Probability-Based Characterization and Algorithms for the Maximum Satisfiability Problem

Dissertation submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Yochai Twitto

Submitted to the Senate of Ben-Gurion University
of the Negev

May 2020
Beer-Sheva

Probability-Based Characterization and Algorithms for the Maximum Satisfiability Problem

Dissertation submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Yochai Twitto

Submitted to the Senate of Ben-Gurion University
of the Negev

Approved by the advisor ________________________
Approved by the Dean of the Kreitman School of Advanced Graduate
Studies ________________________

May 2020
Beer-Sheva

This work was carried out under the supervision of Prof. Daniel Berend
and Prof. Aryeh Kontorovich
Department of Computer Science
Faculty of Natural Sciences

I Yochai Twitto, whose signature appears below, hereby declare that (Please mark the appropriate statements):

✓ I have written this Thesis by myself, except for the help and guidance offered by my Thesis Advisors.

✓ The scientific materials included in this Thesis are products of my own research, culled from the period during which I was a research student.

__ This Thesis incorporates research materials produced in cooperation with others, excluding the technical help commonly received during experimental work. Therefore, I am attaching another affidavit stating the contributions made by myself and the other participants in this research, which has been approved by them and submitted with their approval.

Date: May 2020 Student's name: Yochai Twitto Signature:__________

Contents

List of Figures

List of Tables

Abstract

In the Maximum Satisfiability (Max Sat) problem, we are given a sequence of clauses over some boolean variables. Each clause is a disjunction of literals (a variable or its negation) over different variables. We seek 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. We restrict our attention mainly to instances for which the clauses consist of exactly r literals each. This restricted problem is also known as Max Er-Sat.

In this work we provide a probabilistic characterization of the random Max r-Sat problem. We study the variance of the number of clauses satisfied by a random assignment, and the covariance of the numbers of clauses satisfied by a random pair of assignments of an arbitrary distance. We show that the correlation between the numbers of clauses satisfied by a random pair of assignments of distance $d = cn$, $0 \leq c \leq 1$, converges in probability to $((1 - c)^r - 1/2^r)/(1 - 1/2^r)$. We also show that the so-called normalized autocorrelation length of random Max r-Sat converges in probability to $(1 - 1/2^r)/r$.

We explore the correlation between the quality of initial assignments provided to local search heuristics and that of the corresponding final assignments. We show that the correlation in question 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. We use this insight to improve a state-of-the-art local search solver.

We present and study a new algorithm for the Maximum Satisfiability (Max Sat) problem. The algorithm is based on the Method of Conditional Expectations (MOCE), and applies an efficient greedy variable ordering to MOCE. We call our algorithm Efficient Exhaustive Method of Conditional

Expectations (EEMOCE) as its greediness efficiently exhausts all unassigned variables at each step.

We conduct a comprehensive empirical evaluation of EEMOCE and show that it performs much better than MOCE, while keeping the additional runtime overhead relatively low. In particular, EEMOCE reduces the number of unsatisfied clauses by tens of percents, while the time complexity increases only by a logarithmic factor. The actual runtime is typically up to 3 times longer even for very large instances. We also point out how to eliminate the logarithmic factor added to the time complexity, in practical usages.

To conclude, we introduce general techniques for awarding combinatorial dominance certificates to arbitrary solutions of various optimization problems. We demonstrate the techniques by applying them to the Maximum Satisfiability and Traveling Salesman problems, and experiment their usability.

Chapter 1

Introduction

In the Maximum Satisfiability (Max Sat) problem, we are given a sequence of clauses over some boolean variables. Each clause is a disjunction of literals (a variable or its negation) over different variables. We seek 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. We restrict our attention mainly to instances for which the clauses consist of exactly r literals each. This restricted problem is also known as Max Er-Sat.

Let n be the number of variables. Denote the variables by $v_1, v_2, \ldots, v_n$. The number of clauses is denoted by m, and the clauses by $C_1, C_2, \ldots, C_m$. We denote the clause-to-variable ratio by $\alpha = m/n$, to which we also refer as density. We use the terms “positive variable” and “negative variable” to refer to a variable and to its negation, respectively. Whenever we find it convenient, we consider the truth values `true` and `false` as binary 1 and 0, respectively.

As Max r-Sat (for $r \geq 2$) is NP-hard [11, pp. 455–456], large-sized instances cannot be exactly solved in an efficient manner (unless $P = NP$), and one must resort to approximation algorithms and heuristics. Numerous methods have been suggested for solving Max r-Sat, e.g. [17, 99, 97, 69, 24, 78, 8, 32, 56], and an annual competition of solvers has been held since 2006 [10].

Satisfiability related questions attracted a lot of attention from the scientific community. As an example, one may consider the well-studied satisfiability threshold density question [27, 37, 2, 73, 28, 34]. For a comprehensive overview of the whole domain of satisfiability we refer to [16].

1.1 Probabilistic quantities

Using Walsh analysis [42], an efficient way of calculating moments of the number of satisfied clauses of a given instance of Max r-Sat was suggested in [54]. 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.

An interesting study of Max 3-Sat was provided in [85]. 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 [99] and WSAT [97].

A slightly better version of this novel heuristic, based on clustering instead of averaging, is provided in another paper [89] of the same authors. This version turned out to outperform all the heuristics implemented by that time in the Sat solver framework UBCSAT [104].

In [58], the authors analyze how the way random instances are generated 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. In [5], the autocorrelation coefficient of several problems was calculated, and problem hardness was classified accordingly.

Elaboration on correlations and on the way of harnessing them to design well-performing local search heuristics and memetic algorithms is provided in [74]. 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.

In [103], it is shown how to use the Walsh decomposition [42] 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 above approximation is based on mean-field approximation [106] with some presumed assumption on the statistical fluctuation of the approximated quantity. Formulas for these expectations are provided only in terms of Walsh coefficients, and thus give less insight as to their actual values.

The autocorrelation length, which is closely related to the ruggedness of landscapes, is of interest in the area of landscape analysis [72, 103, 5, 58, 36, 6, 25]. It is fundamental to the theory and design of local search heuristics [26, 74]. According to the autocorrelation length conjecture [101], in many landscapes, the number of local optima can be estimated using an expression based on this quantity.

1.2 Local search

Local search heuristics [59] explore the assignment space. They usually start from 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) [69] 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.

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 leads to the maximum number of satisfied clauses. Ties are broken randomly.

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

In CCLS, a variable is a *configuration changed variable* if at least one of its neighboring variables has been flipped, since its most recent flip. Here, the neighbors of a variable are those variables appearing together with it in at least one clause.

Works related to local search, configuration checking, CCLS, and algorithms of the same spirit, include [79, 22, 20, 70, 71, 1, 18, 19, 21, 98, 76, 100, 23].

1.3 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 Method of Conditional Expectations (MOCE) [35, 107], 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 [53]. We 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, one should 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 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 is independent of the number of variables or clauses in the instance. Thus, a step takes a constant time on the average. Note, though, that this requires us to continuously track all clauses containing each variable. This is in addition to the map between clauses and their variables.

Theoretical and empirical works related to MOCE, and algorithms of the same spirit, include [29, 82, 84, 83, 30].

1.4 Dominance analysis

While approximation ratio analysis does give some information on heuristics, it does not provide the whole picture regarding their performance in practice. Algorithmic solutions used in practice are often some form of local improvement heuristic, based on techniques such as Simulated Annealing [62], HC [94], GRASP [81], Tabu Search [39], or Genetic Algorithms [57, 77]. Properly implemented, these techniques may lead to short, efficient programs which yield reasonable solutions. However, these heuristics often come with no theoretical guarantee as to the quality of the provided solution.

An $f(I)$ *combinatorial dominance guarantee* is a certificate that a solution is not worse than at least $f(I)$ solutions for a particular problem instance I. The intuition behind this performance measure rests on the letter of recommendation one could write on behalf of a given person, or heuristic solution. A recommendation like *"She is the best of the 75 students in my class this year"* is analogous to a combinatorial dominance guarantee. It certifies the candidate as superior to a certain number of members of a given pool, with the implied assumption that this says something meaningful about the candidate's global ranking as well. The larger the number of competitors dominated by the candidate, the stronger the recommendation.

The issue of measuring the quality of approximate solutions has been addressed in [108]. A formulation of the very basic properties expected from a function measuring the quality of approximate solutions was given, and the notion of a *proper* quality measure stated accordingly. The author suggested considering some measures, such as *z-approximation* [52] and *location ratio*, which is more familiar recently as *dominance ratio* [44, 3]. Both of these measures are proper.

The latter measure has been studied primarily within the operations research community. The basic notion appears to have been independently discovered several times. The primary focus has been on algorithms for TSP, specifically designing polynomial-time algorithms which dominate exponentially large neighborhoods. The first TSP heuristic with an exponential dominance number is presented in [93] (see also [95, 96]).

The question whether there exists a polynomial-time algorithm which yields a solution dominating $(n-1)!/p(n)$ tours, where $p(n)$ is polynomial, appears to have first been raised in [40]. Dominance bounds for TSP have been most aggressively pursued by Gutin, Yeo, and Zverovich in a series of papers (cf. [45, 46]), culminating in a polynomial-time algorithm which finds a solution dominating $\Theta((n-1)!)$ tours. These bounds follow by applying certain Hamiltonian cycle decomposition theorems to the complete graph. We refer to [47] for more information.

In [33], the authors survey the complexity of optimizing TSP over several well-defined but exponentially large neighborhoods. Such optima by definition have large dominance numbers. In [12], the authors perform an experimental study of certain linear-time dynamic programming algorithms for TSP, which dominate exponentially many solutions.

Gutin, Vainshtein, and Yeo [44] appear to have been the first to consider the complexity of achieving a given dominance bound. In particular, they define complexity classes of DOM-easy and DOM-hard problems. They prove that weighted Max k-Sat and Max Cut are DOM-easy while (unless P=NP) Vertex Cover and Clique are DOM-hard.

Alon, Gutin, and Krivelevich [3] provide several algorithms which achieve large dominance *ratios* for versions of Integer Partition, Max Cut, and Max r-Sat. These algorithms share a common property — they provide solutions of quality guaranteed to be not worse than the average solution value. This property has been used also in other dominance proofs [87, 44, 45, 86, 63]. In [105], the author showed that this property by itself does not necessarily ensure good dominance.

Other works on dominance analysis include [48, 87], where it is proved that the nearest neighbor, minimum spanning tree, and greedy heuristics perform extremely poorly for symmetric and asymmetric TSP. Various combinatorial optimization problems and classical heuristics for them have been analyzed in [14, 13, 43]. In [80], a model for analyzing heuristic search algorithms (such as simulated annealing and backtracking), based on the ideas of combinatorial dominance, has been developed.

In [64], the authors studied a polynomial-time algorithm for ATSP, and showed that it provides a dominance ratio of at least $1/2 - o(1)$. In [65], they gave a polynomial-time algorithm with dominance ratio of $1 - n^{-1/29}$ for a special case of TSP in which the edges may take only two possible weights.

In [88], the authors analyzed the BBQP problem with $m + n$ variables. They proved that any solution for this problem, with quality no worse than the average, dominates at least 2^{m+n-2} solutions, and that this bound is the best possible. They provided an $O(mn)$ algorithm to identify such a solution.

1.5 Overview

In Chapter 2, we provide a probabilistic characterization of the random Max r-Sat problem. We study the variance of the number of clauses satisfied by a random assignment, and the covariance of the numbers of clauses satisfied by a random pair of assignments of an arbitrary distance. Closed-form formulas for the expected value and the variance of these quantities are provided. We asymptotically and probabilistically analyze these formulas and use them to gain insights on the similarity of instances.

Based on the above probabilistic characterization, we show that the correlation between the numbers of clauses satisfied by a random pair of assignments of distance $d = cn$, $0 \leq c \leq 1$, converges in probability to $((1-c)^r - 1/2^r)/(1-1/2^r)$. Our main result is that the so-called normalized autocorrelation length of Max r-Sat converges in probability to $(1-1/2^r)/r$. The latter quantity is of interest in the area of landscape analysis as a way to better understand problems and assess their hardness for local search heuristics. A former result regarding the same quantity only expressed it in terms of Walsh coefficients. All the results in this chapter apply to random r-Sat as well.

In Chapter 3, we explore the correlation between the quality of initial assignments provided to local search heuristics and that of the corresponding final assignments. We restrict our attention to the Max r-Sat problem and to one of the leading local search heuristics – Configuration Checking Local Search (CCLS). We use a tailored version of the Method of Conditional Expectations (MOCE) to generate initial assignments of diverse quality.

We show that the correlation in question is significant and long-lasting. Namely, even when we delve deeper into the local search, we are still in the shadow of the initial assignment. Thus, under practical time constraints, the quality of the initial assignment is crucial to the performance of local search heuristics.

To demonstrate our point, we improve CCLS by combining it with MOCE. Instead of starting CCLS from random initial assignments, we start it from excellent initial assignments, provided by MOCE. Indeed, it turns out that this kind of initialization provides a significant improvement of this state-of-the-art solver. This improvement becomes more and more significant as the instance grows.

In Chapter 4, we present and study a new algorithm for the Maximum Satisfiability (Max Sat) problem. The algorithm is based on the Method of Conditional Expectations (MOCE), and applies an efficient greedy variable ordering to MOCE. We call our algorithm Efficient Exhaustive Method of Conditional Expectations (EEMOCE) as its greediness efficiently exhausts all unassigned variables at each step.

We conduct a comprehensive empirical evaluation of EEMOCE and show that it performs much better than MOCE, while keeping the additional runtime overhead relatively low. In particular, EEMOCE reduces the number of unsatisfied clauses by tens of percents, while the time complexity increases only by a logarithmic factor. The actual runtime is typically up to 3 times longer even for very large instances.

We empirically study the main quantities managed by EEMOCE during its execution, exposing a relatively large residual randomality that may be harnessed for further improvement of the performance. Based on this study, we also point out how to eliminate the logarithmic factor added to the time complexity, in practical usages.

In Chapter 5, we introduce simple but general techniques for awarding combinatorial dominance certificates to arbitrary solutions of various optimization problems. We demonstrate the techniques we introduce by applying them to the Traveling Salesman and Maximum Satisfiability problems, and briefly experiment their usability.

A brief conclusion is provided in Chapter 6. While we find it best reading this work chapter-by-chapter according to their order, it is not a must. In fact, all the result chapters of this work, namely, chapters 2–5, are self-contained. Each of them may be read on its own, without reading any of the other results chapters.

Chapter 2

Probabilistic Characterization of Random Max r-Sat

In this chapter we provide a probabilistic characterization of the random Max r-Sat problem. We study the variance of the number of clauses satisfied by a random assignment, and the covariance of the numbers of clauses satisfied by a random pair of assignments of an arbitrary distance. Closed-form formulas for the expected value and the variance of these quantities are provided. We asymptotically and probabilistically analyze these formulas and use them to gain insights on the similarity of instances.

Based on the above probabilistic characterization, we show that the correlation between the numbers of clauses satisfied by a random pair of assignments of distance $d = cn$, $0 \leq c \leq 1$, converges in probability to $((1-c)^r - 1/2^r)/(1-1/2^r)$. Our main result is that the so-called normalized autocorrelation length of Max r-Sat converges in probability to $(1-1/2^r)/r$. The latter quantity is of interest in the area of landscape analysis as a way to better understand problems and assess their hardness for local search heuristics. A former result regarding the same quantity only expressed it in terms of Walsh coefficients. All our results apply to random r-Sat as well.

2.1 Introduction

In the Maximum Satisfiability (Max Sat) problem, we are given a sequence of clauses over some boolean variables. Each clause is a disjunction of literals (a variable or its negation) over different variables. We seek 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. Here we restrict our attention to instances for which the clauses consist of exactly r literals each. This restricted problem is also known as Max Er-Sat.

Let n be the number of variables. Denote the variables by $v_1, v_2, \ldots, v_n$. The number of clauses is denoted by m, and the clauses by $C_1, C_2, \ldots, C_m$. We use the terms "positive variable" and "negative variable" to refer to a variable and to its negation, respectively. Whenever we find it convenient, we consider the truth values `true` and `false` as binary 1 and 0, respectively.

As Max r-Sat (for $r \geq 2$) is NP-hard [11, pp. 455–456], large-sized instances cannot be exactly solved in an efficient manner (unless $P = NP$), and one must resort to approximation algorithms and heuristics. The simple randomized approximation algorithm, which assigns a truth value to each variable independently and uniformly at random, satisfies $1 - 1/2^r$ of all clauses on the average. Furthermore, this simple algorithm can also be easily derandomized using the Method of Conditional Expectations [35], yielding an assignment that is guaranteed to satisfy at least this number 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 [53].

Using Walsh analysis [42], an efficient way of calculating moments of the number of satisfied clauses of a given instance of Max r-Sat was suggested in [54]. 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. We provide closed-form formula, asymptotics, and convergence proof for the variance.

An interesting study of Max 3-Sat is provided in [85]. 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 [99] and WSAT [97]. A slightly better version of this heuristic, based on clustering instead of averaging, is provided in another paper [89] of the same authors. This version turned out to outperform all the heuristics implemented at that time in the Sat solver framework UBCSAT [104]. Our convergence in probability proofs mathematically validate their simulative results regarding similarity for several statistical properties, including the long-range correlation they used for their heuristics.

In [58], the authors analyze how the way random instances are generated 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. In [5], the autocorrelation coefficient of several problems was calculated, and problem hardness was classified accordingly. We contribute one more result for this classification.

Elaboration on correlations and on the way of harnessing them to designing well-performing local search heuristics and memetic algorithms is provided in [74]. 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.

In [103], it is shown how to use the Walsh decomposition [42] 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 approximation is based on mean-field approximation [106] with some presumed assumption on the statistical fluctuation of the approximated quantity. Formulas for these expectations are provided only in terms of Walsh coefficients, and thus give less insight as to their actual values. We substantially improve the result regarding the autocorrelation length, by showing its normalized version converges in probability to an explicit constant.

Numerous methods have been suggested for solving Max r-Sat, e.g. [17, 99, 97, 69, 24, 78, 8, 32, 56], and an annual competition of solvers has been held since 2006 [10]. Satisfiability related questions attracted a lot of attention from the scientific community. As an example, one may consider the well-studied satisfiability threshold question [27, 37, 2, 73, 28, 34]. For a comprehensive overview of the whole domain of satisfiability we refer to [16].

This chapter deals with the variance of the number of clauses satisfied by a random assignment, and the covariance of the numbers of clauses satisfied by a random pair of assignments at an arbitrary distance. We obtain explicit formulas for the expected value and the variance of these quantities. Asymptotics of these expressions are provided as well. From the asymptotics we conclude that the variance of the number of clauses satisfied by a random assignment is usually quite close to the expected value of this variance.

Based on the above probabilistic characterization, we show that the correlation between the numbers of clauses satisfied by a random pair of assignments of distance $d = cn$, $0 \leq c \leq 1$, converges in probability to $((1-c)^r - 1/2^r)/(1 - 1/2^r)$. Our main result is that the so-called normalized autocorrelation length [38] of Max r-Sat converges in probability to $(1 - 1/2^r)/r$.

The latter quantity, which is closely related to the ruggedness of landscapes, is of interest in the area of landscape analysis [72, 103, 5, 58, 36, 6, 25]. It is fundamental to the theory and design of local search heuristics [26, 74]. According to the autocorrelation length conjecture [101], in many landscapes, the number of local optima can be estimated using an expression based on this quantity. Our result reveals the normalized autocorrelation length of Max r-Sat, improving a former result [103] expressed it only in terms of Walsh coefficients [42].

In Section 2.2 we present our main results, and in Section 2.3 the proofs. Some elaboration and discussion are provided in Section 2.4. All our results immediately apply to random r-Sat, as both random Max r-Sat and random r-Sat deal with the same collection of random instances – the collection of random r-CNF formulas. We choose to present our results in the context of Max r-Sat, and to omit the prefix "random", assuming this is the default when not mentioned otherwise.

2.2 Main results

Throughout the chapter we deal with three basic probability spaces. The first consists of all instances with m clauses of length r over n variables. As any r of the variables may appear in a clause, and each may be positive of negative, the number of instances is $\left(\binom{n}{r}2^r\right)^m$. All instances are equally likely, namely each has a probability of $1/\left(\binom{n}{r}2^r\right)^m$. The second probability space consists of all 2^n equally likely truth assignments. The third consists of all $2^n\binom{n}{d}$ equally likely pairs of truth assignments of distance d. The distance between two assignments is the Hamming distance, i.e., the number of variables they assign differently.

We use the subscripts $\mathcal{I}$, A, and d to specify that a certain quantity is associated with the first, second, or third probability space, respectively. We use I, a, and (a, b) to denote a random instance, a random assignment, and a random pair of assignments of distance d, respectively. Let the random variable $S(I, a)$ ($R(I, a)$, resp.) be the number of clauses of I satisfied (unsatisfied, resp.) by the assignment a.

For a given instance I, let $\rho(d) = \mathrm{Corr}_d(S(I, a), S(I, b))$ be the correlation (coefficient) between the numbers of clauses satisfied by a random pair of assignments at distance d from each other. The autocorrelation length [38], given by $l = -1/\ln(|\rho(1)|)$, is a one-number summary of the ruggedness of the landscape of the instance. The higher its value, the smoother is the landscape. The *normalized* autocorrelation length is simply l/n.

A slightly different quantity for summarizing ruggedness is the autocorrelation coefficient [4], defined by $\xi = 1/(1 - \rho(1))$. Similarly, the *normalized* autocorrelation coefficient is ξ/n. These two measures, l/n and ξ/n, are asymptotically the same. I.e., their quotient approaches 1 as n grows larger. We arbitrarily choose to work with the latter. For convenience, these quantities are summarized in Table 2.1.

Before stating our results, it worth mentioning the notion of convergence in probability, which we heavily use below. A sequence $(X_n)_{n=1}^{\infty}$ of random variables *converges in probability* to a constant c if for all $\varepsilon > 0$:

$$P(|X_n - c| \geq \varepsilon) \xrightarrow[n\to\infty]{} 0.$$

Quantity	**Notation**	**Defined by**
Correlation Coefficient	$\boldsymbol{\rho(d)}$	$\mathrm{Corr}_d(S(I,a),S(I,b))$
Autocorrelation Length	$\boldsymbol{\ell}$	$-1/\ln(\lvert\rho(1)\rvert)$
Autocorrelation Coefficient	$\boldsymbol{\xi}$	$1/(1-\rho(1))$

Table 2.1: Some quantities and notations.

Such convergence is denoted by:

$$X_n \xrightarrow[n\to\infty]{\mathcal{P}} c.$$

For Max r-Sat, the normalized autocorrelation length converges in probability to a constant. This constant is independent of the clause-to-variable ratio (a.k.a. density) $\alpha = m/n$, as stated in our main theorem, which improves the result of [103]. There, this quantity is provided only in terms of Walsh coefficients [42], along with a mean-field approximation [106]. In our results regarding convergence in probability, here and afterward, the random variables are always defined on $\mathcal{I}$. Namely, they are defined on the probability space consisting of all (equally likely) instances with m clauses of length r over n variables.

Theorem 1. *For Max r-Sat:*

$$\frac{\xi}{n} \xrightarrow[n\to\infty]{\mathcal{P}} \frac{1-1/2^r}{r}.$$

Notice that, for some instances, ξ/n is not well defined. As an example, one may consider instances that are composed of bunches of clauses, where each bunch consists of all 2^r possible clauses over some r specific variables. Figure 2.1 depicts the normalized autocorrelation length for common values of r.

To prove Theorem 1, we will first prove Theorems 2 and 3. Besides being building blocks for the proof of the main theorem, each of these is of independent interest.

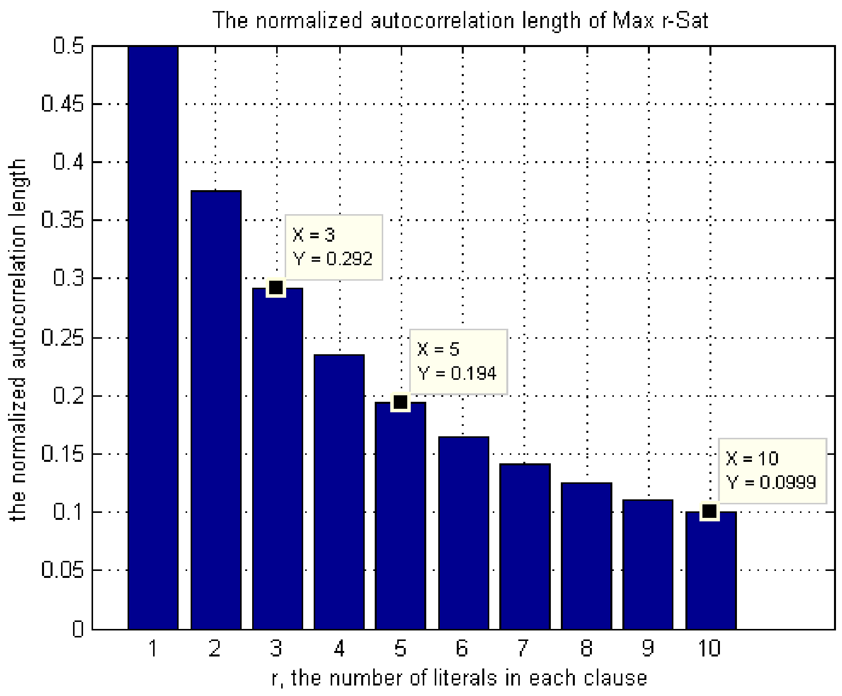


Figure 2.1: The normalized autocorrelation length.

For Max r-Sat, the expected number of clauses satisfied by a random assignment is $m(1-1/2^r)$, regardless of the instance. Thus, the calculation of the expected value and the variance of this quantity, over the ensemble of all instances, is trivial. The following theorem summarizes our results regarding the variance of the number of clauses satisfied by a random assignment. In the provided asymptotics, we assume that $n \to \infty$, $m = \alpha n$ for some constant $\alpha > 0$, and that r is constant.

Theorem 2. *For Max r-Sat, the expected value and the variance (over all instances) of the variance of the number of clauses satisfied by a random assignment are given by:*

$$E_{\mathcal{I}}(V_A(S(I,a))) = \frac{m}{2^r}\left(1-\frac{1}{2^r}\right), \tag{2.A}$$

$$V_{\mathcal{I}}(V_A(S(I,a))) = \frac{2m(m-1)}{2^{4r}}\left(\sum_{t=0}^{r}\frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}}\cdot 2^t - 1\right) \tag{2.B.1}$$

$$= \frac{\alpha^2 r^2}{2^{4r-1}}n + O(1). \tag{2.B.2}$$

In particular,

$$\frac{V_A(S(I,a))}{n} \xrightarrow[n\to\infty]{\mathcal{P}} \frac{\alpha}{2^r}\left(1-\frac{1}{2^r}\right). \tag{2.C}$$

Let $\varepsilon > 0$, and consider the proportion of instances for which the absolute difference between $V_A(S(I,a))/n$ and $\alpha(1-1/2^r)/2^r$ exceeds ε. By (2.C), this proportion tends to 0 as n grows. Nonetheless, one can easily construct such instances for arbitrarily large n. As an example, one may consider instances for which all clauses are the same. For such instances, $V_A(S(I,a))/n$ is asymptotically $n\cdot\alpha^2(1-1/2^r)/2^r$, namely much larger than the limit given in (2.C). In the other direction, instances composed of bunches (as mentioned right after Theorem 1) have $V_A(S(I,a))/n = 0$.

The next theorem generalizes the results further, and summarizes our results regarding the covariance of the numbers of clauses satisfied by a random pair of assignments at an arbitrary distance from each other. Here, in the asymptotics we also assume that $d = cn$, $0 \le c \le 1$.

Theorem 3. *For Max r-Sat, the expected value and the variance (over all instances) of the covariance of the numbers of clauses satisfied by a random pair of assignments of distance d are given by:*

$$E_{\mathcal{I}}(\mathrm{Cov}_d(S(I,a),S(I,b))) = \frac{m}{2^r}\left(\frac{\binom{n-r}{d}}{\binom{n}{d}} - \frac{1}{2^r}\right) \tag{3.A.1}$$

$$= \frac{\alpha}{2^r}\left((1-c)^r - \frac{1}{2^r}\right)n + O(1), \tag{3.A.2}$$

$$V_{\mathcal{I}}(\mathrm{Cov}_d(S(I,a),S(I,b)))$$

$$= \frac{2m(m-1)}{2^{4r}}\left(\sum_{t=0}^{r}\left(\frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}}\cdot 2^t\cdot\sum_{s=0}^{t}\frac{\binom{t}{s}\binom{n-t}{d-s}}{\binom{n}{d}}\cdot\frac{\binom{n-t}{d-s}}{\binom{n}{d}}\right)-1\right) \tag{3.B.1}$$

$$= \frac{\alpha^2 r^2(2c-1)^2}{2^{4r-1}}n + O(1). \tag{3.B.2}$$

In particular,

$$\frac{\mathrm{Cov}_d(S(I,a),S(I,b))}{n} \xrightarrow[n\to\infty]{\mathcal{P}} \frac{\alpha}{2^r}\left((1-c)^r - \frac{1}{2^r}\right). \tag{3.C}$$

As one may see, pairs of assignments of relatively small distance, $c < 1/2$,

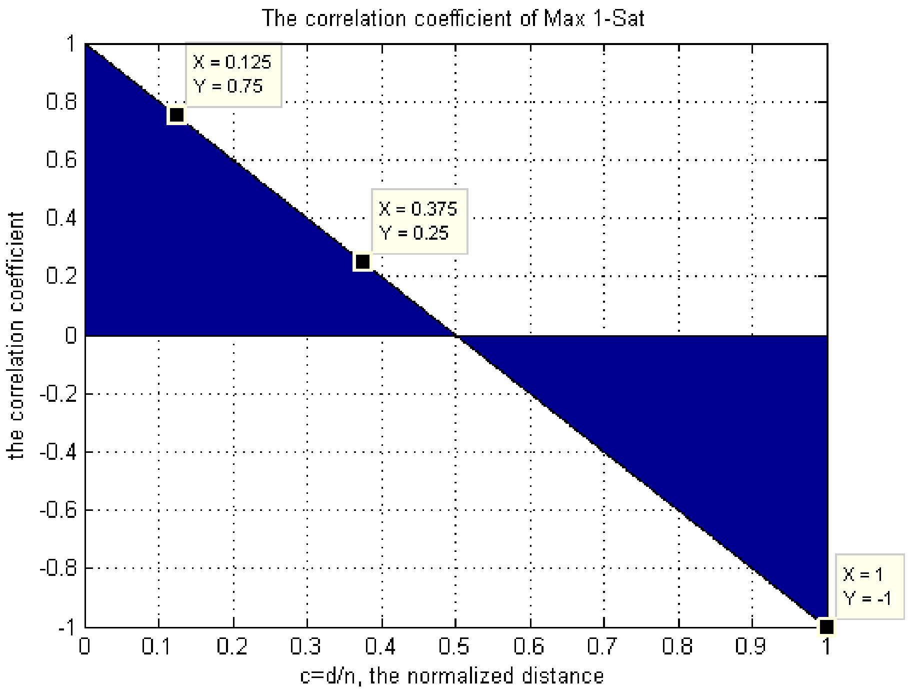


Figure 2.2: The correlation coefficient of Max 1-Sat.

tend to be positively correlated. On the other hand, if the distance is relatively large, $c > 1/2$, the assignments tend to be negatively correlated. In case $c = 1/2$, the assignments become almost uncorrelated, as one assignment may be roughly viewed as obtained from the other by flipping each variable's truth value randomly, so that the assignments are almost independent.

