Better understandings and configurations in MaxSAT stochastic local search solvers via anytime performance analysis

Ye, F.; Luo, C.; Cai, S.

Better Understandings and Configurations in MaxSAT Stochastic Local Search Solvers via Anytime Performance Analysis

Furong Ye[1,2], **Chuan Luo**[3*], **Shaowei Cai**[1]

[1]Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

[2] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands

[3] School of Software, Beihang University, Beijing, China

f.ye@ios.ac.cn, chuanluo@buaa.edu.cn, caisw@ios.ac.cn

Abstract

Though numerous solvers have been proposed for the MaxSAT problem, and the benchmark environment such as MaxSAT Evaluations provides a platform for the comparison of the state-of-the-art solvers, existing assessments were usually evaluated based on the quality, e.g., fitness, of the best-found solutions obtained within a given running time budget. However, concerning solely the final obtained solutions regarding specific time budgets may restrict us from comprehending the behavior of the solvers along the convergence process. This paper demonstrates that Empirical Cumulative Distribution Functions can be used to compare MaxSAT stochastic local search solvers' anytime performance across multiple problem instances and various time budgets. The assessment reveals distinctions in solvers' performance and displays that the (dis)advantages of solvers adjust along different running times. This work also exhibits that the quantitative and high variance assessment of anytime performance can guide machines, i.e., automatic configurators, to search for better parameter settings. Our experimental results show that the hyperparameter optimization tool, i.e., SMAC, can achieve better parameter settings of solvers when using the anytime performance as the cost function, compared to using the metrics based on the fitness of the best-found solutions.

Code and Datasets —
https://github.com/FurongYe/AAAI25-MaxSAT/

Introduction

The Maximum Satisfiability (MaxSAT) problem is the optimization version of the influential Boolean Satisfiability (SAT) problem that aims at finding the maximum number of satisfied clauses (Biere, Heule, and van Maaren 2009). (Weighted) partial MaxSAT, which is an important generalization of MaxSAT, divides the clauses into hard and soft ones, and a feasible solution requires all hard clauses to be satisfied (Li and Manya 2021). Among various complete and incomplete solvers developed for solving MaxSAT, stochastic local search (SLS) (Cai et al. 2016; Lei and Cai 2018), which commonly follows an iterative framework, is a significant category of incomplete solvers. While new methods have been continuously proposed, the corresponding comparison results are presented in numerous documents. The famous MaxSAT Evaluations (Berg et al. 2023a) provide a platform for practitioners to understand the performance of state-of-the-art solvers, and the existing comparisons among MaxSAT solvers mainly concern the quality, i.e., fitness, of best solutions obtained within a given time budget, namely, cutoff time (Luo et al. 2014; Cai et al. 2014; Cai, Luo, and Zhang 2017; Luo et al. 2017; AlKasem and Menai 2021; Chu, Cai, and Luo 2023; Liu et al. 2025), which is a reasonable option when we require a concrete judgment in competitions. However, the assessments concerning best-found solutions may produce bias in the cutoff time and hinder us from understanding the behavior of solvers during the iterative optimization process. Though some work (Zheng et al. 2022) investigated the convergence process when analyzing SLS solvers' behavior, the current mechanism usually plots individual convergence lines for particular instances. Hickey and Beacchus's work (Hickey and Bacchus 2022) addressed anytime performances of MaxSAT solvers. Still, their work focused on proposing a solution that would obtain good solutions on multiple cutoff times without addressing the issue of measuring anytime performance. Meanwhile, recent work (Hansen et al. 2022) has discussed that anytime performance assessment is vital for benchmarking algorithms.

Two perspectives, i.e., fixed-budget and fixed-target, are commonly considered when measuring iterative algorithms' performance. The fixed-budget approach is usually applied to problems in which prior knowledge about the performance, e.g., the fitness scale, is unavailable, and it can assess the order of the obtained solutions. The fixed-target approach is commonly applied in well-understood practical scenarios or benchmarking environments, and it uses quantitative indicators to explain how fast an algorithm obtains a solution meeting specific requirements. However, we usually need to consider both perspectives when developing an algorithm for practical problems. Therefore, addressing the concerns of both approaches, an anytime performance metric, which examines algorithms' performance based on empirical cumulative distribution functions (ECDF) of a set of cutoff times, has recently presented its superiority in black-box optimization benchmarking scenarios (Doerr et al. 2020; Hansen et al. 2016, 2022). The ECDF value at a given time is computed based on the fraction of solved prob-

*Chuan Luo is the corresponding author of this work.

lem instances or obtained solutions reaching the required target, e.g., fitness. In this way, we can obtain a ratio scaled value independent of the scale of solution fitness, and results regarding multiple cutoff times can be reasonably aggregated.

This paper introduces the first anytime performance assessments of MaxSAT SLS solvers. Our anytime performance assessments reveal intriguing behaviors that were invisible for fixed-budget assessments. The observations can provide valuable insights for improving the design and parameter settings of future SLS solvers. The hybrid solvers, which are the current state-of-the-art MaxSAT solvers, operate complete solvers and SLS solvers independently and alternatively (Cai and Lei 2020). Recent work on improving hybrid solvers mainly works on enhancing the performance of the SLS part (Berg et al. 2023b). Therefore, this work focusing on analyzing SLS will certainly contribute to future enhancement of the state-of-the-art MaxSAT solvers. In addition, the assessment techniques employed in this work can be transparent to both complete and hybrid solvers. In practice, our results *reveal (dis)advantages of the solvers across different problem instances* and *demonstrate algorithms' convergence progress aggregated across multiple instances*. Apart from providing a comprehensive assessment that can inspire experts to improve the design of algorithms further, the quantitative assessment also creates an alternative way for algorithmic tools, e.g., hyperparameter optimizers, to recognize algorithms' performance.

