A priori knowledge for learning combinations of heuristics in CSPs

Anastasia Paparrizou
University of Montpellier, CNRS, France

Michael Sioutis
University of Montpellier, CNRS, France

Yoan Thomas
University of Montpellier, CNRS, France

Christian Bessiere
University of Montpellier, CNRS, France

Abstract

Selecting the right variable ordering heuristic to solve a given constraint satisfaction problem (CSP) instance can dramatically influence solver efficiency. Yet, no single heuristic universally outperforms others across all instances. Recent works have shown that interleaving multiple heuristics during search can significantly improve performance. In this paper, we propose to use a priori knowledge on the performance of heuristics to provide a better initial configuration to the solver. To represent this knowledge we provide a formal definition to quantify the degree of complementarity between pairs of heuristics. To validate this approach, we solved instances from the XCSP competition using every possible combination of modern heuristic pairs. Our results reveal a strong correlation between the complementarity score of a pair of heuristics and the performance improvement achieved by interleaving them in the search strategy. Even complementary heuristics with poor standalone performance can, when combined, deliver surprisingly good results. These findings suggest that our notion of complementarity represents a critical step forward in understanding why and how alternating heuristics enhances CSP solving.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Branching heuristics; Adaptive search; Reinforcement Learning

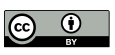
Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Constraint Programming (CP) involves assigning values to variables so that all the constraints of the problem are satisfied. Such a task is NP-hard, thus using a good variable ordering heuristic is a key component for the efficiency of constraint solvers. The research community has devoted a lot of effort in designing variable ordering heuristics that perform well on large and hard problems. Before the 2000s, heuristics were using features of the instance to order the variables. For instance `maxdeg` selects the variable involved in the maximum number of constraints, `mindom` selects the variable with the smallest domain [11], and `dom/deg` selects the variable minimizing the ratio size of the domain over number of constraints involving the variable [5].

Since the beginning of the 2000s, a new generation of heuristics has appeared, often called *adaptive* heuristics. Adaptive heuristics select the next variable to instantiate based on stored information about what happened in the search space until that point. Adaptive heuristics rely on internal weights to dynamically adjust the importance of variables during the search process. These weights are updated using conflict information in `dom/wdeg` [6], `cacd` [22], and `chs` [10], through propagation events in `activity` [16], and by combining both conflicts and propagation in `pick/dom` [1]. Despite the significant improvements these heuristics provide to constraint solvers, the results indicate that no single heuristic consistently outperforms the others across all problem classes and instances.

Another approach towards the goal of designing an efficient and stable general-purpose variable ordering heuristic consists in combining several heuristics. Pioneering attempts used arithmetic-based



aggregation operations, such as dom/deg, or multi-agent voting mechanism [8]. Modern approaches leverage machine learning techniques, such as Multi-Armed Bandits (MAB) [23, 21, 17, 7, 12, 20]. A MAB formalizes the problem of sequentially choosing among a set of options (arms) under uncertainty, while balancing exploration and exploitation based on observed rewards. It has been shown in [21] that MABs are effective for combining multiple heuristics. Results in [17] show that even combining only a single heuristic with the random heuristic (i.e., the strategy that selects variables uniformly at random) can outperform the original heuristic used alone. This behavior holds for any heuristic combined with the random heuristic and is independent from the MAB exploration policy. Even a static policy is sufficient to produce gains. The SAT community immediately adopted this dual-bandit framework to dynamically select between two SAT state-of-the-art heuristics: the Variable State Independent Decaying Sum (VSIDS) and the Conflict History-Based (CHB) heuristic. The resulting Kissat-MAB [7] solver was ranked first in the Main Track of SAT competition in 2021.

Following these results in CP and SAT solvers, our goal is to further study the interaction of heuristics when used in pairs, and to try to understand which criteria make a combination fruitful or not. We introduce an original measure, called p-*complementarity*, that will allow us to quantify how much two heuristics lead the solver to exhibit different performance depending on the instance to solve. Our hypothesis is that the more two heuristics are complementary, the more promising their combination is. This information is provided prior search to the solver, as an a priori knowledge in order to set up a better initial configuration to the solver. To confirm this hypothesis, we conducted a comprehensive evaluation using the CSP22to25 dataset.[1]

To extract the complementarity knowledge, we first used a random sample of the instances to compute the p-complementarity of all the pairs of adaptive heuristics that we selected for our study. We then combined these pairs of heuristics inside a solver in the same way as in [17]. We first restricted the solver to follow a uniform policy, where each heuristic is selected with equal probability. This choice allows us to isolate the intrinsic effect of interactions between the two heuristics without introducing additional bias induced by learning or reward-driven adaptation. We then solved all instances in the dataset with a selection of pairs of heuristics that are representative of the possible scenarios. The results demonstrate a strong correlation between the degree of p-complementarity in a pair of heuristics and the performance of a solver using these two heuristics uniformly. This correlation holds regardless of whether the individual heuristics are effective on their own or not. A solver using two strong (resp. weak) heuristics performs better than either heuristic alone if they exhibit high p-complementarity, but worse otherwise.

