Best Heuristic Identification for Constraint Satisfaction

Frederic Koriche[1] , **Christophe Lecoutre**[1] , **Anastasia Paparrizou**[1] and **Hugues Wattez**[2]
[1]CRIL, Univ. Artois & CNRS
[2]LIX CNRS, École Polytechnique, Institut Polytechnique de Paris
{koriche,lecoutre,paparrizou}@cril.fr, wattez@lix.polytechnique.fr

Abstract

In constraint satisfaction problems, the *variable ordering heuristic* takes a central place by selecting the variables to branch on during backtrack search. As many hand-crafted branching heuristics have been proposed in the literature, a key issue is to identify, from a pool of candidate heuristics, which one is the best for solving a given constraint satisfaction task. Based on the observation that modern constraint solvers are using restart sequences, the *best heuristic identification problem* can be cast in the context of multi-armed bandits as a *non-stochastic best arm identification problem*. Namely, during each run of some given restart sequence, the bandit algorithm selects a heuristic and receives a reward for this heuristic before proceeding to the next run. The goal is to identify the best heuristic using few runs, and without any stochastic assumption about the constraint solver. In this study, we propose an adaptive variant of *Successive Halving* that exploits *Luby's universal restart sequence*. We analyze the convergence of this bandit algorithm in the non-stochastic setting, and we demonstrate its empirical effectiveness on various constraint satisfaction benchmarks.

1 Introduction

Constraint satisfaction is a recurring problem that arises in numerous computer science applications including, among others, bio-informatics, configuration, planning, scheduling and software validation. Given a set of decision variables and a set of constraints, each specifying a relation holding among some variables, the Constraint Satisfaction Problem (CSP) is to find an assignment of all variables that satisfies all constraints. To cope with this NP-complete problem, constraint solvers typically interleave backtracking search and inference in order to efficiently explore the space of assignments.

The *variable ordering heuristic* plays a key role in constraint satisfaction by iteratively selecting the next variable to branch on during backtrack search. Choosing the right branching strategy for a given CSP instance can significantly speed up the resolution, because different variable orderings can lead to entirely different search trees [Gent *et al.*, 1996]. Unfortunately, discovering an optimal variable ordering is computationally infeasible, since the complexity of finding the next variable to branch on for deriving a minimal-size search tree is DP-hard [Liberatore, 2000]. For this reason, most constraint satisfaction tasks are handled by using existing variable ordering heuristics, carefully designed by experts in Constraint Programming. To this point, a wide spectrum of hand-crafted heuristics have been proposed in the literature, ranging from *static* heuristics, in which variables are ordered before search starts, to *dynamic* heuristics for which the variable to branch on is selected using information that is collected during the search process (e.g. [Bessiere and Régin, 1996; Smith and Grant, 1998; Boussemart *et al.*, 2004; Refalo, 2004; Michel and Hentenryck, 2012; Habet and Terrioux, 2021; Wattez *et al.*, 2019]). In presence of such a diversity, an important question arises:

Given a CSP instance and a set of candidate (variable ordering) heuristics available in the solver, which heuristic is the best for solving the instance?

This question naturally calls for a "bandit" approach, as recently advocated in [Xia and Yap, 2018; Wattez *et al.*, 2020]. Multi-armed bandit problems are sequential decision tasks in which the learning algorithm has access to a set of arms, and observes the reward for the chosen arm after each trial. In the context of constraint satisfaction, each arm is a candidate heuristic and the sequence of trials can be derived using restarts [Wattez *et al.*, 2020]. More precisely, modern constraint solvers use a *restart scheme* that generates a sequence of "cutoffs" at which backtrack search is restarted if unsuccessful [Gomes *et al.*, 2000]. Based on this restart scheme, the solver explores at each run a search tree using the heuristic selected by the learner. When the cutoff is reached, the solver abandons the current search and restarts, while the learner receives a reward for the chosen arm, which reflects the effectiveness of the corresponding heuristic at the current run.

Conceptually, the performance of a bandit algorithm lies in its ability to interleave *exploration* by acquiring new information about arms, and *exploitation* by selecting an optimal arm based on the available information. In the standard *cumulative regret* setting, this performance is measured by the difference of cumulative rewards between the best arm taken from the benefit of hindsight, and the sequence of arms selected by the learner [Bubeck and Cesa-Bianchi, 2012]. This performance metric was used in [Wattez *et al.*, 2020], together with well-known low-regret bandit algorithms, such as UCB [Auer *et al.*, 2002a] and EXP3 [Auer *et al.*, 2002b]. However, when solving a CSP instance using search-and-restart, maximizing the cumulative reward of selected heuristics is not necessarily the best option. Indeed, evaluating the reward of a heuristic at some trial has a *cost* that depends on the cutoff associated with the trial. To this point, most restart schemes used in practice are far from uniform: the cutoff may vary from one run to another. A prototypical example is *Luby's universal scheme* [Luby *et al.*, 1993], whose sequence of cutoffs $(1, 1, 2, 1, 1, 2, 4, 1, \ldots)$ grows linearly in a non-monotonic way. Thus, by taking into account a given restart sequence, the learner should focus on exploration at runs associated with low cutoffs, and gradually turn to exploitation at runs with larger cutoffs.