Finally, notice that (2.C) and (3.C) together lead immediately to the following corollary regarding the convergence in probability of the correlation of the numbers of clauses satisfied by a random pair of assignments at an arbitrary distance from each other.

Corollary 1. *For Max r-Sat:*

$$\rho(d) \xrightarrow[n\to\infty]{\mathcal{P}} \frac{(1-c)^r - 1/2^r}{1-1/2^r}.$$

Figures 2.2, 2.3, and 2.4 depict this correlation coefficient for Max 1-Sat, Max 3-Sat, and Max 5-Sat, respectively.

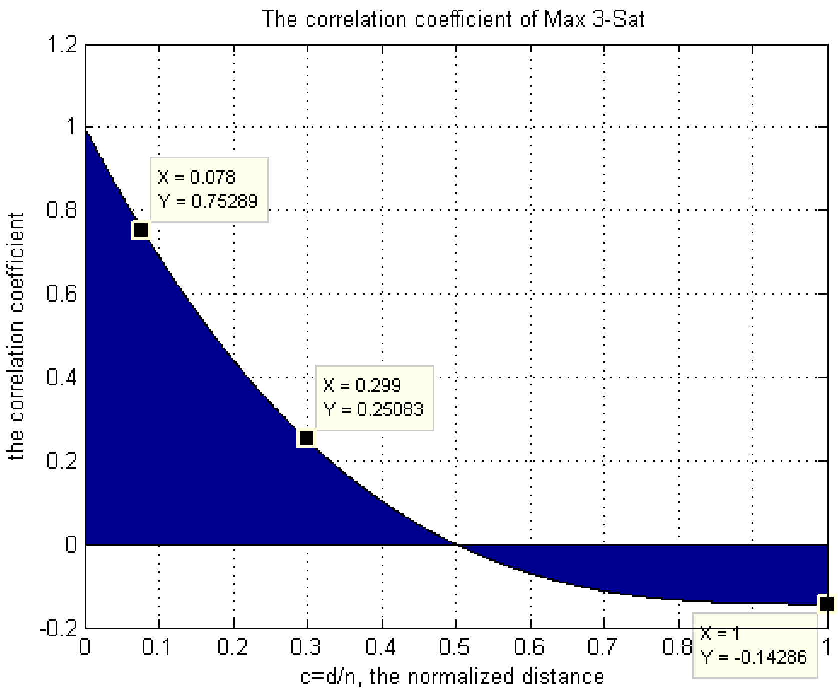


Figure 2.3: The correlation coefficient of Max 3-Sat.

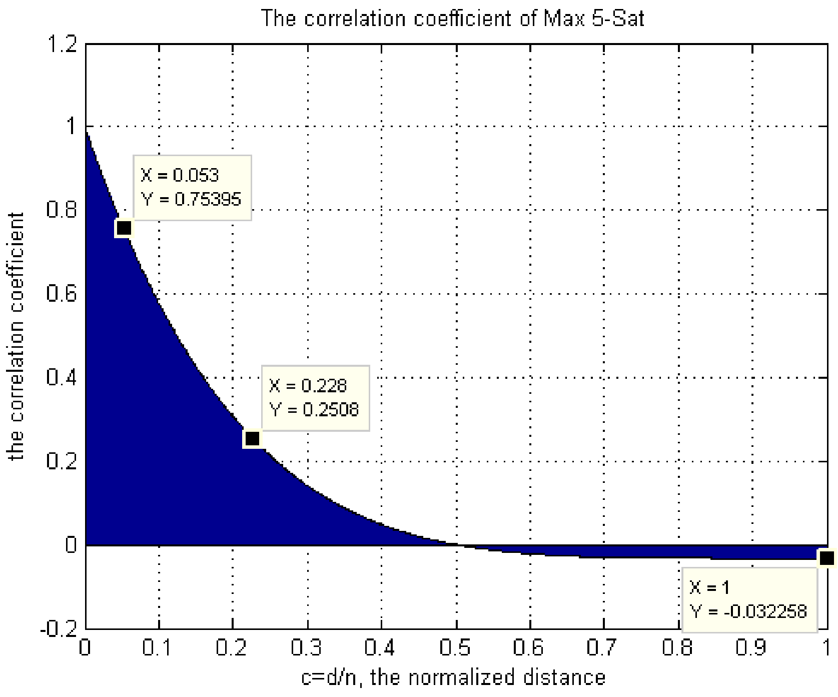


Figure 2.4: The correlation coefficient of Max 5-Sat.

2.3 Proofs

Theorem 2 follows immediately from Theorem 3, by applying the latter with $d = 0$. Yet, it will be instructive to provide an independent short proof of (2.A). Afterward, we provide the proof of Theorem 3. Theorem 1, which relies heavily on the former theorems, is proved at the end.

Proof of (2.A). The expected number of clauses satisfied by a random assignment is the same for all instances. Thus, in our case, the law of total variance [92, pp. 347–349] reduces to:

$$V_{(\mathcal{I},A)}(S(I,a)) = E_{\mathcal{I}}(V_A(S(I,a))).$$

Observing that the random variable $S(I,a)$ on the left-hand side is binomially distributed, $B(m, 1 - 1/2^r)$, we obtain the theorem. □

Proof of Theorem 3. As $R(I,a) = m - S(I,a)$, we may work with the covariance of the numbers of *unsatisfied* clauses, instead of that of the numbers of satisfied clauses. Define the following random variable:

$$R_i(I,a) = \begin{cases} 1, & \text{the assignment } a \text{ does not satisfy the clause } C_i, \\ 0, & \text{otherwise.} \end{cases}$$

We start with a single clause. For the sake of readability, we write $R_i(a)$ instead of $R_i(I,a)$.

$$\begin{aligned} &\mathrm{Cov}_d(R_i(a), R_i(b)) \\ &\quad = E_d(R_i(a) \cdot R_i(b)) - E_d(R_i(a)) \cdot E_d(R_i(b)) \\ &\quad = P_d(R_i(a) = R_i(b) = 1) - P_d(R_i(a) = 1) \cdot P_d(R_i(b) = 1) \\ &\quad = P_d(R_i(a) = 1) \cdot P_d(R_i(b) = 1 \mid R_i(a) = 1) - \frac{1}{2^{2r}} \\ &\quad = \frac{1}{2^r}\left(\frac{\binom{n-r}{d}}{\binom{n}{d}} - \frac{1}{2^r}\right). \end{aligned}$$

Next, we calculate the covariance for any specific instance. Again, we use the shorthand $R(a)$ instead of $R(I, a)$.

$$\begin{aligned}
\mathrm{Cov}_d(R(a), R(b)) &= \mathrm{Cov}_d\left(\sum_{i=1}^{m} R_i(a), \sum_{j=1}^{m} R_j(b)\right) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} \mathrm{Cov}_d(R_i(a), R_j(b)) \\
&= \sum_{i=1}^{m} \mathrm{Cov}_d(R_i(a), R_i(b)) + 2\sum_{1\le i<j\le m} \mathrm{Cov}_d(R_i(a), R_j(b)) \\
&= \frac{m}{2^r}\left(\frac{\binom{n-r}{d}}{\binom{n}{d}} - \frac{1}{2^r}\right) + 2\sum_{1\le i<j\le m} \mathrm{Cov}_d(R_i(a), R_j(b)).
\end{aligned} \tag{2.1}$$

For $1 \le i < j \le m$ we have:

$$\begin{aligned}
E_{\mathcal{I}}(\mathrm{Cov}_d(R_i(a), R_j(b))) &= E_{\mathcal{I}}(E_d(R_i(a)\cdot R_j(b))) \\
&\quad - E_{\mathcal{I}}(E_d(R_i(a))\cdot E_d(R_j(b))) \\
&= E_d(E_{\mathcal{I}}(R_i(a)\cdot R_j(b))) - \frac{1}{2^{2r}} \qquad (2.2)\\
&= E_d(E_{\mathcal{I}}(R_i(a))\cdot E_{\mathcal{I}}(R_j(b))) - \frac{1}{2^{2r}} \\
&= 0.
\end{aligned}$$

Here, the second last transition stems from the fact that, for a given pair of assignments a, b, the variables $R_i(a), R_j(b)$ are independent, as their associated clauses are selected uniformly at random and with repetitions from all possible clauses. Overall, the second addend on the right-hand side of (2.1) vanishes, which proves (3.A.1). The asymptotics provided in (3.A.2) is immediate.

Now, let us prove (3.B.1). We have:

$$\begin{aligned}
&V_{\mathcal{I}}(\mathrm{Cov}_d(R(a), R(b))) \\
&= V_{\mathcal{I}}\left(\frac{m}{2^r}\left(\frac{\binom{n-r}{d}}{\binom{n}{d}} - \frac{1}{2^r}\right) + 2\sum_{1\le i<j\le m}\mathrm{Cov}_d(R_i(a), R_j(b))\right) \\
&= 4\sum_{1\le i<j\le m} V_{\mathcal{I}}(\mathrm{Cov}_d(R_i(a), R_j(b))) \\
&\quad + 8\sum_{\substack{1\le i<j\le m\\ 1\le k<l\le m\\ (i<k)\vee(i=k\wedge j<l)}} \mathrm{Cov}_{\mathcal{I}}(\mathrm{Cov}_d(R_i(a), R_j(b)), \mathrm{Cov}_d(R_k(a), R_l(b))) \\
&= 4g + 8\sum_{\substack{1\le i<j\le m\\ 1\le k<l\le m\\ (i<k)\vee(i=k\wedge j<l)}} h_{ijkl} \\
&= 4g + 8h.
\end{aligned}$$

Next, we calculate g as follows:

$$\begin{aligned}
g &= \sum_{1\le i<j\le m} V_{\mathcal{I}}(\mathrm{Cov}_d(R_i(a), R_j(b))) \\
&= \sum_{1\le i<j\le m} \left(E_{\mathcal{I}}\left(\mathrm{Cov}_d^2\left(R_i(a), R_j(b)\right)\right) - E_{\mathcal{I}}^2\left(\mathrm{Cov}_d\left(R_i(a), R_j(b)\right)\right)\right) \\
&= \sum_{1\le i<j\le m} \left(E_{\mathcal{I}}\left(\left(E_d(R_i(a)\cdot R_j(b)) - E_d(R_i(a))\cdot E_d(R_j(b))\right)^2\right) - 0\right) \quad \text{// by (2.2)} \\
&= \sum_{1\le i<j\le m} E_{\mathcal{I}}\left(\left(E_d(R_i(a)\cdot R_j(b)) - \frac{1}{2^{2r}}\right)^2\right) \\
&= \sum_{1\le i<j\le m} E_{\mathcal{I}}\left(E_d^2(R_i(a)\cdot R_j(b)) - \frac{E_d(R_i(a)\cdot R_j(b))}{2^{2r-1}} + \frac{1}{2^{4r}}\right) \\
&= \sum_{1\le i<j\le m} \left(E_{\mathcal{I}}\left(E_d^2(R_i(a)\cdot R_j(b))\right) - \frac{E_{\mathcal{I}}(E_d(R_i(a)\cdot R_j(b)))}{2^{2r-1}} + \frac{1}{2^{4r}}\right) \\
&= \sum_{1\le i<j\le m} \left(E_{\mathcal{I}}\left(E_d^2(R_i(a)\cdot R_j(b))\right) - \frac{1}{2^{2r}2^{2r-1}} + \frac{1}{2^{4r}}\right) \quad \text{// as done in (2.2)} \\
&= \sum_{1\le i<j\le m} \left(E_{\mathcal{I}}\left(P_d^2(R_i(a) = R_j(b) = 1)\right) - \frac{1}{2^{4r}}\right).
\end{aligned}$$

To continue the calculation, we consider the clauses C_i and C_j. We denote by t the number of variables shared by C_i and C_j. Let s be the number of variables, out of the t common ones, whose sign (as literals) is different in the two clauses. The remaining $t-s$ shared variables have the same sign in the two clauses.

The probability that C_i and C_j share exactly t variables, from which exactly s are of different sign, is

$$\frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot \frac{\binom{t}{s}}{2^t}.$$

Given t and s, we have

$$P_d(R_i(a) = R_j(b) = 1)) = \frac{1}{2^r} \cdot \frac{\binom{n-t}{d-s}}{\binom{n}{d}} \cdot \frac{1}{2^{r-t}}.$$

Thus,

$$\begin{aligned}
&E_{\mathcal{I}}\left(P_d^2(R_i(a) = R_j(b) = 1)\right) \\
&\quad = \sum_{t=0}^{r}\sum_{s=0}^{t} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot \frac{\binom{t}{s}}{2^t} \cdot \left(\frac{2^t}{2^{2r}} \cdot \frac{\binom{n-t}{d-s}}{\binom{n}{d}}\right)^2 \\
&\quad = \frac{1}{2^{4r}} \sum_{t=0}^{r} \left(\frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot 2^t \cdot \sum_{s=0}^{t} \frac{\binom{t}{s}\binom{n-t}{d-s}}{\binom{n}{d}} \cdot \frac{\binom{n-t}{d-s}}{\binom{n}{d}} \right).
\end{aligned}$$

Plugging the last expression into the expression for g, we arrive at the final form of g:

$$g = \frac{m(m-1)}{2} \cdot \frac{1}{2^{4r}} \cdot \left(\sum_{t=0}^{r} \left(\frac{\binom{r}{t}\binom{n-r}{n-t}}{\binom{n}{r}} \cdot 2^t \cdot \sum_{s=0}^{t} \frac{\binom{t}{s}\binom{n-t}{d-s}}{\binom{n}{d}} \cdot \frac{\binom{n-t}{d-s}}{\binom{n}{d}} \right) - 1 \right).$$

The expression for $4g$ is exactly the right-hand side of (3.B.1), so to conclude the proof it suffices to show that $h = 0$. In fact, we will see that every single term h_{ijkl} in the sum appearing in the expression for h vanishes. Let us expand this term.

$$\begin{aligned}
h_{ijkl} &= \mathrm{Cov}_{\mathcal{I}}\left(\mathrm{Cov}_d\left(R_i(a), R_j(b)\right), \mathrm{Cov}_d\left(R_k(a), R_l(b)\right)\right) \\
&= \mathrm{Cov}_{\mathcal{I}}\left(E_d\left(R_i(a)\cdot R_j(b)\right) - \frac{1}{2^{2r}}, E_d\left(R_k(a)\cdot R_l(b)\right) - \frac{1}{2^{2r}}\right) \\
&= \mathrm{Cov}_{\mathcal{I}}\left(P_d(R_i(a) = R_j(b) = 1), P_d(R_k(a) = R_l(b) = 1)\right).
\end{aligned}$$

To prove that $h_{ijkl} = 0$, we will show that $P_d(R_i(a) = R_j(b) = 1)$ and $P_d(R_k(a) = R_l(b) = 1)$ are independent. Observe that the addends in the sum can be classified to two categories: addends for which $|\{i,j,k,l\}| = 4$, and those for which $|\{i,j,k,l\}| = 3$. Either way, we have

$$P_d(R_i(a) = R_j(b) = 1) = \frac{2^{T_1}}{2^{2r}} \cdot \frac{\binom{n-T_1}{d-S_1}}{\binom{n}{d}},$$
$$P_d(R_k(a) = R_l(b) = 1) = \frac{2^{T_2}}{2^{2r}} \cdot \frac{\binom{n-T_2}{d-S_2}}{\binom{n}{d}},$$

where T_1 is the number of variables shared by C_i and C_j, S_1 is the number of common variables whose sign (as literals) is different in C_i and C_j, and T_2 and S_2 are defined similarly with respect to C_k and C_l.

The variables T_1 and S_1 are determined solely by the way C_j is selected in relation to C_i. Similarly, T_2 and S_2 are determined solely by the way C_l is selected in relation to C_k. This holds even if, for example, $i = k$. Thus, the variables (T_1, S_1), and the variables (T_2, S_2), are independent. Consequently, $P_d(R_i(a) = R_j(b) = 1)$ and $P_d(R_k(a) = R_l(b) = 1)$ are independent as well. This means that $h_{ijkl} = 0$, which proves (3.B.1).

Regarding (3.B.2), suppose $0 < c < 1$. As $\binom{n-r}{r-t}/\binom{n}{r} = \Theta(1/n^t)$ and $\binom{n-t}{d-s}/\binom{n}{d} = \Theta(1)$, the addends in the sum appearing in (3.B.1) behave as $\Theta(1/n^t)$, for given values of t and s. The largest addends are obtained in the following three settings of t and s:

$$\begin{aligned}
t=0, s=0: &\quad \left(1-\frac{r}{n}\right)\left(1-\frac{r}{n-1}\right)\cdots\left(1-\frac{r}{n-r+1}\right) \\
&\quad = 1-\frac{r^2}{n}+O\left(\frac{1}{n^2}\right). \qquad (2.3) \\
t=1, s=0: &\quad \frac{2r^2(1-c)^2}{n-r+1}\left(1-\frac{r}{n}\right)\left(1-\frac{r}{n-1}\right)\cdots\left(1-\frac{r}{n-r+2}\right) \\
&\quad = \frac{2r^2(1-c)^2}{n}+O\left(\frac{1}{n^2}\right). \qquad (2.4) \\
t=1, s=1: &\quad \frac{2r^2c^2}{n-r+1}\left(1-\frac{r}{n}\right)\left(1-\frac{r}{n-1}\right)\cdots\left(1-\frac{r}{n-r+2}\right) \\
&\quad = \frac{2r^2c^2}{n}+O\left(\frac{1}{n^2}\right). \qquad (2.5)
\end{aligned}$$

For $c = 0$, the addends in the sum appearing in (3.B.1) behave as

$\Theta(1/n^{t+2s})$. The largest ones are obtained in the first and second settings above. Their values are given in (2.3) and (2.4), respectively. For $c = 1$, the addends in this sum behave as $\Theta(1/n^{3t-2s})$, the largest ones are obtained in the first and third settings of t and s, and their values are given in (2.3) and (2.5), respectively.

Summing up the leading addends in each case, we see that, regardless of the value of $0 \leq c \leq 1$, the sum appearing in (3.B.1) is

$$1 + \frac{r^2(2c-1)^2}{n} + O\left(\frac{1}{n^2}\right).$$

Plugging this approximation into (3.B.1), and using the fact that $m = \alpha n$, we arrive at (3.B.2).

Finally, to prove (3.C), denote:

$$X_n = \mathrm{Cov}_d(R(a), R(b))/n.$$

It suffices to show that the expected value of X_n converges to the right-hand side of (3.C), and that its variance converges to 0. These two convergences follows from (3.A.2) and (3.B.2) directly:

$$\begin{aligned}
E_{\mathcal{I}}(X_n) &= \frac{1}{n} \cdot E_{\mathcal{I}}(\mathrm{Cov}_d(R(a), R(b))) \\
&= \frac{\alpha}{2^r}\left((1-c)^r - \frac{1}{2^r}\right) + O\left(\frac{1}{n}\right) \\
&\xrightarrow[n\to\infty]{} \frac{\alpha}{2^r}\left((1-c)^r - \frac{1}{2^r}\right), \\
V_{\mathcal{I}}(X_n) &= \frac{1}{n^2} \cdot V_{\mathcal{I}}(\mathrm{Cov}_d(R(a), R(b))) \\
&= \frac{\alpha^2 r^2 (2c-1)^2}{2^{4r-1}} \cdot \frac{1}{n} + O\left(\frac{1}{n^2}\right) \\
&\xrightarrow[n\to\infty]{} 0.
\end{aligned}$$

This proves (3.C), which means the whole theorem is proved. $\square$

Proof of Theorem 1. Denote

$$\begin{aligned}
Y_n &= V_A(R(a))/n, \\
Z_n &= V_A(R(a)) - \mathrm{Cov}_1(R(a), R(b)).
\end{aligned}$$

By (2.C), the random variable Y_n converges in probability to $\alpha(1-1/2^r)/2^r$. In the following, we will show that Z_n converges in probability to $\alpha r/2^r$. As $\xi/n = Y_n/Z_n$, this will imply the theorem. To this end, it suffices to show that

$$E_{\mathcal{I}}(Z_n) \xrightarrow[n\to\infty]{} \frac{\alpha r}{2^r}, \tag{2.6}$$

$$V_{\mathcal{I}}(Z_n) \xrightarrow[n\to\infty]{} 0. \tag{2.7}$$

Using (2.A) and (3.A.1), we get:

$$\begin{aligned} E_{\mathcal{I}}(Z_n) &= E_{\mathcal{I}}(V_A(R(a))) - E_{\mathcal{I}}(\mathrm{Cov}_1(R(a), R(b))) \\ &= \frac{m}{2^r}\left(1 - \frac{1}{2^r}\right) - \frac{m}{2^r}\left(\frac{\binom{n-r}{1}}{\binom{n}{1}} - \frac{1}{2^r}\right) \\ &= \frac{\alpha n}{2^r} \cdot \frac{r}{n} = \frac{\alpha r}{2^r}. \end{aligned}$$

To prove (2.7), denote:

$$g = V_A(R(a)) \cdot \mathrm{Cov}_1(R(a), R(b)).$$

Then:

$$\begin{aligned} V_{\mathcal{I}}(Z_n) &= V_{\mathcal{I}}(V_A(R(a))) + V_{\mathcal{I}}(\mathrm{Cov}_1(R(a), R(b))) \\ &\quad - 2\mathrm{Cov}_{\mathcal{I}}(V_A(R(a)), \mathrm{Cov}_1(R(a), R(b))) \\ &= V_{\mathcal{I}}(V_A(R(a))) + V_{\mathcal{I}}(\mathrm{Cov}_1(R(a), R(b))) \\ &\quad - 2E_{\mathcal{I}}(g) + 2E_{\mathcal{I}}(V_A(R(a)))E_{\mathcal{I}}(\mathrm{Cov}_1(R(a), R(b))). \end{aligned} \tag{2.8}$$

Theorems 2 and 3 provide an explicit form for each of the terms on the right-hand side, with the exception of $E_{\mathcal{I}}(g)$.

By (2.1),

$$V_A(R(a)) = \frac{m}{2^r}\left(1 - \frac{1}{2^r}\right) + 2\sum_{1\le i<j\le m} \mathrm{Cov}_0(R_i(a), R_j(b)),$$

and

$$\mathrm{Cov}_1(R(a), R(b)) = \frac{m}{2^r}\left(1 - \frac{1}{2^r} - \frac{r}{n}\right) + 2\sum_{1\le k<l\le m} \mathrm{Cov}_1(R_k(a), R_l(b)).$$

Thus,

$$\begin{aligned}
E_{\mathcal{I}}(g) &= E_{\mathcal{I}}(V_A(R(a)) \cdot \mathrm{Cov}_1(R(a), R(b))) \\
&= \frac{m^2}{2^{2r}}\left(1 - \frac{1}{2^r}\right)\left(1 - \frac{1}{2^r} - \frac{r}{n}\right) \\
&\quad + \frac{m}{2^r}\left(1 - \frac{1}{2^r} - \frac{r}{n}\right) \cdot 2 \sum_{1 \le i < j \le m} E_{\mathcal{I}}(\mathrm{Cov}_0(R_i(a), R_j(b))) \\
&\quad + \frac{m}{2^r}\left(1 - \frac{1}{2^r}\right) \cdot 2 \sum_{1 \le k < l \le m} E_{\mathcal{I}}(\mathrm{Cov}_1(R_k(a), R_l(b))) \\
&\quad + 4 \sum_{\substack{1 \le i < j \le m \\ 1 \le k < l \le m}} E_{\mathcal{I}}(\mathrm{Cov}_0(R_i(a), R_j(b)) \cdot \mathrm{Cov}_1(R_k(a), R_l(b)))
\end{aligned} \tag{2.9}$$

By (2.2), the second and third addends on the right-hand side of (2.9) both vanish. Moreover, following an argumentation similar to that applied to h_{ijkl} in the proof of Theorem 3, we conclude that $\mathrm{Cov}_0(R_i(a), R_j(b))$ and $\mathrm{Cov}_1(R_k(a), R_l(b))$ are independent whenever $3 \le |\{i, j, k, l\}| \le 4$. Thus, the last sum on the right-hand side of (2.9) reduces to the terms for which $i = k$ and $j = l$. Applying those insights to $E_{\mathcal{I}}(g)$, we arrive at:

$$\begin{aligned}
E_{\mathcal{I}}(g) = \frac{m^2}{2^{2r}}\left(1 - \frac{1}{2^r}\right)\left(1 - \frac{1}{2^r} - \frac{r}{n}\right) \\
+ 4 \sum_{1 \le i < j \le m} E_{\mathcal{I}}(\mathrm{Cov}_0(R_i(a), R_j(b)) \cdot \mathrm{Cov}_1(R_i(a), R_j(b))).
\end{aligned} \tag{2.10}$$

Now, let us convert the expectation appearing in the last expression to a simpler form, which will allow us to calculate it directly:

$$\begin{aligned}
&E_{\mathcal{I}}(\mathrm{Cov}_0(R_i(a), R_j(b)) \cdot \mathrm{Cov}_1(R_i(a), R_j(b))) \\
&\quad = E_{\mathcal{I}}((E_0(R_i(a)R_j(b)) - 1/2^{2r}) \cdot (E_1(R_i(a)R_j(b)) - 1/2^{2r})) \\
&\quad = E_{\mathcal{I}}(E_0(R_i(a)R_j(b)) \cdot E_1(R_i(a)R_j(b))) \\
&\qquad - E_{\mathcal{I}}(E_0(R_i(a)R_j(b)))/2^{2r} - E_{\mathcal{I}}(E_1(R_i(a)R_j(b)))/2^{2r} + 1/2^{4r} \\
&\quad = E_{\mathcal{I}}(E_0(R_i(a)R_j(b)) \cdot E_1(R_i(a)R_j(b))) \\
&\qquad - 1/2^{4r} - 1/2^{4r} + 1/2^{4r} \quad // \text{ as done in (2.2)} \\
&\quad = E_{\mathcal{I}}(P_0(R_i(a) = R_j(b) = 1) \cdot P_1(R_i(a) = R_j(b) = 1)) - 1/2^{4r}.
\end{aligned}$$

As in the proof of Theorem 3, we now consider the clauses C_i and C_j.

We denote by t the number of variables shared by C_i and C_j. Let s be the number of variables, out of the t common ones, whose sign (as literals) is different in the two clauses. The remaining $t-s$ shared variables have the same sign in the two clauses. A direct calculation, as in that proof, with $d=0$ and $d=1$, yields:

$$\begin{aligned}
&E_{\mathcal{I}}(P_0(R_i(a)=R_j(b)=1)\cdot P_1(R_i(a)=R_j(b)=1)) \\
&= \sum_{t=0}^{r}\sum_{s=0}^{t} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot \frac{\binom{t}{s}}{2^t} \cdot \frac{\binom{n-t}{-s}}{2^{2r-t}} \cdot \frac{\binom{n-t}{1-s}}{2^{2r-t}n} \\
&= \frac{1}{2^{4r}} \sum_{t=0}^{r} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot 2^t \cdot \left(1-\frac{t}{n}\right).
\end{aligned}$$

The last equation follows by a routine simplification, after observing that terms for which $s>0$ vanish. Plugging these values into (2.10), we arrive at the final form of $E_{\mathcal{I}}(g)$:

$$\begin{aligned}
E_{\mathcal{I}}(g) &= \frac{m^2}{2^{2r}}\left(1-\frac{1}{2^r}\right)\left(1-\frac{1}{2^r}-\frac{r}{n}\right) \\
&\quad + \frac{2m(m-1)}{2^{4r}}\left(\sum_{t=0}^{r} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot 2^t \cdot \left(1-\frac{t}{n}\right) - 1\right).
\end{aligned}$$

Now that we have an explicit form for all the terms on the right-hand side of (2.8), we can obtain an explicit expression for $V(Z_n)$:

$$\begin{aligned}
V_{\mathcal{I}}(Z_n) &= \frac{2m(m-1)}{2^{4r}}\left(\sum_{t=0}^{r} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot 2^t - 1\right) \\
&\quad + \frac{2m(m-1)}{2^{4r}}\left(\sum_{t=0}^{r} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot 2^t \cdot \left(1-\frac{2t}{n}+\frac{t(t+1)}{n^2}\right) - 1\right) \\
&\quad - \frac{2m^2}{2^{2r}}\left(1-\frac{1}{2^r}\right)\left(1-\frac{1}{2^r}-\frac{r}{n}\right) \\
&\quad - \frac{4m(m-1)}{2^{4r}}\left(\sum_{t=0}^{r} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot 2^t \cdot \left(1-\frac{t}{n}\right) - 1\right) \\
&\quad + 2\cdot\frac{m}{2^r}\left(1-\frac{1}{2^r}\right)\cdot\frac{m}{2^r}\left(1-\frac{1}{2^r}-\frac{r}{n}\right).
\end{aligned}$$

A routine simplification leads to:

$$V_{\mathcal{I}}(Z_n) = \frac{2m(m-1)}{2^{4r}} \cdot \frac{1}{n^2} \cdot \sum_{t=0}^{r} \frac{\binom{r}{t}\binom{n-r}{r-t}}{\binom{n}{r}} \cdot 2^t \cdot t(t+1).$$

Recall that $m = \alpha n$ for some $\alpha > 0$. Thus, the product of the two factors outside the sum (in the last expression) is $\Theta(1)$. The leading addend in the sum is obtained for $t = 1$, and it is $\Theta(1/n)$. Thus, we conclude that

$$V_{\mathcal{I}}(Z_n) = \Theta(1/n) \xrightarrow[n\to\infty]{} 0,$$

which completes the proof. □

2.4 Discussion

In this chapter we have provided some results characterizing the ensemble of all (equally likely) r-CNF formulas. These results apply to both random Max r-Sat and random r-Sat. Along the chapter we chose to present them in the context of random Max r-Sat instances.

In this section we discuss how our results directly apply and characterize random Max r-Sat instances by giving some examples and interpretations of our results. We give some general motivation and implications of our results in the context of local search and landscape theory. Finally, we elaborate on the results of [103], compare them with ours, and clarify our relative contribution.