We also experimented when the solver selects the heuristics following the standard MAB-UCB1 policy [3]. The results are consistent with those observed when using a uniform policy. The conclusion we draw from this work is that our measure of complementarity is indeed a relevant criterion to evaluate how much a combination of heuristics will be beneficial to the performance of a solver that interleaves these two heuristics, independently of the selection policy. This is one step forward in understanding why and how combinations of heuristics improve constraint solving.

The rest of the paper is organized as follows. Section 2 provides the necessary definitions and background to understand the paper. Section 3 contains our definition of complementarity of heursitics and experiments to compute the complementarity of pairs of heuristics. Section 4 presents the results of running a solver combining pairs of heuristics with a uniform policy or a MAB-UCB1 policy. These results demonstrate how strong the correlation between p-complementarity and performance is. Section 5 concludes the paper.

[1] https://xcsp.org/instances/

2 Background

Constraint Satisfaction Problems

Constraint Programming (CP) is the art of solving problems expressed as conjunctions of constraints. A constraint is a relation specifying which combinations of values are acceptable for a given set of variables. The Constraint Satisfaction Problem (CSP) consists in deciding whether there exists an assignment of values to problem variables that satisfies all the constraints. The CSP is NP-complete.

Most constraint solvers (aka CP solvers) solve CSPs by performing backtrack search, assigning values to variables, enforcing a consistency property called *generalized arc consistency* [15], at each assignment, and backtracking when a failure occurs. This procedure is called *Maintaining Arc Consistency* (MAC) [19]. At each node, the MAC procedure selects the next variable to assign according to a *variable ordering heuristic*.

Search with restarts

Deterministic variable ordering heuristics have been shown to exhibit heavy-tailed behavior on both random instances and real-world instances of CSPs [9]. This issue can be alleviated by using *randomization* and *restart* strategies, which incorporate some random choices in the search process, and iteratively restart search from the beginning. A commonly used cutoff strategy is driven by the Luby sequence [14]. It is worth noting that, upon restart, if the solver uses an adaptive variable ordering heuristic, it can retain the heuristic weights computed during previous runs and continue updating them in subsequent runs. The variable ordering is therefore influenced by both the current and past executions.

Multi-armed bandits

Reinforcement learning, and in particular Multi-Armed Bandit (MAB) algorithms, has been widely exploited over the past decade in CP across various contexts. These include guiding search strategies [13], learning the appropriate level of consistency [4], and –most prominently– selecting the best variable ordering heuristics [23, 21, 17, 12]. In this paper, we adopt the MAB framework proposed by [17], which makes use of restarts and is designed for double-armed bandits, making it particularly suitable for our study. The algorithm (Algorithm 1 in [17]) is already implemented in the Choco solver (as well as in ACE), and can be easily integrated into any CP solver. It works as follows: At the start of each run (i.e., at each restart), the MAB selects either heuristic H_1 or H_2 (the two arms) provided as inputs and applies it throughout the whole run. At the end of each run, the selected arm/heuristic receives feedback reflecting the efficiency of the run under the chosen heuristic, referred to as the ESB reward function. ESB captures how efficient the given run has been in avoiding inconsistent parts of the search tree. More precisely, ESB returns the ratio of number of visited nodes to the size of the complete sub-tree defined over the variables selected during the given run. In reinforcement learning, the trade-off between exploration and exploitation is governed by various policies, among which the Upper Confidence Bound (UCB1) [2] is the most widely used.

In the experiments of this paper, we also use a version of the solver in which the two arms/heuristics are chosen uniformly at random. The purpose is to evaluate the contributions in isolation of any bias induced by sophisticated MAB learning or reward-driven adaptation.