These considerations warrant the use of an alternative bandit setting, called *pure exploration* or *best arm identification*. Here, the goal is to find an optimal arm as quickly as possible, and the performance of the learner is typically measured by the number of exploration steps needed to converge. Although most approaches to pure exploration operate in the stochastic regime, where the rewards of each arm are drawn at random according to a probability distribution (e.g. [Audibert *et al.*, 2010; Kaufmann *et al.*, 2016]), recent studies have considered the more general *non-stochastic* regime, where the sequence of rewards for each arm is convergent, but nothing is known about its rate of convergence [Jamieson and Talwalkar, 2016; Li *et al.*, 2017]. The last regime is more appropriate for constraint satisfaction tasks, since the rewards observed for some heuristic may depend on the trials at which the heuristic has been selected.

With these notions in hand, we call *best heuristic identification* the problem of finding as quickly as possible a variable ordering heuristic with optimal (asymptotic) reward, given a CSP instance, a pool of candidate heuristics, *and* a predefined restart scheme. Our main focus in this paper is the aforementioned Luby's universal scheme, which is regularly used in practice. Based on the binary tree structure of Luby's sequence, we propose a best arm identification algorithm inspired from *Successive Halving* [Karnin *et al.*, 2013; Jamieson and Talwalkar, 2016]. Our algorithm, called *Adaptive Single Tournament* (AST), uses single-elimination tournaments for successively eliminating half of all currently remaining heuristics, before proceeding to trials with larger cutoffs. We examine the convergence of this algorithm in the non-stochastic setting, and we demonstrate its effectiveness on various constraint satisfaction benchmarks. Notably, by selecting the best heuristic on large cutoffs of Luby's sequence, AST outperforms the standard bandit methods UCB and EXP3 advocated in [Wattez *et al.*, 2020].

The rest of the paper is organized as follows. The background about best arm identification is given in Section 2, and the best heuristic identification problem is presented in Section 3, together with an analysis of our algorithm. Comparative experiments are reported in Section 4. Finally, we conclude with some perspectives of research in Section 5. [1]

2 Best Arm Identification

For ease of reference, we use $[n]$ to denote the set $\{1, \cdots, n\}$. Multi-armed bandit problems can be formulated as infinitely repeated games between a learning algorithm and its environment. During each trial $t \in \mathbb{N}$, the learner plays an arm i taken from a predefined set $[K]$ of arms, and the environment responds with a reward $r_i(t)$ for this arm. In this study, we assume that for each trial $t \in \mathbb{N}$ and each arm $i \in [K]$, the feedback $r_i(t)$ is generated from a predefined reward function $r_i(\cdot) : \mathbb{N} \to [0, 1]$ for which the limit $\lim_{t\to\infty} r_i(t)$ exists and is equal to ν_i. Since reward functions do not change over time, the environment is "oblivious" to the learner's actions.

This *non-stochastic* setting introduced in [Jamieson and Talwalkar, 2016; Li *et al.*, 2017] lies between two extreme cases: the *stochastic* regime where each reward $r_i(t)$ is an i.i.d. sample from a fixed probability distribution supported, and the *adversarial* regime where each reward $r_i(t)$ is an arbitrary value that can be chosen adaptively by the environment, based on all the arms that the learner has played so far. As observed in [Jamieson and Talwalkar, 2016], the stochastic regime is a special case of the non-stochastic setting, which in turn is a special case of the adversarial regime.

In the *(non-stochastic) best arm identification* problem, the learning algorithm is given an exploration period during which it is allowed to gather information about reward functions. The goal of the learner is to minimize the number of trials required to explore before committing to an optimal arm on all subsequent trials. More formally, suppose without loss of generality that $\nu_1 > \nu_2 \geq \cdots \geq \nu_K$ Then, the performance of the learner is given by the smallest integer $\tau \in \mathbb{N}$ such that for any trial $t \geq \tau$, the learner is playing 1. Now, let

$$\Delta_{\min} = \min_{i=2,\cdots,K} \Delta_i \text{ where } \Delta_i = \nu_1 - \nu_i$$

are the *gaps* of suboptimal arms. In the *stochastic* best arm identification setting, these gaps are sufficient to derive bounds on the probability that the learner is selecting the right arm at horizon τ (e.g. [Kaufmann *et al.*, 2016]). However, in the non-stochastic setting, we also need to approximate the convergence rate of reward functions $r_i(\cdot)$. To this end, let

$$\rho_{\max}(t) = \max_{i\in[K]} \rho_i(t) \text{ where } \rho_i(t) = \sup_{t'\geq t} |\nu_i - r_i(t')|$$

Each "envelope" $\rho_i(\cdot)$ is the point-wise smallest non-increasing function from $\mathbb{N}$ to $[0, 1]$ satisfying $|\nu_i - r_i(t)| \leq \rho_i(t)$ for all t. The quasi-inverse of $\rho_{\max}(\cdot)$ is given by

$$\rho^{-1}_{\max}(v) = \min\{t \in \mathbb{N} : \rho(t) \leq v\}$$

By coupling the smallest gap $\Delta_{\min}$ with the function $\rho^{-1}_{\max}(\cdot)$, which together capture the intrinsic difficulty of the learning problem, we can derive a simple upper bound on the length of the exploration phase required by the learner to converge towards an optimal arm. Namely,

Proposition 1. *There exists a learning algorithm such that, given as input $[K]$, $\rho^{-1}_{\max}(\cdot)$ and $\Delta_{\min}$, after an exploration period of length*

$$\tau = K + \rho^{-1}_{\max}\left(\frac{\Delta_{\min}}{2}\right)$$

the algorithm always returns the best arm.

