**UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS**

Heitor Pascoal de Bittencourt

**Search strategies and phase transition in the Random
Boolean satisfiability problem**

São Carlos

2021

Heitor Pascoal de Bittencourt

**Search strategies and phase transition in the Random
Boolean satisfiability problem**

Dissertation presented to the Graduate Program in Physics at the Instituto de Física de São Carlos, Universidade de São Paulo to obtain the degree of Master of Science.

Concentration area: Basic Physics

Supervisor: Prof. Dr. José Fernando Fontanari

Corrected version
(Original version available on the Program Unit)

São Carlos
2021

I AUTHORIZE THE REPRODUCTION AND DISSEMINATION OF TOTAL OR PARTIAL COPIES OF THIS DOCUMENT, BY CONVENTIONAL OR ELECTRONIC MEDIA FOR STUDY OR RESEARCH PURPOSE, SINCE IT IS REFERENCED.

de Bittencourt, Heitor Pascoal

Search strategies and phase transition in the Random Boolean satisfiability problem / Heitor Pascoal de Bittencourt; advisor José Fernando Fontanari - corrected version -- São Carlos 2021.

74 p.

Dissertation (Master's degree - Graduate Program in Basic Physics) -- Instituto de Física de São Carlos, Universidade de São Paulo - Brasil , 2021.

1. NP-complete problem. 2. Boolean satisfiability problem. 3. Local search algorithms. 4. Phase transitions. 5. Random problems. I. Fontanari, José Fernando, advisor. II. Title.

ACKNOWLEDGEMENTS

To my supervisor, Dr. José Fernando Fontanari, for the trust, patience, and enthusiasm when I knocked on his door to discuss the possibilities for a Master's thesis, and all the support and assistance to make this work happen, thank you.

To Dr. Sandro M. Reia, for the discussions, burgers, and help validating my results, thank you.

To the Professors, Technicians, and staff from the Institute of Physics of São Carlos, for the knowledge I acquired during these years, thank you.

To the Professors, Researches, and staff from Helsinki Institute of Physics and CERN, for the lectures, discussions, and hikes during my exchanges, thank you.

To all the colleagues I met during my Bachelor, Exchange, and Master's studies, for all the nights with unlimited coffee on the exam's eve, for all the game nights, barbecues, and parties, thank you.

To my family, for supporting my studies and investing in my education, for trust and encouragement, for the patience and kindness, thank you.

To my girlfriend, for the support, encouragement, and company during my studies and writing, thank you.

To Wikipedia and those-two-websites-that-cannot-be-named, for sharing free knowledge and allowing me to read and study books and papers that otherwise would be impossible, thank you.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

*"It's a dangerous business, Frodo, going out of your door,"
he used to say.*

*"You step into the Road, and if you don't keep your feet,
there is no knowing where you might be swept off to. "*

Bilbo Baggins in J. R. R. Tolkien's The Lord of The Rings

ABSTRACT

de BITTENCOURT, H. P. **Search strategies and phase transition in the Random Boolean satisfiability problem.** 2021. 74p. Dissertation (Master in Science) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2021.

The Boolean Satisfiability Problem is the problem of deciding if a given Boolean formula, such as $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1) \wedge (x_2 \vee x_3)$ is satisfiable, that is, if there is an assignment of *True* or *False* to the logical variables x_1 , x_2 and x_3 such that the formula evaluates to *True*. This was the first problem proved to be NP-complete, which means that there is no known algorithm that can solve it with a running time that scales polynomially with the problem size in a worst-case scenario. Here we study random Boolean formulas with fixed number of variables N and number of clauses M that are generated by choosing randomly the variables that appear in each clause and negating them with probability $1/2$. We solve those formulas using a random-walk based, local search algorithm known as WalkSAT. We show that the WalkSAT can be used to study a remarkable property of the ensemble of random Boolean formulas – there is a critical value of the clauses-to-variables ratio M/N that separates satisfiable from unsatisfiable formulas in the limit of large N – and we characterize the critical region, or the sharpness of the transition, for finite N using finite-size scaling. From the perspective of computer science, this transition is important because satisfiable random formulas with the ratio M/N near the transition point are hard to solve, in the sense that WalkSAT requires much more time to find their solutions than in the case that ratio is far from the critical region. We show that a collective search strategy where several WalkSATs run in parallel and halt when one of them finds the solution results in a sub-linear speedup, that is, the speedup is less than the number of WalkSATs used in the collective search.

Keywords: NP-complete problem. Boolean satisfiability problem. Local search algorithms. Phase transitions. Random problems.

RESUMO

de BITTENCOURT, H. P. **Estratégias de busca e transição de fase no problema da satisfatibilidade Booleana aleatório**. 2021. 74p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2021.

O problema da satisfatibilidade booliana é o problema de decidir se uma determinada fórmula booliana, como $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1) \wedge (x_2 \vee x_3)$ é satisfatível, ou seja, se há uma atribuição de *True* ou *False* às variáveis lógicas x_1 , x_2 e x_3 de modo que a fórmula seja avaliada como *True*. Este foi o primeiro problema provado ser NP-completo, o que significa que não há algoritmo conhecido que pode resolvê-lo com um tempo de execução que escale polinomialmente com o tamanho do problema em um cenário de pior caso. Nessa dissertação, estudamos fórmulas boolianas aleatórias com número fixo de variáveis N e número de cláusulas M que são geradas escolhendo aleatoriamente as variáveis que aparecem em cada cláusula e negando-as com probabilidade $1/2$. Resolvemos essas fórmulas usando um algoritmo de busca local baseado em passeio aleatório conhecido como WalkSAT. Mostramos que o WalkSAT pode ser usado para estudar uma propriedade notável do conjunto de fórmulas boolianas aleatórias – há um valor crítico da razão entre cláusulas e variáveis M/N que separa as fórmulas satisfatíveis das insatisfatíveis no limite de N grande – e caracterizamos a região crítica, ou a agudeza da transição, para N finito usando a teoria de escala de tamanho finito. Do ponto de vista da ciência da computação, essa transição é importante porque fórmulas aleatórias satisfatíveis com a razão M/N perto do ponto de transição são difíceis de resolver, no sentido que o WalkSAT requer muito mais tempo para encontrar suas soluções do que no caso em essa razão está longe da região crítica. Mostramos que uma estratégia de busca coletiva onde vários WalkSATs rodam em paralelo e param quando um deles encontra a solução resulta em um aumento sublinear da velocidade da busca, ou seja, o aumento de velocidade é menor do que o número de WalkSATs usados na busca coletiva.