3 Complementarity of heuristics

In this section, we define a measure to quantify and compare the performance of a solver when using one or another variable ordering heuristic. We then describe our experimental setting and we report

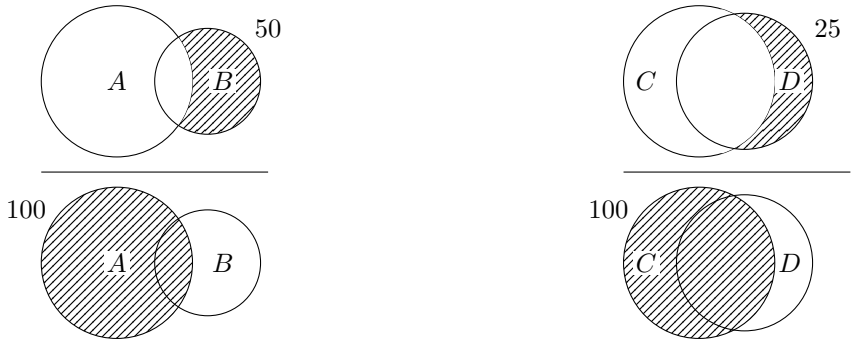


Figure 1 An illustration of p-complementarity. $p(A, B, t) = 0.5$, $p(C, D, t) = 0.25$.

the complementarities computed for all pairs of heuristics involved in the analysis.

3.1 Definition of complementarity

Given two variable ordering heuristics and a time limit, the performance complementarity of the two heuristics compares the set of instances the solver can solve with one or the other heuristic in the given amount of time. We call this measure the p-complementarity of the heuristics.

Given a set P of CSP instances, a solver, a time limit t, and a heuristic H, we denote by $S_P(H, t)$ the set of instances from P that the solver solves using the heuristic H, and with a time limit t for each instance. Formally, p-complementarity is defined as follows.

▶ **Definition 1** (p-complementarity). *Given a solver, let P be a set of CSP instances, t a time limit, and H_1 and H_2 two heuristics. The p-complementarity of H_1 and H_2 at time t is denoted by $p(H_1, H_2, t)$ and is equal to:*

$$\frac{\min(|S_P(H_1,t) \setminus S_P(H_2,t)|, |S_P(H_2,t) \setminus S_P(H_1,t)|)}{\max(|S_P(H_1,t)|, |S_P(H_2,t)|)}$$

Figure 1 illustrates p-complementarity on two cases. On the left, heuristic B solves 50 instances that A could not solve within the time limit t whereas A –the best heuristic– solves 100 instances in total. Hence, $p(A, B, t) = 0.5$. On the right, even though D is almost as efficient as C, it solves only 25 instances that C could not solve within the time limit t. Hence, $p(C, D, t) = 0.25$.

Computing the p-complementarity of pairs of heuristics requires solving instances and then comparing which heuristic solved what subset within the time limit. (Fortunately, as we will detail later on, a small set of instances is sufficient for computing a good approximation of p-complementarity.)

3.2 Experimental setting

We used the benchmark CSP22to25, which is an aggregate of the XCSP3 competition instances from 2022 to 2025,[2] composed of 800 *hard* CSP instances representative of wide range of problem classes. All instances were solved with the Choco constraint programming solver [18], in which we used

[2] https://xcsp.org/instances/

Heuristic	#Solved	Time (in sec.)	PAR2
`cacd`	345	1,147	918,085
`dom/wdeg`	339	1,175	947,618
`activity`	327	1,247	1,013,815
`pick/dom`	318	1,321	1,089,114
`chs`	300	1,483	1,202,340
`rand`	161	2,406	2,147,247

Table 1 Standalone adaptive heuristics on the 800 CSP instances of CSP22to25. Times in seconds (average on the instances solved by at least one heuristic). Timeout = 1 hour.

only existing components. We used 2-way branching, generalized arc consistency as the level of consistency, Luby progression based on node visits as restart policy with the constant fixed to 100, and the timeout was set to 1 hour with 32GB of RAM per instance.[3] Our study involves most of the existing adaptive heuristics in the literature: `activity`, `cacd`, `chs`, `dom/wdeg`, `pick/dom`, to which we added `rand`, the heuristic that always selects the next variable randomly. Apart from `rand`, we have not selected any non-adaptive heuristic because they do no maintain information among restarts and are usually inefficient for solving hard instances.