Since contemporary algorithms, including SLS, are mostly parametric, hyperparameter optimization (HPO) is vital for robust and competitive performance. For example, the MaxSAT Evaluation competition[1] submissions are usually embedded with fitting parameter settings. Meanwhile, automatic HPO tools have been commonly applied to achieve those fine-tuned settings. The HPO scenario for MaxSAT solvers usually addresses the algorithm configuration (AC) problem, which aims to find a configuration minimizing a cost function e. A configuration will be executed to evaluate the cost function e until a cutoff time κ. Based on theoretical analysis of evolutionary algorithms, recent work (Hall, Oliveto, and Sudholt 2022) has studied the impact of κ on the performance of HPO, considering both scenarios using fixed-target and fixed-budget approaches as e. Tuning algorithms' anytime performance has been addressed in the recent work (Ye et al. 2022), which compared the HPO scenarios of tuning fixed-target and anytime performances. While the mentioned studies were tested on classic benchmark problems such as ONEMAX (Doerr et al. 2020), López-Ibáñez and Stützle applied the hypervolume considering two objectives, i.e., the best-found solution quality and the corresponding cutoff time, to take into account the anytime behavior of algorithms (López-Ibáñez and Stützle 2014), and the method was tested on a MAX-MIN Ant system for Traveling Salesperson Problems and the famous mixed-integer solver SCIP (Achterberg 2009). In this work, we investigate tuning the anytime performance of a MaxSAT SLS solver, and the experimental results suggest that *anytime performance is a better mechanism than the fixed-budget one for the cost function of HPO*.

The main contributions of this work are as follows:

- We provide an anytime performance assessment of four state-of-the-art MaxSAT SLS solvers. We aggregate their ECDFs across multiple problem instances and compare the solvers' performance concerning various cutoff times. The assessment illustrates that ECDF, as a universal technique with a ratio scale, can distinguish the solvers that are considered identical when comparing the fixed-budget performance. Moreover, we observe that the solvers' performance varies along the optimization process while investigating their performance in particular instances, addressing the necessity of anytime performance assessment.
- We suggest tuning anytime performance can obtain better parameter settings compared to the commonly applied fixed-budget performance tuning in the MaxSAT community. By tuning the ECDF of SLS solvers, SMAC (Lindauer et al. 2022) can obtain better configurations in terms of anytime performance (i.e., ECDFs) and fixed-budget performance (i.e., scores based on the best-found solution quality). Note that the mechanism of computing incremental aggregated ECDFs for HPO used in this paper can be applied to other scenarios without interfacing with the HPO framework.

Preliminaries

Problem Definition Given a set of n Boolean variables $V = \{x_1, x_2, \ldots, x_n\}$, a *literal* l is either a variable x or its negation $\neg x$, and a *clause* is a disjunction of literals, i.e., $c = l_1 \vee l_2 \ldots \vee l_k$, where k is the length of c. Given an assignment α mapping Boolean values to each variable in V, a clause c is satisfied if at least one literal in c is *True*; otherwise, it is falsified. Given a conjunction normal form (CNF) $F = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, where m is the number of clauses, the MaxSAT problem is to find an assignment that maximizes the number of satisfied clauses. Partial MaxSAT (PMS) is a variant of MaxSAT that divides clauses into *hard* ones and *soft* ones, and a feasible assignment α requires all *hard* clauses to be satisfied. For the weighted PMS (WPMS) problem, each clause is associated with a weight $w(c) > 0$, and the problem is to find a feasible α that maximizes the total weights of satisfied clauses (namely, minimizes the total weights of falsified clauses). In this paper, we denote (W)PMS as a constrained optimization problem minimizing the $cost(\alpha)$ that is the total weight of falsified clauses. Nowadays, MaxSAT solvers are commonly developed and tested for (W)PMS, and we refer to MaxSAT as (W)PMS.

Algorithms SLS is a popular category of incomplete algorithms for MaxSAT. The common procedure of SLS samples an initial solution, i.e., assignment α, and then follows an optimization loop, creating new assignments by flipping one or multiple variables x in α iteratively until reaching the termination condition (e.g., using out the cutoff time). Variants of SLS solvers usually work on the strategies of picking one or multiple variable(s) x to be flipped.

[1] https://maxsat-evaluations.github.io/2023/

The classic SATLike solver (Lei and Cai 2018) proposed a clause weighting scheme, which introduces a bias towards certain variables x and determines which one(s) to be flipped. The newest extension of SATLike, SATLike3.0 (Cai and Lei 2020), is one of the state-of-the-art SLS solvers for MaxSAT and has been included in much work for comparisons (AlKasem and Menai 2021). NuWLS (Chu, Cai, and Luo 2023) proposed a new clause weighting scheme handling hard and soft clauses separately and worked on the weight initialization based on the framework of SATLike3.0. Differently from other methods that flip one variable at each iteration, MaxFPS (Zheng, He, and Zhou 2023) applies a farsighted probabilistic sampling method flipping a pair of variables. Moreover, instead of using the clause weighting technique, BandMax (Zheng et al. 2022) introduced the method of multi-armed bandit to select clauses to be satisfied, correspondingly deciding which variable to be flipped.