Consider, for example, the variance of the number of clauses satisfied by a random assignment, explored in Theorem 2. Let $\varepsilon > 0$, and consider the proportion of instances for which the absolute difference between $V_A(S(I,a))/n$ and $\alpha(1-1/2^r)/2^r$ exceeds ε. By (2.C), this proportion tends to 0 as n grows, which gives a very accurate understanding of this variance over the ensemble of all (equally likely) instances. Similar statements can be made regarding the covariance, correlation coefficient, and normalized autocorrelation length, using the results in Theorem 3, Corollary 1, and Theorem 1, respectively.

Formulas (2.B.1) and (2.B.2), as well as (3.B.1) and (3.B.2), give closed expressions and clear asymptotics for useful quantities, usually simulated or approximated (e.g., [54, 85, 89]).

In the local search community, and especially among researchers in the area of landscape analysis, it is a common belief that the ruggedness of the landscape of an instance is one of the main factors for its hardness. The commonly used measures to summarize ruggedness are the autocorrelation length (or alternatively the autocorrelation coefficient) and the fitness-distance correlation. These measures are believed to assess the hardness of combinatorial optimization problems for local search heuristics. These beliefs have been empirically confirmed in various studies [4, 5, 6], and work aiming at better understanding the landscape of combinatorial optimization problems and classifying their hardness for local search is continuously conducted [25, 26, 58, 72, 85, 89, 101, 103].

Our results provide insights on the structure of the landscapes of instances of Max r-Sat, and their similarity. They are also a step toward a richer classification of hardness of combinatorial optimization problems, in the context of local search. We provide a simple expression for the autocorrelation length of Max r-Sat, and other interesting statistical measures for this problem as well. Results in a similar vein, regarding the autocorrelation length of problems like TSP, QAP, GC, GBP, etc, are summarized in [5, Table 1].

In several problems, the autocorrelation length has been calculated with respect to various neighborhood operators, and a clear suggestion for the designers of local search heuristics came out: use the neighborhood operator with the largest autocorrelation length [74, 4, 6]. Such operators induce smoother landscape, which leads to better performance of local search heuristics. Some researchers even suggest performing amendments to the problem in a way that leads to an equivalent problem with a larger autocorrelation length [5]. Our result regarding the autocorrelation length may serve as a baseline for assessing the relevance of any future neighborhood operator or amendment for Max r-Sat.

In many heuristics, after a local optimum is reached, the whole search is repeated from a different, randomly selected starting point. This is a simple, clean way to restart. Yet, as the heuristic may have already yielded a quite good assignment, one may prompt for further exploration of the landscape in the vicinity of this assignment. In such cases, one may want to perform a jump from the local optimum that is not too large, so as to stay in the vicinity of the local optimum. On the other hand, a too small jump would fall in the basin of attraction of the current local optimum, which will lead to the same local optimum again. To this end, the autocorrelation length may provide assisting information, as it hints on the average distance between local optima [74]. The specific formulation of using it to calculate the size of the jump is a subject for further research.

Research efforts to formulate the autocorrelation length of Max r-Sat, done in the last decades, culminated in [103]. Both here and in [103], there is interest in the correlation between the number of clauses satisfied by a random assignment, and an assignment obtained from it due to some random changes.

In this chapter, we measure the change according to the Hamming distance between the two assignments. In [103], one starts from the initial assignment, makes a number of random steps (bit flips), and then compares the initial assignment with the final one. If the distance in the first version, and the number of steps in the second, are the same, then we may expect to be closer to the initial assignment in the second version, as some of the steps may well bring us closer to it. However, as the autocorrelation length only depends on two assignments at a distance of 1 from one another, the two versions are equivalent in this respect.

Our result regarding the normalized autocorrelation length improves the result of [103]. There, this quantity is provided only in terms of Walsh coefficients [42], along with a mean-field approximation [106]. We find an *explicit* expression that approximates the normalized autocorrelation length. Furthermore, we prove *rigorously* that the normalized autocorrelation length converges in probability to this explicit expression as the number of variables grows.

Finally, we note that the nature of industrial instances, for example, is subtly different from random ones. Their underlying probability model, if any, is different, and they should be addressed separately. We hope our work will encourage a concise analysis of modeled practical instances ensembles.

Chapter 3

Effect of Initial Assignment on Local Search Performance for Max Sat

In this chapter, we explore the correlation between the quality of initial assignments provided to local search heuristics and that of the corresponding final assignments. We restrict our attention to the Max r-Sat problem and to one of the leading local search heuristics – Configuration Checking Local Search (CCLS). We use a tailored version of the Method of Conditional Expectations (MOCE) to generate initial assignments of diverse quality.

We show that the correlation in question is significant and long-lasting. Namely, even when we delve deeper into the local search, we are still in the shadow of the initial assignment. Thus, under practical time constraints, the quality of the initial assignment is crucial to the performance of local search heuristics.

To demonstrate our point, we improve CCLS by combining it with MOCE. Instead of starting CCLS from random initial assignments, we start it from excellent initial assignments, provided by MOCE. Indeed, it turns out that this kind of initialization provides a significant improvement of this state-of-the-art solver. This improvement becomes more and more significant as the instance grows.

3.1 Introduction

In the Maximum Satisfiability (Max Sat) problem [66], we are given 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. We seek 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. Here we restrict our attention to instances with clauses consisting of exactly r literals each (sometimes called Max Er-Sat). We 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, we focus on the case where r and α are constant.

As Max r-Sat (for $r \geq 2$) is NP-hard [11, pp. 455–456], it cannot be exactly solved in polynomial time (unless $P = NP$), and one must resort to approximation algorithms and heuristics. Numerous methods have been suggested for solving Max r-Sat, e.g. [17, 99, 97, 69, 24, 78, 8, 32, 56], and an annual competition of solvers has been held since 2006 [10]. Satisfiability related questions attracted a lot of attention from the scientific community. As an example, one may consider the well-studied satisfiability threshold question for random instances [27, 37, 2, 73, 28, 34]. For a comprehensive overview of the whole domain of satisfiability we refer to [16].

3.1.1 Local search

Local search heuristics [59] explore the assignment space. They usually start from 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) [69] 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.

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 randomly.

Generally, the number of satisfied clauses after flipping a variable is not necessarily larger than prior to the flip. In fact, it is even possible that, flipping any of the candidate variables, we will arrive at a lower quality assignment. Also, the set of candidates may be empty in some of the greedy steps. In such a 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 appearing together with it in at least one clause.

Recent works, related to local search, configuration checking, CCLS, and algorithms of the same spirit, include [79, 22, 20, 70, 71, 1, 18, 19, 21, 98, 76, 100, 23].

3.1.2 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 Method of Conditional Expectations (MOCE) [35, 107], 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 [53]. We 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 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 is independent of the number of variables or clauses in the instance. Thus, a step takes a constant time on the average. Note, though, that this requires us to continuously track all clauses containing each variable. This is in addition to the map between clauses and their variables.

Recent theoretical and empirical works related to MOCE, and algorithms of the same spirit, include [29, 82, 84, 83, 30].

3.1.3 Overview

In Section 3.2, we explore the correlation between the quality of the initial assignments provided to local search heuristics and the quality of the final assignments resulting from them. We restrict our attention to CCLS, which is the winner of several Max Sat competitions held in recent years [10], and is currently one of the best practical Max Sat solvers available. We use a tailored version of MOCE to generate initial assignments of diverse quality, to accommodate the exploration of the correlation.

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. This implies that, even when we delve deeper into the local search, we are 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.

In Section 3.3, we demonstrate our point by improving CCLS. Instead of starting CCLS from a random initial assignment, we start it from excellent initial assignments, provided by MOCE. This kind of initialization provides a significant improvement of this state-of-the-art solver. Moreover, the improvement 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 [50, 49]. A summary and conclusions are presented in Section 3.4.

3.2 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. We explore the ongoing correlation during the execution as well. We have chosen CCLS 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, we manipulate MOCE 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, we assign to the current variable 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.

We emphasize 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 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.

3.2.1 End-to-end correlation

In the following, we describe what we have 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, we started a local search using CCLS, and thus obtained 51 final assignments. By the end of the 51 executions, we had 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 minutes, measured in CPU time.

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

For each of the correlation coefficients, we also calculated the p-value. The p-value is the probability that we would have found this correlation, or

	correlation coefficient			**regression slope**	
density	**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	$1.1 \cdot 10^{-5}$	$3.4 \cdot 10^{-3}$	$0.7 \cdot 10^{-3}$

Table 3.1: End-to-end correlation coefficients and regression slopes.

a higher one, 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, we calculated the average p-value over the 150 correlation coefficients as a measure of the statistical significance of the results.

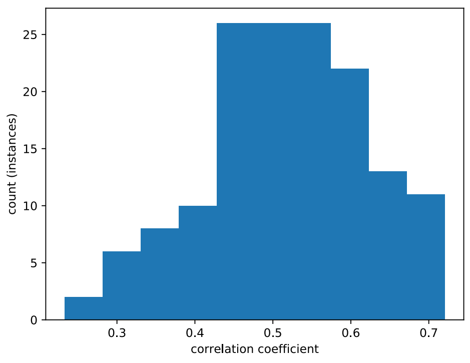
To measure the impact of the improvement of the quality of an initial assignment on the quality of the corresponding final assignment, we applied regression analysis. Specifically, we calculated the regression line of each of the instances of a family, and took its slope as a measure of the strength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

The results are provided in Table 3.1. 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 3.1 depicts histograms of the 150 end-to-end correlation coefficients of the family of density 5 (Figure 3.1a) and for the family of density 15 (Figure 3.1b).

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.

While the correlation is strong, the regression slope suggests that a large improvement in the initial assignment yields only a small improvement in


(a) Family of density 5.

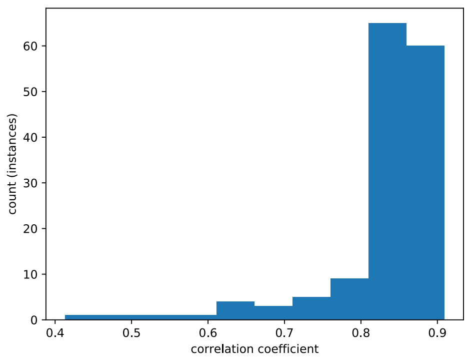

(b) Family of density 15.

Figure 3.1: Histograms of end-to-end correlation coefficients.

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 minutes 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 3.2 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). We 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.

3.2.2 Ongoing correlation

Besides the end-to-end correlation, we explored the ongoing correlation during the experiment. To this end, for each initial assignment, we recorded the minimum number of unsatisfied clauses found so far, not only at the end of

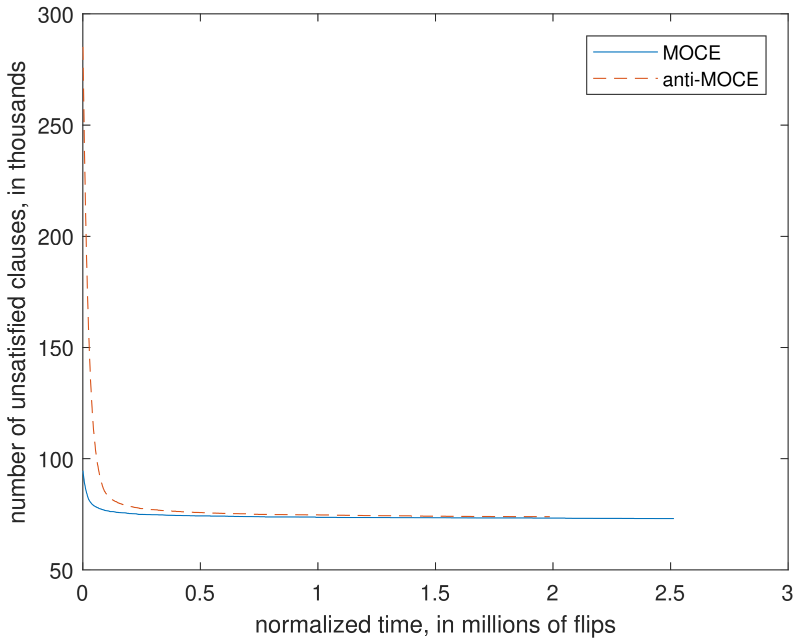


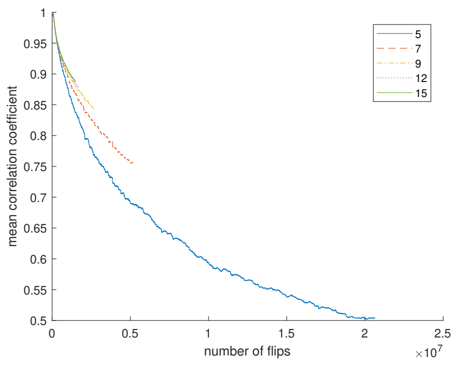
Figure 3.2: Number of unsatisfied clauses as function of the number of flips.

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.

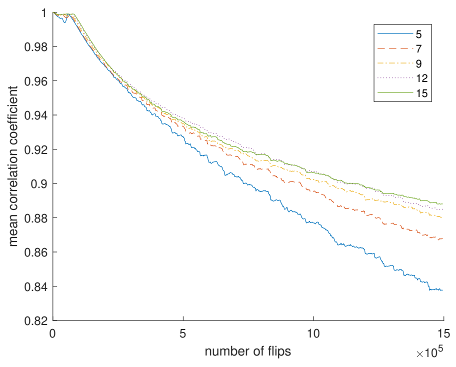
The number of flips made during the execution is very different for different families. In a denser instance, a flip takes longer, so that less flips are made. Even for instances of the same family, the number of flips varies. We provide statistics only up to the minimal number of flips made, over all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

Figure 3.3 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.

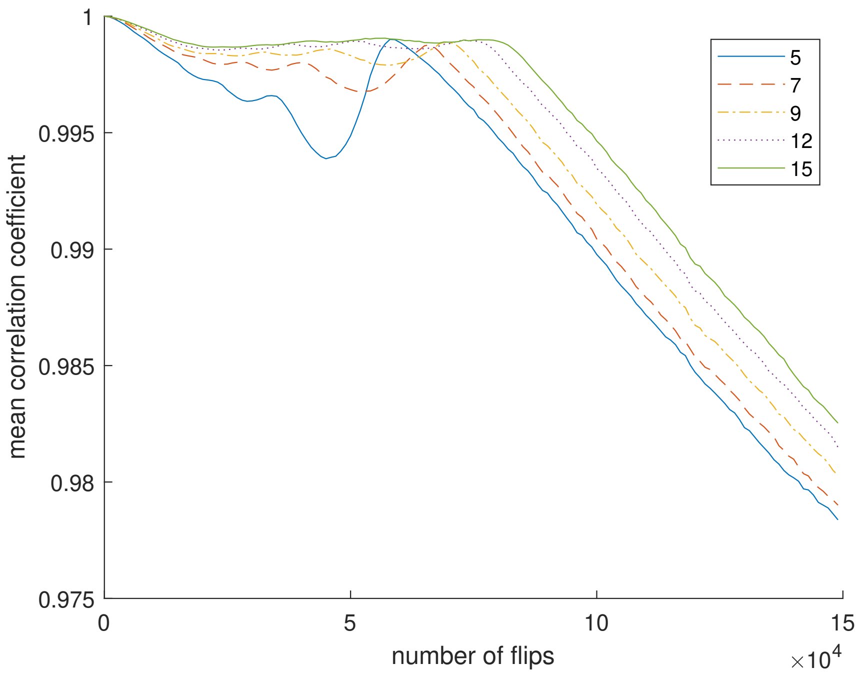
Figure 3.3a shows the full results. It provides the graphs of correlation



(a) Full graphs.



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



(c) First 150,000 flips.

Figure 3.3: Ongoing correlation decay as a function of the number of flips.

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 3.3b 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, we see clearly the faster decay of the correlation in sparser families, as well as the smaller differences between the decay in denser families.

Figure 3.3c zooms in on the first 150,000 flips. During this stage, we 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 characterized by a rapid decrease in the number of unsatisfied clauses, and is aligned with the rapid decrease shown in Figure 3.2.

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 Figure 3.2, 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 Figure 3.3c.

Note that Figure 3.3c 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 we have finished the rapid increase phase for all initial assignments, the correlation returns to a higher level.

3.2.3 Experimentation information

The experiments described in this section were executed on a Sun Grid Engine (SGE) [102] 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, we had 744 computation cores and 1984GB of working memory at hand.

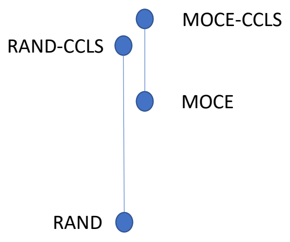
We limited each of the jobs submitted to the cluster to use up to 3GB of working memory. Provided the load on the cluster, we managed to achieve a parallelization of about 300 times, thus reducing the experiments overall sequential time of approximately 2.18 years to around 2.66 days of parallel execution.

3.3 Improving CCLS

The high correlation between the quality of the initial assignment and that of the final one makes it clear that we are searching in the shadow of the initial assignment for a long period of time. Thus, starting the local search from an excellent initial assignment, we may improve its performance. This holds as long as the selection of such an initial assignment does not consume too much time.

In this section, we study 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 we use 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.

We first conducted experiments on several families of random instances. 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. Afterward, we conducted experiments on some public benchmarks. We 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.



In the diagram on the right, we summarize qualitatively what we have observed in 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.

3.3.1 Comparative performance on structured benchmarks

In this section we focus on random instances, for which the clauses are of length 3, the number of variables ranges from 10,000 to 1,000,000, and the density from 3 to 9. Such ranges allow us to systematically study the performance of the algorithms at hand on diverse families. For each family, we selected 100 instances uniformly at random, in the same way elaborated in Section 3.2.

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 [15]. For convenience, the (theoretical) average number of clauses unsatisfied by RAND for the families we studied (namely, $m/8$), is provided in Table 3.2. The rows correspond to the various numbers of variables, n, and the columns to the various densities, α.

α / n	3		5		7		9	
	RAND	MOCE	RAND	MOCE	RAND	MOCE	RAND	MOCE
10000	3750	412	6250	1500	8750	2889	11250	4442
50000	18750	2078	31250	7459	43750	14409	56250	22209
100000	37500	4149	62500	14944	87500	28861	112500	44436
500000	187500	20790	312500	74702	437500	144296	562500	222074
1000000	375000	41559	625000	149383	875000	288572	1125000	444174
% unsat	12.5%	1.4%	12.5%	3%	12.5%	4.1%	12.5%	4.9%

Table 3.2: The number of clauses unsatisfied by RAND and MOCE.

Table 3.2 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 but a few seconds for the larger instances we 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, we obtained a significant improvement. Namely, the number of unsatisfied clauses is significantly reduced at the local search stage.

This improvement is summarized in Table 3.3. 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

α	3		5		7		9	
n	(R−RC)/R	(M−MC)/M	(R−RC)/R	(M−MC)/M	(R−RC)/R	(M−MC)/M	(R−RC)/R	(M−MC)/M
10000	100.0%	100.0%	96.0%	83.6%	85.5%	56.2%	77.4%	42.9%
50000	100.0%	100.0%	95.5%	81.2%	84.8%	54.1%	76.7%	41.2%
100000	100.0%	100.0%	95.1%	79.9%	84.3%	52.9%	76.2%	40.2%
500000	100.0%	100.0%	90.4%	69.4%	77.1%	42.4%	68.3%	31.3%
1000000	80.6%	100.0%	48.7%	49.6%	37.4%	27.1%	31.6%	18.3%

Table 3.3: The improvement by executing CCLS after RAND and MOCE.

MOCE-CCLS, divided by the number of clauses unsatisfied by MOCE. In the table, we also present the improvement of supplementing RAND with CCLS (which is simply the standard version of CCLS), under the columns named "(R−RC)/R".

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 3.3 are based on 30 minutes 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 [67, 68, 60, 61, 55, 51].

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, we comment that, theoretically, if the execution time is unlimited, no algorithm can beat CCLS, or even the baseline RAND. This is due to their incorporated randomness, which eventually will lead to an optimal solution if time is unrestricted. Clearly, this has no practical implications, as the amount of time required is way out of reach.

We conclude this section by comparing MOCE-CCLS and RAND-CCLS head to head. The comparison is provided in Table 3.4 (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

α / n	3			5		
	RC	MC	% improve	RC	MC	% improve
10000	0	0	NaN	248	246	0.81%
50000	0	0	NaN	1417	1403	0.99%
100000	0	0	NaN	3038	3002	1.18%
500000	0	0	NaN	29976	22894	23.63%
1000000	72642	0	100.00%	320674	75260	76.53%
α / n	7			9		
	RC	MC	% improve	RC	MC	% improve
10000	1265	1264	0.08%	2546	2537	0.35%
50000	6647	6617	0.45%	13122	13052	0.53%
100000	13717	13588	0.94%	26770	26554	0.81%
500000	99976	83163	16.82%	178234	152512	14.43%
1000000	548044	210440	61.60%	769640	363037	52.83%

Table 3.4: MOCE-CCLS vs. RAND-CCLS.

RAND-CCLS and the number of those unsatisfied by MOCE-CCLS, divided by the number of clauses unsatisfied by RAND-CCLS.

The results demonstrate our point regarding the importance of the initial solution. Even after 30 minutes 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, we conclude that, 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.

Experimentation information

The experiments described in this section were carried out on the same infrastructure as in Section 3.2.3. Here as well, we limited each of the jobs submitted to the cluster to use up to 3GB of working memory. Provided the load on the cluster, we managed to achieve a parallelization of about 100 times, thus reducing the experiment overall sequential time of approximately 4.11 months to around 1.25 days of parallel execution.

3.3.2 Comparative performance on public benchmarks

An international evaluation of solvers for the Maximum Satisfiability problem has been held annually since 2006 [10]. The random instances of the evaluation of 2016 were tailored mainly for complete solvers. Thus, they are very small and 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 most of the time; the ranking was only according to the time they consumed to reach their best solutions.

In our comparison of RAND-CCLS and MOCE-CCLS, the situation was no different. Both found solutions with the same number of unsatisfied clauses as the leading solvers in the evaluation, and the winner was decided by the time it required. On those instances, RAND-CCLS found the best solution faster, and thus won.

In the following, we consider three additional benchmarks in the same spirit as the 2016 Evaluation – but larger ones. As we wanted to keep the exact same blend of instances, we created the new benchmarks by blowing up the original ones. We enlarged the number of variables and that of clauses in each instance, while keeping the density the same as in the evaluation.

We created three expanded benchmarks 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, we went over the original instances one by one, and for each instance created a new random instance 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 Instance Won measure. This measure is the one used in the Max Sat Evaluation [10] held in 2016, from which we took the original instances. We ran each of the two competitors on each of the instances for a few minutes (CPU time). For each instance, the winner is the competitor that provides the smaller number of unsatisfied clauses. Ties are broken by the time it took each competitor to arrive at its best solution. The overall

measure / blow	By number of unsatisfied clauses				Tie breaking by time-to-best		
	RC	MC	Draw	Winner	RC	MC	Winner
x10 (1 min)	112	115	145	MC	187	185	RC
x100 (2 min)	137	223	12	MC	143	229	MC
x1000 (5 min)	4	368	0	MC	4	368	MC

Table 3.5: MOCE-CCLS vs. RAND-CCLS, Instance Won measure, on random instances blown up from those of Max Sat Evaluation 2016.

winner is the heuristic that wins more instances.

The results, according to the Instance Won measure, are presented in Table 3.5. The first line in the table relates to the benchmark with instances blown up by a factor of 10. The time limit for this benchmark is 1 minute.

The first four entries on the first line show the results according to the Instance Won measure, with ties considered as draw. Namely, if the competitors obtain the same number of unsatisfied clauses in some instance, both within the given time limit, we consider the result as a draw. The first four entries show the number of instances won by RAND-CCLS, the number of instances won by MOCE-CCLS, the number of instances for which we got a draw, and the winner according to the above measure.

The entries afterward provide the results when ties are broken by time-to-best. Namely, if the number of unsatisfied clauses is the same, the winner is the competitor who reached this value faster. The last three entries provide the number of instances won by RAND-CCLS, the number of those won by MOCE-CCLS, and the identity of the winner according to this variant of the measure.

While RAND-CCLS wins on the competition instances, it is enough to blow up the instances tenfold to have MOCE-CCLS achieve an overall draw. 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 MOCE-CCLS wins on the expanded benchmarks in terms of the number of unsatisfied clauses, and not merely by time. Namely, MOCE-CCLS provides solutions with a strictly smaller number of unsatisfied clauses.

We also calculated the average relative improvement of MOCE-CCLS over RAND-CCLS. Overall, this average for the benchmark expanded by a factor of 10 was 0.04%. For the benchmark expanded by a factor of 100, it was 0.16%, and for the benchmark expanded by a factor of 1000 it was

0.64%.

Note that the number of variables in instances of the original Max Sat Evaluation 2016 benchmark is between 70 and 200. Thus, even after blowing up by a factor of 1000, we obtain instances of medium size only.

In the one thousand-fold expanded benchmark, the improvement was manifested in an average of extra 423 clauses satisfied by MOCE-CCLS. 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.

Finally, we note that MOCE alone is not enough. It is the value from the combined solver MOCE-CCLS that leads to the extra satisfied clauses. Moreover, CCLS provides a significant improvement to the excellent initial solutions of MOCE. Thus, the state-of-the-art performance of MOCE-CCLS is attributed to both its ingredients: MOCE and CCLS.

3.4 Summary and conclusions

In this chapter, we have explored the correlation between the quality of initial assignments provided to local search heuristics and that of the corresponding final assignments. We 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.

We demonstrated our point by improving the state-of-the-art solver CCLS, by combining it with MOCE. Instead of starting CCLS from random initial assignments, we started 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. Namely, it handles larger instances better, and shows superior performance on them.

Given the above, we recommend MOCE-CCLS over RAND-CCLS. Furthermore, we recommend starting CCLS from solutions 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).

Chapter 4

Efficient Exhaustive Method of Conditional Expectations for Max Sat

In this chapter we present and study a new algorithm for the Maximum Satisfiability (Max Sat) problem. The algorithm is based on the Method of Conditional Expectations (MOCE), and applies an efficient greedy variable ordering to MOCE. We call our algorithm Efficient Exhaustive Method of Conditional Expectations (EEMOCE) as its greediness efficiently exhausts all unassigned variables at each step.

We conduct a comprehensive empirical evaluation of EEMOCE and show that it performs much better than MOCE, while keeping the additional runtime overhead very low. In particular, EEMOCE reduces the number of unsatisfied clauses by tens of percents, while the time complexity increases only by a logarithmic factor. The actual runtime is typically up to 3 times longer even for very large instances.

We empirically study the main quantities managed by EEMOCE during its execution, exposing a relatively large residual randomality that may be harnessed for further improvement of the performance. Based on this study, we also point out how to eliminate the logarithmic factor added to the time complexity, in practical usages.

4.1 Introduction

In the Maximum Satisfiability (Max Sat) problem [66], we are given 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. We seek a truth (`true`/`false`) assignment for the variables, maximizing the number of satisfied (made `true`) clauses, or equivalently, minimizing the number of unsatisfied (made `false`) clauses.

In the Max r-Sat problem, each clause is restricted to consist of at most r literals. Here we restrict our attention to instances with clauses consisting of exactly r literals each. This restricted problem is also known as Max Er-Sat.

Let n be the number of variables. Denote the variables by $x_1, x_2, \ldots, x_n$. The number of clauses is denoted by m, and the clauses by $C_1, C_2, \ldots, C_m$. We denote the clause-to-variable ratio by $\alpha = m/n$, to which we also refer as density.

We use the terms "positive variable" and "negative variable" to refer to a variable and to its negation, respectively. Whenever we find it convenient, we consider the truth values `true` and `false` as binary 1 and 0, respectively.

As Max r-Sat (for $r \geq 2$) is NP-hard [11, pp. 455–456], large-sized instances cannot be exactly solved in an efficient manner (unless $P = NP$), and one must resort to approximation algorithms and heuristics. Numerous methods have been suggested for solving Max r-Sat, e.g. [17, 99, 97, 69, 24, 78, 8, 32, 56], and an annual competition of solvers has been held since 2006 [10].

Using Walsh analysis [42], an efficient way of calculating moments of the number of satisfied clauses of a given instance of Max r-Sat was suggested in [54]. 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. A closed-form formula, asymptotics, and convergence proof for the variance is provided in [15].

An interesting study of Max 3-Sat is provided in [85]. The authors claim that many instances share similar statistical properties and provide empirical evidence for it. Simulation results on the autocorrelation of a random walk in the assignments space are provided for several instances, as well as predictions for typical instances. In [58], the authors analyze how the way random instances are generated affects the autocorrelation and fitness-distance correlation. They raised the question of similarity of the landscape of different instances.

Overall, satisfiability related questions attracted a lot of attention from the scientific community. Some of them were thoroughly studied, especially the satisfiability threshold density question [27, 37, 2, 73, 28, 34]. For a comprehensive overview of the whole domain of satisfiability we refer to [16].

This chapter is arranged as follows. In Section 4.2 we present MOCE, a classical algorithm for Max r-Sat, and discuss some related algorithmic aspects and implementation details. In Section 4.3 we introduce our algorithm EEMOCE for Max r-Sat. We provide an elaborated description of the algorithm, along with a fully fledged pseudocode. We introduce several notions related to its algorithmics, and get into details regarding various quantities it manages and how they are managed efficiently.

Sections 4.4 and 4.5 are dedicated to presenting the results of a comprehensive empirical study designed to evaluate the performance of EEMOCE. Section 4.4 is focused on practical performance analysis – the analysis of regular density instances, commonly used in practice. On those instances, we summarize the performance of EEMOCE both in general and as compared to MOCE.

Section 4.5 is focused on asymptotics performance analysis – the analysis of high density instances. In this section we summarize the performance of EEMOCE in general and as compared to MOCE and to theoretical bounds regarding the optimum. We also pinpoint a caveat regarding the analysis of performance in those cases and suggest performance measures to overcome it.

In Section 4.6 we study the main quantities managed by EEMOCE during its execution. We expose a relatively large residual randomality that may be harnessed for further improvement of the performance. A summary and conclusion are provided in Section 4.7.

4.2 The Method of Conditional Expectations

Consider the naive randomized approximation algorithm which assigns to each variable a truth value uniformly at random, independently of all other variables. It satisfies $1 - 1/2^r$ of all clauses on the average. Furthermore, it can also be easily derandomized using the Method of Conditional Expectations (MOCE) [35, 107], yielding an assignment that is guaranteed to satisfy at least this number 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 [53]. We note that, typically, this method yields assignments that are much better than this worst-case bound. Theoretical and empirical works related to MOCE, and algorithms of the same spirit, include [29, 82, 84, 83, 30].

