Journal of Combinatorial Optimization (2022) 44:3711–3734
https://doi.org/10.1007/s10878-022-00907-5

Using the method of conditional expectations to supply an improved starting point for CCLS

Daniel Berend[1] · Shahar Golan[2] · Yochai Twitto[3]

Accepted: 1 September 2022 / Published online: 30 September 2022

Abstract

This paper proposes to combine the method of conditional expectations (MOCE, also known as Johnson's Algorithm) with the state-of-the-art heuristic configuration checking local search (CCLS), to solve maximum satisfiability (Max Sat) instances. First, MOCE is used to find an outstanding assignment, and then CCLS explores the solution space, starting at this assignment. This combined heuristic, which we call MOCE–CCLS, is shown to provide a significant improvement over each of its parts: MOCE and CCLS. An additional contribution of this paper is the results of a comprehensive comparative evaluation of MOCE–CCLS versus CCLS on various benchmarks. On random benchmarks, the combined heuristic reduces the number of unsatisfied clauses by up to tens of percents. On Max Sat 2016 and 2021 public competition benchmarks, which include crafted and industrial instances also, MOCE–CCLS outperforms CCLS as well. To provide an empirical basis to the above result, this work further explores the correlation between the quality of initial assignments provided to CCLS and that of the corresponding final assignments. Empirical results show that the correlation is significant and long-lasting. Thus, under practical time

A preliminary version (Berend and Twitto 2020) of this work has been presented at the 18th International Symposium on Experimental Algorithms (SEA 2020). The contribution of this elaborated paper is mainly on the practical solving side. We performed a comprehensive empirical evaluation of the combined solver MOCE–CCLS on all the benchmarks from the Max Sat Evaluation of 2016 and 2021 to assess and establish its usability as a state-of-the-art solver for Max Sat.

✉ Yochai Twitto
yochait@sce.ac.il

Daniel Berend
berend@cs.bgu.ac.il

Shahar Golan
sgolan@jct.ac.il

[1] Departments of Mathematics and of Computer Science, Ben-Gurion University, 84105 Beer Sheva, Israel

[2] Department of Computer Science, Jerusalem College of Technology, 91160 Jerusalem, Israel

[3] Department of Software Engineering, Shamoon College of Engineering, 8410802 Beer Sheva, Israel

constraints, the quality of the initial assignment is crucial to the performance of local search heuristics.

Keywords Combinatorial optimization · Maximum satisfiability · Method of conditional expectations · Local search

1 Paper overview

This paper is organized as follows. Section 2 is dedicated to the introduction, in which we give a general overview of the Max Sat problem and related literature. In particular, in this section, an elaborated description of MOCE and CCLS is provided.

In Sect. 3, MOCE and CCLS are combined into an improved algorithm. Instead of starting CCLS from a random initial assignment, it is started from excellent initial assignments, provided by MOCE. The experiments show that this provides a performance improvement, which becomes more and more significant as the instance grows. It has been noticed in other problems, such as TSP and QAP, that local search heuristics yield excellent results when started from initial solutions selected greedily with respect to expectation (Gutin and Yeo 2002; Gutin and Punnen 2006).

In Sect. 4, the correlation between the quality of the initial assignments provided to local search heuristics and the quality of the final assignments resulting from them is explored. We show that there is a strong long-lasting correlation between the quality of the initial assignment, from which the local search heuristic starts, and that of the final assignment provided by it.

The above correlation implies that, even in later stages of the local search, it is still in the shadow of the initial assignment. Thus, the quality of the initial assignment is crucial under practical time constraints. The observed correlation decays slower for denser instances, and faster for sparser ones. We show that the correlation is statistically significant, and estimate the impact of the improvement in the quality of the initial assignment on the quality of the final assignment.

A summary and conclusions are presented in Sect. 5.

2 Introduction

This section is dedicated to presenting the maximum satisfiability problem, related literature and the current state-of-the-art in the field. An elaboration on the method of conditional expectations, and the so-called configuration checking local search heuristic is provided as well.

In the maximum satisfiability (Max Sat) problem (Li and Manyà 2009), the input is a sequence of clauses over some boolean variables. Each clause is a disjunction of literals over different variables. A literal is either a variable or its negation. The goal is to find a truth (`true`/`false`) assignment for the variables, maximizing the number of satisfied (made `true`) clauses.

In the Max r-Sat problem, each clause is restricted to consist of at most r literals. The case in which each of the clauses consists of exactly r literals, is sometimes called

Max Er-Sat. Denote by n the number of variables and by m the number of clauses. The density of the instance is $\alpha = m/n$. As is customary in the literature, this paper focuses on the case where r and α are constant.

As Max r-Sat (for $r \geq 2$) is NP-hard Ausiello et al. (2003, pp. 455–456), it cannot be exactly solved in polynomial time (assuming $P \neq NP$), and one must resort to approximation algorithms and heuristics. Numerous methods have been suggested for solving Max r-Sat, e.g. (de Boer et al. 2005; Selman et al. 1992, 1996; Luo et al. 2014; Chen and Santhanam 2015; Narodytska and Bacchus 2014; Ansótegui et al. 2013; Davies and Bacchus 2011; Heras et al. 2008; Niedermeier and Rossmanith 2000), and an annual competition of solvers has been held since 2006 (Argelich et al. 2020).

Various complete solvers for Max Sat have been developed during recent years, some of which were presented in several annual evaluations of Max Sat solvers (Argelich et al. 2020). Among these practical solvers, one can find Branch and Bound solvers [e.g., MaxSatz (Li et al. 2007) and Clone (Pipatsrisawat and Darwiche 2007)], Satisfiability based solvers [e.g., SAT4J (Le Berre and Parrain 2010) and QMaxSat (Koshimura et al. 2012)], and Unsatisfiablity based solvers [e.g., WPM1 (Ansótegui et al. 2012, 2009)], etc. Complete solvers which participated in the last evaluations include MaxHS (Davies 2013), Pacose (Paxian et al. 2018), EvalMaxSAT (Avellaneda 2020), and more. Other exact algorithms worth mentioning include the branching algorithm presented by Li et al. (2022) [which improves over the results of Chen and Kanj (2004)], and the algorithm presented by Xiao (2022).

Practical incomplete solvers for Max Sat are actively researched as well. Some of them competed in the incomplete track of the last evaluations—Loandra (Berg et al. 2017), TT-open-WBO-Inc (Nadel 2019), sls-mcs (Guerreiro et al. 2019), and more. More elaboration on randomized techniques can be found in Sect. 2.1. Elaboration on local search based solvers can be found in Sect. 2.2.

Overall, satisfiability related questions have attracted a lot of attention from the scientific community. Some of them were thoroughly studied, especially the satisfiability threshold density question (Chvátal and Reed 1992; Friedgut and Bourgain 1999; Achlioptas and Peres 2004; Mertens et al. 2006; Coja-Oghlan 2014; Ding et al. 2015). According to the phase transition conjecture (Achlioptas 2009), for each $r \geq 2$, there exists a constant α_r, such that the probability of a random Max r-Sat instance over n variables and αn clauses to be satisfiable converges to 1 if $\alpha < \alpha_r$ and to 0 if $\alpha > \alpha_r$, as n approaches infinity.

There has been a lot of work on the conjecture, including (Franco and Paull 1983; Chvátal and Reed 1992; Friedgut and Bourgain 1999; Mézard et al. 2002; Achlioptas and Peres 2004). The conjecture was proved by Ding et al. (2015) for sufficiently large r. Coja-Oghlan (2014) has shown that the threshold is $\alpha_r = 2^r \ln 2 - (1 + \ln 2)/2 + \varepsilon_r$, where $\varepsilon_r \to 0$ as $r \to \infty$. Some lower and upper bounds on the satisfiability threshold density, α_r, are provided in Table 1. Empirical results indicate that $\alpha_3 \approx 4.27$ and $\alpha_4 \approx 9.93$ (Mertens et al. 2006; Crawford and Auton 1996). For a comprehensive overview of the whole domain of satisfiability, we refer to (Biere et al. 2009).

For instances with density much above the satisfiability threshold, Coppersmith et al. (2004) studied the expected number of clauses satisfied by an *optimal* assignment,

Table 1 Theoretical lower and upper bounds on the satisfiability threshold density

α_r	2	3	4	5	7	10	20
Upper bound	1	4.46	10.23	21.33	87.88	708.94	726,817
Lower bound	1	3.52	7.91	18.79	84.82	704.94	726,809

for a uniformly random instance of clause length r and density α. In particular, they provided theoretical lower and upper bounds on this number as the density goes to infinity.