In this paper, we select SATLike3.0, BandMax, NuWLS, and MaxFPS for our first assessment of anytime performance of MaxSAT SLS solvers. Due to the space issue, we denote SATLike as SATLike3.0 in the following text.

Performance Assessment

We test the four state-of-the-art SLS solvers on the 2022 and 2023 anytime tracks of MaxSAT Evaluations, including weighted (MSE23-w and MSE22-w) and unweighted MaxSAT (MSE23-uw and MSE22-uw). All the tested solvers are implemented in C++ and complied with g++ 9.4.0. All experiments are conducted in Ubuntu 20.04.4 with the AMD EPYC 7763 CPU and 1TB RAM. The cutoff time is set as $300s$ for each run following the suggestion of MaxSAT Evaluations, and we conduct a solver of ten independent runs for each instance. Additional detailed results of the cutoff time $60s$, which is another suggestion of MaxSAT Evaluations, are available in our public repository.

We introduce in the following the solvers' fixed-budget performance and anytime performance. The former is commonly applied in the MaxSAT community, and the latter is studied for MaxSAT for the first time in this work. The fixed-target performance is not addressed because MaxSAT is an NP-hard problem, and its optima are unknown.

Fixed-budget Performance

Fixed-budget Performance Metric Existing work on SLS solvers for MaxSAT usually reports their experimental results based on (1) the number of winning instances ("#win") and (2) the score based on the competing solution (Berg et al. 2023a; AlKasem and Menai 2021). The number of winning instances presents the scope in the tested instances where a solver obtains the best solution among all competing solvers, and the score denotes the ratio between the solution quality obtained by a solver and the best-competing solver:

$$score(i) = \frac{\text{cost of the best solution for } i+1}{\text{cost of the found solution for } i+1} \in [0,1] \quad (1)$$

$score(i) = 0$ indicates that a solver can not find a feasible solution for the instance i. These two metrics are both calculated based on the best-found solution within a cutoff time. Consequently, such assessment outcomes may deviate along different cutoff times. Note that we are not critical of using these measures, as they can provide a concrete comparison for algorithm performance orders, as shown in amounts of existing work and MaxSAT Evaluation competitions.

	BandMax	MaxFPS	NuWLS	SATLike
MSE23-w	42(30)	26(20)	81	26(18)
MSE23-uw	73(42)	71(34)	110(99)	51(30)
MSE22-w	49(40)	49(41)	103	35(27)
MSE22-uw	89(65)	86(64)	106(91)	55(39)

Table 1: Number of instances in which a solver obtains the best performance among the competing solvers

	BandMax	MaxFPS	NuWLS	SATLike
MSE23-w	0.849	0.867	0.904	0.827
MSE23-uw	0.800	0.782	0.902	0.705
MSE22-w	0.869	0.889	0.942	0.864
MSE22-uw	0.854	0.852	0.901	0.731

Table 2: Aggregated scores of the solves

Experimental Results We present in Table 1 the number of instances in which a solver obtains the best solution (that is achieved by all runs of the competing solvers) in one of the tested ten runs, followed by the number (in brackets) of instances in which a solver obtains the best average results of ten runs among the competing solvers. Note that we omit numbers in brackets if they are identical to the former ones. Results are categorized for the four tested instance sets. According to Table 1, NuWLS obtains significant advantages against the other solvers, and BandMax and MaxFPS show similar performance across the four tested benchmark tracks. BandMax and MaxFPS win on more instances than SATLike when comparing the best of ten runs, but such advantages diminish when comparing the average of ten runs (as shown in the brackets). This observation indicates the necessity of performing multiple trials when comparing SLS solvers.

Table 2 presents the scores of the tested solvers, of which values are averaged across the instances for each benchmark track. Since we conduct ten independent runs for each instance, the score of a solver for an instance i is the average of ten trails. $score(i) = 1$ indicates that a solver obtains the best solution for the instance i among the competing solvers in all runs, and $score(i) = 0$ indicates that no feasible solution is obtained. We do not take the scores of the instances in which no solver obtains a feasible solution into account. We observe that the orders of the solvers' scores are identical to the ones in Table 1. In addition, according to Figure 1, there exist several instances in which no solver can obtain a feasible solution, and the solvers obtain the same or similar results in many instances, as shown in light yellow color, e.g., $score > 0.9$. Also, Figure 1 shows NuWLS obtains better scores in more instances compared to the other solvers.

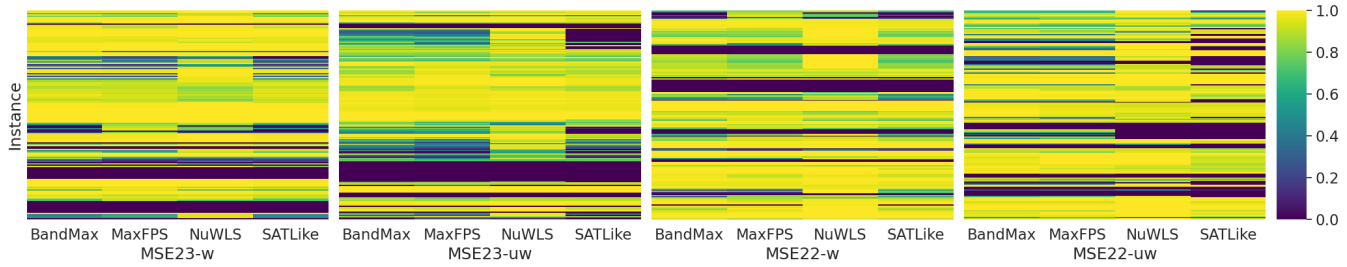


Figure 1: Heatmap illustrating the scores for individual instances. Each row represents an instance, while each column represents a different solver. The color depicts the score achieved by the solvers, with lighter shades indicating better performance. The benchmark tracks "MSE23-w", "MSE23-uw", "MSE22-w" and "MSE22-uw" consist of 160, 179, 197, and 179 instances (as shown by four boxes) respectively. The names of the instances have been omitted from the y-axis due to the limited space. Detailed results regarding groups of instances are available in Appendix B.