[1] Proofs of results and additional experiments are available at: https://hal.archives-ouvertes.fr/hal-03678354.

Algorithm 1: Adaptive Successive Halving (ASH)

```
Input: A set of arms $[K]$
Set $T_0 = K\lceil \log_2 K \rceil$
for $p = 0, 1, \ldots$ do
    Set $S_0 = [K]$
    for $q = 0, 1, \cdots, \lceil \log_2(K) \rceil - 1$ do
        Play each arm in $S_q$ for $s_q$ times, where
            $s_q = \left\lfloor \frac{T_p}{|S_q| \lceil \log_2(K) \rceil} \right\rfloor$
        Set $S_{q+1}$ as the $\lfloor S_q/2 \rfloor$ best arms in $S_q$
        measured according to their $s_q$th reward
    Play the unique arm in $S_{\lceil \log_2(K) \rceil}$
    Set $T_{p+1} = 2T_p$
```

Of course, in multi-armed bandit problems, the reward functions $r_i(\cdot)$ are *hidden*, and hence, nothing is *a priori* known about the envelopes $\rho_i(\cdot)$ and the gaps Δ_i. So, the learner has to deal with this issue by devising an exploration strategy whose performance is reasonably close to τ. A common strategy for the best arm identification task is the *Successive Halving* algorithm which was analyzed in both the stochastic setting [Karnin *et al.*, 2013] and the non-stochastic setting [Jamieson and Talwalkar, 2016].

In Algorithm 1, we present an adaptive, parameter-free version of Successive Halving (called ASH) that uses the so-called "doubling trick" for circumscribing the right amount of exploration. More precisely, each iteration of the outer loop consists of an exploration period, specified by the inner loop, followed by an exploitation period during which ASH is playing the unique remaining arm in its pool $S_{\lceil \log_2(K) \rceil}$. The doubling trick is then performed at the end of the outer loop. Based on a simple extension of the analysis given in [Jamieson and Talwalkar, 2016], we get the following result.

Proposition 2. *ASH is playing an optimal arm at the end of any iteration* p *of the outer loop, whenever*

$$p \geq 2 + \log_2 \left[K \lceil \log_2 K \rceil \left(1 + \rho_{\max}^{-1}\left(\frac{\Delta_{\min}}{2} \right) \right) \right]$$

3 Best Heuristic Identification

How can we adapt the paradigm of Successive Halving to constraint satisfaction tasks? This is the purpose of this section; after introducing some technical background about constraint satisfaction, we present a family of algorithms inspired from ASH that identify an optimal variable ordering heuristic.

3.1 Constraint Satisfaction

Recall that any instance P of the *Constraint Satisfaction Problem* (CSP) consists of a finite set of decision variables $\texttt{vars}(P)$, and a finite set of constraints $\texttt{ctrs}(P)$. Each variable x takes values from a finite domain, denoted $\texttt{dom}(x)$. Each constraint c is specified by a relation $\texttt{rel}(c)$ over a set of variables, called the *scope* of c, and denoted $\texttt{scp}(c)$. The *arity* of a constraint c is the size of its scope, and the *degree* of a variable x is the number of constraints in $\texttt{ctrs}(P)$ involving x in its scope. A *solution* to P is the assignment of a value to each variable in $\texttt{vars}(P)$ such that all constraints in $\texttt{ctrs}(P)$ are satisfied. The instance P is *satisfiable* if it admits at least one solution, and it is *unsatisfiable* otherwise.

A standard approach for solving CSP instances is to perform a backtracking search on the space of partial solutions, and to enforce a property called *generalized arc consistency* [Mackworth, 1977] on each decision. The resulting MAC (*Maintaining Arc Consistency*) algorithm [Sabin and Freuder, 1994] partitions the search space using a binary search tree $\mathcal{T}$. For each internal node of $\mathcal{T}$, a pair (x, v) is selected where x is an unfixed decision variable and v is a value in $\texttt{dom}(x)$. Based on this pair, two branches are generated: the assignment $x = v$ and the refutation $x \neq v$. The choice of x is decided by a *variable ordering heuristic*, and the choice of v is determined by some value ordering heuristic, which is usually the lexicographic order over $\texttt{dom}(x)$ by default.