In the next section we provide a detailed explanation of the MOCE. The two following sections delve into mathematical and algorithmical issues that will help us understand our EEMOCE algorithm in Section 4.3.

4.2.1 Description of the algorithm

MOCE iteratively constructs an assignment by going over the variables in an arbitrary, usually random, order. At each step, 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 assignment to the previously assigned 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 uniformly random assignment to all the currently unassigned variables. The truth value, for which this sum is the larger of the two, is the one selected for the current variable. Ties are broken randomly. The process is repeated until all variables are assigned.

In an efficient implementation, each step of MOCE typically takes a constant time, as each variable appears in $r\alpha$ clauses on the average. The main thing to do in each step is to find the best truth value for the currently assigned variable and residualize the instance accordingly.

To find the best 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.

The expected gain of assigning the current variable `true`, is calculated by going over the clauses the variable appears in. Each as yet unsatisfied clause, 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.

Indeed, let C be a clause of length $|C| = l$. The probability of C to be eventually satisfied, when all variables in it are assigned, is $1 - 2^{-l}$. By giving a variable in the clause the value satisfying the clause, we increase this $1 - 2^{-l}$ to 1. The clauses satisfied by the assignment of the current variable are eliminated from the instance.

On the other hand, each unsatisfied clause that remains unsatisfied by the assignment of `true` to the variable contributes -2^{-l} to the overall expected gain. Indeed, the probability it will eventually satisfied descends from $1 - 2^{-l}$ to $1 - 2^{-(l-1)}$. These clauses remain in the instance, but they are shortened by one literal – the literal associated with the current variable.

The overall expected gain is 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 is independent of the number of variables or clauses in the instance. Thus, a step takes a constant time on the average, as both r and α are assumed to be constant. Note that, for each of the variables, this requires us to keep track at all times of all clauses containing it (either positively or negatively). This is in addition to the usual map between clauses and their variables.

During the overall execution of MOCE, each of the $m = \alpha n$ 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. Its runtime is no longer than a few seconds for instances with hundreds of thousands of variables and clauses. For smaller instances, its execution time is typically less than a second. This time includes loading the instance, building it in the memory, and constructing the assignment.

4.2.2 Mathematical pseudocode

In this section we provide a mathematical pseudocode for MOCE – see Algorithm 1. This pseudocode is simplified, and puts emphasis merely on

Algorithm 1 The Method of Conditional Expectations

Input: An instance I over n variables $x_1, x_2, \ldots, x_n$.
Output: An assignment of truth value for each of the variables.

```
 1: procedure MOCE($I$)
 2:     for $i \leftarrow 1, 2, \ldots, n$ do
 3:         $\mu_T \leftarrow E_{X_{i+1},X_{i+2},\ldots,X_n}[S_I(b_1, b_2, \ldots, b_{i-1}, T, X_{i+1}, X_{i+2}, \ldots, X_n)]$
 4:         $\mu_F \leftarrow E_{X_{i+1},X_{i+2},\ldots,X_n}[S_I(b_1, b_2, \ldots, b_{i-1}, F, X_{i+1}, X_{i+2}, \ldots, X_n)]$
 5:         if $\mu_T > \mu_F$ then
 6:             $x_i \leftarrow T$
 7:         else if $\mu_T < \mu_F$ then
 8:             $x_i \leftarrow F$
 9:         else
10:             set $x_i$ to $T$ or $F$, uniformly at random
11:         end if
12:     end for
13: end procedure
```

the probabilistic idea underlying the algorithm.

For a given instance I and a given assignment a, let $S_I(a)$ be the number of clauses of I satisfied by the assignment a. Let $X_1, X_2, \ldots, X_n$ be the boolean-valued uniformly random variables, corresponding to the variables $x_1, x_2, \ldots, x_n$, respectively. We denote by

$$E_{X_{i+1},X_{i+2},\ldots,X_n}[S_I(b_1, b_2, \ldots, b_i, X_{i+1}, X_{i+2}, \ldots, X_n)]$$

the expected number of clauses of I satisfied by a random assignment, conditioned on the first i variables $x_1, x_2, \ldots, x_n$ being set to the truth values $b_1, b_2, \ldots, b_i$, respectively.

As is typically done, and for simplicity, in the pseudocode MOCE iterates over the variables according to their index order. Of course, one may take any arbitrary order. In principle, we view the order as uniformly random.

In the i-th step, MOCE calculates two conditional expectations. The first, in line 3, is the expected number of clauses of I satisfied by a random assignment conditioned on the first $i-1$ variables being set to the truth values $b_1, b_2, \ldots, b_{i-1}$ (selected in previous iterations), and the i-th variable is set to `true`. The second, in line 4, is the same quantity, but assuming the i-th variable is set to `false`.

In lines 5–10, it sets the i-th variables to the truth value providing the larger quantity of the above two. In case of equality, it sets the i-th variables to either `true` or `false`, uniformly at random.

4.2.3 Algorithmic pseudocode

In this section we provide an algorithmic pseudocode for MOCE – see Algorithm 2. This pseudocode is detailed and puts emphasis on the algorithmics underlying the algorithm. As before, for simplicity, in the pseudocode MOCE iterates over the variables according to their index.

At each iteration, in lines 3–6, MOCE first checks if the instance I is empty, namely, if no clauses remained in the instance. In this case, it sets all the unassigned variables to either `true` or `false`, uniformly at random, and terminates.

If the instance is non-empty, MOCE focuses on the sub-instance consisting of all clauses of I in which x_i appears, either positively or negatively. This instance is denoted by J in line 7. In lines 8–11, MOCE checks if J is empty – a situation that occurs if all the clauses in which x_i appeared in the original instance have already been satisfied by previously assigned variables. In this case, MOCE sets x_i to either `true` or `false`, uniformly at random. Then, it continues directly to consider the next variable.

Lines 12–19 deal with the case in which the variable x_i still appears in some clauses when it is considered. In this case, MOCE calculates the expected gain associated with setting it to `true`. This expected gain g^T is nullified in line 12. Then, in line 14, MOCE iterates over all the clauses C of J and adds (subtracts, respectively) $2^{-|C|}$ to (from, respectively) g^T if x_i appears positively (negatively, respectively) in C, where $|C|$ is the number of literals in C. Thus, by the end of these lines, g^T is the expected gain associated with setting x_i to `true`, taken over all the clauses. MOCE does not explicitly calculate the expected gain associated with setting x_i to `false`, as it is the additive inverse of g^T.

In lines 20–26, MOCE sets x_i to the truth value providing the larger expected gain. In case of equality, MOCE sets it to either `true` or `false`, uniformly at random.

Finally, in line 27, the instance I is residualized. The instance $I \restriction x_i$ is obtained from I by:

1. removing all the clauses satisfied by the selected assignment to x_i,

Algorithm 2 The Method of Conditional Expectations

```
Input: An instance $I$ over $n$ variables $x_1, x_2, \dots, x_n$, with $m$ clauses
       $C_1, C_2, \dots, C_m$.
Output: An assignment of truth value for each of the variables.
 1: procedure MOCE($I$)
 2:     for $i \leftarrow 1, 2, \dots, n$ do
 3:         if $|I| = 0$ then                                  ▷ instance is empty
 4:             set each of the variables of $I$ independently to either $T$ or $F$,
                uniformly at random
 5:             return                                          ▷ terminate algorithm
 6:         end if
 7:         $J \leftarrow$ the sub-instance of $I$, consisting of all the clauses in which
            the variables $x_i$ appears, either positively or negatively
 8:         if $|J| = 0$ then                                  ▷ $x_i$ is a ghost variable
 9:             set $x_i$ to either $T$ or $F$, uniformly at random
10:             continue                   ▷ jump to line 2, and to the next value of i
11:         end if
12:         $g^T \leftarrow 0$                                  ▷ expected gain of $x_i = T$
13:         for each clause $C$ in $J$ do
14:             if $x_i$ appears positively in $C$ then
15:                 $g^T \leftarrow g^T + 2^{-|C|}$
16:             else
17:                 $g^T \leftarrow g^T - 2^{-|C|}$
18:             end if
19:         end for
20:         if $g^T > 0$ then
21:             $x_i \leftarrow T$
22:         else if $g^T < 0$ then
23:             $x_i \leftarrow F$
24:         else
25:             set $x_i$ to either $T$ or $F$, uniformly at random
26:         end if
27:         $I \leftarrow I \upharpoonright x_i$
28:     end for
29: end procedure
```

2. removing the literal associated with x_i from all clauses unsatisfied by it.

Thus, in the next iteration, MOCE operates typically on a smaller instance.

4.3 The Efficient Exhaustive Method of Conditional Expectations

In this section we present a new algorithm for the Maximum Satisfiability (Max Sat) problem. The algorithm is based on the Method of Conditional Expectations (MOCE), and applies an efficient greedy variable ordering to MOCE. We call our algorithm Efficient Exhaustive Method of Conditional Expectations (EEMOCE) as its greediness efficiently exhausts all unassigned variables at each step.

In Section 4.3.1, we briefly explain the basic idea underlying EEMOCE and present a mathematical pseudocode for it. This pseudocode, if implemented simple-mindedly, leads to quadratic time complexity. Thus, it should be regarded merely as a conceptual presentation of EEMOCE. Afterward we dive into the algorithmics of EEMOCE in several sections, altogether leading to our linearithmic time complexity algorithm EEMOCE, whose pseudocode is presented in Section 4.3.8.

In Section 4.3.2, we describe how we efficiently represent instances. Then, in Section 4.3.3, we present the residualization operation and explain how it is done efficiently. Section 4.3.4 is dedicated to the concept of gain – the way EEMOCE conveys information regarding the profitability of assigning a given variable to a given truth value.

Section 4.3.5 is focused on efficient maintenance of gains. In Section 4.3.6, we briefly describe how the gains are calculated initially. Section 4.3.7 is dedicated to the way gains are updated during the execution of EEMOCE by the so-called marginalization operation. This operation is a delicate and important part of EEMOCE.

An algorithmic pseudocode of EEMOCE is presented in Section 4.3.8. We finish the whole section by analyzing the time complexity of EEMOCE in Section 4.3.9.

4.3.1 Mathematical pseudocode

In this section we provide a mathematical pseudocode for EEMOCE – see Algorithm 3. This pseudocode is provided mainly to obtain a high level view of EEMOCE, and to point out the basic difference between EEMOCE and MOCE. This difference is in the order they consider the variables. Namely, while MOCE goes over the variables in a random order, EEMOCE goes over

Algorithm 3 The Efficient Exhaustive Method of Conditional Expectations

Input: An instance I over n variables $x_1, x_2, \dots, x_n$.
Output: An assignment of truth value for each of the variables.

```
1: procedure EEMOCE($I$)
2:     for $j \leftarrow 1, 2, \dots, n$ do
3:         without loss of generality, assume the as yet unassigned variables
           are $x_j, x_{j+1}, \dots, x_n$
4:         for $i \leftarrow j, j+1, \dots, n$ do
5:             $\mu_{T,i} \leftarrow E_{X_j,\dots,X_{i-1},X_{i+1},\dots,X_n}[S_I(b_1, \dots, b_{j-1}, X_j, \dots, X_{i-1}, T,$
               $X_{i+1}, \dots, X_n)]$
6:             $\mu_{F,i} \leftarrow E_{X_j,\dots,X_{i-1},X_{i+1},\dots,X_n}[S_I(b_1, \dots, b_{j-1}, X_j, \dots, X_{i-1}, F,$
               $X_{i+1}, \dots, X_n)]$
7:         end for
8:         $i \leftarrow \operatorname{argmax}_{j \le i \le n} \max(\mu_{T,i}, \mu_{F,i})$, break ties uniformly at random
9:         if $\mu_{T,i} > \mu_{F,i}$ then
10:            $x_i \leftarrow T$
11:        else if $\mu_{T,i} < \mu_{F,i}$ then
12:            $x_i \leftarrow F$
13:        else
14:            set $x_i$ to $T$ or $F$, uniformly at random
15:        end if
16:    end for
17: end procedure
```

them in a greedy order.

For a given instance I and a given assignment a, denote by $S_I(a)$ the number of clauses of I satisfied by a. We denote by

$$E_{X_j,\dots,X_{i-1},X_{i+1},\dots,X_n}[S_I(b_1, \dots, b_{j-1}, X_j, \dots, X_{i-1}, b, X_{i+1}, \dots, X_n)]$$

the expected number of clauses of I satisfied by a random assignment, conditioned on the first $j-1$ variables being set to the truth values $b_1, b_2, \dots, b_{j-1}$, and the i-th variable set to the truth value b.

In this pseudocode, the loop in line 2 iterates n times as EEMOCE needs to make n steps to assign all n variables. At the beginning of each iteration, in line 3, we rename the variables in such a way that all the variables up to x_{j-1} are assigned, and the variables from x_j on are unassigned. This is done merely for simplicity of presentation.

In lines 4–7, EEMOCE iterates over all the unassigned variables x_i, $j \le i \le n$. For each such variable, it calculates two conditional expectations.

The first, in line 5, is the expected number of clauses of I satisfied by a random assignment, conditioned on the first $j-1$ variables being set to the truth values $b_1, b_2, \ldots, b_{j-1}$ (set to them in previous iterations), and x_i is set to `true`. The second, in line 6, is the same quantity, but assuming x_i is set to `false`.

In line 8, EEMOCE finds the variable whose gain is maximal. Note that there may be more than one such variable. Ties are broken by selecting one of these variables uniformly at random. In lines 9–15, EEMOCE set x_i to the truth value providing the larger of $\mu_{T,i}$ and $\mu_{F,i}$. In case of equality, it sets x_i to either `true` or `false`, uniformly at random.

4.3.2 Efficient instance representation

In our algorithm, an instance is represented by two core data structures:

`clauses` – a mapping of clause index to a clause. A clause is a mapping of variable index to boolean value, where `true` means the variable appears as is in the clause and `false` means the variable appears negated in the clause.

`variables` – a mapping of variable index to a variable. A variable is a mapping of clause index to boolean value, where `true` means the variable appears as is in the clause and `false` means the variable appears there negated.

Figure 4.1 demonstrates how an instance is represented using `clauses` and `variables`. The sample instance has 2 clauses and 3 variables. In the figure, one can see the mapping `clauses`. It maps the index 1 to a representation of clause C_1. The clause C_1 itself is a mapping as well. It maps 10 to `true` as the variables x_{10} appear positively in C_1. It maps 20 to `false` as the variables x_{20} appear negatively in C_1. The mapping of the index 2 to the representation of the clause C_2 should be interpreted similarly.

Figure 4.1 shows `variables` too. The data structure `variables` maps the index 10 to a representation of variable x_{10}. The variable x_{10} itself is a mapping as well. It maps 1 to `true` as the variable x_{10} appears positively in C_1. On the other hand, it maps 2 to `false` as the variable x_{10} appears negatively in C_2. The mapping of the indices 20 and 30 is interpreted similarly.

Sample instance	`clauses`		`variables`	
$C_1: (x_{10} \lor \bar{x}_{20})$	1	$10 \rightarrow T$ $20 \rightarrow F$	10	$1 \rightarrow T$ $2 \rightarrow F$
$C_2: (\bar{x}_{10} \lor x_{20} \lor \bar{x}_{30})$	2	$10 \rightarrow F$ $20 \rightarrow T$ $30 \rightarrow F$	20	$1 \rightarrow F$ $2 \rightarrow T$
			30	$2 \rightarrow F$

Figure 4.1: A sample instance and its representation using the data structures `clauses` and `variables`.

The `variables` data structure is crucial for efficient access to all the clauses in which a variable appears, which is an operation heavily used by the algorithm. As a variable appears in $r\alpha$ clauses on the average, this allows a constant average time traversal on the clauses in which a variable appears.

A hash table based mapping for the above structures is a natural way to allow non-consecutive indices for the variables and clauses. It provides general flexibility for future optimizations as well.

4.3.3 Residualization of an instance

Residualization is the operation in which information, regarding the assignment of a given variable to a given truth value, is used to simplify an instance. The simplification is done by:

1. Removing (from the instance) all the clauses satisfied by the variable assignment. These clauses are listed as satisfied.

2. Removing (from the instance) all the literals of this variable that are unsatisfied by its assignment. Each literal is removed from its clause. Each clause that remained without literals at all is removed from the instance – and it is listed as unsatisfied.

The input to the residualization operation is a variable and its assigned truth value. The result of the operation is a simplified instance. The latter instance is the *residual instance* with respect to the original instance, the assigned variable, and its assigned truth value. Usually, we will refer to it simply as the residual instance. It is worth noting that, besides a possible decrease in the number of clauses, typically some of the clauses shortened during the residualization.

To perform the residualization efficiently, we iterate only over the clauses in which the variable appears. These clauses are efficiently available via `variables`, and their number is bounded on the average. The currently assigned variable itself is removed from the `variables` data structure.

If a clause is satisfied by the variable assignment, we remove it from `clauses`. To keep the instance representation consistent, we then go over all the variables appearing in the removed clause at the removal time, except for the currently assigned variable. For each of these variables, we remove the currently removed clause from its associated clauses in `variables`. If no clauses are associated with a variable, it is removed from `variables`.

If a clause, in which the currently assigned variable appears, is not satisfied by the assignment, we remove the literal associated with the assigned variable from the clause. In case no more literals appear in the clause, we remove the clause from `clauses`. In the latter case, as no literal remained in the clause, it is unsatisfied under the current assignment.

4.3.4 The concept of a gain

A gain conveys information regarding the profitability of assigning a given truth value to a given variable, namely the expected increase in the number of satisfied clauses, or alternatively the expected decrease in the number of unsatisfied clauses.

Consider a variable x and a clause C of length l in which x appears. Recall that the length of a clause is the number of literals appearing in it. The probability that C will be satisfied by a random assignment of the l variables appearing in it is a Bernoulli distributed random variable with parameter $p = 1 - 1/2^l$, and expected value $\mu = p = 1 - 1/2^l$. This expected value is the expected number of satisfied clauses by a random assignment, applied degenerately to the single clause C.

If we assign the variable x a truth value, for which C is satisfied, the increase in the expected number of satisfied clauses, with respect to this clause only, is $1/2^l$. It is increases from $1 - 1/2^l$ to 1.

On the other hand, if we assign the variable x a truth value, for which the clause is not satisfied by the literal associated with the variable x in the clause, this literal is no longer relevant in this clause and can be removed from it, leaving the clause with $l-1$ literals. In this case, the expected number of satisfied clauses, for this single clause, decrease from $1-1/2^l$ to $1-1/2^{l-1}$. This amounts to a change of $1/2^l$ in the expected number of satisfied clauses, but this time it is a decrease instead of an increase.

As the overall gain is the expected increase in the number of satisfied clauses in the whole instance, it is the sum of the gains of all clauses. Thus, the overall gain of assigning a variable a given truth value is the opposite of the overall gain of assigning it the other truth value.

This allows us to maintain only the gain of one truth value for each variable, say the gain of `true`. If this gain is positive, we prefer to set the variable to `true`. If it is negative, we prefer to set the variable to `false`. In case it is zero, we have no preference regarding the best truth value for the variable, and thus assign it uniformly at random to either `true` or `false`. The gain of a variable is defined as the larger between the gain of assigning it `true` and assigning it `false`.

4.3.5 Efficient maintenance of gains

Each variable has its own gain, and as the number of variables is large, we should maintain the gains of all the variables in an efficient manner. We need to accommodate the following operations:

1. Find the gain of a given variable.
2. Update the gain of a given variable.
3. Find the largest gain over all variables.
4. Remove the largest gain from the collection of gains.

To allow performing these operations efficiently, our algorithm maintains a compound data structure which we call `gains`, consisting of two elementary data structures.

The first elementary data structure, which we call `gbv` (for "**g**ains **b**y **v**ariables"), indexes the gains by variables, and allows a constant time access to the the gain of any specific variable. In case the variable indices are not necessarily consecutive, we recommend a hash table based mapping of variables to gains, for the above indexing.

The second elementary data structure, which we call `gbm` (for "**g**ains **b**y **m**agnitudes"), indexes the gains by their magnitude, and allows logarithmic time access to the largest gain at any given moment. To this end, one may utilize a balanced binary search tree or a binary heap. As the sorting key is the gain magnitude, this data structure should allow duplicate keys.

An alternative to allowing duplicate keys is to add some tiny noise to each gain. This can be done one time at the beginning, independent for distinct variables. This noise is a useful random tie breaker as well; it allows obtaining different assignments in different executions of the algorithm. Out of multiple executions, one can select the best assignment of all those obtained in them as the solution for the instance.

When operating on `gains`, these two data structures, `gbv` and `gbm`, are used to find, update, and remove gains efficiently. The operations on `gains` are done in such a way that `gbv` and `gbm` are kept synchronized and consistent (partially sorted) all the time. For example, if some gain is found and updated using `gbv`, then its position in `gbm` (say a balanced binary search tree) is updated too to keep `gbm` consistent and synchronized with `gbv`.

4.3.6 Initial gains calculation

The initial gains calculation is the operation of calculating the gain of each of the variables from scratch, and storing them in the `gains` data structure. To this end, we loop over all the variables, one at a time. For each variable, we perform a full direct calculation of the gains as described in Section 4.3.4.

To each of the calculated gains, we possibly add a tiny positive noise. As the gains are multiples of $1/2^r$, this noise should be strictly less than $1/2^r$. Finally, we insert the gain in the `gains` data structure. After inserting all the variables' gains, we can start calculating the solution (truth assignment) for the instance.

4.3.7 Marginalization of gains

At each step of the algorithm, once we have selected an assignment of truth value to the current variable, and before we residualize the instance, we should update the gains of all affected variables. These are all neighbors of the currently assigned variable, namely all variables appearing with it in at least one clause.

This updating process is called *marginalization*. It is done by iterating over all the clauses in which the currently assigned variable appears. For each such clause, we iterate over all its literals, except for the one associated with the currently assigned variable. For each such literal, we consider the variable associated with it.

This variable is a neighbor of the currently assigned variable, and its gain should be marginalized. Note that this may occur several times for a given neighbor, as a variable may be a neighbor of the currently assigned variable in multiple clauses.

As we explicitly keep track only of the gain of `true` of a variable (the gain of `false` is just the opposite value), we only marginalize the gain of assigning the neighboring variable a `true` value. The magnitude of the marginal value is always $1/2^l$, where l is the size of the clause we consider currently. This marginal value should be either added to or subtracted from the gain of the neighboring variables appearing in this clause, as will now be described.

If the currently assigned variable satisfies the currently considered clause, and the neighboring variable appears positively in the clause, the marginal is negative (i.e., subtracted). As the clause is about to be satisfied by the currently assigned variable, we should detract the gain of the neighboring variable, which includes a contribution of $1/2^l$ that is not relevant anymore.

If the neighboring variable appears negatively in the clause, the marginal is positive (i.e., should be added). The clause is about to be satisfied by the currently assigned variable, and the gain of the neighboring variable contains a negative contribution of $1/2^l$ (because it appears negatively in the clause). We should eliminate this negative contribution from the gain of the neighboring variable, as it is not relevant anymore. Thus, we should add to it $1/2^l$.

The other case is where the currently assigned variable does not satisfy the currently considered clause. In this case, we should add $1/2^l$ to the gain of the neighboring variable if it appears positively in the clause, as its importance now is enlarged by this magnitude. If the neighboring variable appears negatively, we should reduce its gain by $1/2^l$, as assigning it a `true` is more harmful now that the clause is going to be shorten by a literal.

Table 4.1 summarizes the (possibly negative) marginal values that should be added to the gain of `true` of a variable neighboring the currently assigned variable. This values should be added to the neighboring variable per clause it appears with the assigned variables, and according the clause length before the assignment and residualization. The gain of `false` of the neighboring

assigned \ neighbor	appears positively in clause	appears negatively in clause
satisfies clause	$-1/2^l$	$1/2^l$
unsatisfies clause	$1/2^l$	$-1/2^l$

Table 4.1: The marginal value to be added to the gain of `true` of a neighboring variable, for each clause that neighbor shares with the currently assigned variable.

variables is updated with the opposite value.

4.3.8 Algorithmic pseudocode

In this section we provide an algorithmic pseudocode for EEMOCE – see Algorithm 4. This pseudocode is detailed and puts emphasis on the algorithmics underlying the algorithm.

In the pseudocode, for a clause C, a variable x appearing in C, and a truth value b,

$$\text{sign}(C, x) = \begin{cases} 1, & x \text{ appears positively in } C, \\ -1, & x \text{ appears negatively in } C, \end{cases}$$

and

$$\text{sat}(C, x, b) = \begin{cases} 1, & \text{when assigning } x \text{ to } b, \text{ the clause } C \text{ is satisfied}, \\ -1, & \text{when assigning } x \text{ to } b, \text{ the clause } C \text{ is unsatisfied}. \end{cases}$$

The pseudocode is composed of several procedures. The main procedure is the first one, starting in line 1, and named EEMOCE. It details the main flow of our EEMOCE algorithm.

EEMOCE starts by calculating the initial gains of all the variables in line 2. This is done by applying the procedure Calculate Gains, starting in line 10. For more information on this procedure, see Section 4.3.6.

Then, while the instance has clauses (line 3) EEMOCE performs the following operations. It finds the best assignment given the gains, namely the variable and truth value that produce the maximal gain among all unassigned variables. This is done by applying the procedure Find Best Assignment starting in line 19. See Section 4.3.4 and Section 4.3.5 for more information on it. If there are several variables providing a maximal gain,

Algorithm 4 The Efficient Exhaustive Method of Conditional Expectations

```
Input: An instance $I$ over $n$ variables $x_1, x_2, \ldots, x_n$, with $m$ clauses
        $C_1, C_2, \ldots, C_m$.
Output: An assignment of truth value for each of the variables.
 1: procedure EEMOCE($I$)
 2:     gains ← CALCULATE GAINS($I$)
 3:     while $I$ has clauses do                          ▷ instance is not empty
 4:         variable, value ← FIND BEST ASSIGNMENT(gains)
 5:         variable ← value                              ▷ assign variable
 6:         gains ← MARGINALIZE GAINS(gains, $I$, variable, value)
 7:         $I$ ← RESIDUALIZE INSTANCE($I$, variable, value, gains)
 8:     end while
 9: end procedure
```

```
10: procedure CALCULATE GAINS($I$)
11:     for each variable $x$ in $I$ do
12:         $g_x^T = 0$                                    ▷ expected gain of $x = T$
13:         for each clause $C$ in $I$ in which $x$ appears do
14:             $g_x^T = g_x^T + \text{sign}(C, x) \cdot 2^{-|C|}$
15:         end for
16:         update gains (i.e., gbv and gbm) with $g_x^T$
17:     end for                       ▷ gbm is partially sorted according to $|g_x^T|$
18: end procedure
```

```
19: procedure FIND BEST ASSIGNMENT(gains)
20:     variable ← a maximum gain variable extracted from gbm, selected
        uniformly at random
21:     if gbv.$g_{variable}^T > 0$ then
22:         value ← $T$
23:     else if gbv.$g_{variable}^T < 0$ then
24:         value ← $F$
25:     else
26:         value ← either $T$ or $F$, selected uniformly at random
27:     end if
28: end procedure
```

```
29: procedure MARGINALIZE GAINS(gains, $I$, $variable$, $value$)
30:     for each clause $C$ in $I$, in which $variable$ appears do
31:         for each variable $neighbor$ in $C$, except for $variable$ do
32:             gbv.$g^T_{neighbor}$ = gbv.$g^T_{neighbor}$ − sat($C$, $variable$, $value$) ∗
                sign($C$, $neighbor$)/$2^{|C|}$
33:             gbm.$g^T_{neighbor}$ = gbv.$g^T_{neighbor}$
34:         end for
35:     end for
36: end procedure
```

```
37: procedure RESIDUALIZE INSTANCE($I$, $variable$, $value$, gains)
38:     for each clause $C$ in $I$, in which $variable$ appears do
39:         if sat($C$, $variable$, $value$) then
40:             remove $C$ from clauses
41:             for each variable $neighbor$ in $C$, except for $variable$ do
42:                 remove $C$ from $neighbor$ in variables
43:                 if $neighbor$ has no clauses then
44:                     remove $neighbor$ from variables and gains
45:                     assign $neighbor$ to either $T$ or $F$, uniformly at random
46:                 end if
47:             end for
48:         else
49:             remove $variable$ from $C$ in clauses
50:             if $C$ has no variables then
51:                 remove $C$ from clauses
52:             end if
53:         end if
54:     end for
55:     remove $variable$ from variables
56: end procedure
```

EEMOCE selects one of them uniformly at random. In line 5, EEMOCE assigns this best variable to its best truth value.

After that, some updates to the gains and to the instance are done. In line 6, the gains are marginalized with respect to the instance, the assigned variable and its truth value. This is done by applying the procedure Marginalize Gains starting in line 29. An elaborated description of the calculation done by this procedure can be found in Section 4.3.7, and specifically in Table 4.1.

In the last statement of the while loop, in line 7, the instance is residualized, with respect to the assigned variable and its truth value, by applying the procedure Residualize Instance starting in line 37. For more information on this procedure, see Section 4.3.3 and Section 4.3.2.

4.3.9 Time complexity

In this section we provide a time complexity analysis for EEMOCE assuming the commonly used setting for which the clause length r and the density α are constant, while the number of variables, n, grows larger. Given this setting, the size of an instance is $mr = \alpha nr = \Theta(n)$.

Note first that the way we represent an instance allows constant time access to the variables appearing in a given clause, and to the clauses in which a given variable appears. Also, note that the data structure for maintaining the gains allows constant time access to the gain of a given variable. In this context, access is also for updating (insertion and removal).

The key data structure for an efficient implementation of EEMOCE is `gbm`. This data structure allows accessing the largest gain at any given moment. To this end, one may utilize a balanced binary search tree. This allows logarithmic time access to the largest gain at any given moment, as well as logarithmic time updating of any of the gains.

As the sorting key is the gain magnitude, this data structure should allow duplicate keys. An alternative option is to add some tiny noise to each gain (one time at the beginning) to eliminate duplication. This noise is useful for tie breaking as well. It allows us performing random tie breaking in constant time throughout the algorithm. The noise can be added during the initialization of gains without increasing the time complexity.

EEMOCE starts by calculating the initial gain of each of the n variables in the instance, in line 2, by executing the procedure Calculate Gains.

In this procedure, the statement in line 14 is performed exactly mr times, once for each literal in each clause. Thus, this line contributes $O(n)$ to the time complexity of the procedure.

Another line we should consider is line 16. This line is performed exactly n times. Each time it is performed contributes $O(\log n)$, and overall this line contributes $O(n \log n)$ to the time. Thus, the overall time complexity of Calculate Gains is $O(n \log n)$.

The main loop of EEMOCE is executed at most n times, as each execution concludes in an assigned variable. Once a variable is assigned, it never changes its value again, and its consideration is finished. We note that, in some of the procedures, we find it more effective for the analysis to consider the number of times each literal or clause is inspected.

