backtracking algorithms with powerful embedded techniques such as filtering, learning, restarts and search heuristics [1]. A search or branching heuristic determines the manner of visiting the search space by deciding the next variable to assign or to fix. Such a heuristic has a significant impact on the size of the search tree developed by a backtracking algorithm, and consequently on its running time. Although finding the best branching variable while minimizing the search-tree size is NP-hard [2], many search heuristics have been proposed (e.g. [3], [4], [5], [6], [7], [8], [9], [10], [11]) and are intended to dynamically exploit solving related information such as filtering efficiency or constraint hardness.

This work has been published in the proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020. ISBN 978-1-7281-9228-4 DOI 10.1109/ICTAI50040.2020.00050

Recent research has shown the interest of machine learning in designing efficient search heuristics for CSP [12], [13] as well as for other decision problems [14]. One of the motivations is the difficulty of defining a heuristic which can have high performance on any considered instance. Indeed, a heuristic can perform very well on a family of instances while falling drastically on another [13]. In this context, reinforcement learning under the Multi-Armed Bandit (MAB) framework is employed to tend towards a relevant heuristic among a set of candidates which constitute the arms of the MAB. An arm/heuristic is selected accordingly at each node of the search tree [13] or at each restart of the backtracking algorithm [12].

Another motivation for considering reinforcement learning is to estimate the hardness of constraints while relying on the history of the search steps and, in particular, those leading to dead-ends. This estimation is then used to guide the search on variables appearing in hard constraints. Conflict History Search (CHS) [15] is designed with this in mind. The estimation of the hardness is achieved by the Exponential Recency Weighted Average (ERWA) [16]. ERWA calculates the constraint hardness value by considering its current value and a reward value which becomes higher when the constraint leads to dead-ends over short periods. An important parameter is involved in keeping a balance between these two values, namely the step-size value or step parameter. Although it is identified as a hard task, a relevant initialization of the step parameter may lead to significant improvements [15].

Hence, the main contribution of this paper is refining CHS by dealing with the problem of efficiently setting the step-size parameter in CHS. Indeed, a MAB framework is proposed on the basis of CHS while taking advantage of the restart mechanism in backtracking algorithms. Recall that restarts are widely used to prevent the heavy-tailed phenomena [17]. Each run has a duration expressed in terms of the number of decisions, conflicts or backtracks and a restart policy is chosen to define the effective duration of the runs, such as geometric [18] or Luby [19] sequences. The candidate heuristics or arms of the MAB are CHS instances which differ by the initial value of the step parameter taken from a range of given values.

Important components of the MAB framework are the reward assigned to each arm and the policy that selects the next arm to use after a restart. Even if any policy can be employed in this framework, UCB (Upper Confidence Bound) is used as it has shown its relevance in other work

[20], [12], [13]. Furthermore, any policy is supported by the reward function which should be able to estimate the positive impact of a heuristic on the efficiency of the search. Accordingly, a reward function is proposed to favor the arm that reaches dead-ends quickly. A training phase is also proposed consisting in a series of runs with an identical duration conducted according to a round-robin policy on the arms. The aim of this initialization phase is to speed-up the convergence of the MAB framework. The proposed MAB framework based on CHS, denoted by MAB-CHS, is evaluated on XCSP3 benchmarks[1]http://www.xcsp.org. The empirical results confirm the relevance of the proposed approach as MAB-CHS not only improves the performance of CHS but also shows improvements regarding similar approaches and other powerful heuristics.

The paper is organized as follows. Definitions and notations are given in Section II and related work is described in Section III. The proposed MAB-CHS framework is detailed in Section IV and experimentally evaluated in Section V. We conclude and discuss perspective work in Section VI.

II. Background

An instance of a Constraint Satisfaction Problem (CSP) consists of a triplet (X, D, C) such that $X = \{x_1, \cdots, x_n\}$ is a set of n variables, $D = \{D_1, ..., D_n\}$ is a set of finite domains, and $C = \{c_1, \cdots, c_e\}$ is a set of e constraints. The domain of each variable x_i is D_i. Each constraint c_j is defined by its scope $S(c_j) = \{x_{j_1}, \cdots, x_{j_k}\} \subseteq X$ and its compatibility relation $R(c_j) \subseteq D_{j_1} \times \cdots \times D_{j_k}$. The constraint satisfaction problem consists in finding an assignment of the variables $x_i \in X$, within their respective domains D_i ($1 \leq i \leq n$), which satisfies each constraint in C. Such an assignment is said to be consistent and is a solution of the given instance. Checking whether a CSP instance has a solution is NP-complete [1].

CSP solving is mainly based on a backtracking algorithm while maintaining a given level of consistency (e.g. the Generalized Arc Consistency property [21]). A usual algorithm is Maintaining Arc Consistency (MAC) [22]. A search tree is constructed where each node corresponds to a decision ($x_i = v$ or $x_i \neq v$) with respect to a selected variable x_i which is not yet fixed and a domain value v. The choice of the variable x_i is taken according to a search heuristic (i.e. variable branching heuristic). MAC can embed sophisticated techniques, such as nogood recording and efficient handling of global constraints [1], leading to high performance. Furthermore, restarts are used to address the heavy-tail phenomena related to irrelevant decisions during the search [17]. The search algorithm performs a series of runs. Each run has a duration expressed in terms of the authorized number of decisions, conflicts, backtracks, etc. The duration of each run is determined following several policies, such as geometric [18] or Luby [19] sequences.