In modern constraint solvers, the above approach is combined with a *restart scheme* that terminates and restarts search at regular intervals. The effect of such restarts is to early abort long runs, which thus saves on the cost of branching mistakes and resulting "heavy-tailed" phenomena [Gomes *et al.*, 2000]. Formally, any restart scheme is a mapping $\sigma : \mathbb{N} \rightarrow \mathbb{N}^+$, where $\sigma(t)$ denotes the *cutoff* at which the backtracking search is terminated during the tth run. In practice, $\sigma(t)$ is rescaled to capture a relevant cutoff unit such as, for example, the total number of backtracks [Gomes *et al.*, 2000] or the total number of wrong decisions [Bessiere *et al.*, 2004]. In this study, we concentrate on *Luby's universal scheme* $\sigma_{\text{luby}}(\cdot)$ which uses powers of two: when the cutoff 2^i is used twice, the next cutoff is 2^{i+1}. More formally:

$$\sigma_{\text{luby}}(t) = \begin{cases} 2^{i-1} & \text{if } t = 2^i - 1 \\ \sigma_{\text{luby}}\big(t - 2^{i-1} + 1\big) & \text{if } 2^{i-1} \leq t < 2^i - 1 \end{cases}$$

Based on the MAC algorithm and Luby's restart scheme, the solver builds a sequence of search trees $\langle \mathcal{T}_1, \mathcal{T}_2, \ldots \rangle$, where $\mathcal{T}_t$ is the search tree explored by MAC until it reaches the (possibly rescaled) cutoff $\sigma_{\text{luby}}(t)$. Importantly, the behavior of the constraint solver is generally non-stochastic. Indeed, even if it restarts from the beginning, the solver can memorize some relevant information about previous explorations. For example, the solver may keep in a table the number of constraint checks in the past runs, or it can maintain in a cache the nogoods which have frequently occurred in the search trees explored so far [Lecoutre *et al.*, 2007].

3.2 Learning to Branch

As mentioned in the introduction, various heuristics have been proposed for ordering decision variables during tree search. Examples of such variable ordering heuristics commonly used in constraint solvers include dom/ddeg [Bessiere and Régin, 1996], dom/wdeg [Boussemart *et al.*, 2004], impact [Refalo, 2004], activity [Michel and Hentenryck, 2012], chs [Habet and Terrioux, 2021] and cacd [Wattez *et al.*, 2019]. For instance, dom/ddeg branches on the variables with the smallest ratio between the size of their current domain and the current degree formed by counting only unfixed neighboring variables.

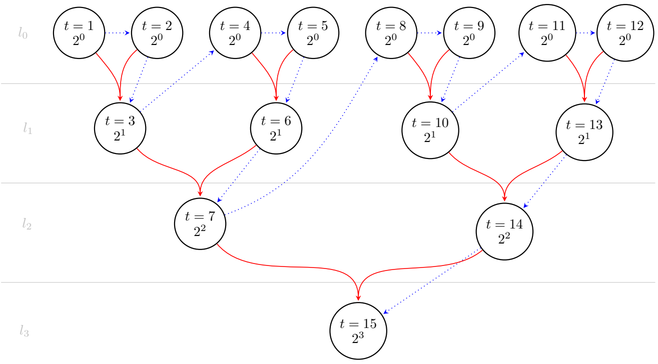


Figure 1: Tree-structured view of Luby's sequence for the first 15 runs. The cutoff sequence is derived by following the blue path, while the behavior of AST is captured by the red paths.

Given a pool $H = \{h_1, \cdots, h_K\}$ of candidate variable ordering heuristics, the problem of identifying the best heuristic for some CSP instance P can be cast as a "best arm identification" task, in which the environment is played by the solver. During each run t of the restart sequence, the learner starts by selecting an index $i \in [K]$. Then, the solver calls the MAC algorithm with the heuristic h_i for building a search tree $\mathcal{T}_t$, which is used to infer a reward $r_t(i) \in [0, 1]$ supplied to the learner. As suggested in [Wattez *et al.*, 2020], this reward can be measured using the (normalized) *pruned tree size*, which captures the ability of the solver to quickly prune large portions of its search space. More formally, let $\texttt{rft}(\mathcal{T}_t)$ be the set of internal nodes where each child is a refutation leaf, and for each such node n, let $\texttt{fut}(n)$ be the set of variables that are left unfixed at n. Then,

$$r_t(i) = \frac{\log_2 \left(\sum_{n \in \texttt{rft}(\mathcal{T}_t)} \prod_{x \in \texttt{fut}(n)} |\texttt{dom}(x)| \right)}{\log_2 \left(\prod_{x \in \texttt{vars}(P)} |\texttt{dom}(x)| \right)}$$

Computing such rewards takes linear space by exploiting the depth-first traversal of the search space performed by the MAC algorithm. However, observing $r_t(i)$ at the end of each run t has a *cost*, which corresponds to the time spent by the constraint solver in exploring the search tree $\mathcal{T}_t$ with the branching heuristic h_i. Because this runtime depends on the tth cutoff in the restart sequence, the learner should take into consideration the solver's restart scheme when trading exploration periods with exploitation phases.

From this perspective, the sequence of cutoffs in Luby's universal scheme $\sigma_{\text{luby}}(\cdot)$ can be viewed as an infinite binary tree organized into layers. This is illustrated in Figure 1 for the first 15 runs. For each layer $l = 0, 1, \cdots$, let t_l be the first run in the sequence with cutoff 2^l. Since by construction t_l is the root of a complete binary tree of size $2^{l+1} - 1$, this suggests using a variant of Successive Halving that starts by exploring 2^l candidate heuristics at the leaves, and successively eliminates half of all currently remaining heuristics until only one heuristic is left at the root t_l. Furthermore, since $t_{l+1} = 2t_l + 1$, this strategy can rely on the doubling trick for reaching the layer l where the best heuristic in H is identified with certainty at t_l.