Palavras-chave: Problema NP-completo. Problema de satisfatibilidade booliana. Algoritmos de busca local. Transição de fase. Problemas aleatórios.

LIST OF FIGURES

Figure 1 – Probability ρ that a random 3-SAT in conjunctive normal form is satisfiable. For small values of $\alpha = M/N$, there is a high probability that a random SAT instance has a solution. The higher the clauses-to-variables ratio, the smaller the probability of existing a solution.	35
Figure 2 – Fraction of the solution space that satisfies a random 3-CNF problem. This fraction is calculated by dividing the number of assignments that satisfy a formula by the total number of possible assignments, 2^N . For small values of α , a considerable fraction of the solution space satisfies the formula. This fraction decreases sharply with increasing values of α . For a fixed α , an increase in the number of variables decreases significantly the number of correct assignments. Each point represents an average over at least 10^5 instances. We note that for $N = 25$ we used about 10^8 instances in order to obtain a reliable statistics. The error bars are smaller than the sizes of the symbols.	36
Figure 3 – Ratio between the standard deviation and the mean number of solutions of satisfiable random 3-CNF problems. The peak in the region around $\alpha_c \approx 4$ is the mark of the threshold phenomenon and we expect that its height will diverge in the limit $N \rightarrow \infty$	37
Figure 4 – Time needed to explore the entire solution space for N variables and check the resulting value of the formula. Code was executed in a modern laptop with an Intel® Core™ i5-4210H CPU running at $3.50GHz$. Both panels show the same data using a linear plot in panel (a) and a semi-log plot in panel (b). Each point represents an average over at least 10^5 and at most 10^7 instances. Note that the horizontal scales varies from 5 to 25, while the vertical scale spans about 6 orders of magnitude: from around $100ns$ to $0.1s$. The exhaustive search is viable for small solution spaces only. For $N = 50$ it would take 26 days to explore the whole solution space and for $N = 60$, 73 years.	38
Figure 5 – Number of unsatisfied clauses as a function of the number of steps t for two different runs of the WalkSAT on the same solvable Boolean formula with $N = 100$ variables and $M = 420$ clauses. The noise parameter is $P = 0.5$. Notice the enormous discrepancy between the search times t^* for the runs illustrated in the panels (a) and (b).	42

Figure 6 – Probability that the WalkSAT solves a random 3-CNF formula at time t^* for $\alpha = 3$. The values of the noise parameter P , as well as the number of variables N , are as indicated. The search time upper bound is $T_{max} = 10^6$	44
Figure 7 – Mean time $\langle t^* \rangle$ for the WalkSAT to solve a random 3-CNF formula as function of the number of variables N , for $\alpha = 3$ and for the noise parameter $P = 0.2, 0.4, 0.5$, and 0.8 as indicated. The search time upper bound is $T_{max} = 10^6$	45
Figure 8 – Probability that the WalkSAT solves a random 3-CNF formula at time t^* for $N = 50$ and $\alpha = 3, 4.1$, and 5 . The values of the noise parameter P are as indicated. The search time upper bound is $T_{max} = 10^6$	46
Figure 9 – Mean time $\langle t^* \rangle$ for the WalkSAT solve a random 3-CNF formula as function of the number of variables N for $\alpha = 4.1$ and $\alpha = 5$. The noise parameter is $P = 0.2, 0.4, 0.5$, and 0.8 as indicated. The search time upper bound is $T_{max} = 10^6$	47
Figure 10 – Mean time $\langle t^* \rangle$ for the WalkSAT solve a random 3-CNF formula as function of the number of variables N for $\alpha = 4.1$ and search time upper bound $T_{max} = 10^3, 10^4, 10^5$, and 10^6 . The noise parameter is $P = 0.2, 0.4, 0.5$ and 0.8 as indicated.	48
Figure 11 – Median $\tilde{t}^*$ of the probability distribution of the search times of the WalkSAT as function of the number of variables N for $\alpha = 4.1$ and search time upper bound $T_{max} = 10^3, 10^4, 10^5$, and 10^6 . The noise parameter is $P = 0.2, 0.4, 0.5$, and 0.8 as indicated.	49
Figure 12 – Fraction of random 3-CNF formulas for which the WalkSAT found a solution within the search time upper bound $T_{max} = 10^6$ as function of ratio of clauses-to-variables α . The number of variables is $N = 20, 30, 40, 50, 60$, and 80 as indicated. The noise parameter is $P = 0.5$	50
Figure 13 – Order parameter ρ against the scaled distance to the critical point $y = N^{1/\nu}(\alpha - \alpha_c)$ for the number of variables $N = 20, 30, 40, 50, 60$, and 80 as indicated. The noise parameter is $P = 0.5$. The critical point is $\alpha_c = 4.1$ and the critical exponent is $\nu = 1.5$	51
Figure 14 – Median search time $\tilde{t}^*$ as function of ratio of clauses-to-variables α . The number of variables is $N = 20, 30, 40, 50, 60$, and 80 as indicated. The noise parameter is $P = 0.5$ and the search time upper bound is $T_{max} = 10^6$	52