Anytime Performance

The assessment of fixed-budget performance provides initial insights into the performance of the tested solvers. However, these results only focus on the cutoff time of $300s$. While we can conduct similar analyses for multiple cutoff times, for example, the MaxSAT Evaluations platform includes cutoff times of $60s$ and $300s$, it remains challenging to aggregate and interpret the fixed-budget results across multiple cutoff time scenarios to understand the convergence of solvers. Therefore, we ask for an indicator that can assess anytime performance across various time budgets. Furthermore, this indicator shall be quantitative, allowing us to measure performance differences on a scale rather than by order. Particularly for instance-based scenarios such as MaxSAT, this indicator needs to be *universal* and independent of the scale of solution quality (e.g., cost of assignments) for each instance, enabling fair performance aggregation across instances.

Anytime Performance Metric. We assess the anytime performance of MaxSAT solvers using the ECDF values of a set of cutoff times. ECDF indicates the fraction of the obtained solutions satisfying a specific quality. We denote ϕ as the solution quality, e.g., the cost of an assignment. Given a set of solutions with $\Phi_i = \{\phi_{i1}, \phi_{i2}, \ldots\}$ for a problem instance i and a solver A that obtains the best-found solution with ϕ_{A_i} within the cutoff t, A's ECDF value at t is:

$$\text{ECDF}(A, i, t) = \frac{\mid \{\phi \in \Phi_i \mid \phi \geq \phi_{A_i}\} \mid}{\mid \Phi_i \mid} \qquad (2)$$

We work on minimization in this paper, and $\{\phi \in \Phi_i \mid \phi \geq \phi_{A_i}\}$ denotes the subset of solutions that are not better than the best-found one of A for instance i. For example, an algorithm A obtains a set of pairs of optimization time and the corresponding best-found solution $O = \{(t_1, \phi_{o1}), (t_2, \phi_{o2}), \ldots\}$, which are plotted by blue dots in Figure 2. Given a set Φ of solutions plotted by red stars, we present on the right subfigure the number of instances in Φ that are not better than the corresponding ϕ_o for each t, and the values normalized by the size of Φ are ECDFs.

Because ECDF is with a ratio scale, we can evaluate the ECDFs of a solver A for a set of optimization times t to measure A's *anytime* performance. ECDFs can also be combined

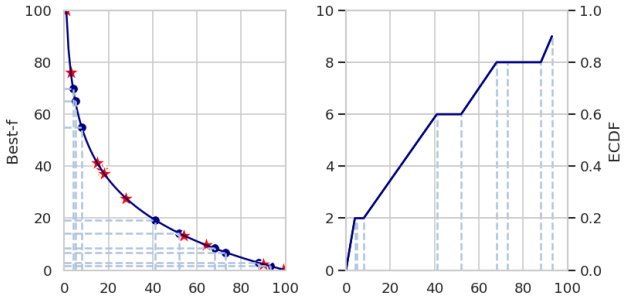
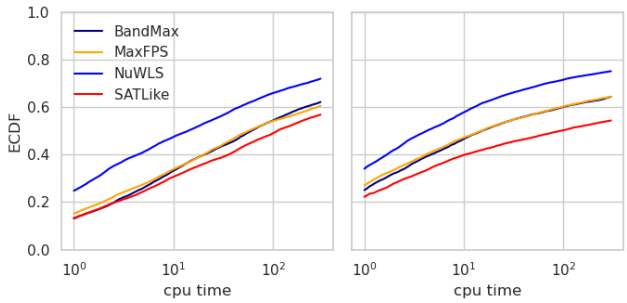


Figure 2: An example of ECDF calculation. **Left:** The blue dots represent pairs (t, ϕ), where a better solution with fitness ϕ (y-axis) is obtained at time t (x-axis). **Right:** the ECDF values based on the given set of solution fitness as presented in red stars on the left.

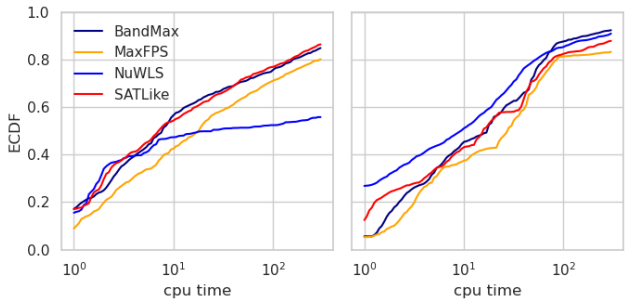
across multiple problem instances i by considering specific Φ for each instance.

To calculate ECDFs, we form a set of solution fitness values Φ_i for each instance i. This solution fitness set comprises the costs of solutions that the solvers have visited during our experiments. We compute the ECDFs at 100 specific optimization times, chosen within the range of $[0, 300s]$ following a logarithmic scale. This set of optimization times introduces a preference for evaluating the performance in terms of attaining high-quality solutions in less time.