An interesting study of Max 3-Sat was provided in Prügel-Bennett and Tayarani-Najaran (2012). The authors claimed that many instances share similar statistical properties and provided empirical evidence for it. Simulation results on the autocorrelation of a random walk in the assignments space were provided for several instances, as well as extrapolation for the typical instance. Finally, a novel heuristic was introduced, ALGH, which exploits long-range correlations found in the problem's landscape. This heuristic outperformed GSAT (Selman et al. 1992) and WSAT (Selman et al. 1996).

A slightly better version of this novel heuristic, based on clustering instead of averaging, is provided in another paper (Qasem and Prügel-Bennett 2010) of the same authors. This version turned out to outperform all the heuristics implemented by that time in the Sat solver framework UBCSAT (Tompkins and Hoos 2005).

Hoos et al. (2004) analyze how the method of generating random instances affects the autocorrelation and fitness-distance correlation. These quantities are considered fundamental to understanding the hardness of instances for local search algorithms. They raised the question of similarity of the landscape of different instances. Angel and Zissimopoulos (2000) calculated the autocorrelation coefficient of several problems and classified problem hardness accordingly.

Merz and Freisleben (1999) provide elaboration on correlations and on the way of harnessing them to design well-performing local search heuristics and memetic algorithms. The importance of selecting an appropriate neighborhood operator for producing the smoothest possible landscape was emphasized. For some landscapes, the autocorrelation length is shown to be associated with the average distance between local optima. This may be used to facilitate the design of mutations that lead memetic algorithms out of the basin of attraction of a local optimum they reached.

Using Walsh analysis (Goldberg 1988), an efficient way of calculating moments of the number of satisfied clauses of a given instance of Max r-Sat was suggested in Heckendorn et al. (1999). Simulation results for the variance and higher moments of the number of clauses satisfied by a random assignment over the ensemble of all instances were provided as well.

Sutton et al. (2009) show how to use the Walsh decomposition (Goldberg 1988) to efficiently calculate the exact autocorrelation function and autocorrelation length of any given instance of Max r-Sat. Furthermore, this decomposition is used to approximate the expectation of these quantities over the ensemble of all instances.

The autocorrelation length, which is closely related to the ruggedness of landscapes, is of interest in the area of landscape analysis (Malan and Engelbrecht 2013; Sutton et al. 2009; Angel and Zissimopoulos 2000; Hoos et al. 2004; Fontana et al. 1993; Angel and Zissimopoulos 2001; Chicano et al. 2012). It is fundamental to the theory and design of local search heuristics (Chicano et al. 2013; Merz and Freisleben 1999). According to the autocorrelation length conjecture (Stadler 2002), in many landscapes, the number of local optima can be estimated using an expression based on this quantity.

2.1 The method of conditional expectations

The simple randomized approximation algorithm, which assigns to each variable a uniformly random truth value, independently of all other variables, satisfies $1 - 1/2^r$ of all clauses on the average. Furthermore, this simple algorithm can be easily derandomized using the well-known method of conditional expectations (MOCE, also known as Johnson's Algorithm) (Erdös and Selfridge 1973; Yannakakis 1994; Coppersmith et al. 2004; Poloczek 2011; Poloczek and Williamson 2016; Poloczek et al. 2017; Costello et al. 2011), yielding an assignment that is guaranteed to satisfy at least this proportion of clauses.

In a sense, this method is optimal for Max 3-Sat, as no polynomial-time algorithm for Max 3-Sat can achieve a performance ratio exceeding $7/8$ unless $P = NP$ (Håstad 2001). Note that, typically, this method yields assignments that are much better than this worst-case bound.

MOCE iteratively constructs an assignment by going over the variables in some (arbitrary) order. At each iteration, it sets the seemingly better truth value to the currently considered variable. This is done by comparing the expected number of satisfied clauses under each of the two possible truth values it may set to the current variable.

For a given truth value, the expected number of satisfied clauses is the sum of three quantities. The first is the number of clauses already satisfied by the values assigned to the previously considered variables. The second is the additional number of clauses satisfied by the assignment of the given truth value to the current variable. The third is the expected number of clauses that will be satisfied by a random assignment to all currently unassigned variables. The truth value, for which the sum in question is larger, is the one selected for the current variable. Ties are broken arbitrarily or randomly. The whole process is repeated until all variables are assigned.

In an efficient implementation, each step of MOCE takes a constant time on the average. The main thing to do at each step is to find the better truth value for the currently assigned variable and residualize the instance accordingly. To find this truth value, we calculate the expected gain in case the variable is assigned `true`. If this gain is positive, the variable is assigned `true`. Otherwise, it is assigned `false`, as the gain in assigning the variable `false` is the additive inverse.

To find the expected gain from assigning the current variable `true`, it suffices to go over the clauses the variable appears in. Each unsatisfied clause, that is made satisfied by the assignment to the current variable, contributes 2^{-l} to the overall expected gain, where l is the number of literals in the clause. In the residualization of the instance, these clauses are eliminated.

On the other hand, each clause, that remains unsatisfied by the assignment of `true` to the variable, contributes -2^{-l} to the overall expected gain. In the residualization of the instance, these clauses remain, but they are shortened by one literal—the one associated with the current variable.

The overall expected gain is defined as the sum of all the contributions obtained from all the clauses the current variable appears in. As each variable appears initially in $r\alpha$ clauses on the average, the whole step of selecting and assigning a variable a truth value depends only on the clauses this variable appears in. Thus, assuming that r and α are constants, a step takes a constant time on the average. Note, though, that in order to execute MOCE efficiently, its implementation requires two auxiliary data structures: a map from each variable to the list of clauses containing it and a map from each clause to the list of variables it contains.

The linear time complexity of MOCE makes it a good candidate for supplying an initial assignment to local search algorithms. It was chosen in this work due to its simplicity and efficiency.

Theoretical and empirical works related to MOCE, randomized algorithms and other algorithms of the same spirit, include (Coppersmith et al. 2004; Poloczek 2011; Poloczek and Williamson 2016; Poloczek et al. 2017; Costello et al. 2011).

2.2 Local search

Local search heuristics (Hoos and Stützle 2004) explore the assignment space. They usually start at a randomly generated assignment, and traverse the search space by flipping variables, usually one at a time. The leading solver configuration checking local search (CCLS) (Luo et al. 2014) follows this scheme and flips variables until some predefined number of flips is executed or the allotted time has been used up. Of course, if a satisfying assignment has been found, the execution is stopped as well. This work focuses on CCLS as a local search algorithm, due to its success in competitive evaluations.

CCLS performs two types of flips: random ones, with some predefined probability p, and greedy ones, with probability $1-p$. Random flips just flip a randomly selected variable from a randomly selected unsatisfied clause. Greedy flips are ones that flip the seemingly best possible variable among all the variables whose configuration has been changed and who satisfy at least one currently unsatisfied clause. This variable is the one with the maximum score out of those variables, i.e., the one whose flipping will lead to the maximum number of satisfied clauses. Ties are broken uniformly at random.

Generally, the number of satisfied clauses after CCLS flips a variable is not necessarily larger than prior to the flip. In fact, it is even possible that flipping any of the candidate variables will result at a lower quality assignment. Also, the set of candidates may be empty in some of the greedy steps. In the latter case, CCLS performs a random flip instead.

In CCLS, a variable is considered as a “configuration changed" variable if, since its most recent flip, at least one of its neighboring variables has been flipped. Here, the neighbors of a variable are those variables sharing with it at least one clause.

Recent works, related to local search, configuration checking, CCLS, and algorithms of the same spirit, include (Pankratov and Borodin 2010; Cai and Su 2011; Cai et al. 2014; Luo et al. 2014, 2013; Abramé et al. 2017; Cai et al. 2015; Cai and Su 2013; Selman et al. 1994; Mills and Tsang 2000; Smyth et al. 2003; Cha et al. 1997).

Iterated Local Search (ILS) metaheuristics (Lourenço et al. 2019; Dong et al. 2013; Den Besten et al. 2001) execute multiple local searches, each time starting from a different initial assignment. A simple approach to iterated local search is to generate the initial assignments randomly. In this approach, there is no learning from the results of the previous local searches. A better approach is to analyze the assignments provided by previous searches in order to produce a better initial assignment for the current local search.

Cai et al. (2016) propose various ideas to apply local search to Partial Max Sat instances. Their ideas include weighting for hard clauses, separating the scores for hard and soft clauses, unit propagation with priority on hard unit clauses, and more. Hains et al. (2013) use Walsh polynomials and exact computation of hyperplane averages to construct initial solutions and to improve local search heuristics.