Algorithm 2: Adaptive Single Tournament (AST)

```
Input: A set of arms $[K]$, a positive integer $m \geq 1$
Set $S = [K]$
for each run $t = 1, 2, \ldots$ do
    if $\sigma_{\text{luby}}(t) = 1$ then
        Select an arbitrary arm $i \in S$
        Set $S = S \setminus \{i\}$ and if $S = \emptyset$ then set $S = [K]$
    else
        Let $i_{\text{left}}$ be the arm played at run $t - \sigma_{\text{luby}}(t)$
        Let $i_{\text{right}}$ be the arm played at run $t - 1$
        Choose $i \in \{i_{\text{left}}, i_{\text{right}}\}$ with best reward $r_i$
    Play $i$ for $m$ times and set $r_i$ to the $m$th observed
      reward at run $t$
```

With these notions in hand, we present the *Adaptive Single Tournament* (AST) algorithm that is based on Luby's restart scheme for identifying the best heuristic on some CSP instance. Of course, AST is intended to be used alongside a constraint solver with $H = \{h_1, \cdots, h_K\}$ available branching heuristics. As specified in Algorithm 2, the learner focuses on exploration at cutoff 1, and progressively exploits at larger cutoffs using single tournaments. Based on the tree-structured view of Luby's sequence (Figure 1), the learner explores with equal frequency each arm in $[K]$ at the leaves of the tree. When it reaches an internal node of the tree, the learner selects the best arm among those played at the children of that node. Finally, AST uses a parameter m for testing the behavior of the constraint solver on candidate heuristics. For each run t of the restart sequence, the learner plays m times the arm i selected at t. Correspondingly, the solver performs m backtracking searches with cutoff $\sigma_{\text{luby}}(t)$, and the learner stores the last observed reward.[2]

Proposition 3. *AST is playing an optimal arm at any root node t_l of Luby's sequence, whenever the layer l satisfies:*

$$l \geq 1 + \frac{\lceil \log_2 K \rceil^2}{2} \left\{ 2 + \log_2 \left\lfloor \frac{1}{m} \left(1 + \rho_{\max}^{-1} \left(\frac{\Delta_{\min}}{2} \right) \right) \right\rfloor \right\}$$

4 Experiments

After an excursion into the theoretical aspects of our framework, we now turn to experiments. Given a set $\mathcal{P}$ of CSP instances and a pool H of variable ordering heuristics, the learning objective is to identify for each instance $P \in \mathcal{P}$ an optimal heuristic $h \in H$, using the MAC algorithm for backtracking search and Luby's sequence for the restart scheme.

For the set $\mathcal{P}$, we have considered all CSP instances selected for the XCSP3 competitions from 2017 to 2019.[3] This amounts to 810 CSP instances classified into 83 families. For the pool H, we have considered 8 heuristics, namely:

$$H = \{\texttt{lex}, \texttt{dom}, \texttt{dom/ddeg}, \texttt{abs}, \texttt{ibs}, \texttt{dom/wdeg}, \texttt{chs}, \texttt{cacd}\}$$

[2] Interestingly, AST shares common features with the top-K rank aggregation algorithm proposed by [Mohajer *et al.*, 2017]. But their algorithm operates in the stochastic dueling bandit setting, which is different from the non-stochastic best arm identification setting.

[3] http://www.xcsp.org/competitions/

	#SOLV.	TIME (S)
VBS	574	$52,175$
AST$_8$	557	$93,922$
AST$_{16}$	556	$94,844$
AST$_1$	555	$97,506$
AST$_2$	555	$99,044$
AST$_4$	553	$100,387$
UCB	550	$115,210$
UNI	547	$115,608$
cacd	543	$140,289$
chs	528	$157,484$
dom/wdeg	520	$179,228$
EXP3	480	$269,192$
abs	420	$426,498$
dom/ddeg	406	$442,194$
dom	404	$467,824$
ibs	383	$495,997$
lex	382	$509,658$

Table 1: Ranking of the branching strategies according to the number of solved instances and the cumulative runtime.

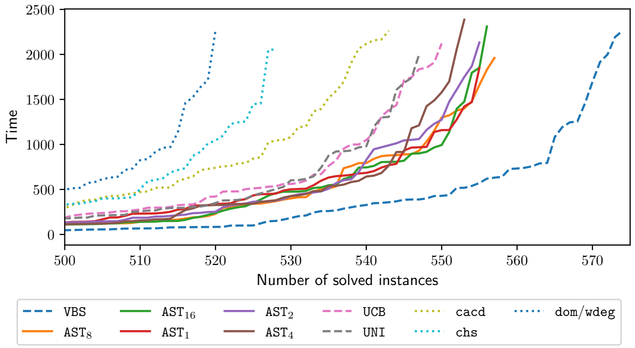


Figure 2: Cactus plots of the best branching strategies.

The best arm identification algorithm AST was compared with the state-of-the-art bandit algorithms UCB [Auer *et al.*, 2002a] and EXP3 [Auer *et al.*, 2002b], together with the baselines UNI and VBS. Here, UNI is the “uniform” strategy which draws uniformly at random a heuristic from H on each run of the restart sequence. VBS is the *Virtual Best Solver* that selects the best variable ordering heuristic in hindsight. This baseline can be derived by first playing each candidate heuristic $h \in H$ on all runs of the restart sequence, and then by selecting the heuristic for which P was solved with the least amount of time. For UCB and EXP3, we have considered the versions used in [Wattez *et al.*, 2020], and for AST, we have evaluated the algorithm using $m \in \{1, 2, 4, 8, 16\}$.

