Comparison of Hyperparameter Optimization Methods for Selecting Search Strategy of Constraint Programming Solvers

1st Hedieh Haddad[1], 2nd Pierre Talbot[2], 3rd Pascal Bouvry[2]
[1]*SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg*
[2]*University of Luxembourg, Esch-sur-Alzette, Luxembourg*
hedieh.haddad@uni.lu, pierre.talbot@uni.lu, pascal.bouvry@uni.lu

***Abstract*—Selecting the most effective search strategy for new problems in constraint programming is a significant challenge. While autonomous search strategy design has been extensively researched, this study adopts a different approach. We present a comprehensive comparison of various *hyperparameter optimization* methods, treating the search strategies of constraint solvers as hyperparameters. This approach aims to enhance the selection process of search strategies, improving solver efficiency and effectiveness, and relieving the users from the burden of choosing the right one for their problems.**

We introduce the *probe and solve algorithm*, a generic method that operates in two phases: a probing phase, where various strategies are explored and ranked using hyperparameter optimization methods, and a solving phase, where the top-ranked strategy is used to solve the problem. We include a comparative analysis between the probe and solve algorithm and the solver's default search strategies.

Our results demonstrate that *Bayesian optimization* is the best hyperparameter optimization method we have tested to select the search strategy. Furthermore, it can outperform state-of-the-art dynamic search strategies across various benchmarks and solvers. Bayesian optimization showed superior performance in 20%-30% of cases, with both our algorithm and baselines achieving equal results in 50%-60% of instances, thus emerging as the most suited for selecting the search strategy.

***Index Terms*—Hyperparameter optimization, constraint programming, search strategies, Bayesian optimization**

I. Introduction

Constraint programming (CP) is a versatile computational method that deals with mathematical relations or constraints. It offers a declarative way to model various real-world problems, from scheduling to musical composition [1]. This versatility sets it apart from other methods like SAT, which focuses on Boolean formulas [2], and linear programming [3] which is designed to solve linear constraints.

The efficiency of CP largely depends on the design of search strategies, which are complex to devise due to the need for deep solver understanding. Several studies have aimed to create a universal search strategy, but no single strategy has proven to dominate others [4], [5]. This highlights the need for a method to select the best search strategy for each problem.

In this paper, we introduce the *probe and solve algorithm (PSA)*, which uses hyperparameter optimization (HPO) techniques to explore and identify the most effective search strategy for a specific problem. HPO, a well-established domain in machine learning [6], has seen limited application in CP, However, viewing search strategies as solver hyperparameters opens up new insights for their automatic optimization, thereby identifying the most effective search strategies for the problem at hand.

When too many parameters are considered, the hyperparameter space becomes too vast for efficient exploration. Evaluating each search strategy is time-consuming as it involves calling the solver. In general, increasing the number of hyperparameters does not necessarily lead to increased efficiency. Therefore, for the purposes of this study, we focus on the commonly used search strategies, namely the variable selection strategy and the value selection strategy.

There are some popular HPO techniques, including: *grid search*, *random search* [7], *hyper-band optimization* [8], and *Bayesian optimization* [9]. Our main contribution is employing these methods for the selection of variable and value selection strategies in CP solvers. However, we exclude grid search due to its exhaustive nature, which requires substantial resources to check all possible sets of hyperparameters, rendering it inefficient and impractical.

Our algorithm unfolds in two stages: probing and solving. In the probing stage, we execute the solver using various search strategies for a fraction of a predetermined global timeout. We then rank these strategies and select the most effective one for solving stage (Section III). A key aspect of this stage is that it provides a ranking of different search strategies for a specific problem, offering insights into the better set of hyperparameters. This information can guide subsequent user decisions. During the solving stage, we run the solver with the chosen strategy for the remaining timeout and deliver the final solution. At this point, the user has the option to either utilize the set proposed by PSA or implement their own strategy based on the provided ranking (Section III-A).

Our method, designed to enhance solver performance by identifying a problem-specific search strategy, is evaluated using benchmark problems from the XCSP3 [10] competition with ACE solver [11]. The results are compared with the default and most commonly used search strategies in XCSP3, as mentioned in [12]. Our algorithm integrates seamlessly with any XCSP3 solvers and dynamically determines the number of probing rounds, ensuring flexibility and adaptability based

on the problem's complexity and requirements (Section IV).