Martin and Otto (1996) combine Simulated Annealing with local search heuristics. To this end, they introduce a new metaheuristic. In this metaheuristic, they embed local search techniques into simulated annealing so that it explores only local optima. Bouhmala (2019) introduces an adapted version of the Kernighan–Lin algorithm for the maximum satisfiability problem. This adapted algorithm is embedded into the simulated annealing algorithm. The combined solver has been shown to provide a state-of-the-art performance.

3 Combining MOCE and CCLS

This section describes the improvement obtained by letting CCLS start its execution from good initial assignments, versus starting it from a random assignment (as done originally). Specifically, the good initial assignments used here are assignments provided by MOCE. We refer to the algorithm that starts from the assignment provided by MOCE as MOCE–CCLS. To emphasize the fact that the original CCLS algorithm starts from a random assignment, we will call it RAND–CCLS.

The first experiments are conducted on several families of random instances. *Random* Max r-Sat instances are constructed as follows. The clauses are selected independently of each other. Each clause is generated by selecting r distinct variables uniformly at random, then negating each of them independently with probability $1/2$.

The families have been selected in a systematic way, so as to reveal trends in the performance, and connect it to the parameters of the family. Additional experiments are conducted on some public benchmarks. The experiments show that MOCE–CCLS scales much better than RAND–CCLS. In particular, as the instance size grows, so does the performance improvement provided by MOCE–CCLS over RAND–CCLS.

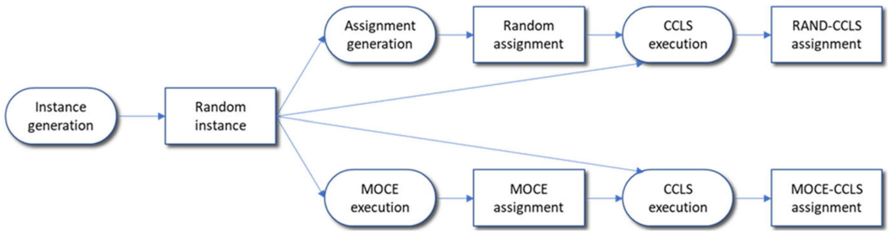
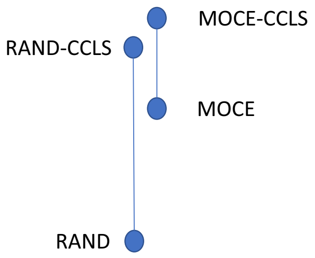


Fig. 1 The flow of the experiment



The points diagram above summarizes qualitatively the insights gained from the experiments. The higher the algorithm appears in the diagram, the better it is. Inspecting the diagram, one can see that MOCE–CCLS performs much better than MOCE, which in turn shows performance very far away from the baseline reference RAND. MOCE–CCLS performs better than RAND–CCLS as well. The last statement holds significantly for large instances, while for small and medium instances MOCE–CCLS maintains or slightly improves the performance of RAND–CCLS.

3.1 Comparative performance on structured benchmarks

This section focuses on random instances, for which the clauses are of length 3. The number of variables in each instance ranges from 10,000 to 1,000,000 in order to represent medium and large instances. The satisfiability threshold for Max r-Sat with $r = 3$ is about 4.27. In order to check cases both slightly below and above the satisfiability threshold, the density in the experiments ranges from 3 to 9. Such ranges allow us to systematically study the performance of the algorithms at hand on diverse families. For each family, 100 instances were selected uniformly at random, in the same way elaborated in Sect. 4.

Figure 1 describes the flow of the experiment. First a random instance is generated. Then two assignments are obtained: the first is generated randomly and the second is the result of a MOCE execution. For each such assignment CCLS is executed, given the instance and the assignment as input. This process is repeated 100 times (to reduce variability) and the performance of each algorithm is taken as the average number of unsatisfied clauses.

For Max r-Sat, the reference baseline RAND unsatisfies $m/2^r$ clauses on average, with an approximate standard deviation of $\sqrt{m(1-1/2^r)/2^r}$ clauses (Berend and Twitto 2016). For convenience, the (theoretical) average number of clauses unsatisfied

Table 2 The number of clauses unsatisfied by RAND and MOCE

n	α							
	3		5		7		9	
	RAND	MOCE	RAND	MOCE	RAND	MOCE	RAND	MOCE
10,000	3750	412	6250	1500	8750	2889	11,250	4442
50,000	18,750	2078	31,250	7459	43,750	14,409	56,250	22,209
100,000	37,500	4149	62,500	14,944	87,500	28,861	112,500	44,436
500,000	187,500	20,790	312,500	74702	437,500	144296	562500	222074
1,000,000	375,000	41,559	625,000	149,383	875,000	288,572	1,125,000	444174
% unsat	12.5%	1.4%	12.5%	3%	12.5%	4.1%	12.5%	4.9%

by RAND for the families we studied (namely, $m/8$), is provided in Table 2. The rows correspond to the various numbers of variables, n, and the columns to the various densities, α.

Table 2 also presents the average number of clauses unsatisfied by MOCE. It turns out that this number scales linearly with the number of clauses, and thus can be described as a proportion of the number of clauses, for any fixed density. The proportion of clauses unsatisfied by MOCE, out of all clauses, was 1.4%, 3%, 4.1%, and 4.9% for the densities 3, 5, 7, and 9, respectively. For each family, the percentage of clauses unsatisfied by each of the algorithms is provided in the last line of the table.

At each of the n steps of MOCE, it selects the seemingly best truth value to an arbitrary unassigned variable. This is done by inspecting all the clauses it appears on, once. Thus, during the overall execution of MOCE, each of the $m = n\alpha$ clauses in the instance is inspected at most r times—once for each of its literals. As the inspection time is constant, the overall time complexity of MOCE is proportional to rm. Thus, MOCE is a linear time algorithm.

MOCE is extremely fast in practice. In fact, its execution time is only a few seconds for the larger instances that were studied, and less than a second for the small and medium size instances. This time includes loading the instance, building it in the memory, and constructing the solution.

Although MOCE returns excellent solutions, it benefits a lot from supplementing it with a highly performing local search. In fact, executing the local search part of CCLS [which we simply call CCLS], starting from the solution returned by MOCE, a significant improvement is obtained. Namely, the number of unsatisfied clauses is significantly reduced at the local search stage.

This improvement is summarized in Table 3. In this table, the columns named "(M−MC)/M" present the relative improvement of MOCE–CCLS over MOCE. This relative improvement is the difference between the number of clauses unsatisfied by MOCE and the number of those unsatisfied by MOCE–CCLS, divided by the number of clauses unsatisfied by MOCE. The table also presents the improvement in the number of unsatisfied clauses of supplementing RAND with CCLS (which is simply the standard version of CCLS), under the columns named "(R−RC)/R".

Table 3 The improvement in the number of unsatisfied clauses by executing CCLS after RAND and MOCE

n	α							
	3		5		7		9	
	(R−RC)/R	(M−MC)/M	(R−RC)/R	(M−MC)/M	(R−RC)/R	(M−MC)/M	(R−RC)/R	(M−MC)/M
10,000	100.0%	100.0%	96.0%	83.6%	85.5%	56.2%	77.4%	42.9%
50,000	100.0%	100.0%	95.5%	81.2%	84.8%	54.1%	76.7%	41.2%
100,000	100.0%	100.0%	95.1%	79.9%	84.3%	52.9%	76.2%	40.2%
500,000	100.0%	100.0%	90.4%	69.4%	77.1%	42.4%	68.3%	31.3%
1,000,000	80.6%	100.0%	48.7%	49.6%	37.4%	27.1%	31.6%	18.3%

It is worth mentioning that this significant improvement comes with a caveat—a significant increase in the execution time. In fact, the results shown in Table 3 are based on 30 min executions of CCLS, after the initial solution (by either RAND or MOCE) has been obtained in just a few seconds.

This prolongation may, sometimes, be too much. This is the case especially when a large number of instances should be solved. Note that some real-world problems may be reduced to the solution of a large number of Max Sat instances. Such a situation arises, for example, in the recovery of encryption keys (Liao 2013; Liao et al. 2013; Kamal 2012; Kamal and Youssef 2010; Heninger 2011; Halderman et al. 2009).

One more caveat is due to the fact that, as the instances grow larger, this improvement decreases. As the instance grows larger, the number of flips CCLS can perform during the allotted time decreases, and with it decreases the obtained improvement as well.

In this context, note that, theoretically, if the execution time is unlimited, CCLS, due to its incorporated randomness, will eventually reach an optimal solution. Clearly, this has no practical implications, as the amount of time required is way out of reach.