In a Multi-Armed Bandit framework (MAB), a bandit, i.e. an agent, has to choose an arm from a set of candidates by relying on information collected through a sequence of trials. The information available to the bandit are rewards attributed to each arm. One of the earliest MAB models, stochastic MAB, was introduced in [23]. As a decision process, a MAB faces an important dilemma which is the trade-off between exploitation and exploration, i.e. the bandit needs to explore underused arms often enough to have a robust feedback while also exploiting good candidates which have the best rewards. To this end, many policies have been devised such as the ϵ-Greedy strategy [16], which does random exploration, Thompson Sampling [24] and the Upper Confidence Bound (UCB) family [25], [26], [27], which conduct smart exploration adequate for uncertain environments, among others.

III. Related Work

The state-of-the-art in terms of search heuristics is very rich. Indeed, many heuristics have been proposed aiming mainly to satisfy the *first-fail principle* [28] which advises "to succeed, try first where you are likely to fail". In this context, efficient heuristics are adaptive and dynamic, by basing the branching decisions on information gathered throughout the solving process, such as the effectiveness of filtering as in ABS [9] and IBS [10] or the hardness of constraints as employed in *dom/wdeg* [5] and its variants [8], [12], [11]. In addition, heuristics such as LC [29] and COS [30] attempt to consider the search history and require the use of auxiliary heuristics.

The Conflict History-Based (CHB) branching heuristic [31], based on the Exponential Recency Weighted Average (ERWA) [16], was introduced for the satisfiability problem. CHB rewards the activity of variables favoring the ones that are involved in recent conflicts. In particular, the reward associated to each variable is updated whenever it is branched on or propagated and its score is calculated after each restart. The Learning Rate Branching (LRB) heuristic [14] extends CHB by exploiting locality and introducing the learning rate of variables. Recently, CHB was also implemented in the context of CSP [32]. The *Conflict-History Search* (CHS) heuristic [15] is also inspired from CHB and similarly considers the history of constraint failures, favoring the variables that are involved in recent failures. Conflicts are dated and the constraints are weighted on the basis of ERWA. Weights are coupled with the variable domains to calculate the Conflict-History scores of the variables. The present work revolves around the CHS heuristic, which we will address more in detail in the next section, and how to refine it using reinforcement learning techniques.

Reinforcement learning techniques have already been studied in constraint programming. In particular, the Multi-Armed Bandit (MAB) framework was used to select adaptively the consistency level of propagation at each node of the search tree [20]. MAB was also used to select a restart strategy in [33]. Moreover, a linear regression method was applied to learn the scoring function of value heuristics [34]. Rewards, calculated for each node of the search tree, were also used to select adaptively a backtracking strategy in [35]. Furthermore, a learning process based on the Least Squares Policy Iteration technique was used to tune adaptively the parameters of stochastic local search algorithms [36]. Upper Confidence Bound (UCB) and Thompson Sampling techniques were employed to select

[1]http://www.xcsp.org

automatically, at each node of the search tree, a branching heuristic for CSP, among a set of candidate ones [13]. A recent work proposes a MAB framework to select at each restart in a backtracking algorithm a search heuristic among a set of candidate ones, such as *ABS*, *IBS* or CHS [12]. In particular, a reward function was designed to favor, among the set of candidate heuristics, those able to prune large parts of the search tree and the UCB policy was identified as the most powerful in this framework. In our work we use an algorithm in the UCB family, namely UCB1 [26], to refine the CHS heuristic as we will explain in the next section.

IV. MAB Framework for the Refinement of CHS

In this section, we explain how MAB can be used to address the refinement of CHS by selecting a relevant value for the step parameter. In particular, we introduce our reward function which is based on search failures and we explain the UCB policy used in the MAB framework. Finally, we present and describe a training phase which can help initialize the MAB parameters in order for it to converge quickly.

A. An overview of CHS

The Conflict History Search (CHS) heuristic considers the history of constraint failures and favors the variables that often appear in recent ones [15]. The conflicts are dated and the constraints are weighted on the basis of the Exponential Recency Weighted Average (ERWA) [16]. These weights are coupled with the variable domains to calculate the Conflict-History scores of the variables. More precisely, CHS maintains for each constraint c_j a score $q(c_j)$ which is initialized to 0. If c_j leads to a failure during the search because the domain of a variable in $S(c_j)$ is emptied by propagation then $q(c_j)$ is updated by the formula below derived from ERWA.

$$q(c_j) = (1 - \alpha) \times q(c_j) + \alpha \times r(c_j)$$

The parameter $0 < \alpha < 1$ is the step-size, also referred to as the step parameter, and $r(c_j)$ is the reward value for constraint c_j. The parameter α fixes the importance given to the old value of the score q at the expense of the reward r and its value decreases over time. Starting from an initial value α_0, α decreases by 10^{-6} at each constraint failure to a minimum of 0.06. Higher rewards are given to constraints that fail regularly over short periods according to the following formula:

$$r(c_j) = \frac{1}{Conflicts - Conflict(c_j) + 1}$$

$Conflicts$ is the number of conflicts that occurred since the beginning of the search. $Conflict(c_j)$ is updated to the current value of $Conflicts$ when c_j leads to a failure. The Conflict-History score of a variable $x_i \in X$ is defined as follows:

$$chv(x_i) = \frac{\sum\limits_{c_j \in C:\ x_i \in S(c_j) \wedge |Uvars(c_j)|>1} (q(c_j) + \delta)}{|D_i|}$$

$Uvars(c_j)$ denotes the set of unassigned variables in $S(c_j)$ and D_i is the current domain of x_i updated by propagation. The δ parameter is a positive real number close to 0 used to guide branching according to the degree of the variables instead of randomly, at the beginning of the search. CHS chooses the variable to branch on with the highest chv value. It focuses the branching on variables with a small domain size belonging to constraints which appear recently and repetitively in conflicts. When CHS is used in a backtracking algorithm with restarts, $Conflict(c_j)$ and $q(c_j)$ are not reinitialized from a run to another. Nevertheless, the scores of all constraints $c_j \in C$ are smoothed at each restart as follows:

$$q(c_j) = q(c_j) \times 0.995^{Conflicts - Conflict(c_j)}$$

More importantly, for the rest of the paper, CHS resets the value of α to α_0 at each new run.