The procedure Find Best Assignment is executed at most n times. Its extraction operation in line 20 is performed in $O(\log n)$. Thus, the overall time complexity of Find Best Assignment is $O(n \log n)$.

Next, consider the procedure Marginalize Gains. Lines 33 takes $O(\log n)$. It is performed at most $mr(r-1) = O(n)$ times, as it is performed at most $r-1$ times for each literal. Thus, throughout the whole execution of EEMOCE, the overall time complexity of Marginalize Gains is $O(n \log n)$.

Finally, consider the procedure Residualize Instance. Line 42 is performed at most mr times throughout the whole execution of EEMOCE, as a clause can be removed one time from each component of `variables` it appears in. Thus, this line contributes $O(n)$ to the time complexity of EEMOCE.

Line 44 is performed at most n times. The removal of the neighbor from `gains`, and specifically from `gbm`, may take $O(\log n)$. But, performing the removal lazily, by only indicating it using `gbv`, the time can be reduced to $O(1)$. In this case, in line 20, each of the extracted variables should be verified as unremoved using `gbv`. A variable indicated as removed should be skipped, and the next maximal gain variable should be extracted instead of it, until an unremoved variable is encountered. Thus, this line contributes $O(n)$ to the time complexity of EEMOCE as well.

Line 51 can be performed at most m times. Altogether, throughout the whole execution of EEMOCE, the overall time complexity of Residualize Instance is $O(n)$.

Altogether, the worst-case time complexity of EEMOCE is $O(n \log n)$. In our implementation, we have used a balanced binary search tree for `gbm`, as it is commonly used and available. Thus, we get the above time complexity. Yet, it should be noted that the time complexity of EEMOCE may be reduced to linear time in the typical/common case.

There are two observations making this improvement possible. The first is that the logarithmic factor in the time complexity of EEMOCE is due to `gbm`. The second is that the number of possible gains at each given moment is typically very limited. Our experiments show that, even for $n = 1000000$, $r = 10$ and $d = 10$, there are approximately 50 distinct gain magnitudes at any given moment. For more information regarding the latter observation, see Section 4.6.2.

Thus, for each of the distinct gain magnitudes, introduce a bucket that holds all the variables with this gain. Then, use the logarithmic data structure `gbm` to maintain the buckets instead of each of the gains directly. As the number of buckets at any given moment is very small, this eliminates the logarithmic factor related to the usage of `gbm`.

4.4 Practical performance analysis

In this section, we perform a comparative evaluation of EEMOCE versus its baseline algorithm MOCE. We focus on densities commonly used in practice. We compare the performance of the algorithms by comparing the number of unsatisfied clauses provided by each of them. This comparison is complemented by comparing their runtimes.

We conducted experiments on several families of random instances. A family is defined by three parameters: r, α, and n. 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.

Instances are constructed as follows. The clauses are selected independent 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.

First, we focus on random Max 3-Sat, with the number of variables ranging from 1000 to 1000000, and the density from 3 to 10. Such ranges allow us to study the performance of the algorithms at hand both below and above

α / n	4		5		7		10	
	MOCE	EEMOCE	MOCE	EEMOCE	MOCE	EEMOCE	MOCE	EEMOCE
1000	90	25	149	63	288	172	526	377
10000	899	250	1492	633	2886	1719	5262	3765
100000	8995	2508	14943	6331	28856	17199	52629	37646
1000000	89915	25056	149438	63316	288588	171955	526267	376454
% Unsat	2.25%	0.63%	2.99%	1.27%	4.12%	2.46%	5.26%	3.76%

Table 4.2: The average number of clauses unsatisfied by MOCE and EEMOCE, for random Max 3-Sat instances.

the satisfiability threshold density (which is approximately 4.27 for Max 3-Sat [73, 31]). We executed both MOCE and EEMOCE on 1000 instances of each of the families, and recorded the number of clauses unsatisfied by them.

Table 4.2 presents the average number of clauses unsatisfied by MOCE and EEMOCE. The rows of the table (but the last) record the number of variables, while the columns record the densities. For the sake of readability, all the numbers in those rows are rounded to the nearest integer.

It turns out that the average number of clauses unsatisfied by each of the algorithms scales linearly with the number of clauses, and thus can be described as a proportion of the number of all clauses, for any fixed density. This proportion is provided in the last row of the table. In fact, in the case of MOCE, the empirical results align with [29, Theorem 5]. The data in the table indicates that the same holds for EEMOCE.

Given this property of the performance of both MOCE and EEMOCE, we empirically explored it further. Figure 4.2 provides graphs of the average proportion of clauses unsatisfied by each of the two algorithms for $r = 3$ and α ranging from 3 to 10 in steps of 0.25. For each density, we provide the average proportion of unsatisfied clauses over 1000 instances.

Next, in Table 4.3, we provide the standard deviation of the number of clauses unsatisfied by the two algorithms, for each of the families. As one can see, each standard deviation is much smaller than the respective average. The ratio between each standard deviation and the corresponding average decreases as $1/\sqrt{n}$. Thus, the averages provided in Table 4.2 are representative. Unsurprisingly, the standard deviation in the number of clauses unsatisfied by EEMOCE is smaller than for MOCE.

Considered as function of n, the standard deviation is proportional to $\sqrt{n}$ (up to noise). Thus, in the last row of the table we provide the ratio between

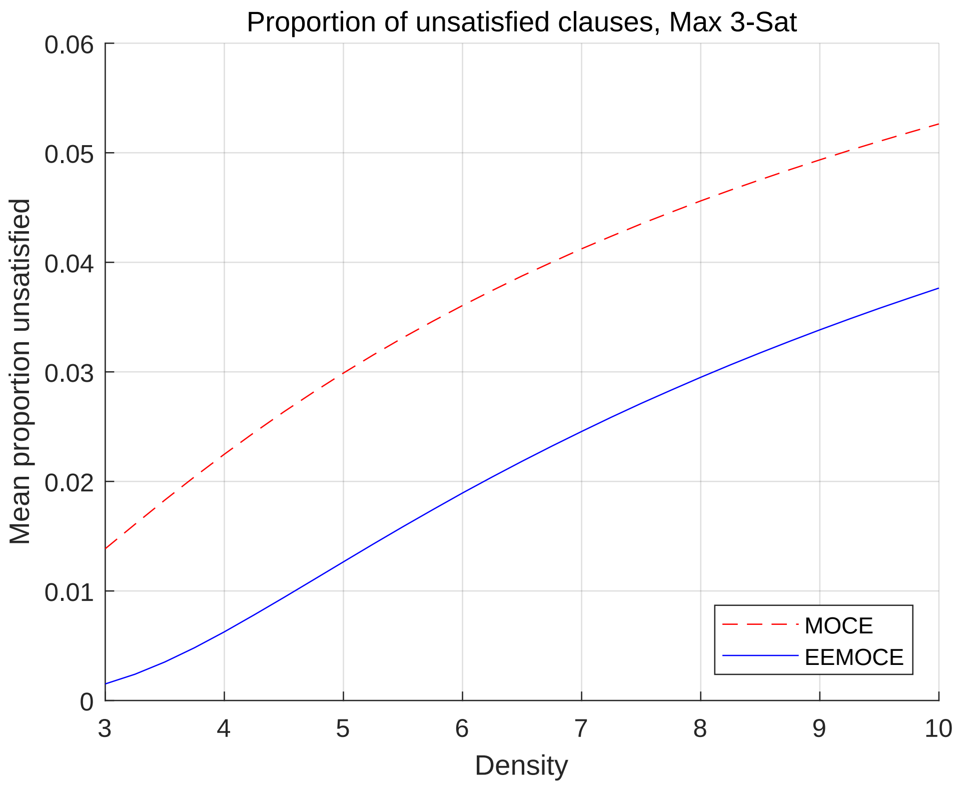


Figure 4.2: The average proportion of clauses unsatisfied by MOCE and EEMOCE, for random Max 3-Sat instances.

n \ α	4		5		7		10	
	MOCE	EEMOCE	MOCE	EEMOCE	MOCE	EEMOCE	MOCE	EEMOCE
1000	7.47	4.14	9.33	5.46	11.82	7.98	15.70	10.91
10000	23.86	12.40	29.27	17.85	37.09	24.64	48.79	33.48
100000	70.99	39.85	93.21	54.91	118.97	83.84	149.80	103.53
1000000	231.12	124.60	284.84	176.04	372.75	248.94	494.16	346.09
$\sigma/\sqrt{n}$	0.2386	0.1240	0.2927	0.1785	0.3709	0.2464	0.4879	0.3348

Table 4.3: The standard deviation of the number of clauses unsatisfied by MOCE and EEMOCE, for random Max 3-Sat instances.

the standard deviation and $\sqrt{n}$. We refer to this ratio as the normalized standard deviation. For given values of r and α, it turns to be basically fixed (up to noise).

As in the case of the mean, we see that the standard deviation follows a well-defined pattern. Thus, we are able to provide an approximation to the standard deviation. Figure 4.3 provides graphs of the normalized standard deviation of the number of clauses unsatisfied by each of the algorithms, for $r = 3$ and α ranging from 3 to 10 in steps of 0.25. For each of the densities,

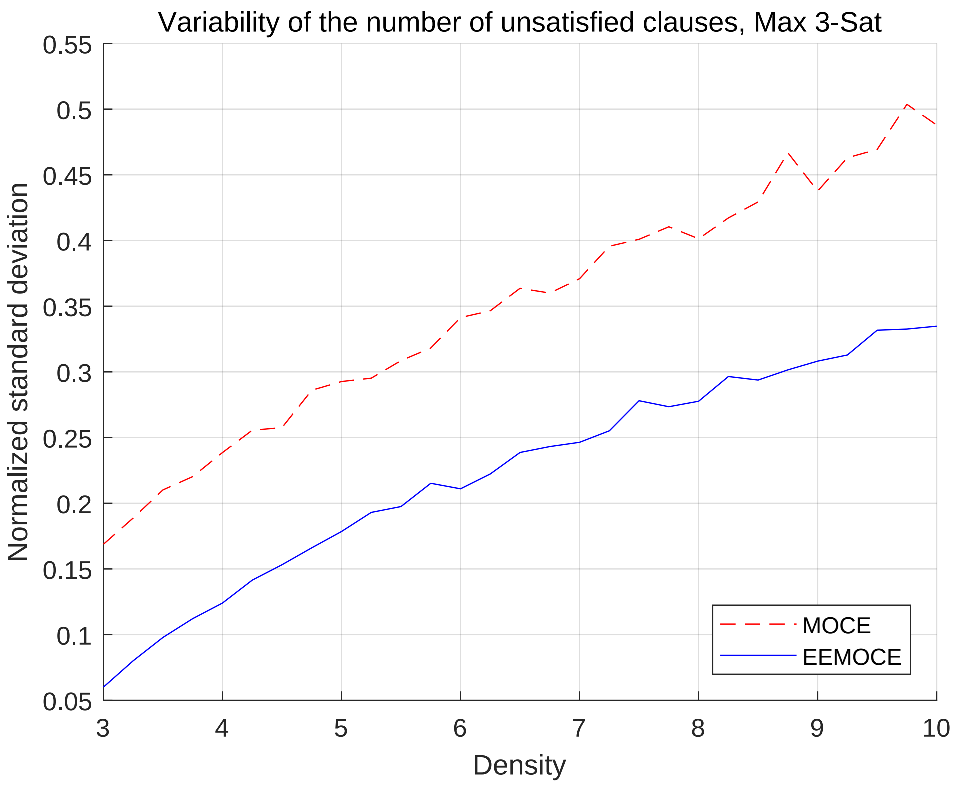


Figure 4.3: The normalized standard deviation of the number of clauses unsatisfied by MOCE and EEMOCE, for random Max 3-Sat instances.

we provide the normalized standard deviation of the number of unsatisfied clauses, based on 1000 random instances.

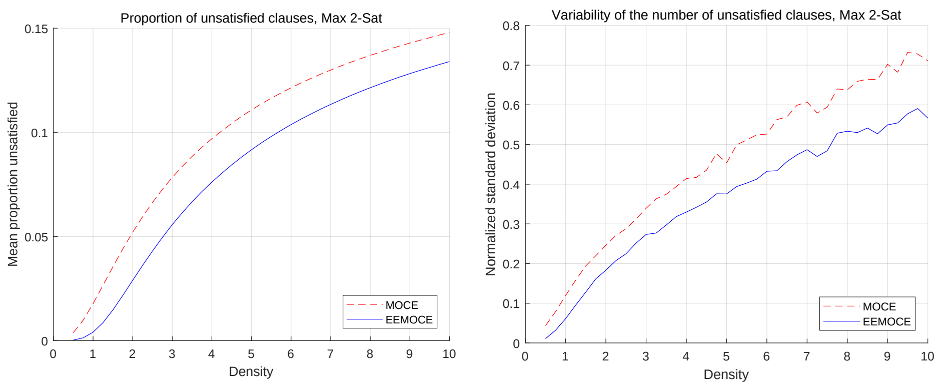
Given the advantage of EEMOCE over MOCE, one may want to consider the runtime overhead EEMOCE introduces. Besides the analysis of the time complexity analysis provided in Section 4.3.9, we measured the actual runtimes of both algorithms on the families we studied. Table 4.4 lists these runtimes. We provide the mean runtime over the 1000 instances as well as the standard deviation (in parenthesis). The table is wrapped for readability.

Examining the runtimes, one can see the EEMOCE does introduce an overhead over MOCE. Yet, overall, the prolonged runtimes seem to be reasonable, given the significant reduction in the number of unsatisfied clauses.

We have also examined instances of Max 2-Sat. The pattern of performance, whereby the proportion of unsatisfied clauses is independent of n, holds here as well, meaning this is a general property of MOCE and EEMOCE. As the satisfiability threshold density is 1 for Max 2-Sat [41],

n \ α	4		5	
	MOCE	EEMOCE	MOCE	EEMOCE
1000	0.096 (0.011)	0.118 (0.016)	0.102 (0.010)	0.130 (0.012)
10000	0.381 (0.031)	0.590 (0.043)	0.443 (0.035)	0.704 (0.055)
100000	3.183 (0.217)	6.142 (0.386)	3.674 (0.231)	7.416 (0.523)
1000000	27.485 (3.571)	78.228 (10.115)	33.877 (5.134)	98.595 (14.029)
n \ α	7		10	
	MOCE	EEMOCE	MOCE	EEMOCE
1000	0.118 (0.012)	0.157 (0.011)	0.138 (0.0128)	0.186 (0.017)
10000	0.559 (0.044)	0.901 (0.056)	0.721 (0.052)	1.199 (0.078)
100000	4.815 (0.346)	10.130 (0.824)	6.198 (0.459)	13.378 (0.994)
1000000	47.945 (8.144)	134.558 (21.945)	99.723 (11.939)	230.167 (31.715)

Table 4.4: The average and standard deviation of the runtime (CPU seconds) of MOCE and EEMOCE, for random Max 3-Sat instances.



(a) Average proportion.

(b) Normalized standard deviation.

Figure 4.4: Clauses unsatisfied by MOCE and EEMOCE, for random Max 2-Sat instances.

the range of densities here was taken to start at $\alpha = 1/2$.

Figure 4.4 depicts the average proportion of clauses unsatisfied by each of the algorithms, as well as the normalized standard deviation of the number of clauses unsatisfied by each of the algorithms. These quantities are provided for α ranging from 0.5 to 10 in steps of 0.25. For each density, these quantities are based on 1000 random instances. Our observations in the case of $r = 3$ are true here as well.

Finally, Table 4.5 lists the runtimes of the two algorithms on several families. We provide the mean runtime over 1000 random instances, for each density and algorithm.

The experiments described in this section, and the following ones, were

n \ α	4		5		7		10	
	MOCE	EEMOCE	MOCE	EEMOCE	MOCE	EEMOCE	MOCE	EEMOCE
1000	0.088	0.105	0.095	0.113	0.103	0.130	0.118	0.153
10000	0.327	0.456	0.373	0.522	0.458	0.660	0.578	0.852
100000	2.745	4.215	3.203	4.882	3.873	6.433	4.750	8.225
1000000	20.603	44.818	25.029	55.982	36.624	78.228	53.743	115.674

Table 4.5: The average runtime (CPU seconds) of MOCE and EEMOCE, for random Max 2-Sat instances.

executed on a Sun Grid Engine (SGE) [102] 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, we had 744 computation cores and 1984GB of working memory at hand.

Each of the executions was limited to use up to 8GB of working memory. We used Java to implement the algorithms, both MOCE and EEMOCE, and other elements related to the solving process, like instance loading, etc. The experiments were conducted using Python scripts, and the results were analyzed and visualized using Matlab.

It is worth emphasizing that the infrastructure we used, the limit on the working memory, and the implementation language, affect the running times of the algorithms (provided in Table 4.4 and Table 4.5), but not their performance (the number of unsatisfied clauses).

4.5 Asymptotic performance analysis

In this section we analyze the asymptotic performance of EEMOCE. Namely, we consider the mean proportion of clauses unsatisfied by assignments returned by EEMOCE as the density grows larger.

As seen in the previous section, the number of variables, n, does not affect the mean proportion of clauses unsatisfied by assignments provided by EEMOCE. Thus, we omit this parameter from the analysis, as it is meaningless in this respect. Clearly, this simplifies the analysis, as the behaviour depends only on two parameters: the clause length r, and the density α.

To consider the performance of EEMOCE, we utilize two extreme reference points. The first is the mean proportion $1/2^r$ of clauses unsatisfied by a random assignment. This reference point gives an inferior performance relative to the other algorithms presented here. The second reference point is a theoretical lower bound on the mean proportion of clauses unsatisfied by an optimal assignment [29]. It yields a (perhaps unachievable) superior performance. We will also add information on a theoretical upper bound on this mean proportion [29], when we find it relevant.

We first explicitly state the result regarding the theoretical bounds. In the following theorem, $f_r(n, \alpha n)$ denotes the mean *number* of *satisfied* clauses by an optimal assignment, taken over all instances for which the clause length is r, the density is α, and the number of variables is n. We write $f(n) \lesssim g(n)$ if $\limsup_{n\to\infty} f(n)/g(n) \leq 1$. Note that, if this asymptotic inequality holds, it is still possible even that $f(n) > g(n)$ for all n. The notation $f(n) \gtrsim g(n)$ is defined similarly. Finally, $o_\alpha(1)$ denotes a quantity which is arbitrarily close to 0 for all α sufficiently large.

Theorem 4 (Coppersmith et al., Theorem 15 in [29])**.** *For all clause length r, for density α large,*

$$f_r(n, \alpha n) \gtrsim \left(\frac{2^r - 1}{2^r}\alpha + \frac{1}{r+1}\sqrt{\frac{\alpha r}{\pi 2^r}}\,(1 - o_\alpha(1))\right) n \tag{4.1}$$

$$f_r(n, \alpha n) \lesssim \left(\frac{2^r - 1}{2^r}\alpha + \sqrt{\alpha}\sqrt{\frac{(2^r - 1)\ln 2}{2^{2r-1}}}\right) n \tag{4.2}$$

We note that, for the specific case of $r = 2$, a better version of (4.1) is stated in [29, Theorem 5]. Yet, we have chosen to use the above generalized theorem, as we use it for various values of r along this section, and as for the reference points we use (4.2).

As we are interested in the mean *proportion* of clauses *unsatisfied* by an optimal assignment, our lower bound is taken as the number of all clauses minus the upper bound on $f_r(n, \alpha n)$ stated in (4.2), divided by the number of all clauses. The upper bound on our quantity of interest is calculated in a similar manner.

As in the previous section, we first focus on Max 3-Sat. We fixed the number of variables to be $n = 1000$. We let the density α vary from 20 to 1000, first in steps of 10, and after reaching 100 in steps of 100. For each density, we selected 1000 instances uniformly at random. We executed both MOCE and EEMOCE on each of these 1000 instances, and recorded the proportion of clauses unsatisfied by them for each of the instances.

Table 4.6 presents the mean proportion of clauses unsatisfied by MOCE

α	mean OPTIMUM lower bound	mean EEMOCE	mean MOCE	mean OPTIMUM upper bound
20	3.79%	6.03%	7.16%	8.64%
30	5.39%	7.13%	8.07%	9.35%
40	6.34%	7.80%	8.62%	9.77%
50	6.99%	8.27%	9.01%	10.06%
60	7.47%	8.62%	9.30%	10.27%
70	7.85%	8.89%	9.53%	10.44%
80	8.15%	9.12%	9.71%	10.57%
90	8.40%	9.30%	9.87%	10.68%
100	8.61%	9.46%	10.00%	10.77%
200	9.75%	10.33%	10.71%	11.28%
300	10.25%	10.72%	11.03%	11.50%
400	10.55%	10.95%	11.22%	11.64%
500	10.76%	11.12%	11.36%	11.73%
600	10.91%	11.23%	11.46%	11.79%
700	11.03%	11.33%	11.53%	11.85%
800	11.12%	11.40%	11.59%	11.89%
900	11.20%	11.47%	11.65%	11.92%
1000	11.27%	11.52%	11.69%	11.95%

Table 4.6: The asymptotics of the average proportion (percent) of clauses unsatisfied by MOCE, EEMOCE, and OPTIMUM, for random Max 3-Sat instances.

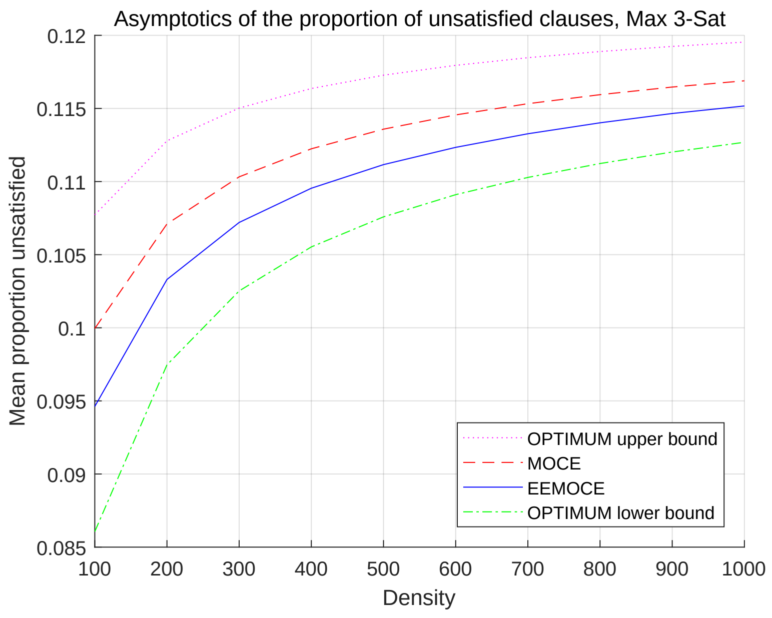


Figure 4.5: The asymptotics of the average proportion of clauses unsatisfied by MOCE, EEMOCE, and OPTIMUM, for random Max 3-Sat instances.

and EEMOCE. We also add to the table the lower and upper bound on the proportion of clauses unsatisfied by an optimal assignment. These bounds are calculated as described above, based on Theorem 4.

For the sake of readability, in the table, we provide the proportions as percentages. Also, the table may be thought of as containing an additional ghost column on the right, with the mean number of clauses unsatisfied by a random assignment. We have omitted this column, as all its entries would be 12.5%.

Figure 4.5 depicts the same quantities for densities from 100 to 1000. As one may see, all the proportions converge gradually as the density grows larger. This convergence aligns with Theorem 4.

How should one measure the asymptotic performance of an algorithm for Max r-Sat? A seemingly reasonable way to do this is by considering the mean proportion of clauses unsatisfied by the assignment returned by this algorithm, and comparing it to the mean proportion of clauses unsatisfied by a random assignment, $1/2^r$. For example, if some algorithm unsatisfies on the average a proportion of $0.8 \cdot 1/2^r$ of all clauses, we could say that it yields an improvement of $0.2 \cdot 1/2^r$.

Unfortunately, this way of measuring the improvement is not suitable for our case. In fact, from (4.2) it follows that, given any $\varepsilon > 0$, if the density is sufficiently large, even the optimal assignment leaves at least a proportion of $1/2^r - \varepsilon$ of the clauses unsatisfied on the average.

We note in passing that the situation is similar on the other side. Namely, even the pessimal assignment will have, on the average, a proportion of less than $1/2^r + \varepsilon$ of the clauses unsatisfied. In other words, for sufficiently large densities, the mean proportion of unsatisfied clauses will be almost the same for all algorithms. As the density grows larger, the improvement with respect to the mean drops to zero, for all algorithms – making this measure useless for asymptotic analysis.

Due to this diminishing improvement phenomenon, we introduce two measures for comparing the asymptotic performance of algorithms for this problem. The first measure is the ratio between the distances of the numbers of clauses unsatisfied by the algorithms from the mean, minus 1. More formally, suppose we have two algorithms A_1 and A_2. Denote by a_1 the mean number of clauses unsatisfied by A_1, by a_2 the mean number of clauses unsatisfied by A_2, and by μ the mean number of clauses unsatisfied by a random assignment. Then, the measure we suggest is $(\mu - a_1)/(\mu - a_2) - 1$. We recommend taking the seemingly better algorithm as A_1.

This measure neutralizes the vanishing of the improvement, and focuses on the relative distances from the mean of the two competing algorithms. Note that the measure may well be close to 0, which means that A_1 and A_2 are of very similar performances.

The second measure we suggest is similar to the first. However, instead of taking μ as a reference point for measuring distances, we take the lower bound on the number of clauses unsatisfied by an optimal assignment as our reference point. Denoting this lower bound by l, the second measure we suggest is $1-(a_1-l)/(a_2-l)$. Here too, we recommend taking the seemingly better algorithm as A_1. In both measures, we take the differences so as to make the measures positive.

Both measures gives us numbers that quantify by how much algorithm A_1 improves the distance of algorithm A_2 from the underlying reference point. The larger the number, the larger is the improvement. In the first measure, this number may be larger than 1, while in the second measure it is bounded by 1. Both measures give a value of 0, when an algorithm is compared to itself.

We name the first measure "Improvement of the Gap from the Mean", or in short "IGM". We name the second measure "Improvement of the Gap to the Optimum", or in short "IGO". Figure 4.6 depicts the asymptotic improvement of EEMOCE over MOCE, measured by IGM and IGO.

The graphs reveal a remarkable phenomenon. As the density grows larger, the improvement of EEMOCE over MOCE does not diminish. Moreover, according to the graphs, it converges to a constant. The IGM of EEMOCE over MOCE converges to approximately 21%, while the IGO of EEMOCE over MOCE converges to approximately 41%. Thus, asymptotically, EEMOCE improves MOCE by 21% or 41%, depending on one's viewpoint.

It is worth mentioning that, although the IGO measure seems to be more important than the IGM measure, it has its own drawbacks. While IGM is based on a stable exact reference point, IGO is based on an approximate lower bound of the number of clauses unsatisfied by an optimal assignment. Thus, IGO is sensitive to the inaccuracy of its reference point.

The phenomenon of convergence to a constant of the IGM and IGO

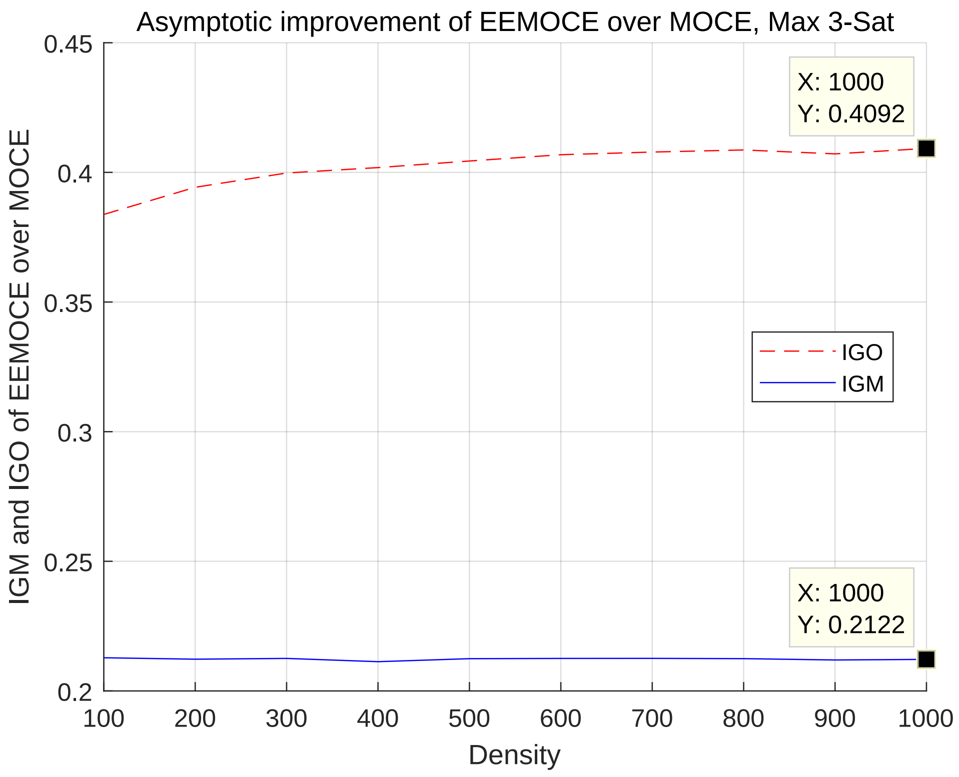


Figure 4.6: The asymptotic improvement of EEMOCE over MOCE, measured by IGM and IGO, for random Max 3-Sat instances.

measures was observed in general, not only for clause length 3. Figure 4.7 provides the IGM and IGO values for commonly used values of r. For each clause length, these measures were calculated based on 1000 random instances, each over 1000 variables and with density 1000. At this high density, the measures are fairly close to the limit. As the proportion of clauses unsatisfied by MOCE and EEMOCE is indifferent to the number of variables – those measures are indifferent to it as well.

We observed a similar convergence pattern for larger clause lengths too. In fact we experimented with clause lengths as large as 10. The convergence is observed clearly in these cases too. Yet, for higher clause lengths, a much larger density is needed to estimate the limiting constant.