Note that measuring ECDFs has been adopted in the black-box optimization community, where researchers quantify "time" in terms of function evaluations, i.e., the number of solutions that have been evaluated (Hansen et al. 2016; Wang et al. 2022). Function evaluation ensures reproducibility and erases disturbances caused by hardware environment, programming design, and other factors. However, MaxSAT solvers are usually applied in practical scenarios concerning cpu time, and cpu time has been commonly used when measuring MaxSAT solvers' performance. We have conducted analyses by using cpu time and function evaluations as the time metrics, respectively. The analyses using these two time metrics show identical conclusions. Therefore, we present the results using cpu time in this paper. The



(a) **Left:** WPMS (instances of MSE22-w and MSE23-w) and **Right:** PMS (instances of MSE22-uw and MSE23-uw)



(b) **Left:** "decision-tree" instances (15 in total), and **Right:** "ParametricRBACMaintenance" instances (13 in total).

Figure 3: Aggregated ECDFs of each solver for different sets of problem instances. x-axis indicates cpu time, and y-axis indicates the corresponding aggregated ECDF values.

data using function evaluations that are supported in a favored black-box benchmarking platform (Wang et al. 2022; de Nobel et al. 2023) is available in our repository.

Observing Anytime Performance. Figure 3a shows that NuWLS outperforms the other solvers for both weighted and unweighted MaxSAT. BandMax and MaxFPS show similar performance, but by examining the "anytime" analysis, we can see that MaxFPS slightly outperforms BandMax within a time limit of $10s$, and the order of their performances alter as the optimization time increases for the weighted MaxSAT. Furthermore, detailed analysis of anytime performance for specific instances can reveal exciting findings. Although NuWLS demonstrates advantages over other solvers considering ECDFs aggregated across the entire set of tested problem instances, it does not exhibit superiority on "decision-tree" instances after being trapped in local optima around $10s$, as shown in Figure 3b. On the other hand, while BandMax obtains similar performance to SATLike for "ParametricRBACMaintenance" instances when cpu time is less than $100s$, it outperforms the other solvers afterward. Compared to the fixed-budget performance assessment, these anytime performance observations offer more insights into the behavior of algorithms for specific problem instances and can be valuable for enhancing algorithm design.

Quantitative Assessment with Variation As mentioned previously, ECDF has a ratio scale that allows aggregation of values across multiple optimization times to obtain quantitative assessments for each instance, which is the so-called (approximated) area under the ECDF curve (AUC) (Ye et al. 2022). Table 3 presents the aggregated ECDFs for the four benchmark tracks, showing that the order of the solver's assessment based on ECDF remains the same as the one based on scores, as presented in Table 2. When analyzing the behavior in specific instances, Figure 4 shows higher variances in ECDFs for each instance compared to the ones in scores (see Figure 1). Specifically, instead of having a large set of scores higher than 0.9 in Figure 1, ECDFs are in a wide variation of values as shown in Figure 4. We also provide the average standard deviations of scores and ECDFs across all tested instances in Appendix A, which shows that the standard deviations obtained from ECDFs are much greater than the scores. Moreover, for the instances where solvers can obtain the same best-found solutions, ECDFs can distinguish solvers' performance by quantitatively estimating the convergence process. For example, Figure 4 can show that NuWLS converge faster for the instances plotted in yellow.

	BandMax	MaxFPS	NuWLS	SATLike
MSE23-w	0.444	0.445	0.584	0.409
MSE23-uw	0.546	0.548	0.696	0.466
MSE22-w	0.458	0.470	0.624	0.420
MSE22-uw	0.589	0.602	0.696	0.494

Table 3: Aggregated ECDFs for each benchmark track. The ECDF value for each run on an instance is normalized by the number of considered cutoff times, i.e., 100. The presented values are average across the corresponding instances.

Hyperparameter Optimization

Contemporary algorithms, such as heuristics, evolutionary algorithms, machine learning models, etc., are usually parametric. Therefore, parameter settings are essential for the competitive performance of algorithms. Before the development of automatic tools, parameter settings were manually chosen based on experts' experience or repetitive testing using Grid Search. Nowadays, the tools such as SMAC (Lindauer et al. 2022) and irace (López-Ibáñez et al. 2016) have been commonly applied for automatic HPO. While practical scenarios may propose various requirements for HPO, we address in this paper the algorithm configuration (AC) problem, which meets the requirement of tuning MaxSAT solvers for a set(s) of problem instances. AC (Lindauer et al. 2022) aims at searching a well-performance configuration, i.e., parameter setting, $\lambda \in \Lambda$ of an algorithm A across a set of problem instances $\{i_1, \ldots\} \subset \mathcal{I}$:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} e(\lambda) \tag{3}$$

where the cost function e is usually set as the best-found solutions' quality intuitively when tuning MaxSAT SLS solvers. However, in this section, we introduce AUC as an alternative candidate for e and compare the results obtained by using these two different mechanisms as the cost function for tuning MaxSAT SLS solvers. For ease of reading, we denote AUC and aggregated ECDFs as the metrics used in HPO and experimental comparisons, respectively.