The findings indicate that the algorithm exhibits superior performance within the ACE solver, outperforming the baseline measures. This investigation seeks to determine if the implementation of more intricate search strategies can boost solver performance without necessitating an increase in architectural complexity (Section IV-C).

We hope our work highlights the potential of HPO techniques, particularly Bayesian optimization, in selecting search strategies. We aim to inspire further exploration of these methods to unlock new possibilities for efficiency and effectiveness in problem-solving.

II. Background

A. Constraint Programming

A constraint satisfaction problem (CSP) is a tuple $\langle X, D, C\rangle$ where X is a set of variables, D are the domains, and C is the set of constraints. An extension is the constraint optimization problem (COP) which is a tuple $\langle X, D, C, obj\rangle$ with $obj \in X$ where the goal is to find the best objective possible (either by minimization or maximization). Without loss of generalities, we suppose minimization problems in our definitions and algorithms.

The constraint solver is essentially a backtracking procedure dividing the domains of the variable following a certain search strategy and pruning the domains using a form of logical inference called *propagation* [13].

Different problem domains and characteristics may require different search strategies to achieve the best performance. Therefore, the design and evaluation of search strategies are ongoing research areas in CP, aiming to provide users with guidance and tools for effectively solving complex problems [14], [15].

III. Probe and Solve Algorithm

In this section, we introduce the *Probe and Solve algorithm (PSA)*, a generic method designed to identify effective search strategies for resolving constraint problems. The algorithm unfolds in two stages: the probing stage and the solving stage. The probing stage navigates the realm of search strategies utilizing an HPO method, while the solving stage employs the most effective strategy to address the problem.

We denote S_{var} the set of variable selection strategies' names (as recognized by the solvers), S_{val} the set of value selection strategies' names.

To summarize, the HPO method is in charge of optimizing two arrays of integers representing respectively S_{var} and S_{val}. We write $HP := \mathcal{P}(S_{var} \times S_{val})$ the set of all possible combination of hyperparameters.

A. Algorithm

Consider GT as the global timeout allocated for solving a CSP. A specific portion of this time, denoted as PT (probing timeout), is exclusively reserved for the probing phase. The value of PT is a significant parameter in our algorithm, and we have performed comprehensive experiments to optimize it within the context of the GT.

During the probing phase, PSA assesses a variety of search strategies from the set HP. Each strategy is evaluated for a limited duration, initially set to 5 seconds, denoted CT (current timeout). This duration is not only sufficient for the computational requirements of simpler problems but also considers the necessary overhead for transmitting information to the solver and retrieving the solutions.

The algorithm strives to solve the problem within this timeout. If the solver successfully identifies the objective within this timeframe, it proceeds to execute the next experiment in the probing phase using this timeout. However, if the solver fails to find any objective result within the timeout, the approach adapts by extending the defined timeout. This increment is determined by a geometric coefficient, set to 1.2, which multiplies the current timeout CT, thereby increasing it.

The timeout continues to increase incrementally until an objective is discovered or PT is reached, using the calculation of elapsed time, denoted ET. This adaptive approach ensures that the algorithm can effectively manage problem complexities while maintaining efficiency and providing valuable insights into more complex problems, particularly in scenarios where a solution cannot be found within 5 seconds.

Selecting a large probing timeout will decrease the number of combinations we can test and the effectiveness of the HPO method. However, choosing a small timeout will prevent us from obtaining any solution and make the comparison between the two runs more challenging. In our experiments, we compare two runs using the objective value found and, in the case of ties, the time it took to reach that objective value.

At the conclusion of the probing phase, we obtain a ranking of the tested search strategies, and we select the best one for the solving phase.

We now provide a more explicit definition of our approach in Algorithm 1.

The algorithm accepts a COP $\langle X, D, C, obj\rangle$, a hyperparameter function hpo, the set of available search strategies HP, and two timeouts: GT and PT.

The HPO method, denoted as $hpo(HP, psolve)$, takes the set of search strategies and an evaluation function. It returns the ranking of the best search strategies, the best objective found so far.

We assume a function $solve(\langle X, D, C, obj\rangle, s, T)$ that optimizes a given COP using the search strategy $s \in HP$ under the timeout T. The solving phase operates for the remaining duration of the global timeout and employs the best search strategy identified in the probing phase to solve the constraint problem.