B. A Multi-Armed Bandit for an Adaptive CHS

The experimental evaluation conducted on CHS reveals that the initial value of the parameter α, i.e. α_0, can be better adjusted to improve the effectiveness of CHS [15]. As an indication, when CHS was tested within a backtracking algorithm with nine different α_0 values, ranging from 0.1 to 0.9 (with a step of 0.1), the Virtual Best Solver (VBS) was able to solve several dozen additional instances [15]. In order to achieve this objective, the present work proposes to use the Multi-Armed Bandit (MAB) framework to choose the relevant values of the α_0 parameter which is of particular importance in CHS. Such a choice will be made at each restart done by the backtracking algorithm. Each arm of the MAB corresponds to a CHS with a different initial α value, i.e. α_0. A training phase is also proposed for a better initialization of the parameters of the arms. Next, we will introduce the proposed MAB framework, detail its use and particularly explain our choice of the reward function.

1) MAB Framework: For a given $i \in \{1 \cdots K\}$, $CHS(\alpha_0^i)$ is defined as CHS where α_0 is set to α_0^i at the beginning of the current restart or run. The arms of the MAB are the candidate heuristics $CHS(\alpha_0^i)$, $i = 1 \cdots K$. Considering a set of K CHS heuristics with different α_0 values and a CSP instance to solve, the proposed framework selects a heuristic $CHS(\alpha_0^i)$ where $i \in \{1 \cdots K\}$ at each restart of the backtracking algorithm according to a MAB policy called Upper Confidence Bound (UCB) [26]. To choose an arm, UCB relies on a reward function calculated at the end of each restart to estimate the performance of the backtracking algorithm regarding the employed search heuristic.

2) Reward Function: An important factor in the efficiency of MAB and, particularly UCB, is the reward function. Indeed, the reward function must reflect the impact of a heuristic on the efficiency of the backtracking algorithm while relying on the information that can be assessed during each run in which it is used. Several criteria can be considered, such as the ability of a heuristic to reduce the size of the search space or to quickly reach failures.

In the proposed MAB framework, any reward function can be used and adapted to a suitable case of use regarding, for example, the instance features or any other criteria. For the current work, several functions have been evaluated through extensive experimentation which highlighted a stronger preference for a reward estimating the ability of a heuristic to reach failures quickly. This is not surprising as a good part of state-of-the-art heuristics are based on the first-fail principle [28].

Furthermore, the proposed MAB framework is independent from the restart policy, which can be related to the number of conflicts, backtracks, decisions, etc. In this work, the run duration is based on the maximum number of authorized backtracks before achieving a new restart. nc_t denotes the number of conflicts which are permitted during the run t. At each conflict $j = 1 \cdots nc_t$, nu_j variables remain unassigned. Therefore, at conflict j, the ratio p_j of unassigned variables is $p_j = nu_j/n$ where $n = |X|$. If a heuristic $CHS(\alpha_0^i)$ (arm i) is used at the current run t, its reward is calculated at the end of this run as follows:

$$R_t(i) = \frac{\sum_{j=1}^{nc_t} p_j}{nc_t}$$

Hence, this reward function consists in an average of the ratios of unassigned variables encountered during conflicts, assessing the heuristic's ability to quickly identify inconsistent assignments. In other words, the fewer the variables are assigned, the greater the assigned reward value will be for the corresponding heuristic. Finally, one should observe that the values of the rewards are defined in the interval $[0, 1]$.

3) UCB Policy: Consider that t runs are performed since the beginning of the search. The employed MAB policy for selecting the next heuristic is UCB1 [26] which is an algorithm in the Upper Confidence Bound family [25], [26], [27]. UCB1 maintains the following parameters for each candidate arm i:

- $n_t(i)$ is the number of times that an arm i is selected during the t runs,
- $\widehat{R}_t(i)$ is the empirical mean of the rewards of arm i over the t runs.

Accordingly, UCB1 selects the arm that maximizes the following term:

$$\widehat{R}_t(i) + c.\sqrt{\frac{ln(t)}{n_t(i)}} \quad (1)$$

The left side of the formula, i.e. $\widehat{R}_t(i)$, aims to put emphasis on candidate heuristics (arms) that received the highest rewards. Conversely, the right side, i.e. $\sqrt{\frac{ln(t)}{n_t(i)}}$, ensures the exploration of underused arms. The parameter c can help to appropriately balance the interchange between the exploitation and exploration phases in the MAB framework.

A strategy for MAB is evaluated by its expected cumulative regret, i.e., the difference between the cumulative expected value of the reward if the best arm is used at each restart and its cumulative value for the total runs of the MAB, denoted T. If $a_t \in \{1 \cdots K\}$ denotes the arm chosen at run t, the expected cumulative regret is formally defined as follows:

$$R_T = \max_{1 \leq i \leq K} \sum_{t=1}^{T} \mathbf{E}[R_t(i)] - \sum_{t=1}^{T} \mathbf{E}[R_t(a_t)]$$

In particular, UCB1 guaranties an expected cumulative regret no worse than $O(\sqrt{T.K.\ln K})$.