Figure 15 – Median search time $\tilde{t}^*$ as function of the number of logical variables N for $\alpha = 3, 4.1$ and 5 as indicated. The noise parameter is $P = 0.5$ and the search time upper bound is $T_{max} = 10^6$. The solid lines are the fittings $\tilde{t}^* = \tilde{a}/N^{\tilde{b}}$. For $\alpha = 3$ we find $\tilde{b} \approx 1.09$; for $\alpha = 4.1$ we find $\tilde{b} \approx 2.53$; and for $\alpha = 5$ we find $\tilde{b} \approx 1.98$	53
Figure 16 – Probability that L independent WalkSATs running in parallel solve a random 3-CNF formula at time t^* for $N = 50$, $P = 0.5$ and $\alpha = 3, 4.1$ and 5 as indicated. The search time upper bound is $T_{max} = 10^6$	56
Figure 17 – Cumulative distribution that L independent WalkSATs running in parallel solve a random 3-CNF formula before or at time t^* for $N = 50$, $P = 0.5$ and $\alpha = 3, 4.1$ and 5 as indicated. The search time upper bound is $T_{max} = 10^6$	56
Figure 18 – Mean time $\langle t^* \rangle$ (panel a) and median time $\tilde{t}^*$ (panel b) for L independent WalkSATs running in parallel to solve a random 3-CNF formula as function of the number of agents L for $\alpha = 3, 4.1$ and 5 as indicated. The other parameters are $N = 50$, $P = 0.5$ and $T_{max} = 10^6$. The dashed lines are the fittings $\langle t^* \rangle = a/L^b$ and $\tilde{t}^* = \tilde{a}/L^{\tilde{b}}$ for large L . For $\alpha = 3$ we find $b = \tilde{b} \approx 0.2$; for $\alpha = 4.1$ we find $b \approx 0.41$ and $\tilde{b} \approx 0.29$; and for $\alpha = 5$ we find $b \approx 0.32$ and $\tilde{b} \approx 0.29$	57
Figure 19 – Euler diagrams for P, NP, NP-complete, and NP-hard computational complexity classes. The left panel is valid under the assumption that $P \neq NP$, whereas the right panel is valid under the assumption that $P=NP$	68
Figure 20 – Fraction of unsatisfied clauses for a randomly generated assignment of many random 3-SAT formulas. Our experiment confirms that a random assignment of the literals leaves on average only 12.5% of the clauses unsatisfied. This result does not depend on the numbers of variables and clauses. Note the small range of the vertical axis, which varies from 12.46% to 12.54%.	70
Figure 21 – Minimum search time t_{min}^* for the WalkSAT to solve a random 3-CNF as a function of the number N of variables. The dashed straight line is the linear fitting $t_{min}^* = -32 + 0.44N$. The parameters are $\alpha = 3$, $P = 0.5$ and $T_{max} = 10^6$	71

LIST OF ALGORITHMS

1	Algorithm to generate a random SAT instance in conjunctive normal form, with k literals per clause, M clauses, and N variables.	33
2	Procedure to evaluate a SAT instance in CNF for a given assignment on the input variables.	34
3	The WalkSAT algorithm returns a solution to the Boolean formula F or a message ‘No assignment found’ if the number of iterations exceeds the parameter T_{max} . The noise parameter P allows the algorithm to perform non-optimal moves to avoid getting stuck in sub-optimal regions of the space of assignments.	40
4	Procedure to find the best variable x_l to flip in an unsatisfied clause C of a Boolean formula F . It identifies the label l of the variable that minimizes the number of clauses that are satisfied with S but would become unsatisfied if the variable is flipped.	41

LIST OF ABBREVIATIONS AND ACRONYMS

CNF	conjunctive normal form
DNF	disjunctive normal form
DTM	deterministic one-tape Turing Machine
ETH	exponential time hypothesis
Max-SAT	maximum satisfiability problem
NP	non-deterministic polynomial time
NTM	non-deterministic Turing Machine
P	deterministic polynomial time
SAT	Boolean satisfiability problem
SEHT	strong exponential time hypothesis
WSAT	WalkSAT
#SAT	sharp satisfiability problem

LIST OF SYMBOLS

$\wedge$	logical AND operator
$\neg$	logical NOT operator
$\vee$	logical OR operator

CONTENTS

1	INTRODUCTION	23
2	THE BOOLEAN SATISFIABILITY PROBLEM	27
2.1	Definitions	27
2.2	Complexity Classes	28
2.3	Variations and Extensions	29
2.3.1	k-SAT	29
2.3.2	Max-SAT	30
2.3.3	#SAT	31
2.4	Characterization of 3-SAT	32
3	THE WalkSAT ALGORITHM	39
3.1	Outline of Algorithm	39
3.1.1	Implementation Tips	41
3.2	Analysis of the search times	42
3.2.1	Probability distribution of the search times	43
3.2.2	Phase transition	49
4	COLLECTIVE SEARCH	55
5	CONCLUSIONS	59
	REFERENCES	61
	APPENDIX	65
	APPENDIX A – TURING MACHINES	67
	APPENDIX B – RANDOM ASSIGNMENTS	69
	APPENDIX C – WalkSAT MINIMUM SEARCH TIME	71
	APPENDIX D – COLLECTIVE BLIND SEARCH	73

1 INTRODUCTION

The Boolean Satisfiability Problem (also known as SAT) is probably the most important decision problem of Computer Science: given a Boolean formula, such as $(x_1 \vee (x_2 \wedge \neg x_3)) \wedge (\neg x_1) \wedge (x_2 \vee x_3)$, is it satisfiable? In other words, is it possible to find an assignment of *True* or *False* to the logical variables x_1 , x_2 and x_3 such that the formula evaluates to *True*? In fact, the Boolean Satisfiability Problem was the first problem proved to be NP-Complete.¹ This was accomplished in 1971 by Stephen Cook,² and, independently, by Leonid Levin in 1973.^{3, 4} Being NP-Complete means that SAT is among the hardest problems in the complexity class NP and is as hard as any problem in NP-Hard. SAT was the starting point to prove that many problems are also NP-Complete. In 1972, shortly after Cook's paper, Richard Karp published a paper in which he reduces SAT to 21 problems,⁵ showing they are all NP-Complete as well.