Our experiments have been performed with the constraint solver ACE[4], by keeping the default option for most parameters. Notably, the solver was allowed to record nogoods after each restart [Lecoutre *et al.*, 2007] in order to improve its performance during the sequence of runs. Actually, the unique exception to the default setting was to discard the last conflict policy [Lecoutre *et al.*, 2009], which would introduce a bias in the behavior of variable ordering heuristics. For all bandit algorithms UCB, EXP3, and AST, the reward of each selected heuristic was measured using the (normalized) pruned tree size, as specified in Section 3. Finally, each cutoff $\sigma_{\text{luby}}(t)$ of Luby's sequence was rescaled to $u \cdot \sigma_{\text{luby}}(t)$, where $u = 150$, corresponding to u wrong decisions (conflicts) per cutoff unit. All experiments have been conducted on a 3.3 GHz Intel XEON E5-2643 CPU, with 32 GB RAM, with a timeout set to $2,400$ seconds.

4.1 Main Results

In Table 1 are reported the results comparing the bandit algorithms UCB, EXP3 and AST (for different values of m), and the baselines UNI and VBS, together with the variable ordering heuristics in H. Here, we have discarded instances which could not be solved by any strategy. The competitors, referred to as branching strategies, are ranked according to the number of solved instances and, in case of tie, according to the cumulative runtime.

At the bottom of the ranking lie the heuristics with the worst performance: using lex, ibs, dom, dom/ddeg, or abs, the constraint solver ACE can solve at most 420 instances, taking more than 118 hours to handle them. The bandit algorithm EXP3 is just above, with a resolution of 480 instances in approximately 75 hours. We mention in passing that EXP3 operates in the *adversarial* bandit setting. Such a bad performance indicates that even if the solver can be non-stochastic, its behavior is not necessarily competitive with respect to the learner. For the next three heuristics dom/wdeg, chs and cacd, at least 520 instances could be solved within less than 50 hours. The simple uniform strategy UNI for selecting heuristics achieves a decent performance with 547 solved instances within 32 hours. To this point, we can observe that the bandit algorithm UCB, which operates in the *stochastic* regime is not much better: only 3 additional instances could be solved using almost the same amount of time. Unsurprisingly, VBS lies at the top of the ranking, with 574 instances solved in less than 15 hours. Interestingly, all configurations of AST are just below by solving, on average, 555 instances in less than 27 hours.

These results are corroborated by the cactus plots of the best strategies, as reported in Figure 2. Each plot indicates the number of solved instances (x-axis) at any time (y-axis). The dotted-line plots correspond to the performances of the three best heuristics in H, while the dashed-line plots capture the performances of the baselines UNI and VBS, and the bandit algorithm UCB.[5] Finally, the cactus plots of the different configurations of AST are indicated using solid lines. Again, we can observe that UCB and UNI have similar performances, when considering the number of solved instances per amount of time. We can also see that the behavior of ASH is remarkably stable when varying m.

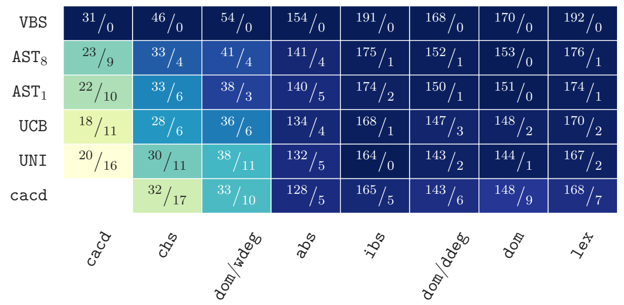
4.2 Pairwise Comparisons

Although our bandit algorithm AST can solve more instances in $\mathcal{P}$ than any single heuristic in H, we would like to determine whether it is capable of solving any instance P that can be solved by *some* heuristic in H. More generally, the goal here is to compare bandit approaches with respect to each $h \in H$, on any CSP instance P for which we know that P can be solved by h before reaching the timeout.

To this end, Figure 3 provides a heatmap of the different branching strategies, by reporting cacd, UNI, UCB, AST (for $m \in \{1, 8\}$) and VBS on the rows, and all variable ordering heuristics in H on the columns. In this heatmap, each entry

[4] https://github.com/xcsp3team/ace

[5] As EXP3 could not solve at least 500 instances, its cactus plot is not reported here.



	cacd	chs	dom/wdeg	abs	ibs	dom/ddeg	dom	lex
VBS	$^{31}/_{0}$	$^{46}/_{0}$	$^{54}/_{0}$	$^{154}/_{0}$	$^{191}/_{0}$	$^{168}/_{0}$	$^{170}/_{0}$	$^{192}/_{0}$
$\mathtt{AST}_8$	$^{23}/_{9}$	$^{33}/_{4}$	$^{41}/_{4}$	$^{141}/_{4}$	$^{175}/_{1}$	$^{152}/_{1}$	$^{153}/_{0}$	$^{176}/_{1}$
$\mathtt{AST}_1$	$^{22}/_{10}$	$^{33}/_{6}$	$^{38}/_{3}$	$^{140}/_{5}$	$^{174}/_{2}$	$^{150}/_{1}$	$^{151}/_{0}$	$^{174}/_{1}$
UCB	$^{18}/_{11}$	$^{28}/_{6}$	$^{36}/_{6}$	$^{134}/_{4}$	$^{168}/_{1}$	$^{147}/_{3}$	$^{148}/_{2}$	$^{170}/_{2}$
UNI	$^{20}/_{16}$	$^{30}/_{11}$	$^{38}/_{11}$	$^{132}/_{5}$	$^{164}/_{0}$	$^{143}/_{2}$	$^{144}/_{1}$	$^{167}/_{2}$
cacd		$^{32}/_{17}$	$^{33}/_{10}$	$^{128}/_{5}$	$^{165}/_{5}$	$^{143}/_{6}$	$^{148}/_{9}$	$^{168}/_{7}$