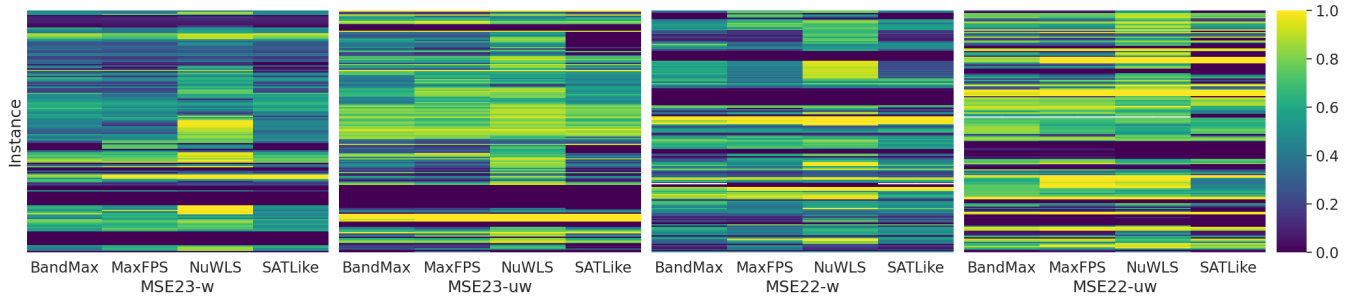


Figure 4: Heatmap illustrating aggregated ECDFs for individual instances. It follows the layout of Figure 1, but the color depicts the ECDF values achieved by the solvers. Detailed results regarding groups of instances are available in Appendix B.

Settings of Hyperparameter Optimization. We use the AC arcade of SMAC3 [2] to compare the results of tuning fixed-budget performance and anytime performance. **Best-f**, which calculates the fitness of the best-found solution obtained within a given cutoff t_B, is a straightforward option for the cost function of tuning fixed-budget performance. In addition, we apply $-score$ in Eq. 1, which is denoted as **Norm-f**, as another cost function of tuning fixed-budget performance. To calculate scores, we use the best-found solutions obtained in the Performance Assessment section. Note that this function can be adopted for competitions but is not doable for the practical scenarios lacking baselines, i.e., the known best-found solutions. As for calculating the cost function of anytime performance **approximated AUC**, we form a set T of 50 optimization times chosen within a range of $[0.1s, t_B]$ following a logarithmic scale. Since the time budget t_B for executing each run of configurations is usually relatively small for HPO scenarios, e.g., $t_B = 100s < 300s$ in this work, following a bias toward the configurations that have the potential to yield better solutions when provided with an additional time budget, we use $t_B - T$ as the set of optimization times to calculate AUCs. Regarding the target (i.e., solution quality) set Φ_i for each instance i, we start with a set F_i of five fitness values chosen from $[f_{\text{init\_min}}, f_{\text{init\_max}}]$ based on the linear scale, where $f_{\text{init\_min}}$ and $f_{\text{init\_max}}$ are the fitness of the best and the worst solutions visited by the initial configuration. When a configuration obtains a better best-found solution for an instance during the tuning process, the corresponding best-found fitness value will be added to F_i. In addition, the AUC of a configuration for a given instance is calculated by aggregating the values across 50 optimization times. In practice, we use a modified $\text{AUC}' = -\text{AUC} \cdot \mid \Phi_i \mid$ measuring e of configurations for each instance i. AUC$'$ decreases accordingly when a better configuration achieves a better best-found solution, thereby increasing the size of Φ. In this way, all of the tuning scenarios are framed as minimization tasks for HPO. Though computing AUC is more complex than computing Best-f and Norm-f, in practice, this computing takes linear time, and its time consumption is negligible compared to the time budget allocated for each run of a configuration.

[2]https://github.com/automl/SMAC3

We have tuned the four solvers for the MSE23-w and MSE23-uw benchmark tracks by using SMAC with three different cost functions. We conduct five runs of each SMAC setting with the cpu time budget of $60,000s$, following the suggestion in the work of NuWLS (Chu, Cai, and Luo 2023). Twenty percent of the instances are randomly selected for training. The obtained configurations of solvers are compared based on scores and AUCs across all the instances. Note that the computation of scores and ACUs follows the same settings as the Performance Assessment section.

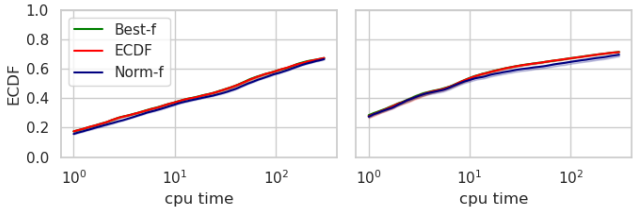
Anytime Performance can Lead to Better Configurations. Table 4 presents the results of configurations obtained by each setting of SMAC, e.g., tuning solvers using different cost functions, and Figure 5 demonstrates the mean and 95% confidence interval of aggregated ECDFs of five configurations along cpu time. More detailed results of each tuned configuration are provided in Appendix C, and all results are from 120 (4 solvers $\times$ 3 cost functions $\times$ 2 benchmark tracks $\times$ 5 SMAC runs) configurations.

We first compare the anytime performance of the obtained configurations for eight scenarios (4 solvers $\times$ 2 benchmark tracks). According to Table 4, using AUC as the cost function can obtain the best anytime performance for six out of eight scenarios, except for BandMax and MaxFPS in MSE23-uw. We can observe in Figure 5a-b that the anytime performance of the tuned BandMax and MaxFPS configurations are close to each other, and the results of using AUC as the cost function generally obtain slight advantages along CPU time for BandMax. When comparing the fixed-budget performance, i.e., scores, using AUC as the cost function obtains the best results for five out of eight scenarios in Table 4. Note that Wilcoxon signed-rank tests indicate that the results of using AUC as the cost function significantly differ from the others. P-values are provided in Appendix C.