The heavy computational science jargon of the previous paragraph is almost unavoidable whenever one wants to advertise the importance and difficulty of a computational problem, regardless of whether it is a decision problem, such as the SAT as stated above, or an optimization problem, such as finding the assignment that solves a Boolean formula. In Chapter 2 we offer a formal description of the computational classes¹ that the computer scientists use to classify the problems and clarify the concepts that are briefly mentioned in this introductory section. Here we note that a proper scientific answer to the question “what is a computation?” was not available until 1936, despite the fact that humans have been calculating things for thousands of years.⁶ Those calculations follow mechanical, step-by-step procedures, which we call algorithms. It was Alan Turing who first realized that it would be possible to construct an abstract computing machine that could carry out all computations specified by any algorithm.⁷ It turned out that the so-called Turing machine is the most general type of computer imaginable: every real-world computer is just a particular realization of Turing's abstract machine. Turing's insight allowed the discovery that certain things are uncomputable and, to the world's surprise, that most real numbers are uncomputable, that is, there is no algorithm that will ever output those numbers.⁶

Then it is only natural to classify the problems according to whether they can or cannot be computed by a Turing machine and, in the affirmative case, how the computation time scales with the size of the problem (for example, the number of logical variables in the SAT). The Turing machine that is capable to implement any real-world algorithm is the deterministic Turing machine: given the state of the machine at a given time and the input to the machine at that time, then the machine specifies a unique action, as any known algorithm which is defined by a set of rules. The class that contains all decision

problems that can be solved by a deterministic Turing machine using a polynomial amount of computation time (in other words, the asymptotic computation time scales as a power of the problem size) is called the P class. Here P stands for polynomial. There are many important problems in P, such as calculating the greatest common divisor and determining if a number is prime.⁸

The computational complexity classification becomes much less intuitive when one goes beyond the P class. This is so because of the introduction of the concept of a non-deterministic Turing machine,⁹ which is essentially an abstract machine that guesses the solution of a problem in a non-deterministic way. It does not matter how it does that, because the relevant point is that the machine must verify if its guess is a solution to the problem and this is done with a deterministic algorithm (in this case, with a deterministic Turing machine). The NP class is defined as the set of decision problems solvable in polynomial time by a non-deterministic Turing machine. Here NP stands for non-deterministic polynomial time. Since the process of guessing the solution is instantaneous, only the verification stage demands computer time and for an NP problem this verification can be done in polynomial time. In fact, we note that although finding an assignment that solves a given Boolean formula may be difficult, we can quickly verify if a particular assignment solves the formula. Of course, the class P is contained in the class NP because if we can solve the problem in polynomial time, we can verify that a candidate solution is a true solution in polynomial time as well. We provide some insights into both versions of the Turing Machine in appendix A.

Another relevant property of decision problems from the perspective of the computational complexity classification scheme is the possibility of mapping, or reduction, of one problem into another problem.⁹ A problem, say H, is NP-hard when every problem in NP can be reduced in polynomial time to H. This means that if we solve quickly an NP-hard problem, then we can quickly solve all problems in NP. The important point here is that the NP-hard problem may not be in NP. By not being in NP means we cannot verify if a candidate solution is a true solution in polynomial time. The NP-complete class is the intersection between the NP-hard and the NP classes. This reveals the importance of designing algorithms for NP-complete problems, such as the SAT: if we find an efficient algorithm to solve SAT (not necessarily in polynomial time) then we can use it to solve all problems in NP. Actually, proving that there is, or there is not, a polynomial time algorithm to solve SAT is equivalent to solve the P versus NP (or $P=NP$) problem, one of the most important problems in Theoretical Computer Science.¹⁰ It is also one of the Millennium Prize Problems selected by the Clay Mathematics Institute that has an award of US\$1 million to the person who solves it.¹¹

From a more pragmatic perspective, the hardness, or difficulty, of a problem is viewed as a measure of the computational resources needed to solve it. The most common

metrics are time and memory, but it is not unusual to consider communication, number of logic gates, and number of CPUs as well. In fact, the *Exponential Time Hypothesis* (ETH), an unproven but widely believed computational hardness assumption, states that no NP-Complete problem can be solved in sub-exponential time in the worst case.¹² We note that a deterministic Turing machine can clearly solve NP-complete problems given enough time, even exponential time on the size of the input. For instance, the SAT can easily be solved by the exhaustive enumeration and test of the 2^N possible assignments of the N logical variables to the *True* and *False* values. Of course, the problem with the exhaustive enumeration is that the computation time scales exponentially with N . Hence, it is believed that SAT cannot be solved faster than $\mathcal{O}(2^{\delta N})$, for any constant $\delta > 0$.¹⁰

Besides its theoretical importance for Computer Science, SAT has many practical applications. It is used in automated theorem proving and verification in the field of satisfiability modulo theories.¹³ SAT is used in conjunction with Electronic Design Automation tools to verify and synthesize hardware designs,^{14, 15} to route field-programmable gate array (FPGA) boards, or demonstrate that a layout is infeasible.¹⁶ SAT is also used in Artificial Intelligence for automated planning,¹³ and as a cryptanalysis tool, for example to create hash collisions in MD4 and MD5.¹⁷ Such is the importance of SAT that there is an yearly conference dedicated to it, the *International Conference on Theory and Applications of Satisfiability Testing*.¹⁸ There is also a very practical yearly event affiliated with that conference, the *SAT Competition*, that benchmarks and ranks SAT solvers.¹⁹

In physics, Ising spin glasses are known to be NP-complete problems.²⁰ In these problems, the coupling between the Ising spins are random variables that can take on positive and negative values which result in a significant number of frustrated links.²¹ The frustration produces complex energy landscapes with many local minima, hence the difficulty to find the global minima (ground states) using the gradient descent methods that work very well for the ferromagnetic version of the Ising model. It is interesting to note that there are Ising formulations for many NP-complete problems, including all of Karp's 21 NP-complete problems.²²

In this dissertation, we address a particular version of SAT, namely, Boolean formulas in the conjunctive normal form (CNF), with exactly 3 variables in each clause. A formula is in CNF if and only if it is a conjunction of clauses, that is, the operation between clauses is the AND and the operation between variables is the OR. For example, the formula $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$ is in 3-CNF. It has $N = 4$ variables and $M = 3$ clauses. In addition, we assume that the variables that appear in each clause are selected at random without replacement from the set of N variables and once selected the variable is negated with probability $1/2$. In sum, our subject is the set of random 3-CNF formulas. These random objects exhibit an interesting property in the limits $N \rightarrow \infty$ and $M \rightarrow \infty$ with the ratio $\alpha = M/N$ fixed: there is a threshold

value α_c below which the formulas are satisfiable (they have a solution) and above which the formulas are unsatisfiable (they do not have a solution).^{23, 24} Although this is a pure property of the ensemble of random 3-CNF formulas, its numerical verification requires the use of efficient algorithms to solve those formulas.