3.3 Computing p-complementarity of our heuristics

We first report the standalone performance for our six heuristics from Table 1. *#Solved* indicates the number of instances the given heuristic solved within the time limit of one hour. *Time* is the average time needed by the given heuristic to solve all the instances that at least one heuristic was able to solve. Such instances not solved by the given heuristic are assigned the time limit. *PAR2* assigns a score to each instance solved by at least one heuristic and reports the sum of the scores, where the score of an instance is its solving time if solved within the time limit, and if not, a penalty equal to twice the time limit. In Table 1 we observe that `cacd` is the one performing the best, followed by `dom/wdeg` and `activity`. In the second half of the table, we find `pick/dom`, `chs`, and `rand`, which show the lowest performance.

As seen in Section 3.1, computing the p-complementarity of pairs of heuristics requires solving a set of instances and comparing which heuristic solved what subset of instances within the time limit. For various sizes of samples of instances from CSP22to25 (those sizes are reported on the right y-axis in Figure 2), we generated 1 000 samples of the given size and computed the p-complementarity ranking of the pairs of heuristics. Figure 2 reports the box plot of the Spearman correlation coefficient (x-axis) of these rankings depending on the size of the sample (y-axis) used for computing p-complementarity. A Spearman coefficient close to 1 indicates that the ranking computed on samples of the given size is almost identical to the ranking obtained when considering the full set of instances. The good news is that a low number of sampled instances is sufficient to obtain a representative ordering of heuristics pairs ranked by their p-complementarity: only 20% of the instances (150 instances), provide an ordering from which the correct ordering (computed on the 800 instances) can be obtained with a few minor permutations.

Figure 3 is a heatmap of the p-complementarity computed for all pairs of heuristics. The value of p-complementarity computed on a sample of 20% is reported in each cell. We observe that `activity`

[3] The hardware we used for our experiments is kept hidden for anonymity reasons

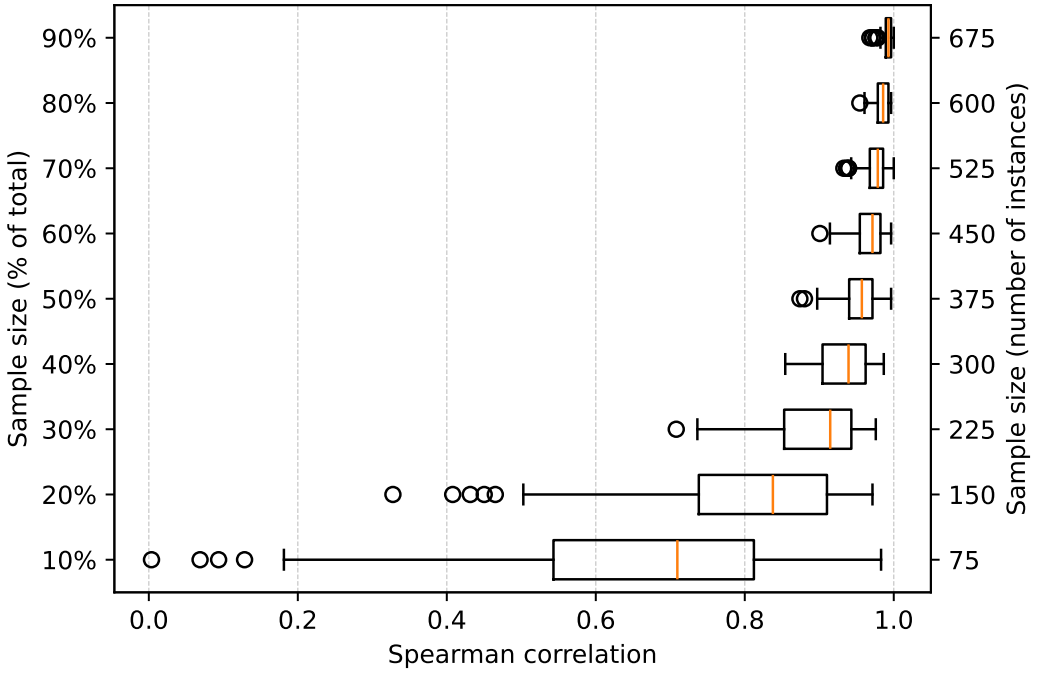


Figure 2 Spearman correlation versus size of sample.

and `chs` are two heuristics that show a great p-complementarity with most of the other heuristics. `activity` has a p-complementarity greater than 0.1 with `cacd`, `chs`, and `dom/wdeg`. `chs` has a p-complementarity greater than or close to 0.1 with `activity`, `cacd`, and `pick/dom`. Overall, the most p-complementary pair is `chs-pick/dom` (0,172). In Figure 3 we also observe that `rand` has a low p-complementarity with all other heuristics. The reason is not that its behavior is close to theirs, but because its performance is so poor that the additional instances it solves cannot contribute much to the p-complementarity formula. On the contrary, the low p-complementarity of the pair `cacd-dom/wdeg`, which are the two most efficient heuristics (see Table 1), is due to the similarity of their behavior.