This section is concluded by comparing MOCE–CCLS and RAND–CCLS head to head. The comparison is provided in Table 4 (which is wrapped for readability). For each density, we provide the number of clauses unsatisfied by RAND–CCLS, the number of clauses unsatisfied by MOCE–CCLS, and the relative improvement of MOCE–CCLS over RAND–CCLS. The latter number is the difference between the number of clauses unsatisfied by RAND–CCLS and the number of those unsatisfied by MOCE–CCLS, divided by the number of clauses unsatisfied by RAND–CCLS.

The results demonstrate the importance of the initial solution. Even after 30 min of local search, and using the excellent local search heuristics CCLS, the initialization with MOCE instead of RAND yields better solutions.

Moreover, MOCE–CCLS proved to be much more scalable than RAND–CCLS. As the instance grows larger, the improvement of MOCE–CCLS over RAND–CCLS becomes more significant. Whereas, for small instances, MOCE–CCLS improves RAND–CCLS by less than 1%, for large instances the improvement exceeds 50%.

In view of the above, when using a local search algorithm for Max Sat, one should strive to start the search from very good assignments. This holds as long as it is not too much time consuming to attain such assignments.

Table 4 MOCE–CCLS vs. RAND–CCLS

n	α					
	3			5		
	RC	MC	% improve	RC	MC	% improve
10,000	0	0	NaN	248	246	0.81%
50,000	0	0	NaN	1417	1403	0.99%
100,000	0	0	NaN	3038	3002	1.18%
500,000	0	0	NaN	29976	22,894	23.63%
1,000,000	72,642	0	100.00%	320,674	75,260	76.53%

n	α					
	7			9		
	RC	MC	% improve	RC	MC	% improve
10,000	1265	1264	0.08%	2546	2537	0.35%
50,000	6647	6617	0.45%	13122	13,052	0.53%
100,000	13717	13,588	0.94%	26,770	26,554	0.81%
500,000	99976	83,163	16.82%	178,234	152,512	14.43%
1,000,000	548,044	210,440	61.60%	769,640	363,037	52.83%

3.1.1 Experimentation information

The experiments described in this section were executed on a Sun Grid Engine (SGE) (2020) managed cluster of 31 identical IBM m4 servers with Intel Xeon E5-2620@2.0GHz processors. Each of the servers consists of 24 computation cores and 64GB of working memory. Thus, the platform had 744 computation cores and 1984GB of working memory at hand.

Each of the jobs submitted to the cluster was limited to use up to 3GB of working memory. Provided the load on the cluster, the parallelization was of about 100 times, thus reducing the experiment overall sequential time of approximately 4.11 months to around 1.25 days of parallel execution.

3.2 Comparative performance on public benchmarks

An international evaluation of solvers for the maximum satisfiability problem has been held annually since 2006 (Argelich et al. 2020). The evaluations are primarily divided to complete/incomplete and weighted/unweighted tracks. Here, we focus on the incomplete unweighted track, as the suggested heuristic is incomplete and it does not take weights into account. Until 2016, it was common to group the benchmarks into 3 categories—Random, Crafted, and Industrial. From 2017 onward, the instances are categorized only by the background problems from which they originated.

In this subsection, we compare the performance of RAND–CCLS and MOCE–CCLS. To this end, the Exclusive Won measure is used. According to this measure, for each instance, the winner is the competitor that satisfies more clauses. If both competitors satisfy the same number of clauses, the result is a draw for this instance. The overall winner of a given benchmark (collection of instances) is the competitor that wins more instances in this benchmark. In the Max Sat 2016 Evaluation, a measure called the Instance Won measure (equivalent to Exclusive Won for two competitors) has been used. In the Max Sat 2021 Evaluation a similar measure has been used. The Instance Won measure can be derived from the Exclusive Won measure by adding the number of draws to the number of wins for each of the competitors.

In Sect. 3.2.1 we compare the performance of RAND–CCLS and MOCE–CCLS on the instances in the Max Sat 2016 Evaluation. In Sect. 3.2.2 we compare the performance of these heuristics for blown up versions of the random instances of the 2016 Evaluation. Finally, in Sect. 3.2.3 we compare the performance of RAND–CCLS and MOCE–CCLS on the Max Sat 2021 Evaluation. Overall, the results indicate that MOCE–CCLS outperforms RAND–CCLS. Moreover, as the instances grow larger, so does the performance gap in favor of MOCE–CCLS.

3.2.1 Max Sat 2016 evaluation benchmarks

Table 5 presents the results of a comparative evaluation, conducted to assess the performance of RAND–CCLS and MOCE–CCLS on the incomplete unweighted track of Max Sat 2016 Evaluation. The table is divided into four parts— Random, Crafted, Industrial, and Max Sat 2016 (overall). Each of the three first parts provides the results for each of the benchmarks of the relevant category and the overall performance in the whole category (denoted by "subtotal"). The "Size" column gives the number of instances in the given collection/benchmark. The column "RC" ("MC", respectively) gives the number of instances exclusively won by RAND–CCLS (MOCE–CCLS, respectively). The last column gives the number of instances ended with a draw. The last row gives the total over all benchmarks of Max Sat 2016 Evaluation.

The random and crafted instances of the Max Sat 2016 Evaluation are very small. They consist of several hundred variables and several thousand clauses per instance. Thus, they are less adequate for evaluation of local search heuristics. Indeed, most of the solvers participating in that evaluation found solutions with the same number of unsatisfied clauses in most of the instances. (Those solutions were probably optimal.) In our comparison of RAND–CCLS and MOCE–CCLS on the random and crafted instances, the situation was no different; both usually found solutions of the same quality. The industrial instances, on the other hand, are quite large. They consist of hundreds of thousands of variables and millions of clauses. On those instances, MOCE–CCLS gives RAND–CCLS a knockout, as one clearly see in the table.

3.2.2 Blown-up versions of instances from the 2016 evaluation

This subsection considers three additional benchmarks in the same spirit as the abrame-habet benchmark (consisting of random instances) of the 2016 Evaluation—but larger ones. In order to keep the exact same blend of instances, the new benchmarks were created by blowing up the number of variables and of clauses of the original instances from the abrame-habet benchmark. The densities (and clause lengths) are the same as in the original benchmark.

Table 5 MOCE–CCLS versus RAND–CCLS, the Exclusive Won measure with a time limit of 5 min, on instances of Max Sat Evaluation 2016

Category	Benchmark	Size	RC	MC	Draw
Random	Abrame-habet	372	1	0	371
	Highgirth	82	9	8	65
	Subtotal	**454**	**10**	**8**	**436**
Crafted	Bipartite	100	0	0	100
	Maxcut	292	0	0	292
	Set-covering	10	4	2	4
	Subtotal	**402**	**4**	**2**	**396**
industrial	Circuit-debugging-problems	3	0	3	0
	Sean-safarpour	51	0	51	0
	Subtotal	**54**	**0**	**54**	**0**
Max Sat 2016	**Total**	**910**	**14**	**64**	**832**

Bold values indicate summaries

Table 6 MOCE–CCLS vs. RAND–CCLS, Exclusive Won measure, on random instances blown up from the abrame-habet benchmark

Blowing factor	RC	MC	Draw
× 10 (1 min)	112	115	145
× 100 (2 min)	137	223	12
× 1000 (5 min)	4	368	0

Three expanded benchmarks were created by enlarging the original one by factors of 10, 100, and 1000. For example, for the new benchmark obtained after blowing up by a factor of 10, for each instance, a new random instance was created whose numbers of variables and clauses are 10 times the corresponding numbers in the original instance. Thus, instances with 70 variables and 700 clauses gave rise in the tenfold blown up benchmark to instances with 700 variables and 7000 clauses.

We compared MOCE–CCLS and RAND–CCLS on the enlarged instances using the Exclusive Won measure. To this end, each of the two competitors ran on each of the instances for a few minutes (CPU time). The results are presented in Table 6. The first row in the table relates to the benchmark with instances blown up by a factor of 10. The time limit for instances in this benchmark is 1 min. Similarly, the second and third rows relate to the benchmarks with instances blown up by a factor of 100 (with a time limit of 2 min) and 1000 (with a time limit of 5 min), respectively.

There is an almost complete draw on the competition instances and on the tenfold blown up the instances. However, when scaling the instances by a factor of 100, MOCE–CCLS wins decisively, and when scaling by a factor of 1000, it beats RAND–CCLS by a knockout.

Note that the number of variables in instances of the original benchmark is between 70 and 200. Thus, even after blowing up by a factor of 1000, the obtained instances are of medium size only.

In the 1000-fold expanded benchmark, the improvement was manifested in an average of extra 423 clauses satisfied by MOCE–CCLS per instance. As RAND–CCLS is a state-of-the-art solver, with excellent performance, this number of extra clauses satisfied by MOCE–CCLS provides a significant improvement.