Accordingly, in this work we study a simple local search heuristic for the SAT, the WalkSAT (or WSAT) algorithm.^{25, 26} In particular, we study how the performance of this algorithm depends on its only adjustable parameter – a noise parameter that, in principle, could prevent the algorithm to enter infinite loops. We show that the use of the WalkSAT allows the verification of the threshold phenomenon (or phase transition) mentioned before and we characterize the threshold using finite-size scaling tools.²⁷ In addition, we find that use of L independent WalkSATs running in parallel does not result in a net performance gain, since the collective search time decreases slower than $1/L$.

This dissertation is organized as follows. In chapter 2, we present the formal definition of the Boolean satisfiability problem and some of its variants, as well as a brief overview of the concepts of computational complexity necessary to appreciate the importance of SAT. In addition, we study the satisfiability and the number of solutions of random 3-CNF formulas using a simple brute-force algorithm, the exhaustive enumeration of assignments. In chapter 3, we introduce a more robust algorithm, the WalkSAT, to tackle problems that are impossible to solve with the brute-force approach. This algorithm is then used to verify and characterize the phase transition of random 3-CNF formulas. In Chapter 4, we investigate the use of several independent WalkSATs to speed up the search for solutions of our random formulas. Finally, Chapter 5 presents our concluding remarks.

2 THE BOOLEAN SATISFIABILITY PROBLEM

This chapter focus on the theory and characterization of the Boolean Satisfiability Problem. In particular, we will first define this classic decision problem of Computer Science and discuss briefly its computational complexity. Then we will use the exhaustive search to explore its space of solutions to reveal the existence of a critical phenomenon that separates the satisfiable and the unsatisfiable problems in the space of the parameters of the problem.

2.1 Definitions

The *Boolean Satisfiability Problem* (also called *SAT*) is the problem of determining if there is a solution to a given Boolean formula F that evaluates to *True*. In other words, the problem is to decide whether there is a combination of *True/False* for the input variables x_i that makes the formula evaluate to *True*, that is, $F(x_1, x_2, \dots, x_n) = \text{True}$.

A *Boolean formula* is a finite number of literals connected by logical operators and, possibly, parentheses. A *literal* is a logical variable x or its negation $\neg x$. The logical operators used are

- AND, or *conjunction*, denoted by $\wedge$.
- OR, or *disjunction*, denoted by $\vee$.
- NOT, or *negation*, denoted by $\neg$.

For example, the formula $F(x_1, x_2, x_3) = (x_1 \vee (x_2 \wedge \neg x_3)) \wedge \neg x_1 \wedge (x_2 \vee x_3)$ can be solved by choosing $x_1 = \text{False}$, $x_2 = \text{True}$ and $x_3 = \text{False}$, since $F(\text{False}, \text{True}, \text{False}) = (\text{False} \vee (\text{True} \wedge \neg \text{False})) \wedge \neg \text{False} \wedge (\text{True} \vee \text{False})$ is equivalent to $(\text{True}) \wedge (\text{True}) \wedge (\text{True})$, that is, *True*.

A Boolean formula F is said *satisfiable* if and only if there is an assignment of the input variables such that the formula evaluates to *True*. Otherwise, the formula is said *unsatisfiable*. In the previous example, the formula has a solution, therefore it is satisfiable. On the other hand, the formula $F(x) = x \wedge \neg x$ is unsatisfiable, as the literal x cannot be *True* and *False* at the same time.

A structure like $(l_1 \vee l_2 \vee \dots \vee l_k)$, where l_i are literals, is called a *clause*.²⁸ A clause has length k if it contains k elements. Note that a clause is a disjunction of literals. Conjunctions are not allowed in a clause. For example, $(x_1 \vee x_2 \vee \neg x_3)$ is a clause with 3 elements, while $(x_1 \vee x_2 \wedge \neg x_3)$ is not a clause.

Two general structures (canonical forms) of Boolean formulas are worth mentioning: formulas in *conjunctive normal form* (CNF) and in *disjunctive normal form* (DNF). A formula is in CNF if and only if it is a conjunction of clauses, that is, it is an *AND of ORs*. A formula is in DNF if and only if it is a disjunction of conjunctions, that is, it is an *OR of ANDs*.²⁹ For example, $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2)$ is in CNF and $(\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2)$ is in DNF.

For CNF problems, it is convenient to define a parameter α as the ratio between the number M of clauses and the number N of variables: $\alpha := M/N$. For example, a k -CNF with $M = 137$ clauses over $N = 42$ variables has $\alpha = 137/42 \approx 3.262$.

In the context of the SAT, the *solution space*, or *search space*, is the space that contains all possible assignments on the input variables. A solution space for N Boolean variables has 2^N unique points. Each point in this space represents a state of *True* or *False* for each variable. For example, for $N = 3$, one point in this space could be $(True, True, True)$.

Two different formulae F and F' are called *equisatisfiable* if and only if they are both satisfiable or they are both unsatisfiable.¹³ This concept is particularly important when converting formulas from a general format to CNF or DNF. It is mandatory that the new and the original formulae are equisatisfiable.

2.2 Complexity Classes

In Theoretical Computer Science, problems can be grouped together based on their computational difficulty. That is, problems with related difficulty, with respect to a computational resource, belong to the same complexity class. Each class is a mathematical definition of how many resources each problem in the set need.

The fundamental complexity class P contains all the decision problems that can be *solved* in polynomial time by a deterministic Turing Machine.²⁹ This means that the time needed to solve each problem in this class scales polynomially with the size of its input.