Note that MAB with UCB has been initially defined for rewards following a stationary probability distribution. In our case, the probability distribution is clearly unknown and we cannot ensure that it is stationary. Indeed, the rewards may evolve during the search since we consider MAC with restarts [37] as solving algorithm. As it records some nogoods when a restart occurs, the problem changes over time. There exists variants of UCB adapted for changing environments. However, it is known that the standard version is unlikely to exhibit pathological behavior in such environments [38]. So using UCB1 in this context is relevant.

4) Training Phase: The number of restarts often seems insufficient to fully evaluate the potential of each candidate heuristic thus rendering difficult for the MAB framework to converge quickly to the most adequate arms for the CSP instance. For this reason, a training phase can be conducted. It consists in choosing each arm one at a turn, like a round-robin policy on the different values of α_0. Each arm, among the K candidates, will be attributed the same restart duration limit. In other words, restarts will have a constant duration d_t in this training phase so as to not favor one arm at the expense of another. The round-robin is repeated a certain number of times denoted by d_p.

This training phase can be considered as an initialization step for the parameters used in the MAB framework, especially the reward values and their means. Once the training is completed, UCB1 takes over to select an arm among the candidate search heuristics. It should be noted that by doing so, the expected regret of UCB1 remains in $O(\sqrt{T.K.\ln K})$. Indeed, the training phase is launched first, it has no effect on the expected regret of UCB1. Furthermore, during the training phase, in addition to the initialization of MAB parameters, the weights of CHS are updated during each run when a conflict occurs or at the end of a run through smoothing. This phase may also lead to the discovery of some nogoods if the solver is able to handle them like in [37]. Finally, the training phase alone may suffice to solve some instances.

V. Experimental Evaluation

In this section, we first describe the experimental protocol we use. In Subsection V-B1, we observe the sensitivity of MAB-CHS(+Train) to some parameters. In Subsection V-C, we assess the benefits of the training phase. Afterwards, we compare our MAB approach with state-of-the-art search heuristics in Subsection V-D. Finally, we combine our MAB framework with Last Conflict (LC) heuristic and measure its performance in Subsection V-E.

A. Experimental Protocol

We consider all the CSP instances from the XCSP3 repository[2] and the XCSP3 2018 competition[3], resulting in 16,947 instances. XCSP3, for XML-CSP version 3, is an XML-based format to represent instances of combinatorial constrained problems. Our solver (the same as [15]) is compliant with the rules of the competition except that the global constraints `cumulative`, `circuit` and some variants of the `allDifferent` constraint (namely `except` and

[2]http://www.xcsp.org/series

[3]http://www.cril.univ-artois.fr/XCSP18/

list) and the noOverlap constraint are not yet supported. Consequently, from the 16,947 obtained instances, we first discard 1,233 unsupported ones. We also remove 2,813 instances which are detected as inconsistent by the initial arc-consistency preprocessing and, thus, having no interest for the present comparison. Hence, our benchmark contains 12,901 instances, including notably structured instances and instances with global constraints.

We consider the following classical heuristics $dom/wdeg$, ABS, wdeg$^{ca.cd}$ [11] and CHS. We do not take into account IBS [10] and CHB [32] which turn to be less relevant [15], [11]. For ABS, we fix the decay parameter γ to 0.999 as in [9]. In CHS, δ is set to 0.0001 while we consider various values of α_0, notably 0.1 and 0.4. For MAB-based heuristics, in addition to our proposed MAB approach, we consider the one defined in [12], which we will refer to as MAB-VH (for MAB with Various Heuristics). MAB-VH exploits five heuristics namely ABS, IBS, dom/ddeg [39], CHS(0.4) and wdeg$^{ca.cd}$ [12]. It aims to take benefit of the heterogeneity of these heuristics in order to identify a relevant one. It relies on a UCB1 policy with $c = 2\sqrt{2}$ and a reward function based on the size of the pruned trees, this size being expressed in terms of the product of the size of the domains of unassigned variables. On the other hand, we discard the MAB approach described in [13] since, in practice, it performs worse than MAB-VH [12]. In order to make the comparison fair, the lexicographic order is used for the choice of the next value to assign within the domains of the variables.

Regarding the solving step, we exploit MAC with restarts [37]. Restarts can be based on various policies. In our experiments, the duration of a run can be measured in the number of backtracks or in the number of decisions. Its evolution can follow a geometric or a Luby sequence. By default, when MAC exploits a classical heuristic alone (resp. MAB-CHS(+Train)), we consider backtrack-based geometric sequences for which the initial cutoff is set to 100 (resp. 50) and the increasing factor to 1.1 (resp. 1.05). For MAB-VH, we consider the setting of [12] with a decision-based Luby sequence whose initial cutoff is set to 100. Moreover, for the sake of simplicity, the duration d_t used in the training phase is equal to the initial cutoff of the restart policy. By default, we set d_p to 10.

We have written our own C++ code to implement all the compared branching heuristics in this section as well as MAC with restarts. By doing so, we avoid any bias related to the way the heuristics or the solver are implemented. In particular, the branching heuristics are all implemented with equal refinement and care, following the recommendations outlined in [40]. The experiments are performed on Dell PowerEdge R440 servers with Intel Xeon Silver 4112 processors (clocked at 2.6 GHz) under Ubuntu 18.04. Each solving process is allocated a slot of 30 minutes and at most 16 GB of memory per instance. The noted time is the cumulative runtime, i.e. the sum of the runtime over all the considered instances.