IV. Experiments

In this section, we evaluate the performance of the *PSA* on a set of benchmark problems from different domains, and we use two metrics to measure the performance: the objective value and the solving time.

Algorithm 1 Probe and Solve Algorithm (PSA)

function PSA($\langle X, D, C, obj\rangle, hpo, HP, GT, PT$)
  Initialize CT to 5 seconds
  Initialize $Geo\_Coefficient$ to 1.2
  Initialize ET to 0 seconds
  Initialize $best\_obj$ to ∞
  while $ET < PT$ **do**
    $psolve \leftarrow \lambda s.solve(\langle X, D, C, obj\rangle, s, CT)$
    $ranking, obj \leftarrow hpo(HP, psolve)$
    $ET \leftarrow ET + CT$
    if obj $\neq\infty$ **then**
      $min(obj, best\_obj)$
    else
      $CT \leftarrow CT \times Geometric\_Coefficient$
    end if
  end while
  if best_obj = ∞ **then**
    return $solve(\langle X, D, C, obj\rangle, ranking[0], GT - PT)$
  else
    return $min(best\_obj, solve(\langle X, D, C \wedge obj < best\_obj, obj\rangle, ranking[0], GT - PT))$
  end if
end function

A. Experimental Setup

The experiments presented in this paper were carried out using the high-performance computing facilities of the University of Luxembourg [16] – see https://hpc.uni.lu. The technical specifications of a cluster compute node are: 2xAMD Epyc ROME 7H12 @ 2.6 GHz [64c/280W] processor with 256 GB RAM.

We detail the experimental setup for our evaluation, encompassing the benchmark problems, solvers, HPO methods, baseline methods, and hyperparameters. Our study utilizes data from the fifth international XCSP3 constraint solver competition conducted in 2023 [11]. We specifically target all the COP instances derived from this competition, totaling 250 instances that cover a variety of constraints and objective functions. For each problem, we align our global timeout setting of 1200 seconds with the one utilized by the XCSP3 challenge, a recognized competition for constraint solvers.

Our objective is to validate that PSA produces encouraging results. To this end, we have opted to set the probing phase timeout to different percentages of the global timeout. This enables us to compare the efficiency and applicability of this timeout and to broadly determine whether a shorter or longer probing phase yields superior results.

B. XCSP3 Benchmark with ACE Solver

In this subsection, we evaluate the performance of PSA on the XCSP3 benchmark problems. XCSP3 is an XML-based format designed to represent instances of combinatorial constrained problems from the perspective of CP. It is an intermediate integrated format that can represent each instance separately while preserving its structure [17].

We compare the results of our algorithm with four baseline strategies. These baselines encompass three popular variable selection strategies including PickOnDom [12], FrbaOnDom [18], and DomWDeg/CACD [12], [19] which are variable selection strategies utilized in constraint programming and the solver's default search strategy. The performance is evaluated based on the objective value and the time taken to solve. For the three popular frameworks, we adhere to the solver's default value selection strategy as done in [12]. For the solver's default search strategy, we refrain from specifying any variable or value selection strategies, leaving this choice to the solver.

Our goal is to observe the influence of these diverse combinations of search strategies on the solver's performance. We strive to identify an effective search strategy using PSA. This is achieved by examining all the variable selection strategies and value selection strategies provided by the ACE solver itself.

The solver offers a wide range of strategies for both variable and value selection strategies, as shown below:

Variable selection strategies:

{RunRobin, Wdeg, Memory, PickOnDom, FrOnDom, WdegOnDom, ProcOnDom, Regret, FrbaOnDom, Ddeg}

Value selection strategies:

{Dist, OccsR, Median, AsgsFp, Flrs, Bivs, First, AsgsFm, Last, Robin, RunRobin, Bivs2, InternDist, Occs, FlrsE}

1) Random Search: In this section, we evaluate the performance of PSA when random search is employed as the HPO approach. Our focus is on comparing the results of PSA with baseline performances, considering all variable selection strategies and value selection strategies provided by the ACE solver as hyperparameters.

Upon applying random search as the HPO approach, it is observed that the results are not as robust as the ones obtained using the baselines. When considering the random search with a ratio of 0.2, it is observed that the results vary across different strategies. For instance, PSA outperformed the CACD strategy in 18.07% of the instances, while CACD achieved a superior objective of 23.11% of the cases. Similarly, PSA surpassed the FRBA strategy in 15.04% of the cases, while FRBA attained a higher objective of 22.36% of the instances, and for the rest of them, both approaches yielded equivalent results.