Decision problems are those that have ‘yes’ or ‘no’ as an answer, while *optimization* problems involve the finding of a solution that optimizes some quantity. Some problems can be stated as a decision or as an optimization problem. For example, the Shortest Tour Problem when stated as ‘given a list of cities and the distances between them, find the shortest tour that visits every city and returns to the origin’ is an optimization problem. However, when it is stated as ‘given a distance L , a list of cities and the distances between them, is there a tour of length at most L ?’, it is a decision problem. We note that the popular Traveling Salesperson Problem is a variant of the Shortest Tour Problem in which there is the additional constraint that each city must be visited only once.

The complexity class NP is the set of all decision problems that can be *verified* in

polynomial time by a deterministic Turing Machine.⁴ This means that, for every problem X in NP, there is an algorithm A that verifies in polynomial time if an input x gives the answer ‘yes’ or ‘no’. As a consequence, $P \in NP$, as any problem that can be solved in polynomial time can also be checked in polynomial time (by solving the problem and comparing with the given answer).

A different, but equivalent, definition for the NP class is that of all decision problems that can be *solved* in polynomial time by a *non-deterministic* Turing Machine. In fact, the letter N in the acronym NP stands for non-deterministic.⁴

An important tool for complexity analysis is the *reduction*. A reduction is an algorithm R that transforms the input for a problem X into an input for a problem Y , in a way that the result of problem Y in the transformed input is the same as the result of X in the original input: $X(x) = Y(R(x))$. The notation used for a reduction from problem X to problem Y is $X \leq Y$. If the reduction runs in polynomial time, it is called a polynomial time reduction, and the notation is $X \leq^P Y$.⁴

With the reduction tool, we can compare different problems to classify them according to their inherent difficulty. Suppose there is a problem X that we don’t know an algorithm that solves it, but we can make a polynomial time reduction to problem Y . If we know an algorithm that solves Y in polynomial time, then we can use it as a subroutine to solve X in polynomial time. We just proved that if $X \leq^P Y$ and $Y \in P$, then $X \in P$. Moreover, if we know that problem X cannot be solved in polynomial time, then Y cannot be solved in polynomial time either. This is easy to demonstrate by absurd: assume that Y can be solved in polynomial time, then we can use Y as a subroutine to solve X in polynomial time. This is a contradiction, therefore the assumption is false. We just proved that if $X \leq^P Y$ and $X \in NP$, then $Y \in NP$.

Reductions are also used to define the NP-Hard class: a decision problem X is in NP-Hard if for every problem $Y \in NP$, there is a polynomial time reduction from Y to X . This class contains all problems that are at least as hard as any NP problem. A problem is said to be NP-Complete if it is both in NP and in NP-Hard.²⁹

2.3 Variations and Extensions

The Boolean Satisfiability Problem for arbitrary formulas, or unrestricted SAT, has variations and extensions that are interesting to study on their own. We discuss some of them in this section.

2.3.1 k-SAT

The problem k-SAT (or k-CNF) is a restricted version of the general SAT problem in which the formula is in conjunctive normal form with at most k literals per clause.²⁹

However, some authors define k -SAT with exactly k literals per clause,^{1, 4, 13, 23} since clauses with $j < k$ literals can be written as clauses with k literals by simply repeating a literal $k - j$ times. For example, consider the clause $(x_1 \vee \neg x_2)$. To make it compatible with a 4-SAT instance, it suffices to rewrite it as $(x_1 \vee \neg x_2 \vee \neg x_2 \vee \neg x_2)$. Hence the definition of k -SAT used in this work considers exactly k literals per clause.

There are two special cases of k -SAT that belong to the complexity class P, namely, 1-SAT and 2-SAT. In fact, 1-SAT is trivial since a formula evaluates to *False* if it contains a contradiction of literals, otherwise it is satisfiable.²⁹ For example, $(x_1) \wedge (x_2) \wedge (\neg x_1)$ is unsatisfiable, as there is a contradiction for x_1 . However, $(x_1) \wedge (\neg x_2) \wedge (x_1)$ is satisfiable since $x_1 = \text{True}$ and $x_2 = \text{False}$ is a solution. 2-SAT is not a trivial problem, nonetheless it can be solved in linear time.³⁰

As for 3-SAT, it is as difficult as SAT. In fact, k -SAT with $k \geq 3$ is an NP-Complete problem and appears in Karp's famous list of 21 NP-Complete problems.⁵ It is important to note that both definitions of the k -SAT, with exactly k literals per clause or with up to k literals per clause, are in the same complexity class. A general SAT formula can be encoded in CNF using the rules of Boolean Algebra, but typically this yields an exponentially large formula. A tractable alternative is done by the Tseytin Transformation, which generates a linear number of clauses and introduces a linear number of new variables to generate an equisatisfiable new formula in CNF. Conversion from CNF to k -CNF can be done with a linear number of new clauses and variables.¹³ We note that this new formula is linear on the size of the original one, so evaluating this new formula still takes linear time on N .

According to the Exponential Time Hypothesis, for every k -SAT there is a constant s_k defined as the infimum of $\{\delta : \text{there is an algorithm that solves } k\text{-SAT in } \mathcal{O}(2^{\delta N})\}$. In other words, s_k is the largest lower bound of the set $\{\delta\}$ of all algorithms that solve k -SAT in $\mathcal{O}(2^{\delta N})$. Hence, s_1 and s_2 are zero, as there is a linear time algorithm for solving them. ETH states that for $k \geq 3$, $s_k > 0$ and $s_k \leq s_{k+1}$.¹² This implies, for example, that 4-SAT is at least as difficult as 3-SAT. In addition, the Strong Exponential Time Hypothesis (SETH) states that $s_\infty = 1$.³¹

2.3.2 Max-SAT

There is an optimization version of SAT named Maximum Satisfiability Problem (Max-SAT): it is the problem of finding the assignment that satisfies the maximum number of clauses of a given Boolean formula in CNF. It is easy to see that MAX-SAT is NP-Hard: if there were a polynomial time algorithm to discover the maximum number of satisfied clauses, one could use this algorithm to solve SAT in polynomial time.