3.2.3 Max Sat 2021 evaluation benchmarks

In order to assess the performance of the heuristics on more recent benchmarks, we have also compared them on the incomplete unweighted track of the Max Sat 2021 Evaluation. This track consists of 30 benchmarks, with 155 instances altogether. Note that the amount of memory, provided for running each of the heuristics on each of the instances, was insufficient for 12 out of the 155 instances. Thus, our statistics is based on the 143 other instances (representing 28 benchmarks) only.

The results are given in Table 7. As one can see, MOCE–CCLS wins exclusively on all instances of 6 of the benchmarks and non-exclusively on 5 additional benchmarks. RAND–CCLS wins exclusively on all instances of 4 of the benchmarks (3 of which have but a single instance), and non-exclusively on 3 additional benchmarks. In 10 of the benchmarks there is a draw. Altogether, MOCE–CCLS clearly wins, with 59 exclusive wins, compared to RAND–CCLS with 31 only.

4 Correlation between the quality of initial and final assignments

In this section, we explore the correlation between the number of clauses unsatisfied by an initial assignment and the number of those unsatisfied by the corresponding final assignment, where the transition is by CCLS. The ongoing correlation during the execution is explored as well. CCLS was chosen for its excellent performance; a local search heuristic of lower quality may well be expected to yield an even stronger correlation.

To generate initial assignments of diverse quality, MOCE is revised by adding to it a parameter that allows us to invert its decision regarding the truth value for the current variable. This parameter, to which we refer as the inversion probability, is the probability to assign to a variable the truth value opposite to the one chosen by MOCE. Namely, for a given inversion probability $0 \leq p \leq 1$, at each step, the current variable is assigned the truth value chosen by MOCE with probability $1 - p$, and the opposite truth value with probability p. Thus, for $p = 0$ the algorithm is simply MOCE, while for $p = 1$ it is “anti-MOCE”. We refer to this tailored algorithm as PMOCE.

It is worth emphasizing that there is no reason to use this algorithm (with $p \neq 0$) to solve Max Sat instances. Its sole purpose is to construct initial assignments with a wide range of qualities. Indeed, our experiments showed a strong correlation between the value of p and the quality of the assignment provided by PMOCE.

We have generated a benchmark, consisting of 5 families of instances of Max 3-Sat. Each of the families consists of 150 instances over 100,000 variables. The densities of the 5 families are 5, 7, 9, 12, 15. The instances in each family were generated uniformly at random as follows. The clauses of an instance were generated independently of each

Table 7 MOCE–CCLS versus RAND–CCLS, the Exclusive Won measure with a time limit of 5 min, on instances of Max Sat Evaluation 2021

Benchmark	Size	RC	MC	Draw
Aes	6	1	1	4
Atcoss	7	2	5	0
Decision-tree	23	0	10	13
Extension-enforcement	2	0	2	0
Gen-hyper-tw	8	0	8	0
hs-timetabling	1	0	1	0
Large-graph-commmunity-detection_jabbour	3	0	3	0
Logic-synthesis	1	1	0	0
Maxclique	3	0	0	3
Maxcut	11	0	0	11
MaximumCommonSub-GraphExtraction	1	0	0	1
MaxSATQueriesinInterpretableClassifiers	1	1	0	0
Min-fill	7	7	0	0
Optic	1	1	0	0
Planning-bnn	6	0	6	0
Program_disambiguation-Ramos	1	0	1	0
Railroad_reisch	8	1	7	0
Railway-transport	4	2	2	0
Ramsey	11	0	0	11
Reversi	7	1	6	0
scheduling	3	1	2	0
Scheduling_xiaojuan	8	6	1	1
Set-covering	9	4	2	3
Setcover-rail_zhendong	4	0	0	4
Treewidth-computation	3	2	1	0
uaq	1	0	0	1
uaq_gazzarata	1	0	0	1
Xai-mindset2	2	1	1	0
Total	**143**	**31**	**59**	**53**

Bold values indicate summaries

other. Each of the clauses was generated by selecting 3 distinct variables uniformly at random, and then negating each of them with probability $1/2$, independently.

4.1 End-to-end correlation

In the following, we describe what was done in the experiment for each family. For each instance in the family, we executed PMOCE with 51 inversion probabilities, ranging from 0 to 1 in steps of 0.02. Thus, we obtained 51 initial assignments with presumed diverse quality. From each of these initial assignments, a local search was started using

Table 8 End-to-end correlation coefficients and regression slopes

Density	Correlation coefficient			Regression slope	
	Mean	Std	p value	Mean	Std
5	0.52	0.11	$1.7 \cdot 10^{-3}$	$0.5 \cdot 10^{-3}$	$0.1 \cdot 10^{-3}$
7	0.74	0.06	$3.6 \cdot 10^{-7}$	$1.5 \cdot 10^{-3}$	$0.2 \cdot 10^{-3}$
9	0.79	0.12	$2.1 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$	$0.5 \cdot 10^{-3}$
12	0.73	0.17	$1.2 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
15	0.83	0.08	$1.1 \cdot 10^{-5}$	$3.4 \cdot 10^{-3}$	$0.7 \cdot 10^{-3}$

CCLS, and thus obtained 51 final assignments. The result of the 51 executions, was 51 pairs of numbers. Each pair consisted of the number of clauses unsatisfied by the initial assignment generated by PMOCE, and the number of unsatisfied clauses at the end of the search done by CCLS. The cutoff time of CCLS was set to 30 min, measured in CPU time.

For each instance, the correlation coefficient was calculated over the corresponding 51 pairs. After going over the whole family, we had 150 correlation coefficients—one for each instance. Then, the mean and standard deviation of these 150 values of correlation coefficients were calculated.

For each of the correlation coefficients, we also calculated the p value. The p value is the probability that this correlation, or a higher one, would have been found if the correlation coefficient was in fact zero (null hypothesis). If this probability is lower than the conventional 5% (i.e., the p value is less than 0.05), the correlation coefficient is considered statistically significant. For each family, the average p value was calculated over the 150 correlation coefficients as a measure of the statistical significance of the results.

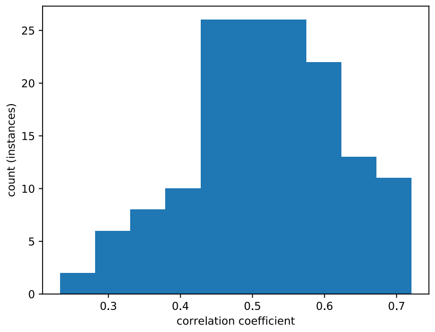
To measure the impact of the improvement of the quality of an initial assignment on the quality of the corresponding final assignment, regression analysis was applied. Specifically, we calculated the regression line of each of the instances of a family, and took its slope as a measure of the strength of the impact. The average of these 150 slopes was taken as a measure of the strength of this impact in a given family.

The results are provided in Table 8. Each line summarizes the results of one family. For example, the first line summarizes the results of the family with density 5. In this family, instances are of 100,000 variables and 500,000 clauses. The mean correlation coefficient measured (over 150 random instances) was 0.52, with a standard deviation of 0.11. The mean p value was $1.7 \cdot 10^{-3}$, and the mean and standard deviation of the regression slope were $0.5 \cdot 10^{-3}$ and $0.1 \cdot 10^{-3}$, respectively.

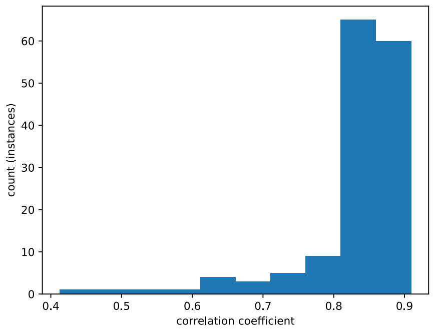
Figure 2 depicts histograms of the 150 end-to-end correlation coefficients of the family of density 5 (Fig. 2a) and for the family of density 15 (Fig. 2b).

The results show a strong positive correlation between the quality of the initial and final assignment for all densities. The correlation is stronger for denser families. The p value is lower by far than the conventional 0.05, which indicates that the correlation coefficients obtained in the experiments are statistically very significant.

The high correlation between the quality of the initial assignment and that of the final one makes it clear that the search continues to be in the shadow of the initial



(a) Family of density 5.



(b) Family of density 15.