B. Parameter Sensitivity

1) Exploration vs. Exploitation: We are interested in assessing the impact of the value of parameter c (in the right term of Equation 1) on the ability of MAC with MAB-CHS to solve instances. In the literature, various values of c are considered. This is an important parameter since it controls the balance between the exploitation and exploration phases in the MAB framework. As mentioned before, the reward values in MAB-CHS, and also in MAB-VH, belong to $[0, 1]$. Nonetheless, the values we observe in practice are seldom close to 1. On the contrary, they are often less than 0.5. This trend is even more pronounced when considering $\widehat{R}_t(i)$ due to the effect of the mean. As a consequence, the MAB tends more often towards the exploration part than the exploitation one and, therefore, may be close to a random MAB. Figure 1 depicts the number of solved instances as a function of the elapsed time for different values of c. Clearly, the three values of c we consider lead to close results. However, the value $c = 1$ obtains slightly better results. Indeed it succeeds in solving 6 additional instances (resp. 5) w.r.t. $c = 0.5$ (resp. $c = 2$). Note that this behavior is close to that of MAC with a random MAB, which solves six instances less than MAC with our MAB for $c = 1$. Therefore, in the following experiments, c is set to 1.

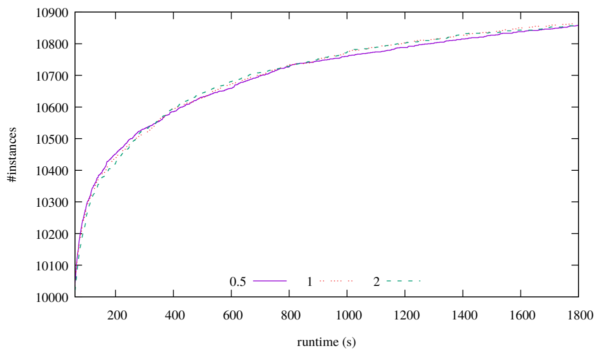


Fig. 1. Number of solved instances as a function of the elapsed time (from 60 s) for MAC with MAB-CHS when c is set respectively to 0.5, 1 or 2.

2) Restart Policy Setting: We consider the increasing factor of the geometric sequence. Indeed this parameter can also influence the behaviour of our MAB. As we can see in Figure 2, values 1.05 and 1.075 turn out to be relevant trade-offs between a weak increase (but more restarts) and a more rapid one (but less restarts). Note that more restarts can mean a better chance for the MAB to make a relevant choice. But, at the same time, it can lead to a less effective search. We can also observe that value 1.05 leads to the best results.

3) Training Phase Setting: We assess the impact of the value of d_t. As the value of d_t is the initial cutoff of the used restart policy, the considered values do not exceed 75 in order to ensure that the duration of runs grows slowly. By doing so, we expect to have enough restarts for the MAB to be able to identify a relevant arm. Figure 3 provides the results of MAC with MAB-CHS+Train. Clearly, $d_t = 50$ turns out to be the best value. Indeed, it allows MAC to solve 10,875 instances against 10,861 and 10,843 respectively for 25 and 75. This gain in number of solved instances is accompanied by a slight reduction of cumulative runtime (1,038 h for 50 against 1,042 h and 1,041 h for 25 and 75). So, $d_t = 50$

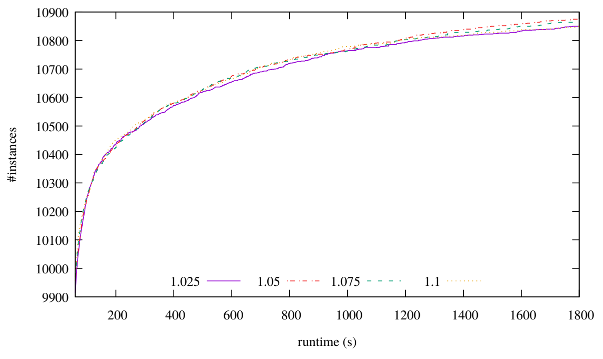


Fig. 2. Number of solved instances as a function of the elapsed time (from 60 s) for MAC with MAB-CHS+Train when the increasing factor is set respectively to 1.025, 1.05, 1.075 or 1.1.

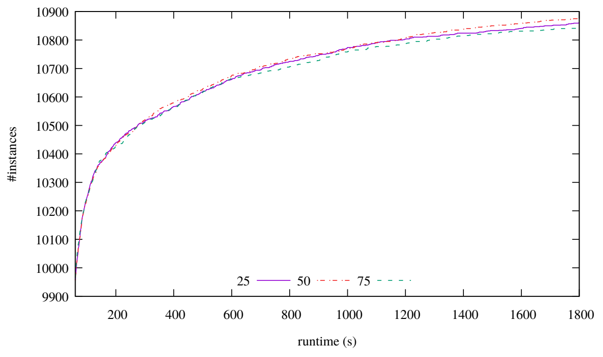


Fig. 3. Number of solved instances as a function of the elapsed time (from 60 s) for MAC with MAB-CHS+Train when d_t is set respectively to 25, 50 or 75.

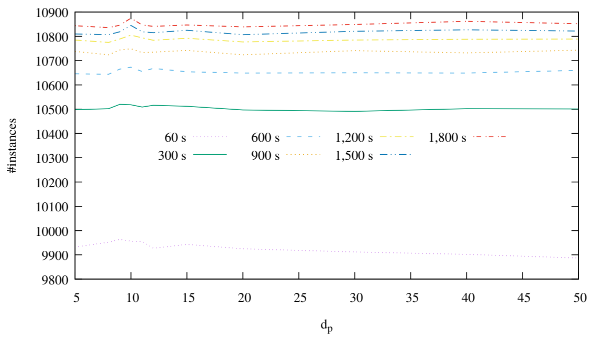


Fig. 4. Number of solved instances within x seconds as a function of the value of d_p for MAC with MAB-CHS+Train.