Figure 3: Heatmap of the branching strategies.

with row i and column j is a pair $^{a}/_{b}$, where a is the number of instances in $\mathcal{P}$ that could be solved by i but not by j, and conversely, b is the number of instances in $\mathcal{P}$ that could be solved by j but not by i. For example, the learning algorithm AST with $m = 8$ (second row) was able to solve 41 instances which could not be solved using the single heuristic dom/wdeg (third column), but the last one could solve 4 instances upon which AST failed. For the sake of readability, the colors of the entries are specified using the ratio $^{b}/_{a+b}$; a value close to 1 corresponds to a light color, which in turn indicates that the branching strategy at row i often fails in handling instances solved by the heuristic at column j.

Obviously, VBS never fails since it is selecting the best heuristic in hindsight. More interestingly, we can observe on the last row that cacd is not systematically better than other heuristics. The row comparisons between UNI, UCB and AST reveal important differences, when focusing on the first three columns of the heatmap, which correspond to the heuristics cacd, chs and dom/wdeg. Notably, the uniform strategy UNI is often dominated by these heuristics when they prove to be efficient on solving some CSP instances. For example, UNI was unable to deal with 16 instances which could be solved by cacd. On the other hand, UCB and AST ($m = 8$) are more robust by respectively solving 5 and 8 additional instances. Similar observations hold for chs and dom/wdeg. Finally, it turns out that UCB is always dominated by AST (for both $m = 1$ and $m = 8$), when they are compared with respect to the most efficient heuristics cacd, chs and dom/wdeg.

4.3 Finer-Grained Analysis

We conclude these experiments by comparing UCB and AST on some specific instances. Recall that UCB aims at maximizing the cumulative reward of selected heuristics, whatever being the sequence of cutoffs. Contrastingly, the goal of AST is to identify the best heuristic on the runs with large cutoffs. Figure 4 illustrates the resulting behaviors on two CSP instances (from families *Primes* and *Rlfap*). For both UCB and AST, each cutoff κ ranging from 1 to 1024 is associated with a bar-plot indicating the proportion of heuristics selected at the runs $\{t : \sigma_{\text{luby}}(t) = \kappa\}$. In light of these graphics, the divergence between UCB and AST is remarkable: while UCB fails at identifying a unique heuristic at large cutoffs, AST quickly converges through successive halving, which is easily recognizable from one bar-plot to the next.

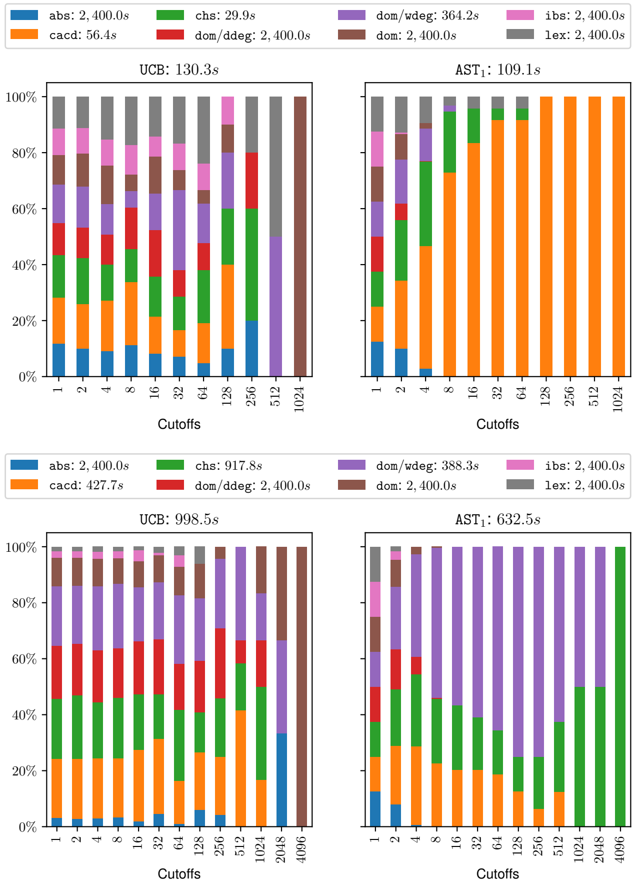


Figure 4: Proportions of heuristics selected by UCB (left) and AST (right) at each cutoff of Luby's sequence, for the CSP instances *Primes-15-60-3-5* and *Rlfap-scen-11-f01_c18*.