Fig. 2 Histograms of end-to-end correlation coefficients

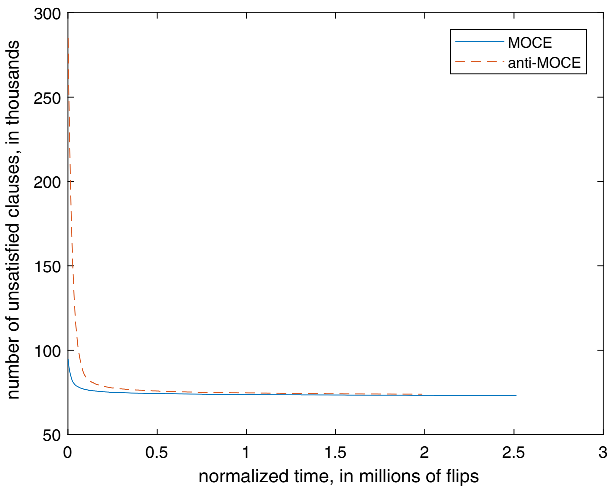


Fig. 3 Number of unsatisfied clauses as function of the number of flips

assignment for a long period of time. Thus, starting a local search from an excellent initial assignment, rather than a random assignment, its performance may improve (as long as the selection of such an initial assignment does not consume too much time).

While the correlation is strong, the regression slope suggests that a large improvement in the initial assignment yields only a small improvement in the final assignment. As CCLS eventually converges to the optimal solution, there is little room for improvement by the end of its execution, so that this regression slope makes sense. Moreover, it is to be expected that the slope becomes even smaller as one runs CCLS longer.

Note that, after 30 min of execution, CCLS is way beyond its rapid improvement stage. In fact, it is deep in its convergence stage and shows relatively minor improvements as time goes by. This validates the correlation observed as meaningful.

Figure 3 depicts the number of unsatisfied clauses as a function of the number of flips made, for an arbitrary (but representative) instance from the family of density 15. The graph shows this number for inversion probability of 0 (MOCE) and 1 (anti-MOCE). One can see that CCLS enters its convergence stage quite early in the execution.

We also emphasize the two phases seen in the graphs. The first phase is the rapid improvements phase. In this phase, the number of unsatisfied clauses is decreasing rapidly. This phase ends after about 100,000 flips. The second phase, which we call the convergence phase, continues from there onward. In this phase, the improvements are rarer and smaller.

4.2 Ongoing correlation

Besides the end-to-end correlation, we explored the ongoing correlation during the experiment. To this end, for each initial assignment, the minimum number of unsatisfied clauses found so far was recorded, not only at the end of the execution, but also after every 1000 flips made by CCLS. Then we calculated the correlation coefficient between the number of clauses unsatisfied by the initial assignment and the number of unsatisfied clauses recorded at each 1000 flips snapshot.

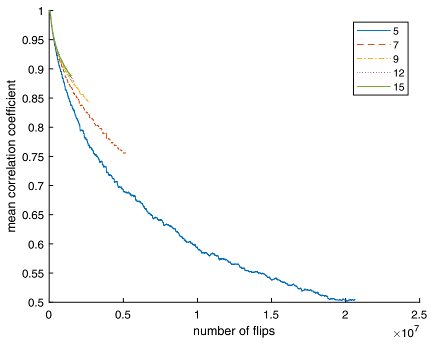
The number of flips made during the execution is very different for different families. In a denser instance, a flip takes longer, so that less flips are made. Even for instances of the same family, the number of flips varies. The statistics are provided only up to the minimal number of flips made, over all instances in the family.

Figure 4 depicts the decay in the correlation as a function of time, where time is measured in number of flips made from the beginning of the local search. It seems that the number of flips is the natural time scale to measure the correlation decay. While the graphs are noisy, the trend is clear—the correlation gradually decays as a function of the number of flips made, and it does so slower for denser families. Moreover, as the density grows larger, the differences in the decay seem to be smaller and the graphs are almost overlapping.

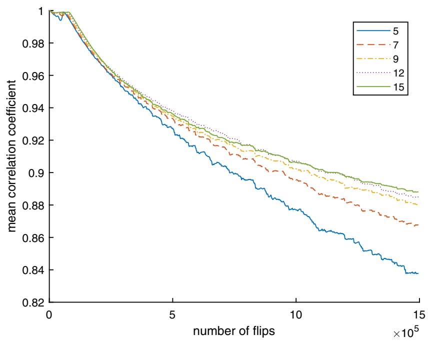
Figure 4a shows the full results. It provides the graphs of correlation decay of all the families. In the figure, one may observe that the number of flips made in denser families is much smaller than the number of flips made in sparser ones. For example, the minimal number of flips over all instances and inversion probabilities for the family of density 5 was about 20,600,000, while for the family of density 15 it was about 1,500,000. The reason is that, in denser families, each variable appears in a larger number of clauses, and a flip makes a larger number of variables available for selection subsequently. Thus, at each step, CCLS has to deal with a larger pool of candidates for flipping, which consumes more time per flip.

Figure 4b depicts the same graphs, but only up to about 1,500,000 flips, which is the place where the graph of the family of density 15 ends. In this figure, one can clearly see the faster decay of the correlation in sparser families, as well as the smaller differences between the decay in denser families.

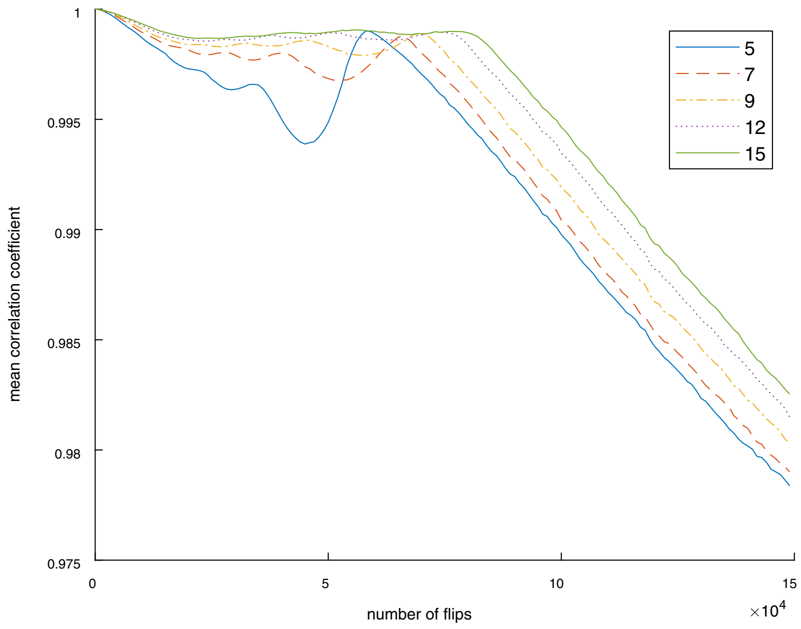
Figure 4c zooms in on the first 150,000 flips. During this stage, one may observe a phenomenon of phase transition in the decay of the correlation. The empirical results suggest two phases of decay. The first phase starts at the beginning and ends after about 60,000–80,000 flips. In this phase, the correlation decays very slowly. This phase is



(a) Full graphs.



(b) Up to 1,500,000 flips.



(c) First150,000flips.

Fig. 4 Ongoing correlation decay as a function of the number of flips

characterized by a rapid decrease in the number of unsatisfied clauses, and is aligned with the rapid decrease shown in Fig. 3.

The second phase is from about 60,000–80,000 flips onward. This phase is characterized by a faster decay in the correlation. It is aligned with the convergence stage of CCLS, shown in Fig. 3, in which the number of unsatisfied clauses is decreasing slowly over time.

The position of the phase transition around 60,000-80,000 flips may be explained by the fact that the initial assignment provided by MOCE is expected to be at a distance of about 50,000 flips from an optimal solution. So the first 50,000 flips are significant.

But, as about 30% of the flips of CCLS are random, and not all the flips are useful in general, this area stretches further to about 60,000–80,000 flips.

During the first phase, the initial assignment is very important in determining the correlation. In all the executions, the decrease in the number of satisfied clauses is rapid and considerable at this phase. Thus, the different executions maintain their relative positions, which leads to a very slow decrease in the correlation. Afterward, the correlation decays at about the same speed, as can be seen in Fig. 4c.

Note that Fig. 4c demonstrates an initially quite strange-looking phenomenon. Namely, between about 50,000 flips and 80,000 flips, depending on the family, the correlation increases. The reason is the large variability in the quality of the initial assignments. Recall that we managed to get initial assignments in a very wide range, starting at very inferior "anti-MOCE" assignments and ending at superior MOCE assignments. When starting from a high-quality assignment, the rapid improvement phase is shorter, and when starting from a low-quality assignment, it is longer. In the time interval, where for good initial assignments the rapid improvement phase has ended already, while for other assignments it has not, the correlation is understandably smaller. After the rapid increase phase was finished for all initial assignments, the correlation returns to a higher level.