We attribute the slower convergence to the fact that, as the clause length gets larger, the satisfiability threshold density grows exponentially. In fact, the threshold is approximately $2^r \ln 2 - (1 + \ln 2)/2$ [28, 73, 75]. While this threshold is 1 for Max 2-Sat [41], and approximately 4.27 for Max 3-Sat

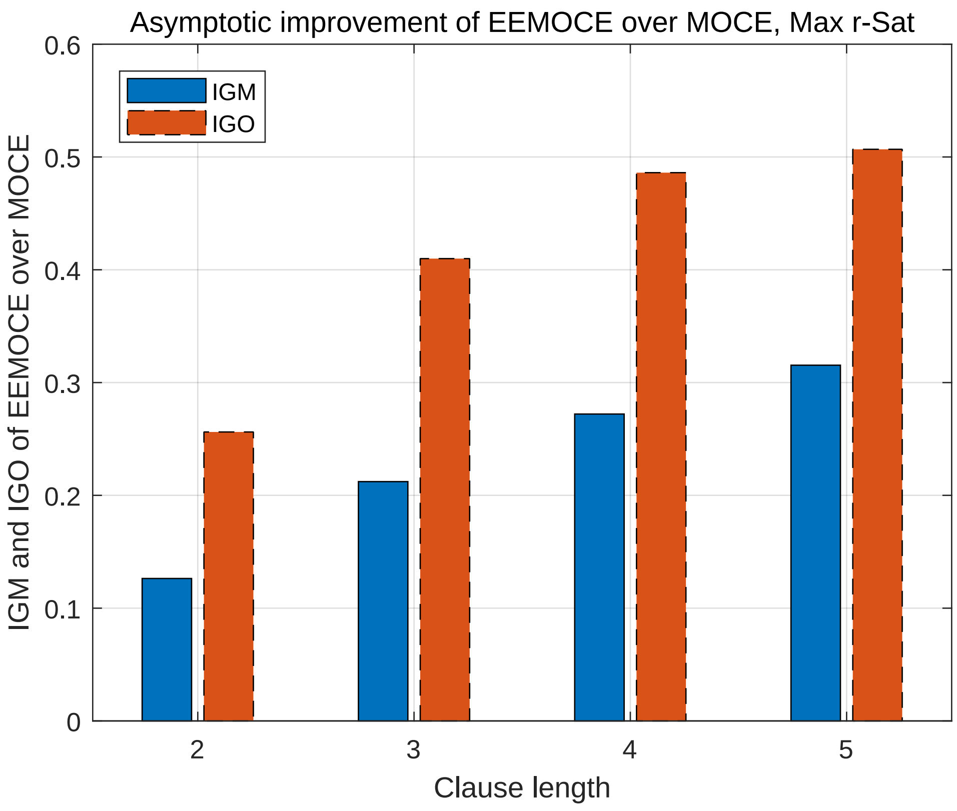


Figure 4.7: The asymptotic improvement of EEMOCE over MOCE, measured by IGM and IGO, for random Max r-Sat instances.

[73, 31], for Max 7-Sat it is already about 88, and it reaches about 709 for Max 10-Sat. Thus, the density we used is not large enough to estimate the limiting constant of the convergence.

We conclude this section by introducing yet another measure for asymptotic performance analysis for Max r-Sat. We name this measure "Intra Gap Location", or in short "IGL". In this measure we use both reference points μ and l. We consider the gap $\mu - l$, and measure the percentage of this gap that is closed by the analyzed algorithm. Namely, denoting by a the mean number of clauses unsatisfied by the analyzed algorithm, our suggested measure is $(\mu - a)/(\mu - l)$.

Assuming the algorithm provides a performance at least as good as the mean μ, IGL yields a number between 0 to 1. The larger this number, the better is the algorithm. Note that, as l is only an approximate lower bound on the optimum, it is not necessarily possible to reach 1, and that one may

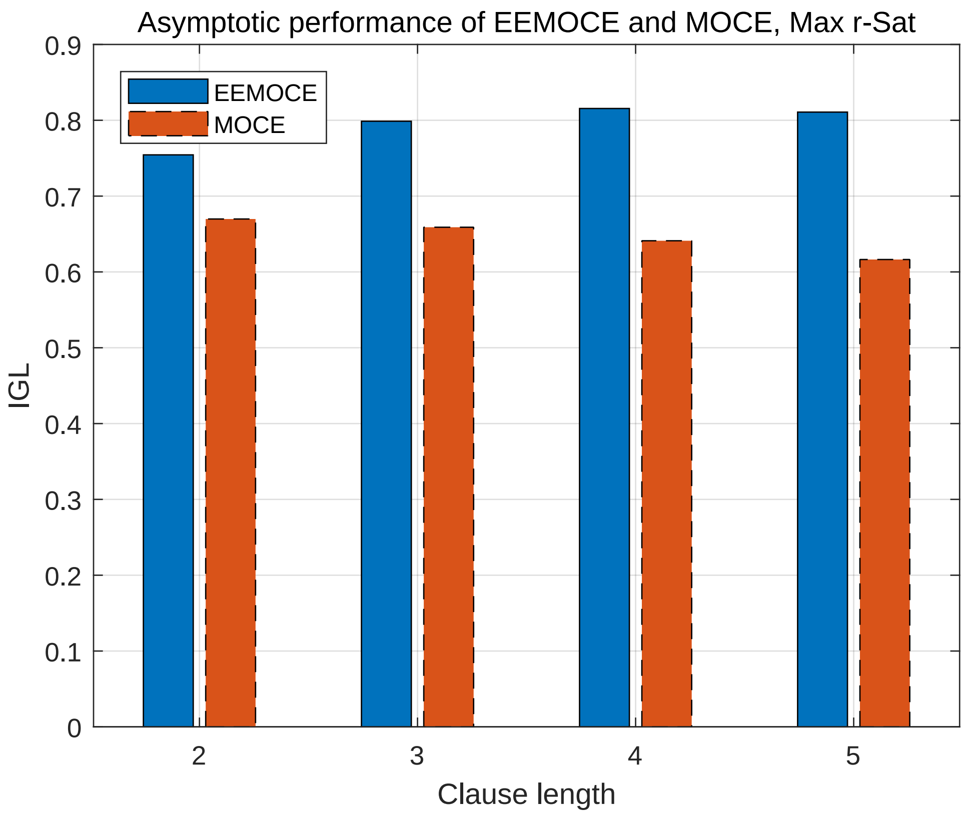


Figure 4.8: The asymptotic performance of EEMOCE and MOCE, measured by IGL, for random Max r-Sat instances.

get a number larger than 1 as well. Also, in the untypical case where the algorithm's performance is worse than the mean μ, this measure yields a negative value.

For each clause length, we calculated this measure based on the same 1000 random instances used before (each over 1000 variables and with density 1000). At this high density, this measure is fairly close to the convergence limit, and is indifferent to the number of variables due to the indifference of MOCE and EEMOCE to it.

Figure 4.8 depicts the IGL of MOCE and EEMOCE for commonly used values of r. As one can see, MOCE closes about 65% of the gap to the optimum, while EEMOCE closes about 80% of it. As we use a lower bound on the optimum, the percentage of the gap that has been closed may be even higher.

4.6 Execution analysis

In this section, we empirically analyze the execution of EEMOCE. We provide details and insights on its main algorithmic aspects and illustrate them. In Section 4.6.1, we compare the expected proportion of unsatisfied clauses during the assignment process, for MOCE and EEMOCE.

In Section 4.6.2, we analyze the number of variables that are candidates for being assigned at each step, namely the number of variables having the same maximal gain at each step during an EEMOCE execution. Based on this, we point out further possible improvements of EEMOCE. Section 4.6.3 illustrates the execution of EEMOCE on many instances and analyzes the distribution of clause lengths during the assignment process.

4.6.1 Expected assignment quality during the execution

In order to illustrate the difference between EEMOCE and MOCE, we track the expected gain and expected proportion of unsatisfied clauses as the process of assigning truth values to the variables proceeds. Recall that the gain is the expected decrease in the number of unsatisfied clauses, as a result of a variable assignment (see Section 4.3.4). This expectation is based on the assumption that each variable has a probability of 1/2 of being assigned each truth value. We executed MOCE and EEMOCE on the same Max 3-Sat instance over 1000000 variables, with density 4.

Figure 4.9 shows the gain of the variables as a function of the step at which they are assigned (each variable assignment is considered as a step). Every 1000 steps are averaged in order to get a reasonably smooth function. Since EEMOCE prioritizes the high-gain variables, the gains at the beginning of the run are much higher. The noticeable gain drops are also of interest, and we refer to them in the next section. For MOCE there is a gradual increase in the average gain, caused by the shortening of clauses, which doubles the contribution to the gain of each variable in each shortened clause.

Figure 4.10 depicts the expected number of unsatisfied clauses throughout the execution. Since both MOCE and EEMOCE guarantee a non-negative gain assignment for each variable, both graphs are non-increasing. Initially, both algorithms have the same expectation, since each clause (is considered as if it) has a probability 1/8 for being unsatisfied. However,

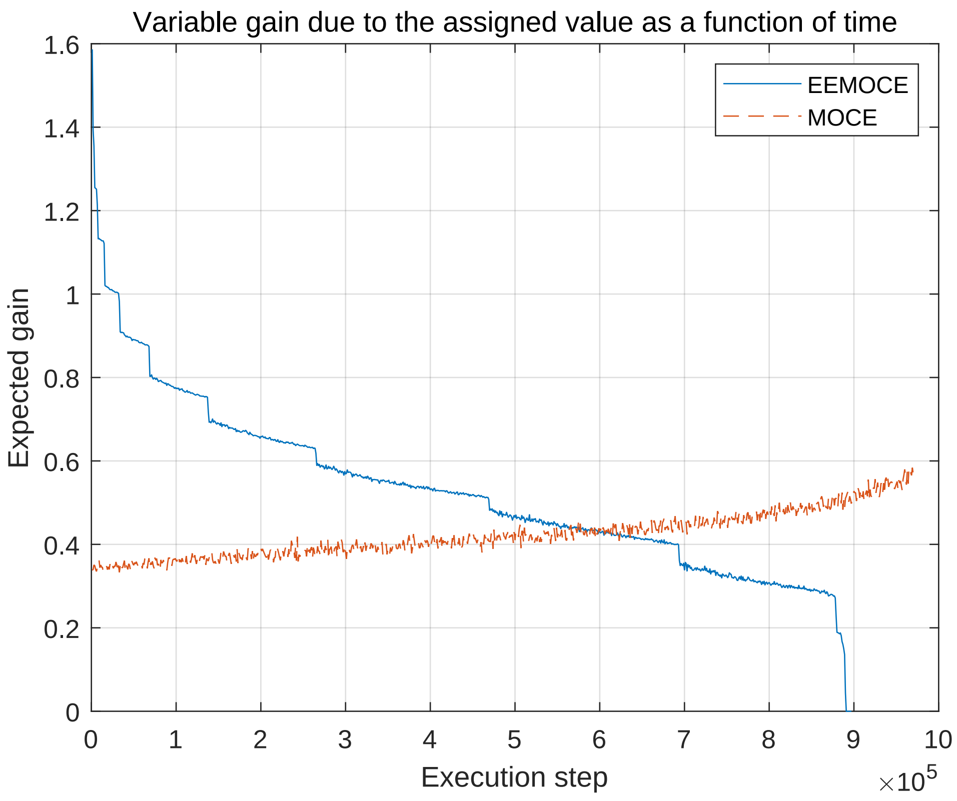


Figure 4.9: The expected variable gain upon its assignment, for MOCE and EEMOCE execution on a typical Max 3-Sat instance over 1000000 variables and with density 4.

since EEMOCE chooses in each step a variable with highest gain, the corresponding absolute slope during most of the execution is higher than that of MOCE. Moreover, choosing the variables in a greedy order also improves the average gain for other variables. In particular, the gain of a balanced variable becomes non-zero after one of its neighbors has been assigned. Thus, the final assignment of EEMOCE is better than that of MOCE.

It is interesting to note that EEMOCE ends after about 900000 steps, which is significantly less than the number of steps MOCE makes. This is due to variables having all of their clauses satisfied by other variables. These variables typically appear in less clauses, and hence have low gain, which means that EEMOCE tends to delay their assignment.

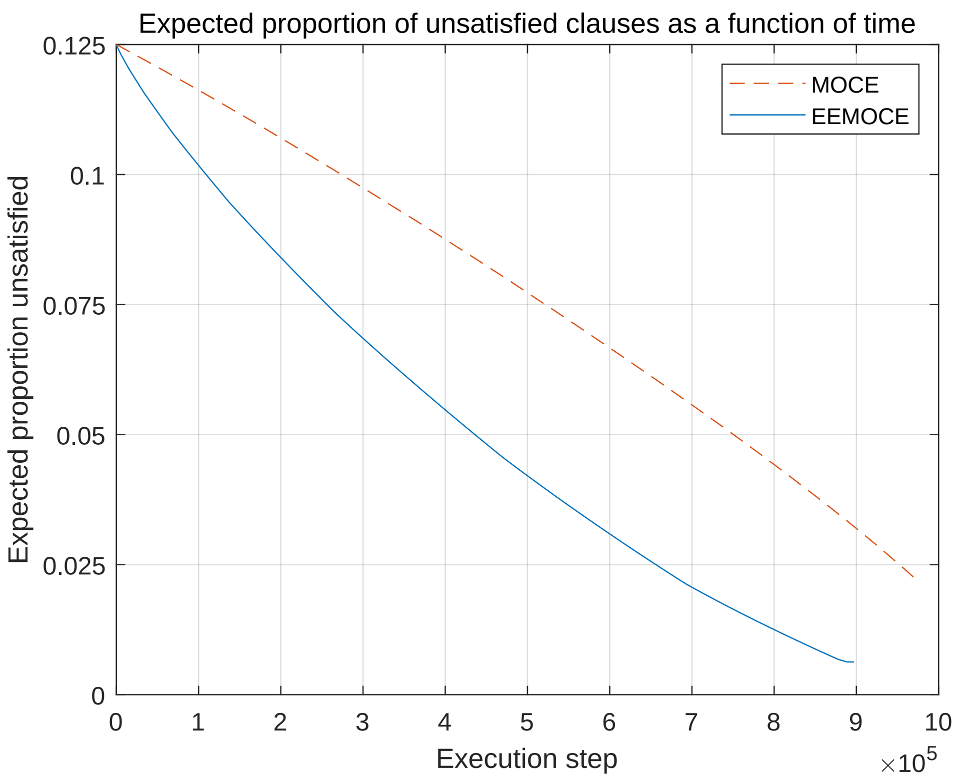


Figure 4.10: The expected proportion of clauses unsatisfied by MOCE and EEMOCE during their execution on a typical Max 3-Sat instance over 1000000 variables and with density 4.

4.6.2 Number of maximal gain variables during the execution

MOCE is considered as a derandomization of the algorithm selecting a random assignment, choosing the assignment value by the estimated gain. EEMOCE may be viewed as a further derandomization of MOCE, fixing the variable assignment order according to a greedy criterion. However, even after taking into account the expected gain, there is some level of randomality, as there are typically several variables having the same maximal gain, and tie breaks are done randomly in EEMOCE.

In this section, we examine the number of variables of maximal gain throughout the execution of the algorithm. The solid blue graph in Figure 4.11 depicts this number for a typical execution of EEMOCE on a Max 3-Sat instance over 1000000 variables, with density 4. Note that this clause length dictates all gains to be multiples of 1/8.

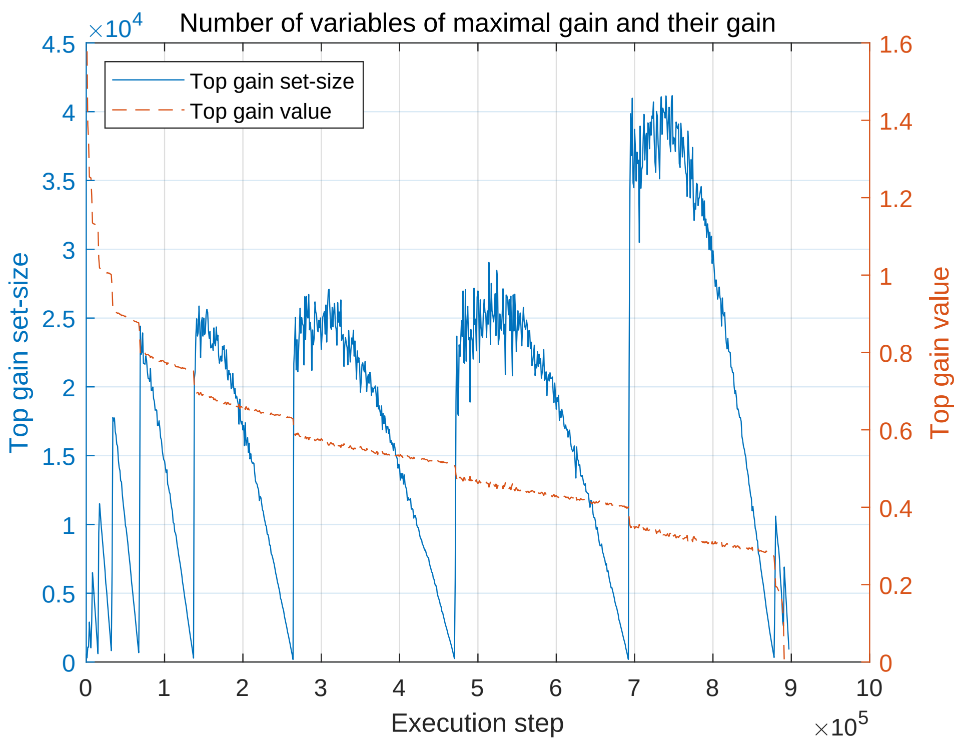
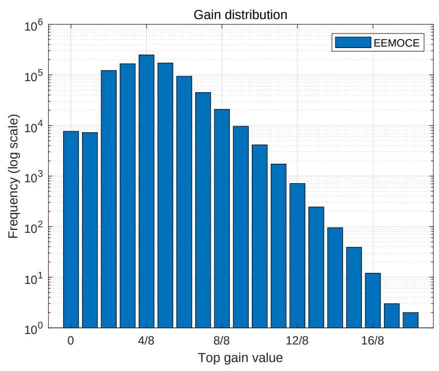


Figure 4.11: The number of variables of maximal gain and their gain throughout an EEMOCE execution on a typical Max 3-Sat instance over 1000000 variables and with density 4.

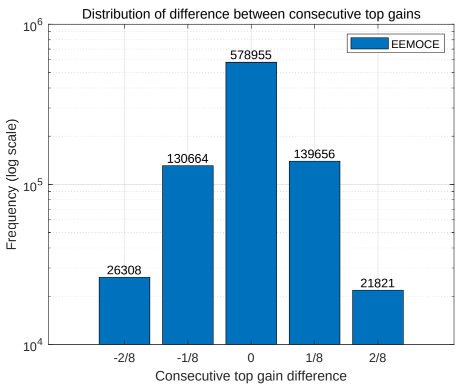
The dashed red graph shows the gain value, and thus explains how the graph from Figure 4.9 interacts with the graph showing the number of variables at the top level. Each gain level can be identified by a jump in the top gain layer size, followed by a gradual descent. The figure shows that the algorithm depletes one gain level after another. During the process, there are up to 40000 variables the algorithm arbitrarily chooses from.

This insight may be a key for improving EEMOCE even more. First, we can further derandomize the assignment order by adding a tie breaking criterion. Second, it is possible to run EEMOCE several times, thus getting different random orders, and choose the execution with the best performance.

It is interesting to examine the relationship between the size of the maximal gain and the number of variables having this gain. In particular, Figure 4.11 shows that, every time a gain level is depleted, there is a noticeable drop in the gain.



(a) Distribution of the top gain.



(b) Distribution of the top gain change.

Figure 4.12: Distribution of the top gain and its change over time throughout an EEMOCE execution on a typical Max 3-Sat instance over 1000000 variables and with density 4.

Figure 4.12a shows the distribution of the maximal gain, which is the actual decrease of the expected number of unsatisfied clauses, over the steps. The gains reflect the imbalance between positive and negative literals of a variable (weighted as a function of the clause lengths).

Since the instances have been generated uniformly at random, high gains are rare. Another important factor is that EEMOCE prioritizes variables by gain. Assignments of variables of low gain are postponed, and their gain may increase by the assignment of other variables. Hence, only a small minority of variables are assigned with zero gain.

In fact, assigning a variable with zero gain happens only when all remaining variables have zero gain. Such variables typically remain in two clauses of length 1, as a literal and its negation (or some other equal number of appearances in each of the two forms). Other variables of gain 0 will be very seldom assigned a value, as there will be a neighbor, or another variable, with a positive gain to be assigned first.

Figure 4.12b shows the distribution of gain changes between consecutive steps. A gain change is the difference between the gain at a given step and the gain at the preceding step.

In more than half of the steps there is no change, indicating that the corresponding variables belonged to the same gain level. Since a variable assignment changes the gain of neighboring variables, some of these variables may rise to have a higher gain than the variable just assigned. This causes an increase, usually followed by an immediate decrease, of the maximal gain, as the algorithm returns to its former gain level. This explains the rough symmetry between positive and negative maximal gain changes.

Typically, the changes in the maximal gain tend to occur after we descend from some level to the next lower level. Since now there are many variables at the top level, it happens often that, assigning a value to some variable, one of the affected neighbors is also a top level variable, which may now rise to a higher level. This corresponds neatly to the fact that the graph in Figure 4.11 is noisy at the beginning of each “hill” and becomes smoother towards the next descent.

Note that a gain change of 2/8 is rare. It usually occurs when we assign some variable, having a neighbor of top level, sharing with it a clause of length 2. The change of the neighbor’s gain is 2/8, and if it happens to be an upward change, the top level gain will increase by 2/8 in a single step. This happened in fact 21821 times during the run, and explains most of the 26308 double decreases in Figure 4.12b.

4.6.3 Clause length distribution during the execution

In this section we consider another aspect of the execution of EEMOCE. When executing EEMOCE with any clause length r, the number of clauses of length r decreases throughout the run. Clauses of shorter lengths are more interesting to observe. We start with no clauses of length $1, 2, \ldots, r-1$. In the initial stages, some of the length r clauses will be satisfied and removed from the instance, while those yet unsatisfied by the partial assignment will shorten to length $r-1$. After a while, when we will give values to some of the variables in these $(r-1)$-length clauses, we will start seeing clauses of length $r-2$, and so on.

An analysis of the expected number of clauses of each length at each stage would be trivial if our algorithm assigned values to all the variables uniformly at random. However, MOCE, and more so EEMOCE, strive to assign values that satisfy more clauses than they leave unsatisfied. Hence, the changes in the distribution of the number of clauses of each length throughout the execution are not easy to analyze (see, for example, the proof of [29, Theorem 5]).

We consider this behavior for instances of Max 2-Sat over 1000 variables with density 5. We have selected 10000 random instances from this family

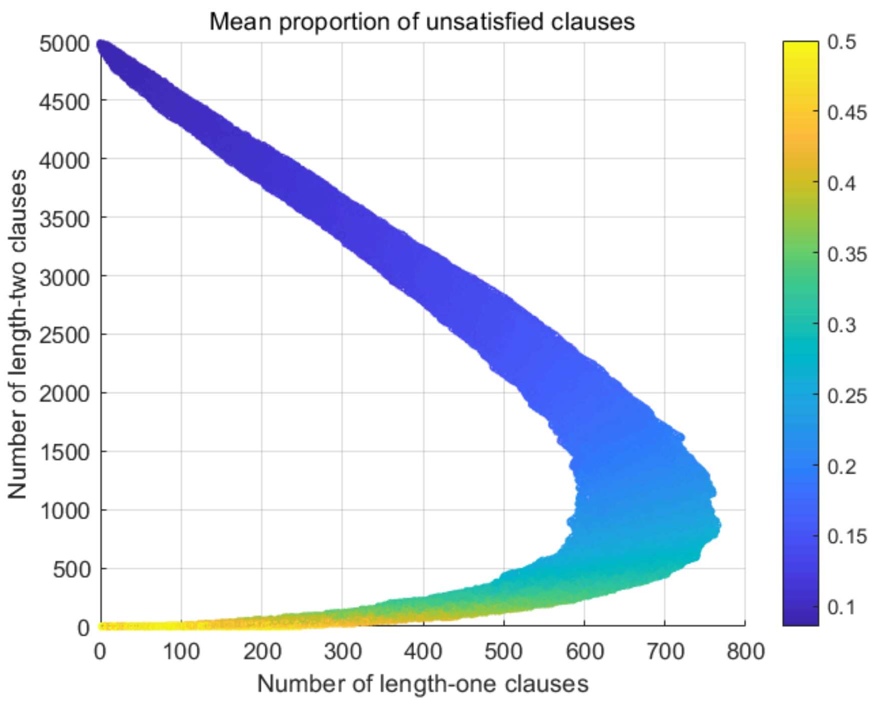


Figure 4.13: The mean proportion of clauses unsatisfied by EEMOCE, as a function of the composition of the residual instance. An aggregation of 10000 Max 2-Sat instances over 1000 variables and with density 5.

and executed EEMOCE on each of them. In principle, when executing EEMOCE on such an instance, we have 1001 stages, starting at the initial instance with no variables assigned, and ending with the final assignment, in which all the variables are assigned. In fact, as some variables do not appear in the instance, or are eliminated before being assigned a value, the number of stages is somewhat smaller. For each of these stages, we recorded the number of remaining clauses of length 1, the number of remaining clauses of length 2, and the proportion of clauses, out of all remaining clauses, that were eventually unsatisfied by the final assignment.

In Figure 4.13 we depict the results. The shaded region indicates combinations of numbers of length-1 and of length-2 clauses encountered during the various executions of the algorithm. The coloring indicates the proportion of clauses eventually unsatisfied (averaged over all instances). As most of the clauses in the initial instance get to be satisfied, the region near the point $(0, 5000)$ is colored dark blue. Towards the end, gains are small, and little over a half of the remaining clauses are to be satisfied. This region is colored yellow.

Another interesting trend is the residual instance hardness. At the beginning of the execution, the instance contains mostly clauses that eventually satisfied. As we progress, the residual instance contains more and more balanced variables that satisfy half of their clauses and unsatisfy the other half.

4.7 Conclusion

In this chapter we have presented and studied a new algorithm for the Maximum Satisfiability (Max Sat) problem. The algorithm is based on the Method of Conditional Expectations (MOCE), and applies an efficient greedy variable ordering to it. We call our algorithm Efficient Exhaustive Method of Conditional Expectations (EEMOCE) as its greediness efficiently exhausts all unassigned variables at each step.

MOCE may be seen as a derandomization of RAND, the naive algorithm that sets the variables uniformly at random to either `true` or `false`. While MOCE still considers the variables in a random order, at each step 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 assign to the current variable. Then, it sets the variable to the truth value with the larger expectation.

EEMOCE may be seen as a derandomization of MOCE. It derandomizes the order in which the variables are assigned. While this order is random in MOCE, in EEMOCE we set the seemingly better variable at each step. This is done by comparing the expected number of satisfied clauses under each of the remaining variables, when each is assumed to be set to its seemingly better truth value (as done by MOCE). Finally, we set the variable for which this expected number is larger.

4.7.1 Algorithmics and implementation

A naive design of EEMOCE results in quadratic-time complexity, as in each step the algorithm has to examine all remaining variables, and recalculate their expected number of satisfied clause.

To avoid a quadratic-time complexity, we represented Max r-Sat instances using two data structures. The first allows constant-time access to all variables of a given clause, and the second allows amortized constant-time access to all the clauses in which a given variable appears.

We complemented the efficient instance representation by an efficient maintenance of gains – the quantities that guide EEMOCE regarding the ordering of the variables. To this end, we use two additional data structures. The first allows constant-time access to the gain of a given variable, and the second allows logarithmic-time access to a variable with a maximal gain.

The efficient maintenance of gains is also required for avoiding quadratic-time complexity. The second structure we used is a balanced binary search tree. The time complexity of EEMOCE is quasilinear. In fact, it is linearithmic – it adds only a logarithmic factor to the linear time complexity of MOCE. By replacing the balanced binary search tree with a tailored data structure, one may reduce the time complexity even further. In fact, we have pointed out how to linearize the algorithm.

We provided a detailed description and pseudocode for EEMOCE. The description includes various notions and aspects related to its design and execution. Among these are efficient instance representation, residualization of an instance, the concept of gain, efficient maintenance of gains, initial gain calculation, the notion of marginalization of gains and how it is done, and more.

4.7.2 Performance and measures

We conducted a comprehensive study of EEMOCE. We studied its performance in practical low densities instances of Max r-Sat. We showed that the average number of clauses unsatisfied by EEMOCE scales linearly with the number of clauses, and thus can be described as a proportion of the number of all clauses, for any fixed density. We also showed that, considered as a function of the number of variables n, the standard deviation is roughly proportional to $\sqrt{n}$.

Examining the runtimes, we showed that EEMOCE does introduce an overhead over MOCE. Yet, overall, the prolonged runtimes seem to be reasonable, given the significant reduction in the number of unsatisfied clauses. As MOCE, EEMOCE is also a very fast algorithm.

We also studied the asymptotic performance of EEMOCE as the density grows larger. We pointed out the phenomenon of diminishing relative improvement in the asymptotic case. As the density grows larger, the possible improvement drops to zero with respect to the mean proportion of clauses unsatisfied by a random assignment.

To overcome the above problem, we have introduced three measures for comparison of the asymptotic performance of algorithms for this problem: IGM, IGO, and IGL. These measures neutralize the decrease of the possible relative improvement by using reference points. They may possibly be relevant for other combinatorial optimization problems as well.

We reveal a remarkable phenomenon. As the density grows larger, the improvement of EEMOCE over MOCE does not diminish – it converges to a constant. For Max 3-Sat, according to the suggested measures, EEMOCE has an advantage of 21%-41% over MOCE. The phenomenon of convergence to a constant of the IGM and IGO measures was observed in general, not only for clause length $r = 3$. We provide the values of these measures for other values of r as well.

We also measured the IGL of MOCE and EEMOCE for commonly used values of r. Using this measure, we showed that MOCE closes about 65% of the gap from the mean to the optimum, while EEMOCE closes about 80%.

4.7.3 Execution and internals

We analyzed the execution of EEMOCE and compared it to MOCE. The analysis shows that EEMOCE expected solution is consistently better than that of MOCE throughout its execution. The number of steps EEMOCE requires for reaching a solution is also lower.

We showed that EEMOCE divides the variables into levels, and assigns them level after level. Our experiments show that the top level may contain many variables, which may open a door to additional improvements. We also discussed the distribution of the clause length throughout the execution.

4.7.4 Further research leads

While EEMOCE removes a lot of randomality from MOCE, it does not remove it all. EEMOCE still includes two types of randomality. The first type occurs when assigning a given variable, for whom both truth values give the same expected number of satisfied clauses. This tie breaking is done randomly by EEMOCE, same as by MOCE.

The second, and more important, type of randomality occurs when selecting the variable to be assigned, out of all the remaining unassigned variables, in a given step. Usually, there are numerous variables that give the same maximal expected number of satisfied clauses.

The remaining randomality in EEMOCE opens possibilities for further research focusing on the derandomization of EEMOCE. This derandomization may be expected to provide an even better algorithm.