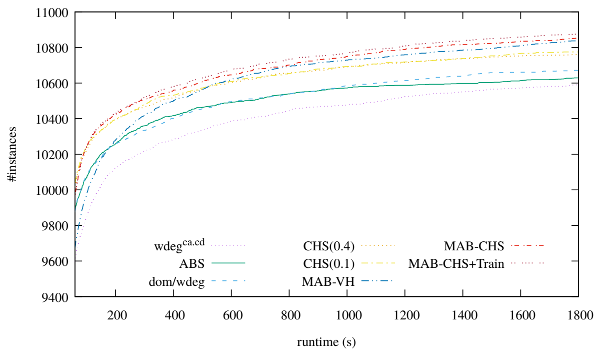


Fig. 5. Number of solved instances as a function of the elapsed time (from 60 s) for MAC with each considered heuristic.

seems to make a good trade-off between the duration of runs and their number.

Finally, We study the MAB-CHS+Train behavior w.r.t. the value of d_p. By varying the value of d_p, the number of instances solved by MAC with MAB-CHS+Train ranges between 10,836 and 10,875. We can clearly observe in Figure 4 the existence of a peak for $d_p = 10$ whatever the limit of runtime we consider above 300 s. Furthermore, the gap between $d_p = 10$ and the other values of d_p is quite significant. For instance, for a timeout of 1,800 s, in most cases, MAC with $d_p = 10$ solves between 20 and 39 additional instances.

To sum up, in the next experiments, we consider the following settings $c = 1$, $d_t = 50$, $d_p = 10$ and an increasing factor of 1.05.

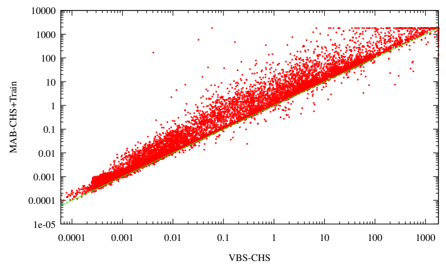
C. Benefits of the Training Phase

First, we compare the results of MAC with MAB-CHS and MAB-CHS+Train (see Figure 5). MAB-CHS+Train leads to the best results. Notably, MAC with MAB-CHS+Train succeeds in solving 10,875 instances against 10,853 for MAC with MAB-CHS. The gain may seem relatively low. However, as we can see in Figure 5, most of the instances are solved easily since, more than 10,000 of them are solved in less than one minute. If we consider the Best Virtual Solver (denoted VBS-CHS) which returns the best answer obtained by MAC with a given $CHS(\alpha_0)$ among the nine considered here, we observe that it solves 10,987 instances. Most of the additional instances it solves are solved by a small number of $CHS(\alpha_0)$ (often only one). So we can suppose that these instances are hard. Note that about 74% of solved instances are solved during the training phase.

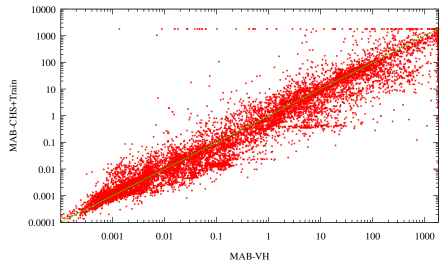
Now, we compare MAC with MAB-CHS+Train to VBS-CHS (see Figure 6(a)). Note that by using a MAB approach, our aim is to get as close as possible to VBS-CHS. Of course, MAC with MAB-CHS+Train solves less instances than VBS-CHS and is slower, but this comparison is not fair since the virtual best solver is only a theoretical tool. However, we can observe that the runtime of MAC with MAB-CHS+Train is generally competitive w.r.t. that of VBS-CHS. Interestingly, we observe that MAC with MAB-CHS+Train is able to outperform VBS-CHS on some instances, and even solve instances which are not solved by VBS-CHS within the timeout.

D. MAB-CHS(+Train) vs. Other Search Heuristics

In this subsection, we first compare MAB-CHS(+Train) to some classical branching heuristics of the state-of-the-art. According to Figure 5, MAC with MAB-CHS or MAB-CHS+Train outperforms MAC with any classical heuristic. For MAC with MAB-CHS+Train, the gain in the number of



(a)



(b)

Fig. 6. Runtime comparison of MAB-CHB+Train vs. VBS-CHB (a) and MAB-VH (b).

solved instances ranges between 95 and 279 instances. The best classical heuristic is CHS with $\alpha_0 = 0.1$. Note that 0.1 corresponds to the value of α_0 giving the best results when α_0 varies between 0.1 and 0.9 by step of 0.1. One could consider that the good performance of MAC with MAB-CHS(+Train) is related to the use of the best settings. This is not the case. Indeed, whatever the settings we have considered, we observe that MAC with MAB-CHS(+Train) solves at least 10,836 instances, that is more than 50 additional instances than any classical heuristic.

Next, we compare MAB-CHS(+Train) to another MAB-based approach, namely MAB-VH [12]. MAC with MAB-CHS+Train (resp. MAB-CHS) solves 38 additional instances (resp. 26) than MAB-VH. Roughly, the results of MAB-VH are close to the ones with the worst settings we have considered for MAB-CHS(+Train). Regarding the cumulative runtime, MAB-CHS+Train allows MAC to be faster than MAB-VH (1,038 h vs. 1,068 h). Figure 6(b) confirms this trend where a majority of the instances are solved faster by MAC with MAB-CHS+Train. This result was not a foregone conclusion. Indeed, if both MAB frameworks aim to identify the most relevant heuristic, one may expect that a MAB with different heuristics is more likely to find a relevant one than a MAB using the same heuristic with a different settings in each arm. This can be all the more surprising since the number of instances solved by each CHS variant are close to each other as shown in [15]. One possible explanation lies in the fact that, in practice, different values of α_0 enable us to obtain diversified heuristics. This trend seems to be sustained by the fact that VBS-CHS is able to solve several dozen additional instances than any considered $CHS(\alpha_0^i)$ [15].