This can be attributed to several factors, the most significant being the inherent randomness of the approach. Despite this, random search remains a popular choice in the HPO field due to its simplicity and low computational overhead, which ensures minimal system strain during execution.

The results of PSA with random search in probing ratio 0.2, are illustrated in Figure 1a.

2) Hyper-band Search : For the second step, we assessed the performance of PSA when hyper-band search is utilized as the HPO method taking into account all variable selection strategies and value selection strategies mentioned above.

Hyper-band search is a resource-efficient variant of random search that uses a bandit-based approach to allocate resources to different configurations. This method is known for its ability to handle a large number of hyperparameters effectively, which could potentially lead to improved results.

When employing hyper-band search with a probing ratio of 0.2, the results demonstrate variability across different strategies. For instance, PSA outperformed the PICK3 strategy in 17.01% of the instances, while PICK3 achieved a superior objective in 22.45% of the instances. This indicates that while the hyper-band method can effectively handle a large number of hyperparameters, the performance can vary significantly depending on the specific strategy employed and the complexity of the problem at hand.

The results of PSA with hyper-band search are illustrated in Figure 1b. This figure will provide insights into the performance of PSA with hyper-band search compared to the baseline strategies, thereby offering a comprehensive view of the effectiveness of different HPO methods in enhancing the solver's performance.

3) Bayesian Optimization: In the next step, we examined the performance of PSA when Bayesian optimization is employed as the HPO method. Upon implementing Bayesian optimization as the HPO approach, we anticipate a different set of results compared to random search and Hyper-band search.

Bayesian optimization is a sequential design strategy for the global optimization of black-box functions that works by constructing a posterior distribution of functions to find the best balance between exploration and exploitation. This method is known for its efficiency and effectiveness in high-dimensional spaces, which could potentially lead to improved results.

It is observed that PSA outperforms the default solver in 29.49% of the models. Interestingly, in 44.87% of the models, PSA and the baselines produce identical results, suggesting that no single strategy consistently excels across all situations.

When employing Bayesian optimization with a probing ratio of 0.2, the results demonstrate variability across different strategies. For instance, PSA outperformed the FRBA strategy in 25.20% of the instances, while FRBA achieved a superior objective of 19.92% of the instances. Indeed, the results suggest that employing Bayesian optimization with a probing ratio of 0.2 can yield better results than all the baselines. This demonstrates the potential of Bayesian optimization as an effective HPO approach, particularly when dealing with complex problems with a large number of hyperparameters.

The results of PSA with Bayesian optimization in probing ratio 0.2, are illustrated in Figure 1c.

C. Detailed Analysis

The detailed results obtained from the various HPO methods and different probing ratios are presented in Table I. We conducted tests using a range of probing ratios, specifically 5%, 10%, 20%, 50%, and 100% of the global timeout. Our results indicated that, for our specific set of problems, a probing ratio of 20% generally performed better than the others. Also, this table incorporates two additional columns: *Fallback to Default* and *Same Search Strategy*. The *Fallback to Default* column measures the instances where PSA could not determine a search strategy within the given timeframe, leading to the use of the default search strategy during the solving phase. The *Same Search Strategy* column signifies the instances where PSA identified a search strategy identical to the solver's default search strategy post the probing phase. These additional insights enhance our understanding of PSA's performance and decision-making process.

Furthermore, it is important to note that unlike other approaches which are invasive and require modifications to the solver's code, our approach is non-invasive and generic, capable of working with any solver. Also, it can dynamically change some crucial parameters of HPO, eliminating the need for their initialization beforehand. This addresses a significant limitation as it allows PSA to adaptively set the number of rounds and timeout for each round, offering a more flexible and efficient approach. For instance, in contrast to the static nature of classical Bayesian optimization, our approach can dynamically adjust these parameters based on the problem's complexity and the solver's performance, thereby enhancing the overall efficiency and effectiveness of the solver.