4.3 Experimentation information

The experiments described in this section were carried out on the same platform as those in Sect. 3. Provided the load on the cluster, the parallelization was of about 300 times, thus reducing the experiments overall sequential time of approximately 2.18 years to around 2.66 days of parallel execution.

5 Summary and conclusions

This paper suggests combining the method of conditional expectations (MOCE) with the state-of-the-art heuristic configuration checking local search (CCLS). Instead of starting CCLS from random initial assignments, we suggest starting it from excellent initial assignments, provided by MOCE. The combined MOCE–CCLS solver provided a significant improvement over CCLS. Moreover, MOCE–CCLS proved to be much more scalable than CCLS. Namely, while for small and medium size instances the improvement is minor, for large instances CCLS breaks down while MOCE–CCLS continues with the same performance.

To provide empirical basis to the above result, we have explored the correlation between the quality of initial assignments provided to local search heuristics and that of the corresponding final assignments. The experiment results have shown that this correlation is significant and long-lasting. Thus, under practical time constraints, the quality of the initial assignment is crucial to the performance of local search heuristics.

Given the above, we recommend MOCE–CCLS over RAND–CCLS, and generally recommend starting local search heuristics from assignments even better than those provided by MOCE, as long as such may be obtained in linear time or slightly longer (say, by a logarithmic factor).

Acknowledgements We thank Shaowei Cai for providing us access to the original authors' implementation of the CCLS solver used in Max Sat Evaluation 2016, and André Abramé for providing us access to the abrame-habet benchmark used (partially) in that evaluation. We thank Gregory Gutin for his helpful comments on this paper. We also thank the anonymous referees for their useful comments and beneficial suggestions. This research was partially supported by the Milken Families Foundation Chair in Mathematics and by the Israeli Council for Higher Education (CHE) via Data Science Research Center, Ben-Gurion University of the Negev.

References

Abramé A, Habet D, Toumi D (2017) Improving configuration checking for satisfiable random k-SAT instances. Ann Math Artif Intell 79(1):5–24

Achlioptas D (2009) Random satisfiability. Handb Satisf 185:245

Achlioptas D, Peres Y (2004) The threshold for random k-SAT is $2^k \log 2 - O(k)$. J Am Math Soc 17(4):947–973

Angel E, Zissimopoulos V (2000) On the classification of NP-complete problems in terms of their correlation coefficient. Discrete Appl Math 99(1):261–277

Angel E, Zissimopoulos V (2001) On the landscape ruggedness of the quadratic assignment problem. Theoret Comput Sci 263(1):159–172

Ansótegui C, Bonet ML, Levy J (2013) SAT-based MaxSAT algorithms. Artif Intell 196:77–105

Ansótegui C, Bonet ML, Levy J (2009) Solving (weighted) partial MaxSAT through satisfiability testing. In: International conference on theory and applications of satisfiability testing. Springer, pp 427–440

Ansótegui, C, Bonet ML, Gabas J, Levy J (2012) Improving SAT-based weighted MaxSAT solvers. In: International conference on principles and practice of constraint programming. Springer, pp 86– 101

Argelich J, Li CM, Manyà F, Planes J (2020) MaxSat evaluations. https://maxsat-evaluations.github.io/

Ausiello G, Crescenzi P, Gambosi G, Kann V, M-Spaccamela A, Protasi M (2003) Complexity and approximation: combinatorial optimization problems and their approximability properties, 2nd edn. Springer, New York

Avellaneda F (2020) A short description of the solver EvalMaxSAT. MaxSAT Eval 2020:8

Berend D, Twitto Y (2020) Effect of initial assignment on local search performance for Max Sat. In: Proceedings of the 18th international symposium on experimental algorithms (SEA 2020), vol 160. Leibniz international proceedings in informatics (LIPIcs), pp 8:1– 8:14

Berend D, Twitto Y (2016) The normalized autocorrelation length of random Max r-Sat converges in probability to $(1 - 1/2^r)$/r". In: The 19th international conference on theory and applications of satisfiability testing (SAT 2016). Springer, pp 60–76

Berg J, Korhonen T, Järvisalo M (2017) Loandra: PMRES extended with preprocessing entering MaxSAT evaluation 2017. MaxSAT Eval 2017:13

Biere A, Heule M, van Maaren H (2009) Handbook of satisfiability, vol 185. IOS Press, Amsterdam

de Boer P-T, Kroese DP, Mannor S, Rubinstein RY (2005) A tutorial on the cross-entropy method. Ann Oper Res 134(1):19–67

Bouhmala N (2019) A Kernighan-Lin inspired algorithm for MAX-SAT. Sci China Inf Sci 62(11):1–3

Cai S, Jie Z, Su K (2015) An effective variable selection heuristic in SLS for weighted Max-2-SAT. J Heuristics 21(3):433–456

Cai S, Su K (2013) Local search for Boolean Satisfiability with configuration checking and subscore. Artif Intell 204:75–98

Cai S, Su K (2011) Local search with configuration checking for SAT. In 2011 IEEE 23rd international conference on tools with artificial intelligence. IEEE, pp 59–66

Cai S, Luo C, Lin J, Su K (2016) New local search methods for partial MaxSAT. Artif Intell 240:1–18

Cai S, Luo C, Thornton J, Su K (2014) Tailoring local search for Partial MaxSAT. In: Proceedings of the 28th AAAI conference on artificial intelligence. AAAI'14. AAAI Press, Québec City, Québec, pp 2623–2629

Cha B, Iwama K, Kambayashi Y, Miyazaki S (1997) Local search algorithms for partial MAXSAT. In: AAAI'97/IAAI'97, pp 263–268

Chen J, Kanj IA (2004) Improved exact algorithms for Max-Sat. Discrete Appl Math 142(1–3):17–27

Chen R, Santhanam R (2015) Improved algorithms for sparse MAXSAT and MAX-k-CSP. In: Proceedings of the 18th international conference on theory and applications of satisfiability testing (SAT 2015). Springer, pp 33–45

Chicano F, Luque G, Alba E (2012) Autocorrelation measures for the quadratic assignment problem. Appl Math Lett 25(4):698–705

Chicano F, Luque G, Alba E (2013) Problem understanding through landscape theory. In: Proceedings of the 15th annual conference companion on genetic and evolutionary computation. ACM, pp 1055–1062

Chvátal V, Reed B (1992) Mick gets some (the odds are on his side) (satisfiability). In: Proceedings of the 33rd annual symposium on foundations of computer science. IEEE, pp 620–627

Coja-Oghlan A (2014) The asymptotic k-SAT threshold. In: Proceedings of the 46th annual ACM symposium on theory of computing. ACM, pp 804–813

Coppersmith D, Gamarnik D, Hajiaghayi M, Sorkin GB (2004) Random MAX SAT, random MAX CUT, and their phase transitions. Random Struct Algorithms 24(4):502–545

Costello KP, Shapira A, Tetali P (2011) Randomized greedy: new variants of some classic approximation algorithms. In: Proceedings of the 22nd annual ACM-SIAM symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, pp 647–655

Crawford JM, Auton LD (1996) Experimental results on the crossover point in random 3-SAT. Artif Intell 81(1–2):31–57

Davies J (2013) Solving MaxSAT by decoupling optimization and satisfaction. PhD thesis. University of Toronto

Davies J, Bacchus F (2011) Solving MAXSAT by solving a sequence of simpler SAT instances. In: Proceedings of the 17th international conference on principles and practice of constraint programming. Springer, pp 225–239

Den Besten M, Stützle T, Dorigo M (2001) Design of iterated local search algorithms. In: Workshops on applications of evolutionary computation. Springer, pp 441–451

Ding J, Sly A, Sun N (2015) Proof of the satisfiability conjecture for large k. In Proceedings of the 47th annual ACM symposium on theory of computing (STOC'15), pp 59–68

Dong X, Chen P, Huang H, Nowak M (2013) A multi-restart iterated local search algorithm for the Permutation Flow Shop problem minimizing total flow time. Comput Oper Res 40(2):627–632

Erdös P, Selfridge JL (1973) On a combinatorial game. J Comb Theory Ser A 14(3):298–301

Fontana W, Stadler PF, Bornberg-Bauer EG, Griesmacher T, Hofacker IL, Tacker M, Tarazona P, Weinberger ED, Schuster P (1993) RNA folding and combinatory landscapes. Phys Rev E 47(3):2083–2099

Franco J, Paull M (1983) Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. Discrete Appl Math 5(1):77–87

Friedgut E, Bourgain J (1999) Sharp thresholds of graph properties, and the k-SAT problem. J Am Math Soc 12(4):1017–1054