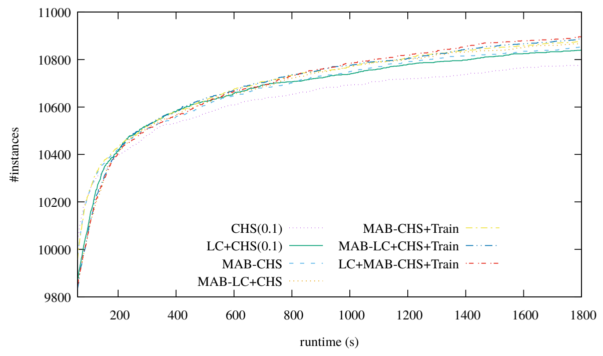


Fig. 7. Number of solved instances as a function of the elapsed time (from 60 s) for MAC with CHS(0.1), MAB-CHS(+Train) and their combination with LC.

E. Combination with LC

The Last Conflict (LC) heuristic [29] aims to focus on the last encountered conflict and chooses a new branching variable only when needed, i.e. when there is no current conflict. In the latter case, it relies on an auxiliary heuristic to make this choice. LC can be used in two different ways with a MAB approach. The first one consists in considering the MAB as the auxiliary heuristic exploited by LC. We denote LC+MAB-CHS(+Train) this version. The second one consists in using LC in each arm of the MAB. In other words, for each arm i, instead of $CHS(\alpha_0^i)$, we consider LC with $CHS(\alpha_0^i)$ as an auxiliary heuristic. We denote MAB-LC+CHS(+Train) this version.

In Figure 7, we compare the behavior of MAC when using these two versions with and without training. We also consider $CHS(0.1)$ and LC+$CHS(0.1)$. Clearly, the use of LC improves the behavior of MAC whatever the auxiliary heuristic we use. We can also observe that the gains, in terms of the number of solved instances, is less important when the auxiliary heuristic alone leads to good results. It seems that the less efficient the heuristic is, the easier it is to solve additional instances. Indeed, incorporating LC has no impact on the hierarchy given in the previous subsection. Moreover, using LC does not make better a given auxiliary heuristic than the next auxiliary heuristic in this hierarchy. For instance, LC+CHS(0.1) does not perform better than CHS-MAB and similarly MAB-LC+CHS performs worse in comparison to MAB-CHS+Train. Finally, the best results are achieved by MAC when using LC+MAB-CHS+Train. It succeeds in solving 22 additional instances w.r.t. MAB-CHS+Train. The gain w.r.t. classical heuristics is even more significant. For example, if we consider the best one, i.e. CHS(0.1), 54 additional instances are solved if we combine LC and CHS, 117 otherwise.

VI. Conclusion and Future Work

Algorithms (or solvers) dedicated to CSP are more and more powerful, but may include parameters which are hard to fix while their values depend on the input instance or any other characteristic. The present work has drawn a framework based on MAB to refine a single heuristic, namely CHS, regarding an important parameter for estimating the hardness of the constraints following ERWA. The experimental evaluation has validated this approach. Indeed, the performance of CHS was improved and the proposed framework is shown competitive with the state-of-the-art heuristics. MAB-CHS components were also widely investigated. To summarise, our results establish the following hierarchy between the evaluated heuristics in terms of the number of solved instances on the considered benchmark: LC+MAB-CHS+Train $>$ MAB-CHS+Train $>$ MAB-CHS $>$ MAB-VH $>$ classical heuristics ($dom/wdeg$, ABS, wdeg$^{ca.cd}$ and CHS). The originality of this work relies on using MAB with a set of CHS instances differing by the initialization value of the step parameter, while similar work use different heuristics in the MAB framework. The proposed training phase is another contribution which shows its benefits in improving the behavior of the proposed framework.

As perspective of our work, it would be interesting to study the effect of combining our MAB framework with the one introduced in [12], MAB-VH, by adding different heuristics from CHS as arms to MAB-CHS(+Train). Furthermore, refining the reward function by relying on a combination of different criteria [41], [42] may lead to the improvement of our framework as it is difficult to fully reflect the behavior of solvers, which have complex environments, while relying on a single criterion. Finally, it would be relevant to apply the same framework on close problems such as SAT, where the reward function may be related to the quality of clause learning [43], which is an important module in modern SAT solvers.

Acknowledgment

This work has been funded by the Agence Nationale de la Recherche project ANR-16-CE40-0028.

References

[1] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence. Elsevier, 2006, vol. 2.

[2] P. Liberatore, "On the complexity of choosing the branching literal in DPLL," *Artificial Intelligence*, vol. 116, no. 1-2, 2000.

[3] C. Bessière, A. Chmeiss, and L. Saïs, "Neighborhood-based variable ordering heuristics for the constraint satisfaction problem," in *Proceedings of CP*, 2001, pp. 565–569.

[4] C. Bessière and J.-C. Régin, "MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems," in *Proceedings of CP*, 1996, pp. 61–75.

[5] F. Boussemart, F. Hemery, C. Lecoutre, and L. Saïs, "Boosting Systematic Search by Weighting Constraints," in *Proceedings of ECAI*, 2004, pp. 146–150.

[6] P. A. Geelen, "Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems," in *Proceedings of ECAI*, 1992, pp. 31–35.

[7] S. W. Golomb and L. D. Baumert, "Backtrack programming," *Journal of the ACM*, vol. 12, pp. 516–524, 1965.