4 Evaluation of our four pairs of heuristics

In this section, we evaluate our hypothesis that p-complementarity of pairs of heuristics is a good predictor of the performance of a solver interleaving these two heuristics. We first present our selection of the four most remarkable pairs that will be used for thorough analysis. Then as a first step, we present results where the use of the two heuristics is monitored by a simple uniform policy, each heuristic being selected with equal probability. Once our hypothesis is confirmed in this simple framework, we present results where a MAB with the ubiquitous UCB1 policy is in charge of deciding how to alternate heuristics. These results show that even in the case of the sophisticated UCB1 policy, p-complementarity of a pair of heuristics is a good predictor of the performance of the solver using that pair. We finally address the case where we increase the number of arms in the MAB.

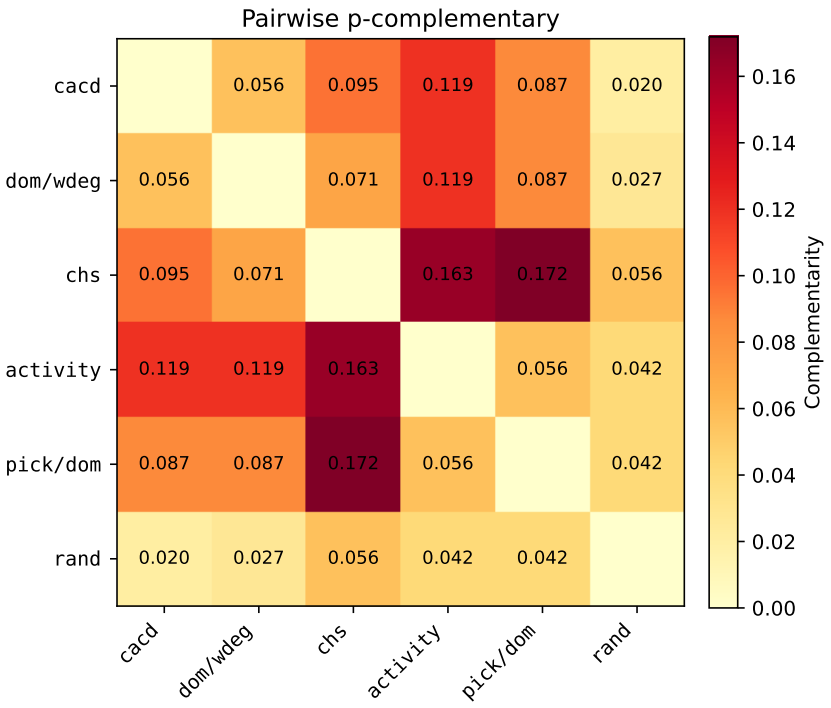


Figure 3 Heatmap of p-complementarity for all pairs of heuristics computed on 150 instances only.