Goldberg DE (1988) Genetic algorithms and Walsh functions: a gentle introduction. Department of Mechanical Engineering, University of Alabama, Clearinghouse for Genetic Algorithms

Guerreiro AP, Terra-Neves M, Lynce I, Figueira JR, Manquinho V (2019) Constraint-based techniques in stochastic local search MaxSAT solving. In: International conference on principles and practice of constraint programming. Springer, pp 232–250

Gutin G, Punnen AP (2006) The traveling salesman problem and its variations. Springer, New York

Gutin G, Yeo A (2002) Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number. Discrete Appl Math 119(1–2):107–116

Hains D, Whitley D, Howe A, Chen W (2013) Hyperplane initialized local search for MAXSAT. In Proceedings of the 15th annual conference on genetic and evolutionary computation, pp 805–812

Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, Feldman AJ, Appelbaum J, Felten EW (2009) Lest we remember: cold-boot attacks on encryption keys. Commun ACM 52(5):91–98

Håstad J (2001) Some optimal inapproximability results. J ACM (JACM) 48(4):798–859

Heckendorn RB, Rana S, Whitley D (1999) Polynomial time summary statistics for a generalization of MAXSAT. In: Proceedings of the 11th annual conference on genetic and evolutionary computation. Morgan Kaufmann, pp 281–288

Heninger NA (2011) Error correction and the cryptographic key. PhD thesis. Princeton University
Heras F, Larrosa J, Oliveras A (2008) MiniMaxSAT: an efficient weighted Max-SAT solver. J Artif Intell Res (JAIR) 31:1–32
Hoos HH, Smyth K, Stützle T (2004) Search space features underlying the performance of stochastic local search algorithms for MAXSAT. In: Proceedings of the 8th international conference on parallel problem solving from nature (PPSN VIII). Springer, pp 51–60
Hoos HH, Stützle T (2004) Stochastic local search: foundations and applications. Elsevier, New York
Kamal AA (2012) Cryptanalysis and secure implementation of modern cryptographic algorithms. PhD thesis. Concordia University
Kamal AA, Youssef AM (2010) Applications of SAT solvers to AES key recovery from decayed key schedule images. In: Proceedings of the 4th international conference on emerging security information, systems and technologies. IEEE, pp 216–220
Koshimura M, Zhang T, Fujita H, Hasegawa R (2012) QMaxSAT: a partial Max-SAT solver. J Satisf Boolean Model Comput 8(1–2):95–100
Le Berre D, Parrain A (2010) The Sat4j library, release 2.2. J Satisf Boolean Model Comput 7(2–3):59–64
Li CM, Manyà F (2009) MaxSAT, hard and soft constraints. In: Biere A, Heule MJH, van Maaren H, Walsh T (eds) Handbook of satisfiability, vol 185. IOS Press, Amsterdam, pp 613–631
Li CM, Manya F, Planes J (2007) New inference rules for Max-SAT. J Artif Intell Res 30:321–359
Li W, Xu C, Yang Y, Chen J, Wang J (2022) A refined branching algorithm for the maximum satisfiability problem. Algorithmica 84(4):982–1006
Liao X (2013) Maximum satisfiability approach to game theory and network security. PhD thesis. Kyushu University
Liao X, Zhang H, Koshimura M, Fujita H, Hasegawa R (2013) Using MaxSAT to correct errors in AES key schedule images. In: 2013 IEEE 25th international conference on tools with artificial intelligence. IEEE, pp 284–291
Lourenço HR, Martin OC, Stützle T (2019) Iterated local search: framework and applications. In Handbook of metaheuristics. Springer, pp 129–168
Luo C, Cai S, Wu W, Jie Z, Su K-W (2014) CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans Comput 64(7):1830–1843
Luo C, Cai S, Su K, Wu W (2014) Clause states based configuration checking in local search for satisfiability. IEEE Trans Cybern 45(5):1028–1041
Luo C, Cai S, Wu W, Su K (2013) Focused random walk with configuration checking and break minimum for satisfiability. In: International conference on principles and practice of constraint programming. Springer, pp 481–496
Malan KM, Engelbrecht AP (2013) A survey of techniques for characterising fitness landscapes and some possible ways forward. Inf Sci 241:148–163
Martin OC, Otto SW (1996) Combining simulated annealing with local search heuristics. Ann Oper Res 63(1):57–75
Mertens S, Mézard M, Zecchina R (2006) Threshold values of random K-SAT from the cavity method. Random Struct Algorithms 28(3):340–373
Merz P, Freisleben B (1999) Fitness landscapes and memetic algorithm design. In: Corne D, Dorigo M, Glover F, Dasgupta D, Moscato P, Poli R, Price KV (eds) New ideas in optimization. McGraw-Hill, London, pp 245–260
Mézard M, Parisi G, Zecchina R (2002) Analytic and algorithmic solution of random satisfiability problems. Science 297(5582):812–815
Mills P, Tsang E (2000) Guided local search for solving SAT and weighted MAX-SAT problems. J Autom Reason 24(12):205–223
Nadel A (2019) TT-Open-WBO-Inc: tuning polarity and variable selection for anytime SAT-based optimization. In: Proceedings of the MaxSAT evaluations
Narodytska N, Bacchus F (2014) Maximum Satisfiability using core-guided MaxSAT resolution. In Proceedings of the 28th AAAI conference on artificial intelligence. AAAI Press, pp 2717–2723
Niedermeier R, Rossmanith P (2000) New upper bounds for maximum satisfiability. J Algorithms 36(1):63–88
Pankratov D, Borodin A (2010) On the relative merits of simple local search methods for the MAX-SAT problem. In: Proceedings of the 13th international conference on theory and applications of satisfiability testing (SAT 2010). SAT'10. Springer, pp 223–236
Paxian T, Reimer S, Becker B (2018) Pacose: an iterative SAT-based MaxSAT solver. MaxSAT Eval 2018:20

Pipatsrisawat K, Darwiche A (2007) Clone: solving weighted Max-SAT in a reduced search space. In: Australasian joint conference on artificial intelligence. Springer, pp 223–233

Poloczek M (2011) Bounds on greedy algorithms for MAX SAT. In Proceedings of the 19th European symposium on algorithms. ESA'11. Springer, pp 37–48

Poloczek M, Williamson DP (2016) An experimental evaluation of fast approximation algorithms for the maximum satisfiability problem. In: International symposium on experimental algorithms. Springer, pp 246–261

Poloczek M, Schnitger G, Williamson DP, Van Zuylen A (2017) Greedy algorithms for the maximum satisfiability problem: simple algorithms and inapproximability bounds. SIAM J Comput 46(3):1029–1061

Prügel-Bennett A, Tayarani-Najaran M-H (2012) Maximum satisfiability: anatomy of the fitness landscape for a hard combinatorial optimization problem. IEEE Trans Evol Comput 16(3):319–338

Qasem M, Prügel-Bennett A (2010) Learning the large-scale structure of the MAX-SAT landscape using populations. IEEE Trans Evol Comput 14(4):518–529

Selman B, Kautz HA, Cohen B (1996) Local search strategies for satisfiability testing. In: DIMACS series in discrete mathematics and theoretical computer science, pp 521–532

Selman B, Kautz HA, Cohen B (1994) Noise strategies for improving local search. In: Proceedings of the 12th national conference on artificial intelligence, vol 1. American Association for Artificial Intelligence, pp 337–343

Selman B, Levesque H, Mitchell D (1992) A new method for solving hard satisfiability problems. In: Proceedings of the 10th national conference on artificial intelligence. AAAI Press, pp 440–446

Smyth K, Hoos HH, Stützle T (2003) Iterated robust tabu search for MAX-SAT. In: Conference of the Canadian society for computational studies of intelligence. Springer, pp 129–144

Stadler P (2002) Fitness landscapes. In: Lässig M, Valleriani A (eds) Biological evolution and statistical physics. Springer, New York, pp 183–204

Sun Grid Engine (SGE) QuickStart (2020) http://star.mit.edu/cluster/docs/0.93.3/guides/sge.html

Sutton AM, Whitley LD, Howe AE (2009) A polynomial time computation of the exact correlation structure of k-satisfiability landscapes. In: proceedings of the 11th annual conference on genetic and evolutionary computation. ACM, pp 365–372

Tompkins DA, Hoos HH (2005) UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAXSAT. In: Proceedings of the 8th international conference on theory and applications of satisfiability testing. Springer, pp 306–320

Xiao M (2022) An exact MaxSAT algorithm: further observations and further improvements. Preprint, accepted to IJCAI

Yannakakis M (1994) On the approximation of maximum satisfiability. J Algorithms 17(3):475–502