A natural extension of this problem is the Weighted Maximum Satisfiability Problem (weighted Max-SAT), in which each clause has a weight associated. The goal is to maximize the combined weight of the satisfied clauses. We can see that Max-SAT is the particular

case where all clauses have the same weight value. ¹³

2.3.3 #SAT

The Sharp Satisfiability Problem (#SAT) extends SAT as a counting problem: given a Boolean formula F , the task is to discover the number of its solutions. In other words, it is the problem of counting the number of different assignments for the input variables that satisfy F . For example, $(x_1 \wedge \neg x_2)$ has only one solution, whereas $(x_1 \vee \neg x_2)$ has three. We note that counting problems have integer numbers as solutions. We are interested in the cardinality of the set of solutions, not on the solutions themselves. Even though there might be an exponential number of solutions and it would need an exponential time to list them all, the size of this set can be written with a polynomial number of bits.

Counting problems associated with NP-Complete problems are clearly NP-Hard. If we know the cardinality of the set of solutions, we can easily check if it is non-zero and then decide if the problem has a solution. A consequence of this is that #3-SAT, and the general #SAT, is NP-Hard. There are, however, counting problems that can be solved in polynomial time. An example is counting the number of spanning trees in a general graph using Kirchhoff's matrix tree theorem. ¹

Counting problems have their own complexity classes, namely, #P, #P-Hard and #P-Complete as defined by Valiant. ^{32, 33} The #P class is the class that contains all counting problems, that is, all the problems that count the number of accepting paths of a polynomial time non-deterministic Turing Machine. A problem belongs to the class #P-Hard when every other problem in the class #P can be reduced to it in polynomial time. The class #P-Complete contains all problems that are simultaneously in #P and in #P-Hard. Finding a polynomial time algorithm for a #P-Complete problem would solve the P versus NP problem with the conclusion that both classes are equal. With these definitions, we can classify #SAT as #P-Complete. This is a very strong claim: even though we know that there are easy to solve SAT cases, such as 2-SAT and SAT in DNF, #2-SAT and #DNF are #P-Complete. ¹

As a more elaborate example, let us consider the following Boolean formula F^* on N input variables

$$\begin{aligned}
 F^* = & (x_1 \vee x_2 \vee x_3 \vee \dots \vee x_N) \wedge \\
 & (\neg x_1 \vee \neg(x_2 \vee x_3 \vee \dots \vee x_N)) \wedge \\
 & (\neg x_2 \vee \neg(x_1 \vee x_3 \vee \dots \vee x_N)) \wedge \\
 & (\neg x_3 \vee \neg(x_1 \vee x_2 \vee \dots \vee x_N)) \wedge \\
 & \vdots \\
 & (\neg x_N \vee \neg(x_1 \vee x_2 \vee \dots \vee x_{N-1})) .
 \end{aligned} \tag{2.1}$$

The first clause of this formula implies that at least one variable is *True*. The following clauses imply that if $\neg x_i$ is *False*, then all remaining variables must be *False*, so that the negation of their disjunction evaluates to *True*. This formula has N solutions, and each solution has exactly one *True* variable and $N - 1$ *False* variables.

2.4 Characterization of 3-SAT

To better understand the satisfiability problem, we developed a simulation in *C++*. Our code is capable of generating random SAT instances in conjunctive normal form with a given number k of literals per clause. In order to avoid generating near-trivial clauses, we explicitly forbid the repetition of a variable in a same clause. For example, clauses like $(x_1 \vee x_2 \vee x_2)$ or $(x_1 \vee x_2 \vee \neg x_2)$ are not present in our formulas.

It is important to note that all clauses in a generated CNF instance have the same number of literals. This choice is based on the work by Selman and aims to generate difficult SAT instances only. In fact, instances that have on average k literals per clause are easier to solve than those that have exactly k literals per clause.³⁴

As already pointed out, our work focuses on satisfiability in 3-CNF, that is, 3-SAT. This version exhibits a simpler structure than the unrestricted version, making it easier to develop and implement algorithms, as well as to represent the problem in the computer memory. There is a common file format for SAT solvers, the DIMACS file format.^{13, 35} In addition, 3-SAT is one of the usual starting points for proving NP-Completeness, as it appears in Karp's list and in several textbooks.^{1, 4, 5, 29}

To generate an instance of the k -SAT, we need three input parameters: the number N of variables, the number M of clauses, and the number k of literals per clause, which we fix to $k = 3$. We then proceed to generate each one of the M clauses randomly. More pointedly, with probability $1/N$ we pick a variable x_i and with probability $1/2$ we negate it. If x_i is not already present in the clause, we add this literal to the clause. This process is repeated until there are exactly k literals in the clause and there are M clauses in the formula. This procedure is illustrated below in algorithm 1.

Our code can generate instances in a simplified DIMACS format, as well as read previously saved ones. This format is a text file with three parts: an optional header with comments, the format of the instance, and a list of clauses. The header can have an arbitrary number of lines, as long as they begin with the letter *c*. The problem format line starts with the letter *p*, then states *cnf* or *dnf*, and the number of variables and clauses. The list of clauses contains one clause per line, each line is a sequence of distinct integers, from $-N$ to N , negative numbers denote a negation of that variable and the number zero denotes the end of that clause.³⁵ An example is given below:

c DIMACS format example

```

input :  $N, M, k$ 
output:  $k$ -SAT instance with  $M$  clauses over  $N$  variables
1 Formula  $\leftarrow$  empty array;
2 for  $j \leftarrow 0$  to  $M$  do
3   Clause  $\leftarrow$  empty array;
4   while  $|Clause| < k$  do
5     variable  $\leftarrow$  UniformRandomNumber(1,  $N$ );
6     sign  $\leftarrow$  UniformRandomChoice( $-1, 1$ );
7     if variable not in Clause then
8       literal  $\leftarrow$  sign * variable;
9       Clause  $\leftarrow$  add literal;
10    end
11  end
12  Formula  $\leftarrow$  add Clause;
13 end
14 return Formula;

```

Algorithm 1: Algorithm to generate a random SAT instance in conjunctive normal form, with k literals per clause, M clauses, and N variables.

```

c k-SAT with K = 3
p cnf 10 2
10 -5 -9 0
-10 -8 1 0

```

In this example, the header has two lines, the problem is in CNF with $N = 10$ variables and $M = 2$ clauses. The formula is $F = (x_{10} \vee \neg x_5 \vee \neg x_9) \wedge (\neg x_{10} \vee \neg x_8 \vee x_1)$.