4.1 Selection of four remarkable pairs of heuristics

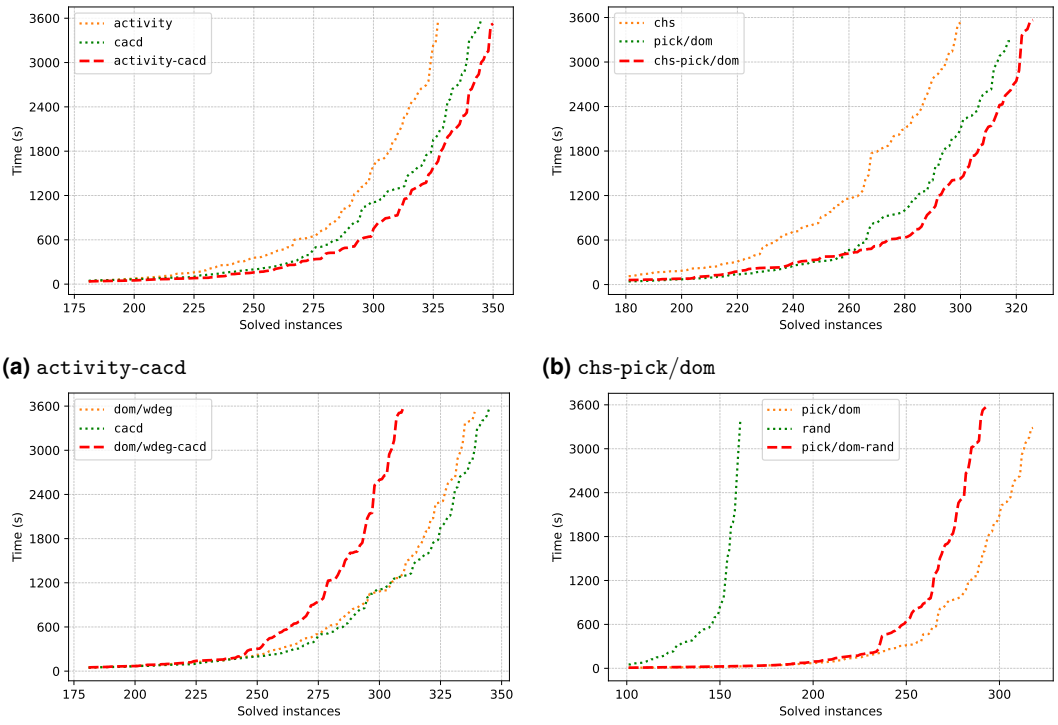
Given the performance of all our heuristics reported in Table 1 and given the value of p-complementarity for all their pairs reported in Figure 3, we select four pairs that characterize the four extreme behaviors a pair of heuristics can exhibit. Among the very good heuristics (`cacd`, `dom/wdeg`, `activity`–see Table 1), we select `activity-cacd`, which is the pair with the highest p-complementarity, and `cacd-dom/wdeg`, which is the pair with the lowest p-complementarity. Among the weak heuristics (`pick/dom`, `chs`, `rand`), we select `chs-pick/dom`, which is the pair with the highest p-complementarity, and `pick/dom-rand`, which is the pair with the lowest p-complementarity. These are the four pair we analyse in the next sections.

4.2 Combining heuristics uniformly

In this first experiment, we restrict our analysis to the uniform policy, where each heuristic is selected with equal probability. This simple policy provides a neutral baseline that is suitable for fundamental and controlled analysis because it allows us to concentrate on the intrinsic interaction between the two heuristics, and not on biases induced by learning or reward-driven adaptation. We solved the 800 instances in the CSP22to25 benchmark with the four remarkable pairs of heuristics chosen in Section 4.1. The setting (solver/restart policy/time limit/...) is exactly the same as the one described in Section 3.2.

Figure 4 and Table 2 report the results. Figure 4 displays them as cactus plots. Each figure shows the performance (in number of solved instances while time increases) of the standalone heuristics and their combination in a uniform strategy. The four cactus plots presented are the results obtained for our four remarkable pairs of heuristics:

- `activity-cacd`: two efficient heuristics with high p-complementarity;



(a) `activity-cacd` **(b)** `chs-pick/dom`

(c) `cacd-dom/wdeg` **(d)** `pick/dom-rand`

Figure 4 Cactus plots for the four remarkable pairs of heuristics combined with a uniform policy (timeout = 1 hour).

- `chs-pick/dom`: two weak heuristics with high p-complementarity;
- `cacd-dom/wdeg`: two efficient heuristics with low p-complementarity;
- `pick/dom-rand`: two weak heuristics with low p-complementarity.

Let us start by `activity-cacd`. These two heuristics have a high p-complementarity (0.119 in Figure 3). Figure 4a shows that despite these two heuristics being already very high in performance as standalone heuristics, their combination is even better. The combination under uniform policy solves a larger number of instances (350) before the time limit (see Table 2), and is faster all along the timeline, compared to both `activity` and `cacd`, which solved 327 and 345 instances respectively.

Figure 4b presents the results of `chs` and `pick/dom`, the pair of heuristics representing the case of high p-complementarity (0.172 in Figure 3) but poor standalone performance. Again we observe a significant improvement for the combination. Their combination solves 326 instances (see Table 2) whereas `chs` and `pick/dom` independently solve 300 and 318 instances.