[8] E. Hebrard and M. Siala, "Explanation-Based Weighted Degree," in *Proceedings of CPAIOR*, 2017, pp. 167–175.

[9] L. Michel and P. V. Hentenryck, "Activity-based search for black-box constraint programming solvers," in *Proceedings of CPAIOR*, 2012, pp. 228–243.

[10] P. Refalo, "Impact-based search strategies for constraint programming," in *Proceedings of CP*, 2004, pp. 557–571.

[11] H. Wattez, C. Lecoutre, A. Paparrizou, and S. Tabary, "Refining Constraint Weighting," in *Proceedings of ICTAI*, 2019, pp. 71–77.

[12] H. Wattez, F. Koriche, C. Lecoutre, A. Paparrizou, and S. Tabary, "Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts," in *Proceedings of ECAI*, 2020.

[13] W. Xia and R. H. C. Yap, "Learning Robust Search Strategies Using a Bandit-Based Approach," in *Proceedings of AAAI*, 2018, pp. 6657–6665.

[14] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning Rate Based Branching Heuristic for SAT Solvers," in *Proceedings of SAT*, 2016, pp. 123–140.

[15] D. Habet and C. Terrioux, "Conflict History based Search for Constraint Satisfaction Problem," in *Proceeding of SAC, Knowledge Representation and Reasoning Technical Track*, 2019, pp. 1117–1122.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[17] C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz, "Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems," *Journal of Automated Reasoning*, vol. 24, no. 1/2, pp. 67–100, 2000.

[18] T. Walsh, "Search in a Small World," in *Proceedings of IJCAI*, 1999, pp. 1172–1177.

[19] M. Luby, A. Sinclair, and D. Zuckerman, "Optimal speedup of Las Vegas algorithms," *Information Processing Letters*, vol. 47(4), pp. 173–180, 1993.

[20] A. Balafrej, C. Bessiere, and A. Paparrizou, "Multi-Armed Bandits for Adaptive Constraint Propagation," in *Proceedings of IJCAI*, 2015, pp. 290–296.

[21] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, p. 99–118, 1977.

[22] D. Sabin and E. C. Freuder, "Contradicting Conventional Wisdom in Constraint Satisfaction," in *Proceedings of ECAI*, 1994, pp. 125–129.

[23] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in applied mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[24] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[25] R. Agrawal, "Sample mean based index policies by o (log n) regret for the multi-armed bandit problem," *Advances in Applied Probability*, vol. 27, no. 4, pp. 1054–1078, 1995.

[26] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, 2002.

[27] E. Kaufmann, O. Cappé, and A. Garivier, "On Bayesian Upper Confidence Bounds for Bandit Problems," in *Proceedings of AISTATS*, 2012, pp. 592–600.

[28] R. M. Haralick and G. L. Elliot, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, pp. 263–313, 1980.

[29] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal, "Last Conflict Based Reasoning," in *Proceedings of ECAI*, 2006, pp. 133–137.

[30] S. Gay, R. Hartert, C. Lecoutre, and P. Schaus, "Conflict Ordering Search for Scheduling Problems," in *Proceedings of CP*, G. Pesant, Ed., 2015, pp. 140–148.

[31] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Exponential Recency Weighted Average Branching Heuristic for SAT Solvers," in *Proceedings of AAAI*, 2016, pp. 3434–3440.

[32] C. Schulte, "Programming branchers," in *Modeling and Programming with Gecode*, C. Schulte, G. Tack, and M. Z. Lagerkvist, Eds., 2018, corresponds to Gecode 6.0.1.

[33] M. Gagliolo and J. Schmidhuber, "Learning Restart Strategies," in *Proceedings of IJCAI*, 2007, pp. 792–797.

[34] G. Chu and P. J. Stuckey, "Learning Value Heuristics for Constraint Programming," in *Integration of AI and OR Techniques in Constraint Programming*. Springer International Publishing, 2015, pp. 108–123.

[35] I. Bachiri, J. Gaudreault, C. Quimper, and B. Chaib-draa, "RLBS: An Adaptive Backtracking Strategy Based on Reinforcement Learning for Combinatorial Optimization," in *Proceedings of ICTAI*, 2015, pp. 936–942.

[36] R. Battiti and P. Campigotto, *An Investigation of Reinforcement Learning for Reactive Search Optimization*. Springer Berlin Heidelberg, 2012, pp. 131–160.

[37] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal, "Recording and Minimizing Nogoods from Restarts," *JSAT*, vol. 1, no. 3-4, pp. 147–167, 2007.

[38] Álvaro Fialho, L. D. Costa, and M. S. Marc Schoenauer, "Analyzing bandit-based adaptive operator selection mechanisms," *Ann. Math. Artif. Intell.*, vol. 60(1-2), pp. 25–64, 2010.

[39] B. M. Smith and S. A. Grant, "Trying Harder to Fail First," in *Proceedings of ECAI*, 1998, pp. 249–253.

[40] J. N. Hooker, "Testing Heuristics: We Have It All Wrong," *Journal of Heuristics*, vol. 1(1), pp. 33–42, 1995.

[41] W. Chu, L. Li, L. Reyzin, and R. Schapire, “Contextual Bandits with Linear Payoff Functions,” in *Proceedings of AISTATS*, 2011, pp. 208–214.

[42] L. Li, W. Chu, J. Langford, and R. E. Schapire, “A Contextual-Bandit Approach to Personalized News Article Recommendation,” in *Proceedings of WWW*, 2010, pp. 661–670.

[43] N. Eén and N. Sörensson, “An extensible SAT-solver,” in *Proceedings of SAT*, 2003, pp. 502–518.