5 Conclusion

In this paper, we have focused on the best heuristic identification problem, which is to learn an optimal variable ordering heuristic for a CSP instance, given a set of candidate heuristics. By formulating this problem as a non-stochastic best-arm identification task, we have presented a bandit algorithm (AST) inspired from Successive Halving that takes into account the structure of Luby's universal sequence. For this algorithm, we have provided a convergence analysis, together with comparative results on various CSP instances.

A natural perspective of research that emerges from this study is to design best-heuristic identification algorithms for other universal restart schemes such as, for example, the geometric sequence examined in [Wu and van Beek, 2007]. An alternative and arguably more challenging perspective is to learn both branching heuristics *and* propagation techniques, for solving constraint satisfaction tasks. Thus, each arm is here a pair formed by a candidate heuristic and a candidate propagator. Yet, since the choice of the right propagation technique depends on the depth at which backtracking search is performed [Balafrej *et al.*, 2015], the corresponding best-arm identification problem should be examined in a *contextual bandit* setting, for which many questions remain open.

References

[Audibert *et al.*, 2010] Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. Best arm identification in multi-armed bandits. In *Proc. of COLT'10*, pages 41–53, 2010.

[Auer *et al.*, 2002a] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2):235–256, May 2002.

[Auer *et al.*, 2002b] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002.

[Balafrej *et al.*, 2015] Amine Balafrej, Christian Bessiere, and Anastasia Paparrizou. Multi-armed bandits for adaptive constraint propagation. In *Proc. of IJCAI'15*, pages 290–296, 2015.

[Bessiere and Régin, 1996] Christian Bessiere and Jean-Charles Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proc. of CP'96*, pages 61–75, 1996.

[Bessiere *et al.*, 2004] Christian Bessiere, Bruno Zanuttini, and Cèsar Fernandez. Measuring search trees. In *Proc. of ECAI'04 workshop on Modelling and Solving Problems with Constraints*, pages 31–40, 2004.

[Boussemart *et al.*, 2004] Frédéric Boussemart, Fref Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proc. of ECAI'04*, pages 146–150, 2004.

[Bubeck and Cesa-Bianchi, 2012] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Found. Trends Mach. Learn.*, 5(1):1–122, 2012.

[Gent *et al.*, 1996] Ian P. Gent, Ewan MacIntyre, Patrick Presser, Barbara M. Smith, and Toby Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proc. of CP'96*, pages 179–193, 1996.

[Gomes *et al.*, 2000] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24(1):67–100, 2000.

[Habet and Terrioux, 2021] Djamal Habet and Cyril Terrioux. Conflict history based heuristic for constraint satisfaction problem solving. *J. Heuristics*, 27:951—-990, 2021.

[Jamieson and Talwalkar, 2016] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proc. of AISTATS'16*, pages 240–248, 2016.

[Karnin *et al.*, 2013] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *Proc. of ICML'13*, pages 1238–1246, 2013.

[Kaufmann *et al.*, 2016] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the complexity of best-arm identification in multi-armed bandit models. *J. Mach. Learn. Res.*, 17:1:1–1:42, 2016.

[Lecoutre *et al.*, 2007] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Recording and minimizing nogoods from restarts. *JSAT*, 1:147–167, 2007.

[Lecoutre *et al.*, 2009] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Reasonning from last conflict(s) in constraint programming. *Artif. Intell.*, 173(18):1592–1614, 2009.

[Li *et al.*, 2017] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.

[Liberatore, 2000] Paolo Liberatore. On the complexity of choosing the branching literal in DPLL. *Artif. Intell.*, 116(1):315–326, 2000.

[Luby *et al.*, 1993] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Inf. Process. Lett.*, 47(4):173–180, 1993.

[Mackworth, 1977] Alan K. Mackworth. Consistency in networks of relations. *Artif. Intell.*, 8(1):99–118, 1977.

[Michel and Hentenryck, 2012] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Proc. of CPAIOR'12*, pages 228–243, 2012.

[Mohajer *et al.*, 2017] Soheil Mohajer, Changho Suh, and Adel Elmahdy. Active learning for top-k rank aggregation from noisy comparisons. In *Proc. of ICML'17*, pages 2488–2497, 2017.

[Refalo, 2004] Philippe Refalo. Impact-based search strategies for constraint programming. In *Proc. of CP'04*, pages 557–571, 2004.

[Sabin and Freuder, 1994] Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proc. of CP'94*, pages 10–20, 1994.

[Smith and Grant, 1998] Barbara M. Smith and Stuart A. Grant. Trying harder to fail first. In *Proc. of ECAI'98*, pages 249–253, 1998.

[Wattez *et al.*, 2019] Hugues Wattez, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. Refining constraint weighting. In *Proc. of ICTAI'19*, pages 71–77, 2019.

[Wattez *et al.*, 2020] Hugues Wattez, Frédéric Koriche, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. Learning variable ordering heuristics with multi-armed bandits and restarts. In *Proc. of ECAI'20*, pages 371–378, 2020.

[Wu and van Beek, 2007] Huayue Wu and Peter van Beek. On universal restart strategies for backtracking search. In *Proc. of CP'07*, pages 681–695, 2007.

[Xia and Yap, 2018] Wei Xia and Roland H. C. Yap. Learning robust search strategies using a bandit-based approach. In *Proc. of AAAI'18*, pages 6657–6665, 2018.