We now focus on the two pairs with low p-complementarity. Figure 4c reports the performance of the two best standalone heuristics, `cacd` and `dom/wdeg`, which are also those which have one of the lowest p-complementarities (0.056 in Figure 3). Figure 4c shows that despite `cacd` and `dom/wdeg` being the top ranked heuristics in the literature, together they are significantly less efficient. Their uniform combination solves only 310 instances (see Table 2). This number is far from the 345 and 339 instances solved respectively by `cacd` and `dom/wdeg` alone, and more impressively less than the 326 instances solved by the combination of `pick/dom` and `chs`, the two weakest tested heuristics –apart from `rand`. Similarly, Figure 4d shows the performance of `pick/dom` and `rand`, two weak heuristics that have a very low p-complementarity (0.042 in Figure 3). We see that their uniform

Heuristic	#Solved	Time (in sec.)	PAR2
`activity-cacd`	350	1,084	871,163
`chs-pick/dom`	326	1,263	1,034,098
`cacd-dom/wdeg`	310	1,368	1,149,246
`pick/dom-rand`	294	1,499	1,266,808

Table 2 Pairs of heuristics combined with a uniform policy on the 800 CSP instances of CSP22to25. Times in seconds (average on the instances solved by at least one heuristic). Timeout = 1 hour.

combination is by far less efficient than `pick/dom` alone, which solves 318 instances. `rand` is the worst, solving only 161 instances.

These results confirm our hypothesis that p-complementarity is an accurate tool to predict how combinations of heuristics will behave when combined in a solver. As p-complementarity can easily be pre-computed once and for all on a small set of instances, p-complementarity should become a useful technique to improve the efficiency of solvers.

4.3 Combining heuristics inside a MAB with UCB1 policy

In the previous section, we have shown evidence that p-complementarity is a good measure to predict efficiency of combinations of heuristics under a uniform policy. The question that remains to answer is whether p-complementarity remains a good predictor when the use of the two heuristics is monitored by a sophisticated reinforcement learning technique. In this section we show that p-complementarity is a good predictor withing a reinforcement learning environment too.

As in Section 4.2, we solved the 800 instances in the CSP22to25 benchmark with the four remarkable pairs of heuristics selected in Section 4.1. The setting (solver/restart policy/time limit/...) is the same as before except that instead of selecting heuristics uniformly at random, we use the dual-bandit framework MAB-UCB1 described in Section 2. The results are reported in Table 3.

When comparing Table 2 (uniform policy) and Table 3 (MAB-UCB1 policy), we observe quite the same pattern. Regardless of whether the solver uses the uniform policy or the MAB-UCB1 to select heuristics, pairs with high p-complementarity perform better than their standalone versions of Table 1, and pairs with low p-complementarity perform worse than their standalone versions of Table 1 –again apart from `rand`, which is the worst of all. The good performance of pairs with high p-complementarity could be expected because MAB-UCB1 is known to be in general better than a uniform policy. What is perhaps more noteworthy is to discover that when the MAB includes two heuristics with low p-complementarity, it fails to combine them in a way that would at least reach the performance of the best of the two. It shows that p-complementarity is a good criterion of choice of pairs of heuristics even when the solver uses a sophisticated MAB strategy.

4.4 Comparing to MABs with more than two arms

In previous sections we studied combinations of two heuristics. One may wonder whether adding extra heuristics would improve even more the performance of the solver or not. We performed a new experiment in which we selected our best pair (`activity-cacd`), and added extra arms. We tried a MAB-UCB1 with `activity-cacd-dom/wdeg` (the three best standalone heuristics of Table 1), another with `activity-cacd-chs-dom/wdeg-pick/dom` (all heuritics except `rand`), and a final one with all six heuristics. The results are reported in Table 4. We observe that the more arms we add, the worse the solver performs.

Heuristic	#Solved	Time (in sec.)	PAR2
`activity-cacd`	355	1,067	850,004
`chs-pick/dom`	325	1,298	1,043,941
`cacd-dom/wdeg`	316	1,345	1,117,302
`pick/dom-rand`	290	1,501	1,281,997

Table 3 Pairs of heuristics combined with a MAB with UCB1 policy on the 800 CSP instances of CSP22to25. Times in seconds (average on the instances solved by at least one heuristic). Timeout = 1 hour.

Heuristic	#Solved	Time (in sec.)	PAR2
`activity-cacd`	355	1,067	850,004
`activity-cacd-dom/wdeg`	353	1,078	862,233
`activity-cacd-chs-dom/wdeg-pick/dom`	344	1,175	914,952
`activity-cacd-chs-dom/wdeg-pick/dom-rand`	341	1,178	922,568

Table 4 Combinations of more than two heuristics with a MAB with UCB1 policy on the 800 CSP instances of CSP22to25. Times in seconds (average on the instances solved by at least one heuristic). Timeout = 1 hour.