Chapter 5

Dominance Certificates for Combinatorial Optimization Problems

Heuristic algorithms, such as simulated annealing, are widely used in practice to solve combinatorial optimization problems. However, they offer no guarantees regarding the quality of the provided solution. An $f(I)$ *combinatorial dominance guarantee* is a certificate that a solution is not worse than at least $f(I)$ solutions for a particular problem instance I.

In this chapter, we introduce simple but general techniques for awarding combinatorial dominance certificates to arbitrary solutions of various optimization problems. We demonstrate these techniques by applying them to the Traveling Salesman and Maximum Satisfiability problems, and briefly experiment their usability.

5.1 Introduction

One of the most active research areas in the theory of combinatorial algorithms is the design of approximation algorithms for NP-hard problems. However, while approximation ratio analysis does give some information on heuristics, it does not provide the whole picture regarding their performance in practice.

Algorithmic solutions used in practice are often some form of local improvement heuristic, based on techniques such as Simulated Annealing [62], HC [94], GRASP [81], Tabu Search [39], or Genetic Algorithms [57, 77]. Properly implemented, these techniques may lead to short, efficient programs which yield reasonable solutions. However, these heuristics often come with no theoretical guarantee as to the quality of the provided solution.

An $f(I)$ *combinatorial dominance guarantee* is a certificate that a solution is not worse than at least $f(I)$ solutions for a particular problem instance I. The intuition behind this performance measure rests on the letter of recommendation one could write on behalf of a given person, or heuristic solution. A recommendation like *"She is the best of the 75 students in my class this year"* is analogous to a combinatorial dominance guarantee. It certifies the candidate as superior to a certain number of members of a given pool, with the implied assumption that this says something meaningful about the candidate's global ranking as well. The larger the number of competitors dominated by the candidate, the stronger the recommendation.

The previous body of work concerns proving existential bounds for particular problems over the space of all problem instances. In this chapter, we demonstrate a general technique for awarding combinatorial dominance "certificates" to arbitrary solutions of various optimization problems. We demonstrate this technique on the Traveling Salesman and Maximum Satisfiability problems, and briefly experiment its usability. Observe that similar approximation ratio certificates are not forthcoming for ad-hoc solutions. Namely, given a particular solution of a problem, it is not at all clear how we can compare its quality with that of the (unknown) optimal solution.

Additionally, we describe how to simulatively estimate the number of solutions better than a given solution up to a given error with high probability. We experiment the usability of the simulative estimation for differentiating between heuristics for Maximum Satisfiability in terms of dominance, and compare the estimate derived from Chebyshev's inequality to simulation results.

In Section 5.1.1 we briefly survey previous work. The notions of combinatorial dominance guarantees are formalized in Section 5.1.2. In Section 5.2 we show how an arbitrary (and, in particular, a randomly selected) solution may be proved to have some combinatorial dominance guarantee. Brief experimental examinations are summarized in Section 5.3. Finally, we discuss directions for future research in Section 5.4.

5.1.1 Previous work

The issue of measuring the quality of approximate solutions has been addressed by Zemel [108]. A formulation of the very basic properties expected from a function measuring the quality of approximate solutions was given, and the notion of a *proper* quality measure stated accordingly. Zemel suggested considering some measures, such as *z-approximation* [52] and *location ratio*, which is more familiar recently as *dominance ratio* [44, 3]. Both of these measures are proper.

The latter measure has been studied primarily within the operations research community. The basic notion appears to have been independently discovered several times. The primary focus has been on algorithms for TSP, specifically designing polynomial-time algorithms which dominate exponentially large neighborhoods. The first TSP heuristics with an exponential dominance number are due to Rublineckii [93] (see also Sarvanov and Doroshko [95, 96]).

The question whether there exists a polynomial-time algorithm which yields a solution dominating $(n-1)!/p(n)$ tours, where $p(n)$ is polynomial, appears to have first been raised by Glover and Punnen [40]. Dominance bounds for TSP have been most aggressively pursued by Gutin, Yeo, and Zverovich in a series of papers (cf. [45, 46]), culminating in a polynomial-time algorithm which finds a solution dominating $\Theta((n-1)!)$ tours. These bounds follow by applying certain Hamiltonian cycle decomposition theorems to the complete graph. We refer to [47] for more information.

In [33], the authors survey the complexity of optimizing TSP over several well-defined but exponentially large neighborhoods. Such optima by definition have large dominance numbers. In [12], the authors perform an experimental study of certain linear-time dynamic programming algorithms for TSP, which dominate exponentially many solutions.

Gutin, Vainshtein, and Yeo [44] appear to have been the first to consider the complexity of achieving a given dominance bound. In particular, they define complexity classes of DOM-easy and DOM-hard problems. They prove that weighted Max k-Sat and Max Cut are DOM-easy while (unless $P = NP$) Vertex Cover and Clique are DOM-hard.

Alon, Gutin, and Krivelevich [3] provide several algorithms which achieve large dominance *ratios* for versions of Integer Partition, Max Cut, and Max r-Sat. These algorithms share a common property — they provide solutions of quality guaranteed to be not worse than the average solution value. This property has been used also in other dominance proofs [87, 44, 45, 86, 63]. Twitto [105] showed that this property by itself does not necessarily ensure good dominance.

Other works on dominance analysis include [48, 87], where it is proved that the nearest neighbor, minimum spanning tree, and greedy heuristics perform extremely poorly for symmetric and asymmetric TSP. Various combinatorial optimization problems and classical heuristics for them have been analyzed in [14, 13, 43]. In [80], a model for analyzing heuristic search algorithms (such as simulated annealing and backtracking), based on the ideas of combinatorial dominance, has been developed.

In [64], the authors studied a polynomial-time algorithm for ATSP, and showed that it provides a dominance ratio of at least $1/2-o(1)$. In [65], they gave a polynomial-time algorithm with dominance ratio of $1 - n^{-1/29}$ for a special case of TSP in which the edges may take only two possible weights.

In [88], the authors analyzed the BBQP problem with $m + n$ variables. They proved that any solution for this problem, with quality no worse than the average, dominates at least 2^{m+n-2} solutions, and that this bound is the best possible. They provided an $O(mn)$ algorithm to identify such a solution.

5.1.2 Definitions

Consider a given instance I of some combinatorial optimization problem P. The instance is represented by a solution space $S_P(I)$ and objective function $C_P(I, x)$. The *solution space* $S_P(I)$ is the set of all combinatorial objects representing possible solutions x to I. The objective function $C_P(I, x)$ is defined for all solutions $x \in S_P(I)$. If P is a maximization (minimization, resp.) problem, we seek an $x_0 \in S_P(I)$ such that $C_P(I, x_0) \geq C_P(I, x)$ ($C_P(I, x_0) \leq C_P(I, x)$, resp.) for all $x \in S_P(I)$.

A *heuristic* H_P for P is a procedure which, for any instance I, selects a solution $x \in S_P(I)$. For a given instance I of P, denote by $F(I)$ the number of solutions that are not better than the heuristic solution $H_P(I)$. The number of all other solutions in $S_P(I)$ (which are better than $H_P(I)$) is denoted by $B(I)$.

Definition 1. A heuristic H_P offers an $F(n)$ *combinatorial dominance guarantee (dominance bound/number)* for problem P if for each n:

1. For all instances I of size n of P, the solution $H_P(I)$ dominates at least $F(n)$ elements of $S_P(I)$.

2. There exists an instance I' of size n for which $H_P(I')$ dominates exactly $F(n)$ elements of $S_P(I')$.

The heuristic *blackball bound/number* of H_P is $B(n) = |S_P(n)| - F(n)$.

The heuristic *dominance* (*blackball*, resp.) *ratio* is defined to be its dominance (blackball, resp.) number divided by the size of the solution space.

5.2 Certified dominance bounds for arbitrary solutions

In this section we demonstrate a general technique for awarding combinatorial dominance certificates to arbitrary solutions of various optimization problems. Such a certificate is a proof that a given solution of some instance (of some optimization problem) is not worse than at least some prescribed number of solutions of that instance. Additionally, we describe how to simulatively estimate the number of solutions better than a given solution up to a given error with high probability.

Assume we have any instance of some optimization problem, and let X be its objective function. Suppose we can calculate the expected value $\mu = E(X)$ and the variance $\sigma^2 = V(X)$ of the value of the objective function at a random solution. (Note that these quantities can be calculated for many problems; see Sections 5.2.1 and 5.2.2 below for two important examples.) Now, suppose we have any solution with an objective value x_0 which happens to be better than μ, i.e., $x_0 > \mu$ ($x_0 < \mu$, resp.) for maximization (minimization, resp.) problems. We may then assert that there is some percentage of the solutions which are not better than this solution. Indeed, denote by x the objective value of a random solution, and assume, say, that we deal with a maximization problem. By the one-sided Chebyshev's inequality [91], we have

$$P(X > x_0) \leq \frac{\sigma^2}{\sigma^2 + (x_0 - \mu)^2} < 1. \tag{5.1}$$

One way to obtain (with probability arbitrarily close to 1) a solution significantly above the mean is to take the best of a large number of randomly selected solutions. For example, suppose the values of the objective function are approximately normally distributed, which is true in many situations. The best solution out of approximately 40 sampled solutions is expected to be about two standard deviations above the mean, as approximately 2.5% of the solutions have this property. By Chebyshev's inequality, such a solution is in any case guaranteed to be not worse than at least four fifths of the solutions. Note that this bound holds whether the values are normally distributed or not, given the distance from the mean is verified somehow. It is the value from Chebyshev's inequality that provides the dominance certificate.

5.2.1 TSP certification

In the Symmetric Traveling Salesman problem (STSP), we are given an edge-weighted complete undirected graph K_V. We seek an ordering p of the $n = |V|$ vertices, minimizing the sum of weights of the edges along the tour induced by p on K_V. The size of the solution space is $(n-1)!/2$, and the size of the problem is taken as $n = |V|$. We apply the above technique to STSP.

Denote by w_{ij} the weight of edge (i,j), and let X be the weight of a random tour. We have to find $E(X)$ and $V(X)$. For $1 \le i < j \le n$, put:

$$X_{ij} = \begin{cases} 1, & \text{the tour contains the edge } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

Then:

$$X = \sum_{1 \le i < j \le n} w_{ij} X_{ij}.$$

For X_{ij}, we have

$$E\left(X_{ij}\right) = P\left(X_{ij} = 1\right) = \frac{n}{\binom{n}{2}} = \frac{2}{n-1},$$

$$V\left(X_{ij}\right) = E\left(X_{ij}^2\right) - E^2\left(X_{ij}\right) = \frac{2(n-3)}{(n-1)^2}.$$

The covariance $\mathrm{Cov}\left(X_{ij}, X_{kl}\right)$ is given by

$$\begin{aligned}\mathrm{Cov}\left(X_{ij}, X_{kl}\right) &= E\left(X_{ij}X_{kl}\right) - E\left(X_{ij}\right)E\left(X_{kl}\right)\\ &= P\left(X_{ij} = X_{kl} = 1\right) - \left(\frac{2}{n-1}\right)^2,\end{aligned}$$

where

$$P\left(X_{ij} = X_{kl} = 1\right) = \begin{cases} \frac{4}{(n-1)(n-2)}, & \{i,j\} \cap \{k,l\} = \emptyset,\\ \\ \frac{2}{(n-1)(n-2)}, & \left|\{i,j\} \cap \{k,l\}\right| = 1,\\ \\ \frac{2}{n-1}, & \{i,j\} = \{k,l\}.\end{cases}$$

Therefore

$$E(X) = \sum_{1 \le i < j \le n} w_{ij} E\left(X_{ij}\right) = \frac{2}{n-1} \sum_{1 \le i < j \le n} w_{ij},$$

and

$$\begin{aligned}V(X) &= V\left(\sum_{1 \le i < j \le n} w_{ij} X_{ij}\right)\\ &= \sum_{1 \le i < j \le n} w_{ij}^2 V\left(X_{ij}\right) + \sum_{(i,j) \ne (k,l)} w_{ij} w_{kl} \mathrm{Cov}\left(X_{ij}, X_{kl}\right)\\ &= \frac{2(n-3)}{(n-1)^2} \sum_{1 \le i < j \le n} w_{ij}^2 + \frac{4}{(n-1)^2(n-2)} \sum_{\{i,j\} \cap \{k,l\} = \emptyset} w_{ij} w_{kl}\\ &\quad - \frac{2(n-3)}{(n-1)^2(n-2)} \sum_{\left|\{i,j\} \cap \{k,l\}\right| = 1} w_{ij} w_{kl}.\end{aligned}$$

Having these explicit formulas for $E(X)$ and $V(X)$, we may easily, and automatically, bound the quality of any given solution for any given instance of STSP. The automation may be obtained by a program that first computes the expectation and variance by the above formulas for the given instance. Then, for any solution thereof, it computes and returns the probability given in (5.1) as the certified dominance bound for the solution.

5.2.2 Max Sat certification

In the Maximum Satisfiability problem (Max Sat), we are given a multiset of clauses over some Boolean variables. Each clause is a disjunction of literals (a variable x_i or its negation $\overline{x}_i$). We seek a `true-false` assignment for the variables, maximizing the number of satisfied clauses.

For disjoint sets $A, B \subseteq \{1, 2, \dots, n\}$, denote:

$$T_{AB} = \bigvee_{i \in A} x_i \vee \bigvee_{j \in B} \overline{x}_j.$$

For example, $T_{\{1,4\}\{2\}} = x_1 \vee x_4 \vee \overline{x}_2$. Suppose the multiset, which we denote by T, consists of c_{AB} occurrences of each T_{AB}. For a random assignment of values, consider the random variable Y — the number of satisfied clauses in T. We have

$$E(Y) = \sum_{A,B} c_{AB} \left(1 - 2^{-|A|-|B|}\right),$$

and

$$\begin{aligned} V(Y) = &\sum_{A,B} c_{AB}^2 2^{-|A|-|B|} \left(1 - 2^{-|A|-|B|}\right) \\ &+ \sum_{(A,B) \neq (A',B')} c_{AB} c_{A'B'} \left(P(T_{AB} = T_{A'B'} = \texttt{false}) - 2^{-|A|-|B|-|A'|-|B'|}\right), \end{aligned}$$

where

$$P(T_{AB} = T_{A'B'} = \texttt{false}) = \begin{cases} 2^{-|A \cup A'| - |B \cup B'|}, & A \cap B' = A' \cap B = \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

Again, automating the computation of the quantities above to bound the quality of any given solution for any given instance of Max Sat is immediate.

5.2.3 A confidence interval for the blackball ratio

Given an instance I of an optimization problem P, and a solution s_0 of I, let:

$$g(s) = \begin{cases} 1, & s \text{ is better than } s_0, \\ 0, & \text{otherwise.} \end{cases}$$

Denote by $P_{s_0}(s)$ the probability that a random solution s of I is better than s_0.

Lemma 1. *Let $\varepsilon, \delta \in (0,1)$. Suppose $s_1, s_2, \ldots, s_t$ are $t \geq \left\lceil \frac{(2+\varepsilon)\ln\frac{2}{\delta}}{\varepsilon^2} \right\rceil$ uniformly and independently sampled solutions of I, and let*

$$\tilde{\mu} = \frac{1}{t}\left|\{1 \leq i \leq t : \, g(s_i) = 1\}\right|.$$

Then, for a uniformly sampled solution s of I, we have

$$\tilde{\mu} \in [P_{s_0}(s) - \varepsilon, P_{s_0}(s) + \varepsilon]$$

with probability at least $1 - \delta$.

Proof. Consider the random variables $Y_i = g(s_i), 1 \leq i \leq t$. Note that $E(Y_i) = P(Y_i = 1) = P_{s_0}(s)$. Let $Y = \sum_{i=1}^{t} Y_i$. By Chernoff's inequality [7] we have

$$\begin{aligned} P(|\tilde{\mu} - P_{s_0}(s)| > \varepsilon) &= P(|Y/t - P_{s_0}(s)| > \varepsilon) \\ &= P(|Y - tP_{s_0}(s)| > \varepsilon t P_{s_0}(s)/P_{s_0}(s)) \\ &\leq 2e^{-\varepsilon^2 t/(2P_{s_0}(s)+\varepsilon)} \\ &\leq 2e^{-\varepsilon^2 t/(2+\varepsilon)} \\ &\leq 2e^{-\frac{\varepsilon^2 (2+\varepsilon)\ln\frac{2}{\delta}}{(2+\varepsilon)\varepsilon^2}} = \delta. \end{aligned}$$

□

The selection between several heuristics for a problem is often made by comparing their performance experimentally. The experiments may include applying the heuristics to random instances of the problem, or a predefined set of representative instances. Note that the above simple method might be used to obtain a single number representing the performance (in the experiments) of each of the heuristics. Namely, the method may yield an estimate of the blackball ratio of each of the heuristics in the experiments. Automation of this method is immediate. Observe that similar estimations of the approximation ratio of heuristics are not forthcoming, as the optimal objective value is usually unknown.

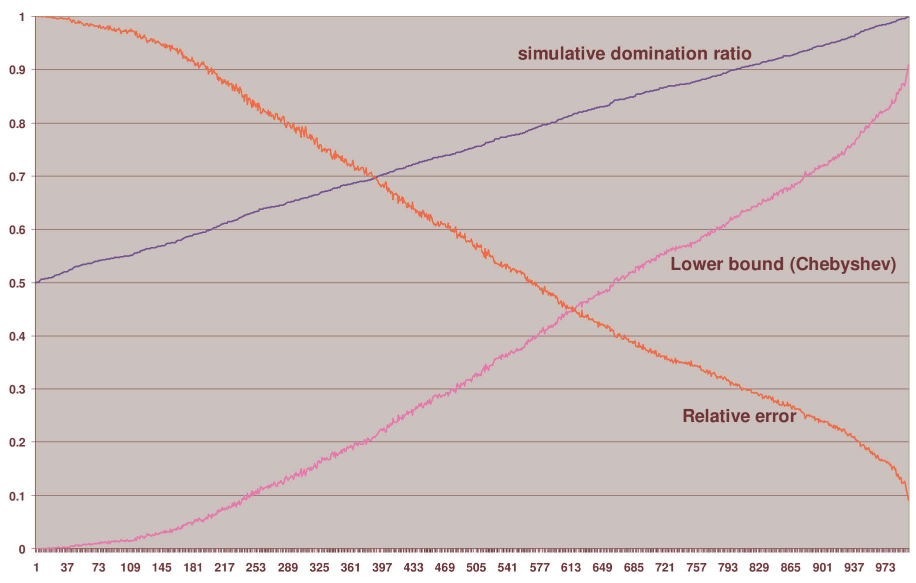


Figure 5.1: Comparing Chebyshev's lower bound to simulation results.

5.3 Experimental results

In this section we present results of experiments we performed in order to examine the techniques presented in Section 5.2.

5.3.1 Results on the Chebyshev's bound based technique

In these experiments, our aim is to check to what extent Chebyshev's bound gives meaningful results. To this end, we took 1000 random instances of STSP, on 20 vertices each, with edge weights selected uniformly and independently in $[0, 1]$. For each of the instances, we randomly selected a solution that is better than the average solution of that instance (by randomly generating solutions until obtaining a solution with this property). We compared Chebyshev's bound on the dominance ratio of the selected solution with an estimate of this ratio, given by simulation. The latter estimate was calculated as the percentage of solutions which outperformed our initial randomly selected solution, taken over a large number of random solutions.

Figure 5.1 shows the results. The instances (horizontal axis) are sorted according to their simulative estimation. The decreasing graph provides the relative error of Chebyshev's bound with respect to the (probably very ac-

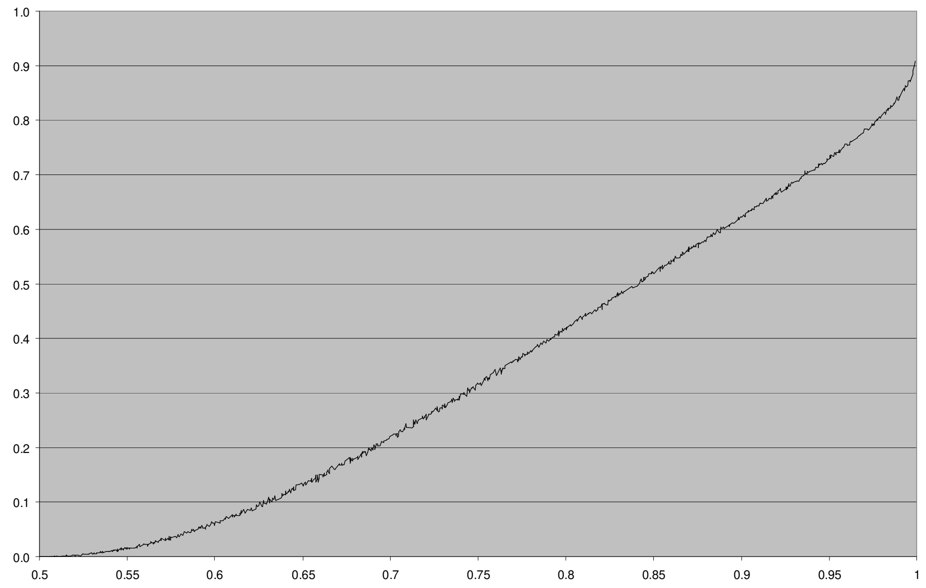


Figure 5.2: Scatterplot of Chebyshev's lower bound and simulation results.

curate) estimate given by the simulation. It shows that Chebyshev's bound gets better and more meaningful as the solutions get better. For solutions close to the average solution value, Chebyshev's bound yields meaningless estimates, whereas for very good solutions it yields good estimates. A scatterplot of Chebyshev's bound (vertical axis) and the estimation provided by simulation (horizontal axis) is provided in Figure 5.2.

Likewise, we considered all the Euclidean instances of size of up to 1000 vertices from TSPLIB [90]. We used the following six heuristics available in the Concorde TSP Solver [9]: Greedy (GR), Boruvka (BV), Quick Boruvka (QBV), Nearest Neighbor (NN), LinKernighan (LK), and Optimal (OPT). (For more information on these heuristics consult the solver's [9] documentation.) We applied each of the heuristics on each of the instances, obtained solutions, and calculated Chebyshev's lower bounds on the dominance ratio of each of the solutions.

The graphs of the dominance of the heuristics are given in Figure 5.3, in which a representative part has been zoomed in. In this figure (as well as in Figure 5.4), the instances are arranged by their size on the horizontal axis. The vertical axis provides the dominance ratio. The location of the two coinciding graphs corresponding to the OPT and LK heuristics, above

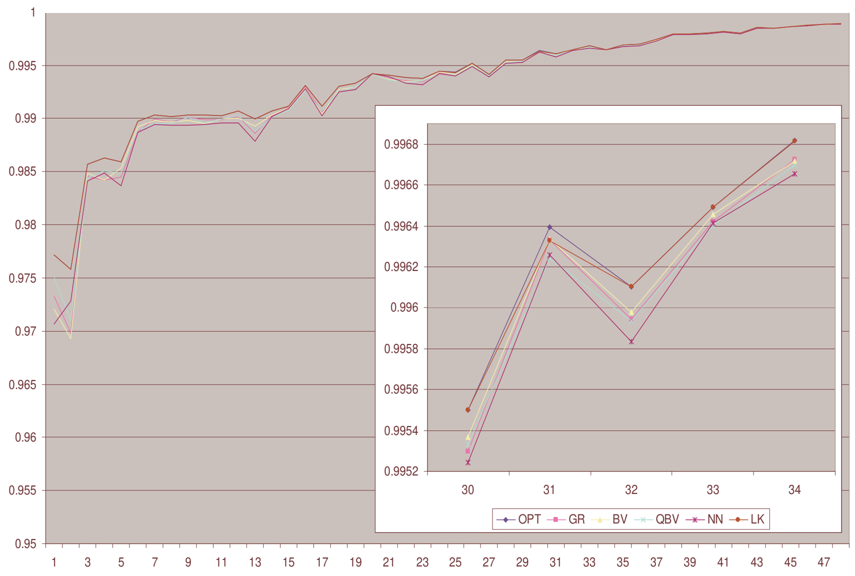


Figure 5.3: Comparison by dominance. Instances from TSPLIB.

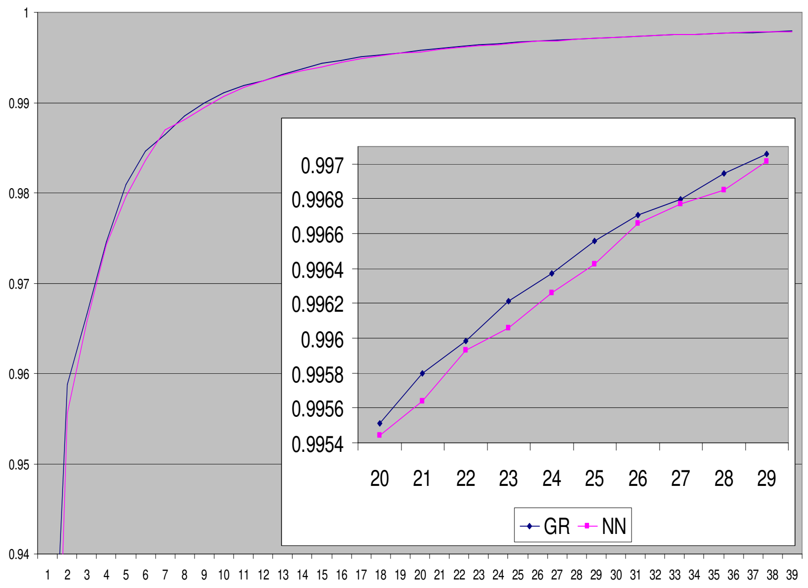


Figure 5.4: GR vs. NN. Applied on randomly generated instances.

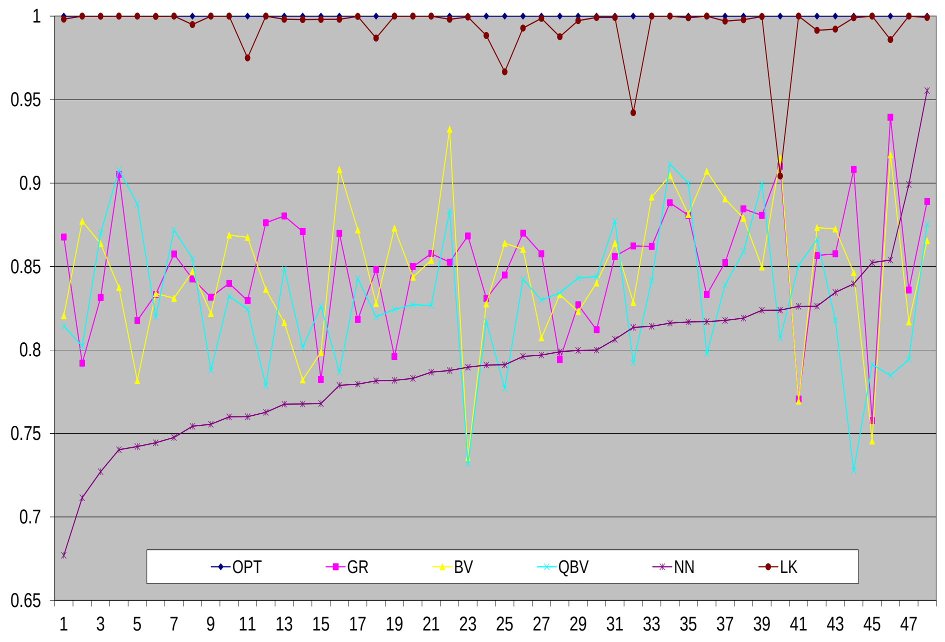


Figure 5.5: Comparison by approximation ratio. Instances from TSPLIB.

all the other graphs, shows that the dominance ratio is able to point to the better methods, and that the LK heuristic usually provides very good solutions. The NN heuristic seems to be the worst all the way long. The other three heuristics (GR, BV, and QBV) are in the middle.

Similar phenomena were observed when we used Chebyshev's lower bound to compare GR and NN on randomly generated instances (Figure 5.4). A comparison using approximation ratio yielded similar results, as can be seen in Figure 5.5. To make it clearer, the instances in this figure are sorted according to the approximation ratio of the NN heuristic on them.

5.3.2 Results on the confidence interval based technique

In the following, our main aim is to examine and demonstrate the usability of the technique presented in Section 5.2.3 as a way to compare and differentiate heuristics according to their estimated domination ratio. To this end, we compared the following heuristics for Max Sat:

Majority Vote (MV). This heuristic assigns the value `true` to a variable if its number of positive occurrences is at least as large as its number

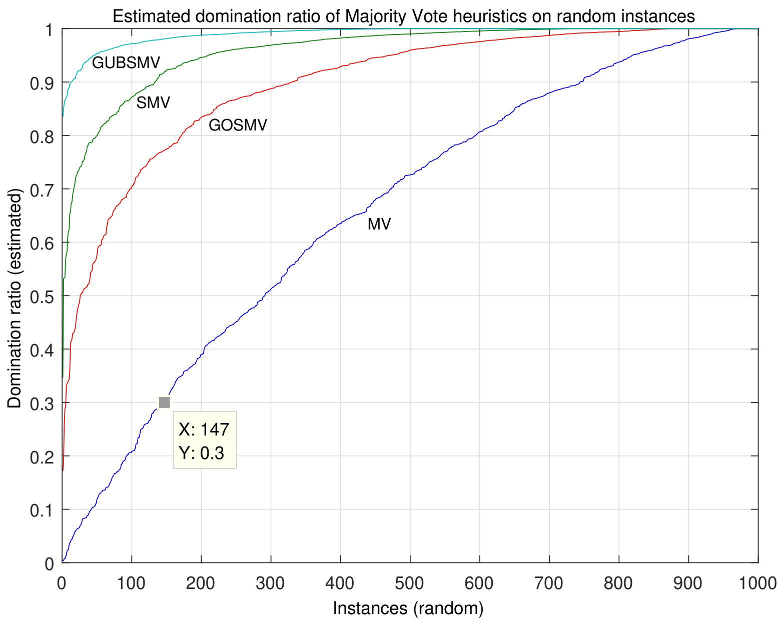


Figure 5.6: Estimated domination ratio of Majority Vote heuristics on random Max Sat instances.

of negative occurrences. Otherwise, it assigns it the value `false`.

Step-by-step Majority Vote (SMV). This heuristic goes over the variables one-by-one according to some random order. It assigns a truth value to the current variable according to the majority vote as before. However, after each assignment it discards all the clauses satisfied so far. The resulting instance is passed for the next step.

Greedy Occurrence SMV (GOSMV). Same as SMV, but at each step assigns a truth value to the currently most frequent variable. Ties are broken arbitrarily.

Greedy Unbalanced SMV (GUBSMV). Same as SMV, but at each step assigns a truth value to the variable for which the absolute value of the difference between its number of positive occurrences and number of negative occurrences is maximal at this point. Ties are broken arbitrarily.