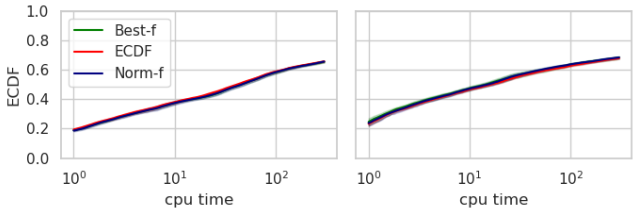
Overall, our results suggest that tuning anytime performance can lead to better configurations in terms of both anytime performance and best-found solution quality. Specifically, when tuning NuWLS and SATLike, using AUC as the cost function obtains significant advantages (4% and 15% by average) compared to the second-best results. For the five out 16 scenarios (in Table 4) that tuning AUC does not obtain the best results, the obtained results are only 0.6% worse than the best ones by average.

	BandMax			MaxFPS			NuWLS			SATLike		
	Best-f	Norm-f	ECDF	Best-f	Norm-f	ECDF	Best-f	Norm-f	ECDF	Best-f	Norm-f	ECDF
Results on MSE23-w												
Score	0.857	0.850	**0.858**	0.844	**0.850**	0.848	0.795	0.809	**0.818**	0.664	0.564	**0.758**
ECDF	**0.423**	0.405	**0.423**	0.424	0.423	**0.430**	0.431	0.450	**0.479**	0.288	0.225	**0.353**
Results on MSE23-uw												
Score	**0.815**	0.799	0.814	**0.776**	0.773	0.769	0.824	0.827	**0.846**	0.617	0.717	**0.790**
ECDF	**0.549**	0.532	0.548	**0.505**	0.503	0.496	0.488	0.491	**0.528**	0.373	0.423	**0.477**

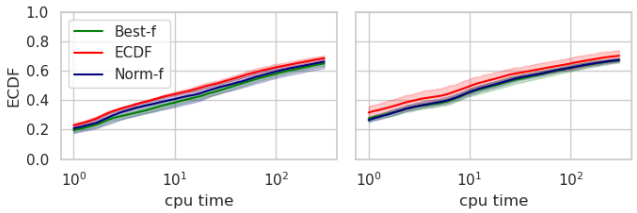
Table 4: Average scores and ECDFs of five configurations obtained by each setting. Larger values indicate better performances.



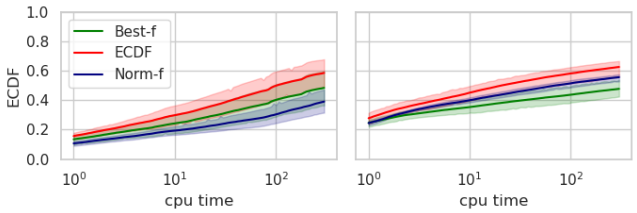
(a) BandMax



(b) MaxFPS



(c) NuWLS



(d) SATLike

Figure 5: Aggregated ECDFs of the configurations obtained by tuning for Best-f, Norm-f, and ECDF for **Left:** MSE23-w and **Right:** MSE23-uw. Results plotted in one line are from five configurations obtained by independent runs of SMAC.

Reasons for the Better Results. One reason for the findings is that AUCs can provide *dense search space* for HPO. For example, within a given time budget t_B, when two configurations λ_1 and λ_2 obtain the same best-found solution for an instance i using different time $t \leq t_B$, Best-f and Norm-f will deliver identical information to HPO tools, but AUC can distinguish these two configurations. Another reason is that AUCs are *normalized values* ranging between 0 and 1 compared to Best-f. Although Norm-f is also normalized, its values range in a smaller domain for each instance. In contrast, AUCs have a wider variation of values. Note that we have also tested using the average of AUCs and Norm-f values across 20 instances as the cost function each time when SMAC evaluates a configuration. However, according to the results in Appendix D, normalization across multiple instances does not help configure MaxSAT solvers. In addition, when conducting HPO, the time budget t_B is usually set as a relatively small value due to time limits. For example, in our tuning process of SMAC, we allocated a budget of $100s$ for each configuration. However, when validating the obtained configurations, the budget is $300s$. In the scenarios where configurations are allocated with limited time budgets, by considering multiple cutoff times, AUCs can *robustly estimate* the potential ability of a configuration to achieve better solutions. On the other hand, using Best-f or Norm-f as the cost function may be misled by fortunate results obtained at a specific cutoff time.

Conclusion

We have introduced ECDF for anytime performance analysis of MaxSAT SLS solvers. Our assessments have been conducted for the four state-of-the-art solvers, namely BandMax, MaxFPS, NuWLS, and SATLike. We illustrate that ECDF can measure solvers' anytime performance regarding multiple cutoff times, provide quantitative assessments with a ratio scale that can be aggregated across multiple instances, and differentiate the solvers that are considered similar in terms of fixed-budget performance. Our experiments show that the AUC can serve as a competitive metric for hyperparameter optimization. Compared to the traditional fixed-budget cost functions, the AUC can help achieve configurations that are, on average, around 10% better than the second-best ones in 11 out of 16 tested scenarios. In the remaining scenarios, its results are, on average, only 0.6% worse than the best ones. Although we have presented the application in the context of MaxSAT SLS, we expect that the applied techniques can be transparent to the hybrid methods integrating complete and incomplete solvers. Furthermore, we have observed that the choice of cost functions can affect the results of HPO. Therefore, we plan to study multi-objective HPO and explore the effect of cost functions. Also, we will study the algorithm portfolio configuration by considering the solvers' anytime performance.