D. Examining More Solvers Parameters

In our study, we explored various solver parameters and strategies. Here, we discuss our findings:

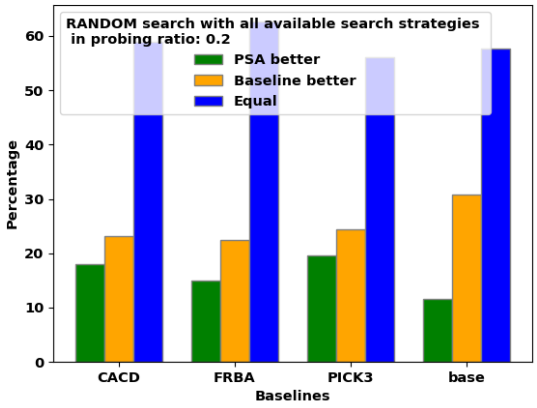
Search Strategies with Restricted Subset of Variables and Value Selection Strategies. We implemented different subsets of search strategies as hyperparameters for PSA to see if PSA with static search strategies could outperform the dynamic strategies. Our results indicate that dynamic approaches surpass static approaches in this context.

Nested Search Strategies. We also experimented with nested search strategies. The idea was to assign different search strategies to each set, hypothesizing that for one objective a pair can work well, while for another there might be a better one. However, the results were not as expected, and using nested search strategies could not improve the results.

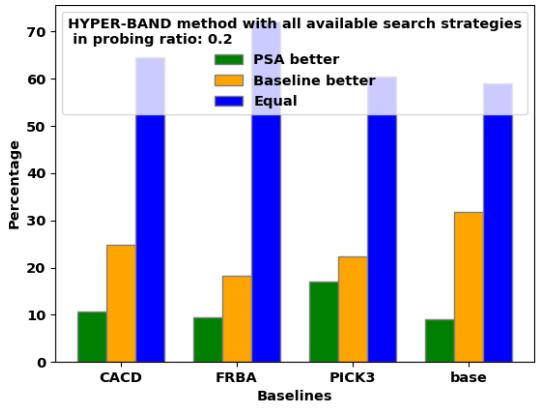
Solver Parameters. We also examined different restart annotations such as Luby sequence restart and geometric restart options, including the number of nodes explored for restarting [20]. We also explored different solver parameters, such as the number of last conflict options. However, due to the vast number of options and limited global timeout, not all of these options were beneficial for our specific approach. In some cases, they even worsened the approach. This could be attributed to the state-space becoming too large and the probing phase is too short to find a good configuration.

V. Conclusion

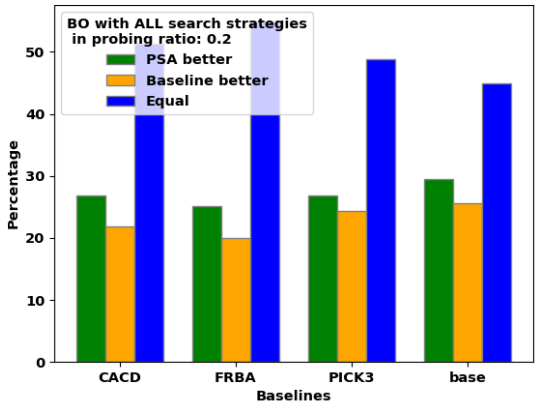
In this research, we explored the application of diverse hyperparameter optimization methods to search strategies within constraint programming solvers. Our aim was to establish a simple and generic approach that enables users, particularly those new to constraint problem modeling, to pinpoint both



(a) Random search with all available S_{var} and S_{val}



(b) Hyper-band with all available S_{var} and S_{val}



(c) Bayesian optimization with all S_{var} and S_{val}

Fig. 1: Comparative performance of PSA and baselines: an examination of variable and value selection strategies with probing timeout ratio 0.2. The comparison is conducted across different baselines and is categorized into three sections: comprehensive analysis with the set of all variable/value selection strategies using (a) Random search (b) Hyper-band search (c) Bayesian optimization.

effective and inefficient search strategies swiftly. This proves particularly beneficial when there is no initial understanding of which search strategies may be successful.

Our results indicate that PSA can surpass the baseline strategies in ACE, which are typically the default in constraint models. Interestingly, of the various hyperparameter optimization methods we assessed, Bayesian optimization proved to be the most suitable for our study. It predominantly exhibited superior performance, outdoing the baselines using different ratios. This highlights the potential of Bayesian optimization to boost the performance of PSA, especially in high-dimensional spaces. However, there are instances where no combination of variable and value selection strategies can exceed the solver's default.