It is worth noting that the MAB with all arms (`activity-cacd-chs-dom/wdeg-pick/dom-rand`) corresponds to the case of a solver exploiting all possible arms blindly, as proposed in [21]. We indeed observe that an appropriate initialisation of the MAB with two arms, according to p-complementarity, results in solving 14 extra instances in less time compared to a mere application of what was proposed in [21].

5 Conclusion and perspectives

Following the recent works that have shown that interleaving multiple variable ordering heuristics during search can significantly improve the performance of the solver, we have initiated a study whose goal is to better understand what criteria explain that a combination of two heuristics works well and another does not. We introduced an original measure that quantifies the degree of complementarity between pairs of heuristics. A priori knowledge about the complementarity of heuristics is used to set up the solver. We validated this approach by experimentally demonstrating a strong correlation between the degree of complementarity of a pair of heuristics and the performance of a solver interleaving them during search. Pairs of highly complementary heuristics –even those with poor standalone performance– can, when combined, yield surprisingly good results. These results suggest that our notion of complementarity captures a fundamental concept for understanding how and when alternating heuristics enhance CSP solving. This provides a straightforward yet powerful approach to improving any state-of-the-art CP solver, provided it supports switching between heuristics after each restart. Our work thus represents a first step toward fully harnessing the performance potential of MABs.

While this paper focused on studying the complementarity of pairs of variable ordering heuristics, such an approach could also prove useful for guiding the selection of other configuration parameters; these include, for example, the level of propagation or value ordering heuristics.

References

1. Gilles Audemard, Christophe Lecoutre, and Charles Prud'homme. Guiding backtrack search by tracking variables during constraint propagation. In *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, Toronto, Canada, August 27-31, 2023*, volume 280 of *LIPIcs*, pages 9:1–9:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
2. Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002.
3. Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002.
4. Amine Balafrej, Christian Bessiere, and Anastasia Paparrizou. Multi-armed bandits for adaptive constraint propagation. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 290–296. AAAI Press, 2015.
5. Christian Bessiere and Jean-Charles Régin. MAC and combined heuristics: Two reasons to forsake FC (and cbj?) on hard problems. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996*, volume 1118 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1996.
6. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 146–150. IOS Press, 2004.
7. Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. Combining VSIDS and CHB Using Restarts in SAT. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
8. Susan L. Epstein and Eugene C. Freuder. Collaborative learning for constraint solving. In *Principles and Practice of Constraint Programming — CP 2001*, pages 46–60, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
9. Carla P. Gomes, Bart Selman, Nuno Crato, and Henry A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24(1/2):67–100, 2000.
10. Djamal Habet and Cyril Terrioux. Conflict history based search for constraint satisfaction problem. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019*, pages 1117–1122. ACM, 2019.
11. Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3):263–313, 1980.
12. Frederic Koriche, Christophe Lecoutre, Anastasia Paparrizou, and Hugues Wattez. Best heuristic identification for constraint satisfaction. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 1859–1865. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Main Track.
13. Manuel Loth, Michèle Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-based search for constraint programming. In *Principles and Practice of Constraint Programming*, pages 464–480, 2013.
14. Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Inf. Process. Lett.*, 47(4):173–180, 1993.
15. Alan K. Mackworth. On reading sketch maps. In Raj Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977*, pages 598–606. William Kaufmann, 1977. URL: http://ijcai.org/Proceedings/77-2/Papers/006.pdf.
16. Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International Conference, CPAIOR 2012, Nantes, France, May 28 - June1, 2012. Proceedings*, volume 7298 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2012.
17. Anastasia Paparrizou and Hugues Wattez. Perturbing branching heuristics in constraint solving. In *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-*

la-Neuve, Belgium, September 7-11, 2020, Proceedings, volume 12333 of *Lecture Notes in Computer Science*, pages 496–513. Springer, 2020.

18 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022.

19 Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the Eleventh European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 8-12, 1994*, pages 125–129. John Wiley and Sons, Chichester, 1994.

20 Zhengyuan Shi, Wentao Jiang, Xindi Zhang, Jin Luo, Yun Liang, Zhufei Chu, and Qiang Xu. DynamicSAT: Dynamic Configuration Tuning for SAT Solving. In *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, volume 340 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:23, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

21 Hugues Wattez, Frédéric Koriche, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. Learning variable ordering heuristics with multi-armed bandits and restarts. In *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 371–378. IOS Press, 2020.

22 Hugues Wattez, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. Refining constraint weighting. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*, pages 71–77. IEEE, 2019.

23 Wei Xia and Roland H. C. Yap. Learning robust search strategies using a bandit-based approach. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6657–6665. AAAI Press, 2018.