Acknowledgements

This work is supported by the National Key R&D Program of China (2023YFA1009500), the National Natural Science Foundation of China (62202025), the Beijing Natural Science Foundation (L241050), Young Elite Scientist Sponsorship Program by CAST (YESS20230566), CCF-Huawei Populus Grove Fund (CCF-HuaweiFM2024005), and Fundamental Research Fund Project of Beihang University.

References

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1): 1–41.

AlKasem, H. H.; and Menai, M. E. B. 2021. Stochastic local search for Partial Max-SAT: an experimental evaluation. *Artificial Intelligence Review*, 54(4): 2525–2566.

Berg, J.; Järvisalo, M.; Martins, R.; and Niskanen, A. 2023a. MaxSAT Evaluation 2023. *Department of Computer Science Series of Publications B*, B-2023-2.

Berg, J.; Järvisalo, M.; Martins, R.; and Niskanen, A. 2023b. MaxSAT Evaluation 2023 : Solver and Benchmark Descriptions0. http://hdl.handle.net/10138/564026.

Biere, A.; Heule, M.; and van Maaren, H. 2009. *Handbook of satisfiability*, volume 185. IOS press.

Cai, S.; and Lei, Z. 2020. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287: 103354.

Cai, S.; Luo, C.; Lin, J.; and Su, K. 2016. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240: 1–18.

Cai, S.; Luo, C.; Thornton, J.; and Su, K. 2014. Tailoring Local Search for Partial MaxSAT. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2623–2629.

Cai, S.; Luo, C.; and Zhang, H. 2017. From Decimation to Local Search and Back: A New Approach to MaxSAT. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 571–577.

Chu, Y.; Cai, S.; and Luo, C. 2023. NuWLS: Improving Local Search for (Weighted) Partial MaxSAT by New Weighting Techniques. In Williams, B.; Chen, Y.; and Neville, J., eds., *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*, 3915–3923.

de Nobel, J.; Ye, F.; Vermetten, D.; Wang, H.; Doerr, C.; and Bäck, T. 2023. IOHexperimenter: Benchmarking platform for iterative optimization heuristics. *Evolutionary Computation*, 1–6.

Doerr, C.; Ye, F.; Horesh, N.; Wang, H.; Shir, O. M.; and Bäck, T. 2020. Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing*, 88: 106027.

Hall, G. T.; Oliveto, P. S.; and Sudholt, D. 2022. On the impact of the performance metric on efficient algorithm configuration. *Artificial Intelligence*, 303: 103629.

Hansen, N.; Auger, A.; Brockhoff, D.; Tusar, D.; and Tusar, T. 2016. COCO: Performance Assessment. *CoRR*, abs/1605.03560.

Hansen, N.; Auger, A.; Brockhoff, D.; and Tusar, T. 2022. Anytime Performance Assessment in Blackbox Optimization Benchmarking. *IEEE Transactions on Evolutionary Computation*, 26(6): 1293–1305.

Hickey, R.; and Bacchus, F. 2022. Large Neighbourhood Search for Anytime MaxSAT Solving. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, 1818–1824.

Lei, Z.; and Cai, S. 2018. Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 1346–1352.

Li, C. M.; and Manya, F. 2021. MaxSAT, hard and soft constraints. In *Handbook of satisfiability*, 903–927. IOS Press.

Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D.; Benjamins, C.; Ruhkopf, T.; Sass, R.; and Hutter, F. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23: 54:1–54:9.

Liu, C.; Liu, G.; Luo, C.; Cai, S.; Lei, Z.; Zhang, W.; Chu, Y.; and Zhang, G. 2025. Optimizing local search-based partial MaxSAT solving via initial assignment prediction. *Science China Information Sciences*, 68(2): 122101:1–122101:15.

López-Ibáñez, M.; Dubois-Lacoste, J.; Cáceres, L. P.; Birattari, M.; and Stützle, T. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3: 43–58.

López-Ibáñez, M.; and Stützle, T. 2014. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal Of Operational Research*, 235(3): 569–582.

Luo, C.; Cai, S.; Su, K.; and Huang, W. 2017. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243: 26–44.

Luo, C.; Cai, S.; Wu, W.; Jie, Z.; and Su, K. 2014. CCLS: an efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7): 1830–1843.

Wang, H.; Vermetten, D.; Ye, F.; Doerr, C.; and Bäck, T. 2022. IOHanalyzer: Detailed Performance Analyses for Iterative Optimization Heuristics. *ACM Transactions on Evolutionary Learning and Optimization*, 2(1): 3:1–3:29.

Ye, F.; Doerr, C.; Wang, H.; and Bäck, T. 2022. Automated Configuration of Genetic Algorithms by Tuning for Anytime Performance. *IEEE Transactions on Evolutionary Computation*, 26(6): 1526–1538.

Zheng, J.; He, K.; and Zhou, J. 2023. Farsighted Probabilistic Sampling: A General Strategy for Boosting Local Search MaxSAT Solvers. In Williams, B.; Chen, Y.; and Neville, J., eds., *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*, 4132–4139.

Zheng, J.; He, K.; Zhou, J.; Jin, Y.; Li, C.; and Manyà, F. 2022. BandMaxSAT: A Local Search MaxSAT Solver with Multi-armed Bandit. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, 1901–1907.