By persistently evaluating and enhancing these strategies, researchers can aid in the development of more potent tools and techniques for resolving complex constraint problems across a broad spectrum of domains. This research serves as a foundation for more advanced and efficient constraint problem-solving methodologies. We anticipate further refining our approach and achieving superior results in future studies.

ACKNOWLEDGMENT

The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [16] – see https://hpc.uni.lu. This work is partially funded by the joint research program UL/SnT–ILNAS on Technical Standardisation for Trustworthy ICT, Aerospace, and Construction.

TABLE I: Comprehensive results for all the possible ratios in comparison with the baselines.

Method		Baselines	Results (%)			Additional Results (%)	
			PSA Better	Baselines Better	Equal Results	Fallback to default	Same search strategy
Random Search All available S_{var} and S_{val}	0.05	CACD	19.67	**22.59**	57.74	24.70	0.00
		FRBA	19.03	**21.46**	59.51		
		PICK3	**26.32**	23.08	50.61		
		Default	12.66	**26.58**	60.76		
	0.1	CACD	16.81	**25.21**	57.98	17.89	0.41
		FRBA	16.26	**23.98**	59.76		
		PICK3	19.51	**26.42**	54.07		
		Default	15.38	**21.79**	62.82		
	0.2	CACD	18.07	**23.11**	58.82	13.01	0.81
		FRBA	15.04	**22.36**	57.32		
		PICK3	19.51	**24.39**	56.10		
		Default	11.54	**30.77**	57.69		
	0.5	CACD	**21.70**	16.98	61.32	11.82	0.00
		FRBA	15.00	**18.18**	66.82		
		PICK3	**20.91**	19.55	59.55		
		Default	17.46	**19.05**	63.49		
	1.0	CACD	1.67	**13.33**	85.00	15.27	0.00
		FRBA	1.56	**13.28**	85.16		
		PICK3	0.76	**14.50**	84.73		
		Default	2.33	**6.98**	90.70		
Hyper-band method All available S_{var} and S_{val}	0.05	CACD	12.66	**28.69**	58.65	25.30	0.81
		FRBA	11.84	**30.20**	57.96		
		PICK3	19.18	**31.84**	48.98		
		Default	10.26	**26.92**	62.82		
	0.1	CACD	13.87	**31.09**	55.04	17.48	0.81
		FRBA	11.79	**29.27**	58.94		
		PICK3	15.04	**32.11**	52.85		
		Default	12.82	**33.33**	53.85		
	0.2	CACD	10.64	**24.82**	64.54	21.77	0.68
		FRBA	9.52	**18.37**	72.11		
		PICK3	17.01	**22.45**	60.54		
		Default	9.09	**31.82**	59.09		
	0.5	CACD	9.38	**22.66**	67.97	17.91	0.75
		FRBA	5.26	**17.29**	77.44		
		PICK3	10.45	**20.9**	68.66		
		Default	8.11	**16.22**	75.68		
	1.0	CACD	0.0	**1.15**	98.85	25.8	0.0
		FRBA	0.0	0.0	100.0		
		PICK3	0.0	**1.08**	98.92		
		Default	0.0	0.0	100.0		
Bayesian Optimization All available S_{var} and S_{val}	0.05	CACD	23.11	**24.37**	52.52	25.60	0.81
		FRBA	20.73	**25.20**	54.07		
		PICK3	21.95	**26.02**	52.03		
		Default	19.23	**30.77**	50.00		
	0.1	CACD	23.85	**25.94**	50.21	16.19	0.80
		FRBA	**25.10**	21.86	53.04		
		PICK3	**25.91**	24.29	49.80		
		Default	**27.85**	24.05	48.10		
	0.2	CACD	**26.89**	21.85	51.26	13.0	1.21
		FRBA	**25.20**	19.92	54.88		
		PICK3	**26.83**	24.39	48.78		
		Default	**29.49**	25.64	44.87		
	0.5	CACD	**25.63**	23.95	50.42	8.82	1.56
		FRBA	**23.98**	21.14	54.88		
		PICK3	**26.02**	24.39	49.59		
		Default	**28.21**	24.36	47.44		
	1.0	CACD	2.94	**26.47**	70.59	8.53	2.40
		FRBA	3.25	**25.20**	71.54		
		PICK3	2.85	**26.42**	70.73		
		Default	2.56	**25.64**	71.79		