The comparisons were done on 1000 randomly generated Max Sat instances, on $n = 50$ variables $x_1, x_2, ..., x_n$ and $m = 300$ clauses. We select

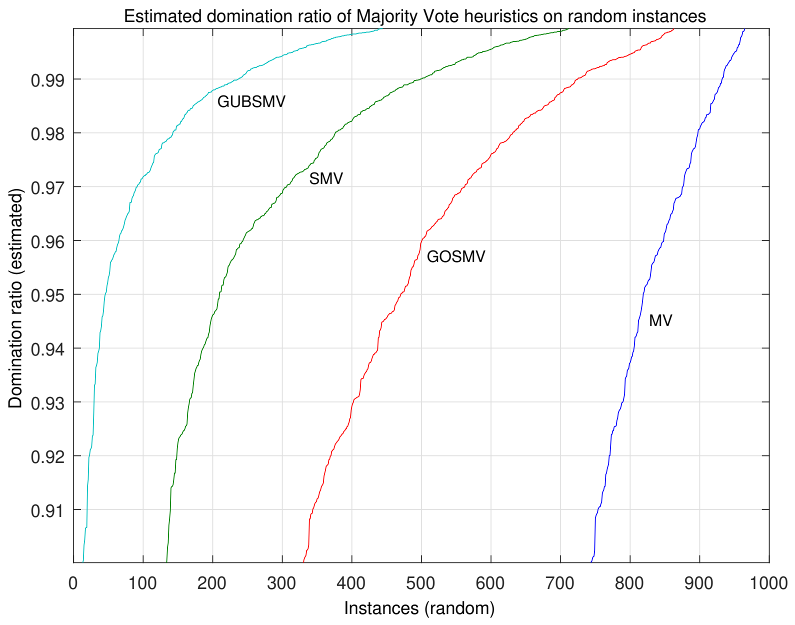


Figure 5.7: Estimated domination ratio of Majority Vote heuristics on random Max Sat instances. A vertical zoom on the top echelon.

the number of variables to appear in each clause uniformly from the interval $[1, n]$. Then, for each variable x_i we draw a random number from the interval $[0, 1/i]$, and select the ones with the largest random numbers to appear in the clause. Each of these variables appears in the clause positively or negatively with probability $1/2$.

For each such randomly generated instance we applied each of the heuristics, and obtained their solutions. The quality of a solution was assessed by Lemma 1. We have chosen $\varepsilon = 0.03$ and $\delta = 0.001$. Each variable of the random solutions generated for the assessment was set to `true` or `false` with probability $1/2$.

The estimated domination ratio of each of the heuristics on the randomly generated instances are depicted in Figure 5.6. On the horizontal axis, the instances are sorted in ascending order of performance of the heuristics on them. The sorting is done for each of the heuristics independently, so that a specific point on the horizontal axis is likely to correspond to distinct instances for the various heuristics. The vertical axis is the estimated domination ratio at this point. For example, the point $(147, 0.3)$ marked on the graph of the MV heuristic indicates that the domination ratio of the instance

	MV	**SMV**	**GOSMV**	**GUBSMV**
min	0.0037	0.3463	0.1716	0.8341
max	1	1	1	1
mean	0.6611	0.9603	0.8991	0.9908
median	0.7263	0.9901	0.9599	1
std	0.2811	0.0735	0.1394	0.0206

Table 5.1: Performance statistics of Majority Vote heuristics on random Max Sat instances.

ranked as the 147-th out of 1000 (from the bottom) for this heuristic is 0.3. A vertical zoom on the top echelon of the domination ratio (above 0.9) is provided in Figure 5.7.

A performance statistics is provided in Table 5.1. For each of the heuristics we give the minimum, maximum, mean, and median, estimated domination ratio measured over the 1000 instances. The standard deviation around the mean estimated domination ratio is also provided. For example, one may learn from the performance statistics that the mean estimated domination ratio of the SMV heuristic was 0.9603, whereas its worst case domination ratio was 0.3463.

Inspecting the results, one can clearly see that the MV heuristic is the worst, as expected. Better performance was shown by the GOSMV heuristics which performs relatively well on average but failed to provide good performance in the worst case. Both were inferior to the SMV heuristic. The best performance was shown by the GUBSMV heuristic, which performed well not only on average but also in the worst case; see the minimum performance in the table. The median of 1 for this heuristic indicates that it provided a solution better than all randomly selected solutions for at least half of the instances.

It is worthwhile mentioning that, by applying the technique demonstrated in this section, we not only obtained a clear differentiation between the explored heuristics, but also gained quantitative insights regarding the performance gap between them in terms of domination ratio.

5.4 Discussion

We have demonstrated analytic and probabilistic methods to obtain a non-trivial combinatorial dominance certificate on the quality of any ad-hoc solution to a given combinatorial optimization problem on any particular instance. We have shown that these methods are easily applied to TSP and Max Sat. We note that similar approximation ratio certificates are not forthcoming for ad-hoc solutions.

This opens up two interesting lines for investigation. The first is to apply these methods to experimentally compare heuristics for other optimization problems. These methods provide ways of identifying relatively hard instances of particular problems and certifying the quality of heuristics even in the absence of known optimal solutions. The second direction concerns theoretical investigations of the power of the Chebyshev-based method. Does the method provably yield more meaningful bounds on some problems than others? To what extent does this method apply to problems with infeasible solutions?

Chapter 6

Conclusion

In Chapter 2 we have provided some results characterizing the ensemble of all (equally likely) r-CNF formulas. These results apply to both random Max r-Sat and random r-Sat. Throughout this chapter we have chosen to present them in the context of random Max r-Sat instances.

In many heuristics, after a local optimum has been reached, the whole search is repeated from a different, randomly selected starting point. This is a simple, clean way to restart. Yet, as the heuristic may have already yielded a quite good assignment, one may prompt for further exploration of the landscape in the vicinity of this assignment. In such cases, one may want to perform a jump from the local optimum that is not too large, so as to stay in the vicinity of the local optimum. On the other hand, a too small jump will leave us in the basin of attraction of the current local optimum, which will lead to the same local optimum again.

To this end, the autocorrelation length may provide assisting information, as it hints on the average distance between local optima [74]. The specific way of using it to calculate the size of the jump is a subject for further research.

We also note that the nature of industrial instances, for example, is subtly different from that of random ones. Their underlying probability model, if any, is different, and they should be addressed separately. We hope our work will encourage a concise analysis of modeled ensembles of practical instances.

In Chapter 3, we have explored the correlation between the quality of initial assignments provided to local search heuristics and that of the corresponding final assignments. We 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.

We demonstrated our point by improving the state-of-the-art solver CCLS, by combining it with MOCE. Instead of starting CCLS from random initial assignments, we started 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. Namely, it handles larger instances better, and shows superior performance on them.

Given the above, we recommend MOCE-CCLS over RAND-CCLS. Furthermore, we recommend starting CCLS from solutions 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).

In Chapter 4 we present and study a new algorithm for the Maximum Satisfiability (Max Sat) problem. The algorithm is based on the Method of Conditional Expectations (MOCE), and applies an efficient greedy variable ordering to it. We call our algorithm Efficient Exhaustive Method of Conditional Expectations (EEMOCE) as its greediness efficiently exhausts all unassigned variables at each step.

MOCE may be seen as a derandomization of RAND, the naive algorithm that sets the variables uniformly at random to either `true` or `false`. While MOCE still considers the variables in a random order, at each step 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. Then, setting the variable to the truth value for which this expected number is larger.

EEMOCE may be seen as a derandomization of MOCE. It derandomizes the order in which the variables are assigned. While this order is random in MOCE, in EEMOCE we set the seemingly best variable at each step. This is done by comparing the expected number of satisfied clauses under each of the remaining variables, when each is assumed to be set to its seemingly better truth value (as done by MOCE). Then, we set the variable for which this expected number is the largest.

While EEMOCE removes a lot of randomality from MOCE, it does not remove it all. EEMOCE still includes two types of randomality. The first type occurs when assigning a given variable, for whom both truth values give the same expected number of satisfied clauses. This tie breaking is done randomly by EEMOCE, same as by MOCE.

The second, and more important, type of randomality occurs when selecting the variable to be assigned, out of all the remaining unassigned variables, in a given step. Usually, there are numerous variables that give the same maximal expected number of satisfied clauses.

The remaining randomality in EEMOCE opens a space for further research focusing on the derandomization of EEMOCE. This kind of derandomization is expected to provide an even better algorithm. Specifically, it seems that the second type of randomality is a prominent derandomization candidate.

Except for a further derandomization of EEMOCE, in the following, we list some other algorithmic ideas worth exploring as possible improvements to EEMOCE. All these ideas are novel and unexplored at the time being, as far as we know.

1. CEEMOCE: Consensus-based Efficient Exhaustive Method of Conditional Expectations.
2. REEMOCE: Reality-based Efficient Exhaustive Method of Conditional Expectations.
3. EEMOCEV: Efficient Exhaustive Method of Conditional Expectations and Variances.

In CEEMOCE we apply EEMOCE several times on an instance, say a hundred times, in each cycle. In each application, we do (the second type) tie breaking randomly. Thus, we obtain many solutions. We collect all these solutions and try to deduce a backbone for the next cycle. The backbone consists of variables on which the vast majority of the solutions consent (give same truth value). This backbone is then locked as a partial assignment which is fed back to the next cycle. A cycle, in which no consensus is obtained, marks the end of the core of the algorithm, and one last application of EEMOCE is done to assign truth values to the variables not included in the last backbone.

One challenge here is to define what we consider as consensus. Setting the threshold optimally to fit various instances is not trivial. Another issue is to properly deal with variables that are passively assigned during the execution of EEMOCE.

REEMOCE is a complicated, yet a clear improvement of EEMOCE. As EEMOCE gives the current variable a truth value based on conditional expectation, assuming the other variables are to be assigned randomly, it clearly misses information. A better way to consider the revenue of each truth value for a given variable is by assuming the tail will be assigned using EEMOCE itself (and not randomly). This way, REEMOCE makes a better and more informative decision for each variable.

The main problem in REEMOCE is to collect data to assist each possible decision, and to do it accurately enough in a reasonable time. This data is collected one time as a preprocessing operation, and may be used afterwards many times for the given instance. A caveat of REEMOCE is that this preprocessing should be done for each family one may want to solve.

The EEMOCEV algorithm is an extension of EEMOCE, which takes variance considerations into account. In particular, it may possibly select an assignment different from the one selected by EEMOCE if the opposite assignment provides larger variance. The motto here is that we are willing to trade expectation with variance, and to select a assignment that is worse in expectation if we know its variance is larger. As we devise EEMOCE to reach solutions far above the expected solution value, it will be best to consider the disparity of qualities together with the expectation.

The weight of the variance with respect to the expectation should be decided dynamically during the algorithm. This might be done using an approximation for the optimal number of satisfied clauses, and specifically its location above the current mean in terms of the number of standard deviations.

Besides the above, another possible future point for research is devising and incorporating various simplification and improvement rules specifically for Max Sat. Such rules allow us to reduce the size of the instances we solve. Moreover, the improvements also allow us to improve the quality of the solution. Finally, we find it useful incorporating local improvements to final solutions, to enhance their quality even further.

In Chapter 5 we have demonstrated analytic and probabilistic methods to obtain a non-trivial combinatorial dominance certificate on the quality of any ad-hoc solution to various combinatorial optimization problems on any particular instance. We have shown that these methods are easily applied to TSP and Max Sat. We note that similar approximation ratio certificates are not forthcoming for ad-hoc solutions.

This opens up two interesting lines for investigation. The first is to apply these methods to experimentally compare heuristics for other optimization problems. These methods provide ways of identifying relatively hard instances of particular problems and certifying the quality of heuristics even in the absence of known optimal solutions. The second direction concerns theoretical investigations of the power of the Chebyshev-based method. Does the method provably yield more meaningful bounds on some problems than others? To what extent does this method apply to problems with infeasible solutions?

Bibliography

[1] A. Abramé, D. Habet, and D. Toumi. “Improving configuration checking for satisfiable random k-SAT instances”. *Annals of Mathematics and Artificial Intelligence* 79.1-3 (2017), pp. 5–24.

[2] D. Achlioptas and Y. Peres. “The threshold for random k-SAT is $2^k \log 2 - O(k)$”. *Journal of the American Mathematical Society* 17.4 (2004), pp. 947–973.

[3] N. Alon, G. Gutin, and M. Krivelevich. “Algorithms with Large Domination Ratio”. *Journal of Algorithms* 50.1 (2004), pp. 118–131.

[4] E. Angel and V. Zissimopoulos. “Autocorrelation coefficient for the graph bipartitioning problem”. *Theoretical Computer Science* 191.1 (1998), pp. 229–243.

[5] E. Angel and V. Zissimopoulos. “On the classification of NP-complete problems in terms of their correlation coefficient”. *Discrete Applied Mathematics* 99.1 (2000), pp. 261–277.

[6] E. Angel and V. Zissimopoulos. “On the landscape ruggedness of the quadratic assignment problem”. *Theoretical Computer Science* 263.1 (2001), pp. 159–172.

[7] D. Angluin and L. G. Valiant. “Fast probabilistic algorithms for hamiltonian circuits and matchings”. In *Proceedings of the ninth annual ACM symposium on Theory of computing (STOC).* Boulder, Colorado, United States, 1977, pp. 30–41. DOI: http://doi.acm.org/10.1145/800105.803393. URL: http://doi.acm.org/10.1145/800105.803393.

[8] C. Ansótegui, M. L. Bonet, and J. Levy. “SAT-based MaxSAT algorithms”. *Artificial Intelligence* 196 (2013), pp. 77–105.

[9] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. *Concorde TSP solver.* See `http://www.tsp.gatech.edu/concorde/`. 2006.

[10] J. Argelich, C. M. Li, F. Manyà, and J. Planes. *MaxSat Evaluations.* URL: `http://www.maxsat.udl.cat/`.

[11] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* 2nd ed. Springer-Verlag, 2003.

[12] E. Balas and N. Simonetti. "Linear time dynamic programming algorithms for some new classes of restricted TSPs: a computational study". *INFORMS Journal on Computing* 13.1 (2001), pp. 56–75.

[13] D. Berend, S. Skiena, and Y. Twitto. "Combinatorial Dominance Guarantees for Heuristic Algorithms". In *Proceedings of the International Conference on Analysis of Algorithms (AofA).* Juan-les-Pins, France, June 2007.

[14] D. Berend, S. Skiena, and Y. Twitto. "Combinatorial Dominance Guarantees for Problems with Infeasible Solutions". *ACM Transactions on Algorithms* 5.1 (2008), pp. 1–29.

[15] D. Berend and Y. Twitto. "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. 2016, pp. 60–76.

[16] A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability.* Vol. 185. IOS press, 2009.

[17] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. "A tutorial on the cross-entropy method". *Annals of Operations Research* 134.1 (2005), pp. 19–67.

[18] N. Bouhmala. "A variable neighborhood Walksat-based algorithm for MAX-SAT problems". *The Scientific World Journal* 2014 (2014).

[19] S. Cai, Z. Jie, and K. Su. "An effective variable selection heuristic in SLS for weighted Max-2-SAT". *Journal of Heuristics* 21.3 (2015), pp. 433–456.

[20] S. Cai, C. Luo, J. Thornton, and K. Su. "Tailoring local search for Partial MaxSAT". In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence.* AAAI'14. Québec City, Québec, Canada: AAAI Press, 2014, pp. 2623–2629.

[21] S. Cai and K. Su. "Local search for Boolean Satisfiability with configuration checking and subscore". *Artificial Intelligence* 204 (2013), pp. 75–98.

[22] S. Cai and K. Su. "Local search with configuration checking for SAT". In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence.* IEEE. 2011, pp. 59–66.

[23] B. Cha, K. Iwama, Y. Kambayashi, and S. Miyazaki. "Local Search Algorithms for Partial MAXSAT". AAAI'97/IAAI'97 (1997), pp. 263–268.

[24] R. Chen and R. Santhanam. "Improved algorithms for sparse MAX-SAT and MAX-k-CSP". In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT 2015).* Springer, 2015, pp. 33–45.

[25] F. Chicano, G. Luque, and E. Alba. "Autocorrelation measures for the quadratic assignment problem". *Applied Mathematics Letters* 25.4 (2012), pp. 698–705.

[26] F. Chicano, G. Luque, and E. Alba. "Problem understanding through landscape theory". In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation.* ACM. 2013, pp. 1055–1062.

[27] V. Chvátal and B. Reed. "Mick gets some (the odds are on his side) [satisfiability]". In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science.* IEEE. 1992, pp. 620–627.

[28] A. Coja-Oghlan. "The asymptotic k-sat threshold". In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing.* ACM. 2014, pp. 804–813.

[29] D. Coppersmith, D. Gamarnik, M. Hajiaghayi, and G. B. Sorkin. "Random MAX SAT, random MAX CUT, and their phase transitions". *Random Structures & Algorithms* 24.4 (2004), pp. 502–545.

DOI: 10.1002/rsa.20015. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.20015.

[30] K. P. Costello, A. Shapira, and P. Tetali. "Randomized greedy: new variants of some classic approximation algorithms". In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms.* Society for Industrial and Applied Mathematics. 2011, pp. 647–655.

[31] J. M. Crawford and L. D. Auton. "Experimental results on the crossover point in random 3-SAT". *Artificial intelligence* 81.1-2 (1996), pp. 31–57.

[32] J. Davies and F. Bacchus. "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, 2011, pp. 225–239.

[33] V. Deineko and G. Woeginger. "A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem". *Mathematical programming* 87.3 (2000), pp. 519–542.

[34] J. Ding, A. Sly, and N. Sun. "Proof of the satisfiability conjecture for large k". In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing.* 2015, pp. 59–68.

[35] P. Erdős and J. L. Selfridge. "On a combinatorial game". *Journal of Combinatorial Theory, Series A* 14.3 (1973), pp. 298–301.

[36] W. Fontana, P. F. Stadler, E. G. Bornberg-Bauer, T. Griesmacher, I. L. Hofacker, M. Tacker, P. Tarazona, E. D. Weinberger, and P. Schuster. "RNA folding and combinatory landscapes". *Physical Review E* 47.3 (1993), pp. 2083–2099.

[37] E. Friedgut and J. Bourgain. "Sharp thresholds of graph properties, and the k-sat problem". *Journal of the American Mathematical Society* 12.4 (1999), pp. 1017–1054.

[38] R. García-Pelayo and P. F. Stadler. "Correlation length, isotropy and meta-stable states". *Physica D: Nonlinear Phenomena* 107.2 (1997), pp. 240–254.

[39] F. Glover. "Tabu search — Part I". *ORSA Journal on Computing* 1.3 (1989), pp. 190–206.

[40] F. Glover and A. Punnen. “The travelling salesman problem: new solvable cases and linkages with the development of new approximation algorithms”. *Journal of the Operational Research Society* 48.5 (1997), pp. 502–510.

[41] A. Goerdt. “A threshold for unsatisfiability”. *Journal of Computer and System Sciences* 53.3 (1996), pp. 469–486.

[42] D. E. Goldberg. *Genetic Algorithms and Walsh Functions: A Gentle Introduction.* Clearinghouse for Genetic Algorithms, Department of Mechanical Engineering, University of Alabama, 1988.

[43] G. Gutin, B. Goldengorin, and J. Huang. “Worst case analysis of max-regret, greedy and other heuristics for multidimensional assignment and traveling salesman problems”. *Journal of Heuristics* 14.2 (2008), pp. 169–181.

[44] G. Gutin, A. Vainshtein, and A. Yeo. “Domination Analysis of Combinatorial Optimization Problems”. *Discrete Applied Mathematics* 129.2 (2003), pp. 513–520.

[45] G. Gutin and A. Yeo. “Polynomial Approximation Algorithms for the TSP and the QAP with a Factorial Domination Number”. *Discrete Applied Mathematics* 119.1 (2002), pp. 107–116.

[46] G. Gutin and A. Yeo. “TSP tour domination and Hamilton cycle decompositions of regular digraphs”. *Operations Research Letters* 28.3 (2001), pp. 107–111.

[47] G. Gutin, A. Yeo, and A. Zverovich. “Exponential Neighborhoods and Domination Analysis for the TSP”. In *The Traveling Salesman Problem and its Variations.* Ed. by G. Gutin and A. Punnen. Boston: Kluwer Academic Publishers, 2002, pp. 223–256.

[48] G. Gutin, A. Yeo, and A. Zverovich. “Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP.” *Discrete Applied Mathematics* 117.1 (2002), pp. 81–86.

[49] G. Gutin and A. P. Punnen. *The Traveling Salesman Problem and Its Variations.* Vol. 12. Springer Science & Business Media, 2006.

[50] G. Gutin and A. Yeo. “Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number”. *Discrete Applied Mathematics* 119.1-2 (2002), pp. 107–116.

[51] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. "Lest we remember: cold-boot attacks on encryption keys". *Communications of the ACM* 52.5 (2009), pp. 91–98.

[52] R. Hassin and S. Khuller. "z-Approximations". *Journal of Algorithms* 41.2 (2001), pp. 429–442.

[53] J. Håstad. "Some optimal inapproximability results". *Journal of the ACM (JACM)* 48.4 (2001), pp. 798–859.

[54] R. B. Heckendorn, S. Rana, and D. Whitley. "Polynomial time summary statistics for a generalization of MAXSAT". In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann, 1999, pp. 281–288.

[55] N. A. Heninger. "Error correction and the cryptographic key". PhD thesis. Princeton University, 2011.

[56] F. Heras, J. Larrosa, and A. Oliveras. "MiniMaxSAT: An efficient Weighted Max-SAT solver". *Journal of Artificial Intelligence Research (JAIR)* 31 (2008), pp. 1–32.

[57] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[58] H. H. Hoos, K. Smyth, and T. Stützle. "Search space features underlying the performance of stochastic local search algorithms for MAX-SAT". In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*. Springer. 2004, pp. 51–60.

[59] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier, 2004.

[60] A. Kamal. "Cryptanalysis and Secure Implementation of Modern Cryptographic Algorithms". PhD thesis. Concordia University, 2012.

[61] A. Kamal and A. M. Youssef. "Applications of SAT solvers to AES key recovery from decayed key schedule images". In *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*. IEEE. 2010, pp. 216–220.

[62] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. “Optimization by simulated annealing”. *Science* 220.4598 (1983), pp. 671–680.

[63] A. E. Koller and S. D. Noble. “Domination analysis of greedy heuristics for the frequency assignment problem”. *Discrete Mathematics* 275.1 (2004), pp. 331–338.

[64] D. Kühn and D. Osthus. “Hamilton decompositions of regular expanders: a proof of Kelly’s conjecture for large tournaments”. *Advances in Mathematics* 237 (2013), pp. 62–146.

[65] D. Kühn, D. Osthus, and V. Patel. “A domination algorithm for $\{0,1\}$-instances of the traveling salesman problem”. *Random Structures & Algorithms* 48.3 (2016), pp. 427–453.

[66] C. M. Li and F. Manyà. “MaxSAT, hard and soft constraints”. In *Handbook of Satisfiability.* Ed. by A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh. Vol. 185. IOS Press, 2009, pp. 613–631.

[67] X. Liao. “Maximum Satisfiability Approach to Game Theory and Network Security”. PhD thesis. Kyushu University, 2013.

[68] X. Liao, H. Zhang, M. Koshimura, H. Fujita, and R. Hasegawa. “Using maxsat to correct errors in aes key schedule images”. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence.* IEEE. 2013, pp. 284–291.

[69] C. Luo, S. Cai, W. Wu, Z. Jie, and K.-W. Su. “CCLS: An efficient local search algorithm for weighted maximum satisfiability”. *IEEE Transactions on Computers* 64.7 (2014), pp. 1830–1843.

[70] C. Luo, S. Cai, K. Su, and W. Wu. “Clause states based configuration checking in local search for satisfiability”. *IEEE transactions on Cybernetics* 45.5 (2014), pp. 1028–1041.

[71] C. Luo, S. Cai, W. Wu, and K. Su. “Focused random walk with configuration checking and break minimum for satisfiability”. In *International Conference on Principles and Practice of Constraint Programming.* Springer. 2013, pp. 481–496.

[72] K. M. Malan and A. P. Engelbrecht. “A survey of techniques for characterising fitness landscapes and some possible ways forward”. *Information Sciences* 241 (2013), pp. 148–163.

[73] S. Mertens, M. Mézard, and R. Zecchina. “Threshold values of random k-SAT from the cavity method”. *Random Structures & Algorithms* 28.3 (2006), pp. 340–373.

[74] P. Merz and B. Freisleben. “Fitness Landscapes and Memetic Algorithm Design”. In *New Ideas in Optimization.* Ed. by D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price. McGraw-Hill, 1999, pp. 245–260.

[75] M. Mézard, G. Parisi, and R. Zecchina. “Analytic and algorithmic solution of random satisfiability problems”. *Science* 297.5582 (2002), pp. 812–815.

[76] P. Mills and E. Tsang. “Guided local search for solving SAT and weighted MAX-SAT problems”. *Journal of Automated Reasoning* 24.1-2 (2000), pp. 205–223.

[77] H. Mühlenbein. “Genetic Algorithms”. In *Local Search in Combinatorial Optimization.* Ed. by E. Aarts and J.-K. Lenstra. Wiley, 1997, pp. 137–171.

[78] N. Narodytska and F. Bacchus. “Maximum Satisfiability using core-guided MaxSAT resolution”. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence.* AAAI Press, 2014, pp. 2717–2723.

[79] D. Pankratov and A. Borodin. “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. Edinburgh, UK: Springer-Verlag, 2010, pp. 223–236.

[80] V. Phan, S. Skiena, and P. Sumazin. “A model for analyzing black-box optimization”. In *Lecture Notes in Computer Science.* Vol. 2748. Springer-Verlag, 2003, pp. 424–438.

[81] L. S. Pitsoulis and M. G. C. Resende. “Greedy randomized adaptive search procedures”. In *Handbook of Applied Optimization.* Ed. by P. M. Pardalos and M. G. C. Resende. Oxford University Press, 2002, pp. 178–183.

[82] M. Poloczek. "Bounds on greedy algorithms for MAX SAT". In *Proceedings of the 19th European Conference on Algorithms.* ESA'11. Saarbrücken, Germany: Springer-Verlag, 2011, pp. 37–48.

[83] M. Poloczek, G. Schnitger, D. P. Williamson, and A. Van Zuylen. "Greedy algorithms for the maximum satisfiability problem: Simple algorithms and inapproximability bounds". *SIAM Journal on Computing* 46.3 (2017), pp. 1029–1061.

[84] M. Poloczek and D. P. Williamson. "An experimental evaluation of fast approximation algorithms for the maximum satisfiability problem". In *International Symposium on Experimental Algorithms.* Springer. 2016, pp. 246–261.

[85] A. Prügel-Bennett and M.-H. Tayarani-Najaran. "Maximum satisfiability: anatomy of the fitness landscape for a hard combinatorial optimization problem". *IEEE Transactions on Evolutionary Computation* 16.3 (2012), pp. 319–338.

[86] A. Punnen and S. Kabadi. "Domination analysis of some heuristics for the asymmetric traveling salesman problem". *Discrete Applied Mathematics* 119.1 (2002), pp. 117–128.

[87] A. Punnen, F. Margot, and S. Kabadi. "TSP Heuristics: Domination Analysis and Complexity". *Algorithmica* 35.2 (2003), pp. 111–127.

[88] A. Punnen, P. Sripratak, and D. Karapetyan. "Domination analysis of algorithms for bipartite boolean quadratic programs". In *Proceedings of the International Symposium on Fundamentals of Computation Theory (FCT).* Liverpool, United Kingdom, Aug. 2013, pp. 271–282.

[89] M. Qasem and A. Prügel-Bennett. "Learning the large-scale structure of the MAX-SAT landscape using populations". *IEEE Transactions on Evolutionary Computation* 14.4 (2010), pp. 518–529.

[90] G. Reinelt. "TSPLIB — a traveling salesman problem library". *ORSA journal on computing* 3.4 (1991). See also `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`, pp. 376–384.

[91] S. Ross. *A First Course in Probability.* 5th. Prentice Hall, 1998.

[92] S. Ross. *A First Course in Probability.* 8th ed. Pearson Education, 2010.

[93] V. I. Rublineckii. "Estimates of the accuracy of procedures in the Traveling Salesman Problem". *Numerical Mathematics and Computer Technology (in Russian)* 4 (1973), pp. 18–23.

[94] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* 2nd. Prentice Hall, 2003.

[95] V. Sarvanov and N. Doroshko. "The approximate solution of the traveling salesman problem by a local algorithm that searches neighborhoods of exponential cardinality in quadratic time". *Software: Algorithms and Programs (in Russian)* 31 (1981), pp. 8–11.

[96] V. Sarvanov and N. Doroshko. "The approximate solution of the traveling salesman problem by a local algorithm that searches neighborhoods of factorial cardinality in cubic time". *Software: Algorithms and Programs (in Russian)* 31 (1981), pp. 11–13.

[97] B. Selman, H. A. Kautz, and B. Cohen. "Local search strategies for satisfiability testing". In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* 1996, pp. 521–532.

[98] B. Selman, H. A. Kautz, and B. Cohen. "Noise strategies for improving local search". In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1).* American Association for Artificial Intelligence, 1994, pp. 337–343.

[99] B. Selman, H. Levesque, and D. Mitchell. "A new method for solving hard satisfiability problems". In *Proceedings of the Tenth National Conference on Artificial Intelligence.* AAAI Press, 1992, pp. 440–446.

[100] K. Smyth, H. H. Hoos, and T. Stützle. "Iterated robust tabu search for MAX-SAT". In *Conference of the Canadian Society for Computational Studies of Intelligence.* Springer. 2003, pp. 129–144.

[101] P. Stadler. "Fitness landscapes". In *Biological Evolution and Statistical Physics.* Ed. by M. Lässig and A. Valleriani. Springer, 2002, pp. 183–204.

[102] *Sun Grid Engine (SGE) QuickStart.* URL: `http://star.mit.edu/cluster/docs/0.93.3/guides/sge.html`.

[103] A. M. Sutton, L. D. Whitley, and A. E. Howe. “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. 2009, pp. 365–372.

[104] D. A. Tompkins and H. H. Hoos. “UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT”. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2005, pp. 306–320.

[105] Y. Twitto. “Dominance guarantees for above-average solutions”. *Discrete Optimization* 5.3 (2008), pp. 563–568.

[106] C. P. Williams and T. Hogg. “Exploiting the deep structure of constraint problems”. *Artificial Intelligence* 70.1 (1994), pp. 73–117.

[107] M. Yannakakis. “On the approximation of maximum satisfiability”. *Journal of Algorithms* 17.3 (1994), pp. 475–502.

[108] E. Zemel. “Measuring the quality of approximate solutions to zero-one programming problems”. *Mathematics of Operations Research* 6.3 (1981), pp. 319–332.