References

[1] F. R. Peter vanBeek, Toby Walsh, "Handbook of Constraint Programming [Book]," iSBN: 9780080463803. [Online]. Available: https://www.oreilly.com/library/view/handbook-of-constraint/9780444527264/

[2] A. Biere, M. Heule, H. Van Maaren, and T. Walsh, "Handbook of Satisfiability: Second Edition," ser. Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press, Feb. 2021. [Online]. Available: http://ebooks.iospress.nl/doi/10.3233/FAIA336

[3] J. M. Bernd Gärtner, *Understanding and Using Linear Programming*, ser. Universitext. Berlin, Heidelberg: Springer, 2007. [Online]. Available: http://doi.org/10.1007/978-3-540-30717-4

[4] H. Simonis and B. O'Sullivan, "Search Strategies for Rectangle Packing," in *Principles and Practice of Constraint Programming*, P. J. Stuckey, Ed. Berlin, Heidelberg: Springer, 2008, pp. 52–66. [Online]. Available: https://doi.org/10.1007/978-3-540-85958-1_4

[5] E. Teppan, G. Friedrich, and A. Falkner, "QuickPup: A Heuristic Backtracking Algorithm for the Partner Units Configuration Problem," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 2, pp. 2329–2334, Jul. 2012. [Online]. Available: https://doi.org/10.1609/aaai.v26i2.18979

[6] T. Agrawal, *Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient*. Berkeley, CA: Apress, 2021. [Online]. Available: https://doi.org/10.1007/978-1-4842-6579-6

[7] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281–305, Mar. 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:15700257

[8] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," Jun. 2018, arXiv:1603.06560 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1603.06560

[9] J. Ungredda and J. Branke, "Bayesian Optimisation for Constrained Problems," May 2021, arXiv:2105.13245 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2105.13245

[10] G. Audemard, C. Lecoutre, and E. Lonca, "Proceedings of the 2023 XCSP3 Competition," Mar. 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2312.05877

[11] C. Lecoutre, "ACE, a generic constraint solver," Jan. 2023, arXiv:2302.05405 [cs]. [Online]. Available: http://arxiv.org/abs/2302.05405

[12] G. Audemard, C. Lecoutre, and C. Prud'homme, "Guiding Backtrack Search by Tracking Variables During Constraint Propagation." Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2023.9

[13] K. Apt, "Constraint propagation algorithms," in *Principles of Constraint Programming*. Cambridge: Cambridge University Press, 2003, pp. 254–298. [Online]. Available: https://doi.org/10.1017/CBO9780511615320.007

[14] P. Talbot, "Spacetime Programming: A Synchronous Language for Composable Search Strategies," in *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming*, ser. PPDP '19. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 1–16. [Online]. Available: https://doi.org/10.1145/3354166.3354183

[15] T. Schrijvers, G. Tack, P. Wuille, H. Samulowitz, and P. J. Stuckey, "Search combinators," *Constraints*, vol. 18, no. 2, pp. 269–305, 2013. [Online]. Available: http://link.springer.com/article/10.1007/s10601-012-9137-8

[16] S. Varrette, H. Cartiaux, S. Peter, E. Kieffer, T. Valette, and A. Olloh, "Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0," in *Proceedings of the 2022 6th High Performance Computing and Cluster Technologies Conference*, ser. HPCCT '22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 14–24. [Online]. Available: https://doi.org/10.1145/3560442.3560445

[17] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette, "XCSP3: An Integrated Format for Benchmarking Combinatorial Constrained Problems," Nov. 2022, arXiv:1611.03398 [cs]. [Online]. Available: http://arxiv.org/abs/1611.03398

[18] H. Li, M. Yin, and Z. Li, "Failure Based Variable Ordering Heuristics for Solving CSPs (Short Paper)," in *LIPIcs, Volume 210, CP 2021*, vol. 210, 2021. [Online]. Available: https://doi.org/10.4230/LIPICS.CP.2021.9

[19] W. Hugues, C. Lecoutre, A. Paparrizou, and S. Tabary, "Refining Constraint Weighting," Nov. 2019, pp. 71–77. [Online]. Available: https://doi.org/10.1109/ICTAI.2019.00019

[20] S. Haim and M. Heule, "Towards Ultra Rapid Restarts," Feb. 2014, arXiv:1402.4413 [cs]. [Online]. Available: http://arxiv.org/abs/1402.4413