In our software, an assignment S is a Boolean vector of length N . For example, an assignment for $N = 4$ could be $S = (False, True, True, False)$. An instance F can evaluate to either *True* or *False* under a given assignment. In other words, $F(S)$ can return *True* or *False*. For example, the formula $F = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_4)$ evaluates to $F(S) = True$ under the previous exemplified assignment, but $F(False, False, True, False) = False$ as the first clause is not satisfied.

To check if a SAT can be satisfied by an assignment, the software loops over all clauses and checks if it contains is at least one literal that is *True*. If this condition is not satisfied, then this instance evaluates to *False* under this assignment. On the other hand, if no clauses are *False*, then the instance evaluates to *True* under this assignment. This is illustrated in algorithm 2 below.

Note that algorithm 2 runs in polynomial time in the length of the formula F . For a k -CNF instance with M clauses, the worst-case running time is $\mathcal{O}(kM)$. This proves that SAT in CNF is an NP problem.

```

input  : Formula  $F$ , assignment  $S$ 
output:  $True$  if  $F(S) == True$ ,  $False$  otherwise

1 for each clause  $C$  in formula  $F$  do
2    $this\_clause \leftarrow False$ ;
3   for each literal  $l$  in  $C$  do
4     // evaluate the clause
5     if  $l > 0$  then
6       // that is, literal not negated
7       if  $S_l == True$  then
8          $this\_clause \leftarrow True$ ;
9       end
10    else
11      // that is, literal negated
12      if  $S_l == False$  then
13         $this\_clause \leftarrow True$ ;
14      end
15    end
16  end
17  if  $this\_clause == False$  then
18    // return  $False$  on first false clause
19    return  $False$ ;
20  end
21 end
22 // at this point, all clauses are  $True$ 
23 return  $True$ ;

```

Algorithm 2: Procedure to evaluate a SAT instance in CNF for a given assignment on the input variables.

With this in mind, we can calculate the probability ρ that a random 3-CNF is satisfiable. This is done the following way: for each value of N and $\alpha = M/N$, we generate at least 10^5 random 3-CNF instances and check all possible solutions. The satisfiability probability is defined then as the number of instances that have a solution divided by the total number of generated instances. Its dependence on the parameter α for different values of N is shown in figure 1 below.

In the case a 3-CNF has few clauses, there are few constraints on the variables. This results in a high probability of existing a solution for the problem. On the other hand, in the case that there are many clauses, there are many constraints on the variables. This makes the problem harder and reduces its chances of being satisfiable. This behaviour is clearly seen in figure 1: for $\alpha < 3$, the satisfiability probability is approximately 1 and it decreases with increasing α , regardless of the value of N .

For small values of α , the random instances are easily solvable, independently of the number of variables N . As α increases, the probability of a random 3-CNF being satisfiable

Figure 1 – Probability ρ that a random 3-SAT in conjunctive normal form is satisfiable. For small values of $\alpha = M/N$, there is a high probability that a random SAT instance has a solution. The higher the clauses-to-variables ratio, the smaller the probability of existing a solution.

Source: By the author.

decreases more sharply with increasing N . There is a special value of α , which is denoted by α_c and is determined by the crossing point of all curves for large values of N , that signals the existence of a threshold or critical phenomenon.^{23, 24, 36} In figure 1, the crossing point of the curves for different N occurs at $\alpha_c \approx 4$. We will offer a thorough analysis of this parameter in section 3.2.2. In the limit $N \rightarrow \infty$, the critical point α_c separates two regimes: instances with $\alpha < \alpha_c$ are satisfiable with probability 1, whereas instances with $\alpha > \alpha_c$ are unsatisfiable with probability 1. For an analysis of the dependence of α_c on the number k of variables in a clause, we refer the reader to Kirkpatrick and Selman²⁴, as in this dissertation we will focus on $k = 3$ only.

For the satisfiable instances, we count the number of assignments that satisfy the random formula and divide by the total number of possible solutions, 2^N . This way we can analyse the fraction of the solution space that evaluates to *True*. Figure 2 below shows this number as function of α for different values of N . Each point is the average over at least 10^5 instances and the error bars are the standard deviations divided by the square root of the number of instances, as usual. We note that for $N = 25$ we used about 10^8 instances in order to obtain reliable statistics.

We can see that, for small values of α , there is a considerable fraction of the solution space that satisfies the given random formula. This fraction decreases as α increases. Again, we see that the higher the number of constraints on the variables, the harder the problem becomes. As expected, the number of solutions tends to zero as α grows.

It is interesting to note that the fraction of the solution space that satisfies a random CNF instance decreases exponentially with increasing N , even for small α . This means that the number of solutions of satisfiable random instances increases much slower

Figure 2 – Fraction of the solution space that satisfies a random 3-CNF problem. This fraction is calculated by dividing the number of assignments that satisfy a formula by the total number of possible assignments, 2^N . For small values of α , a considerable fraction of the solution space satisfies the formula. This fraction decreases sharply with increasing values of α . For a fixed α , an increase in the number of variables decreases significantly the number of correct assignments. Each point represents an average over at least 10^5 instances. We note that for $N = 25$ we used about 10^8 instances in order to obtain a reliable statistics. The error bars are smaller than the sizes of the symbols.

Source: By the author.

than the number of possible assignments, 2^N . In addition, figure 2 shows no indication of the threshold phenomenon that occurs at $\alpha_c \approx 4$.

However, the threshold phenomenon seems to leave a mark on the distribution of the number of solutions of satisfiable random CNF instances, as illustrated in figure 3 below, which shows the standard deviation of the number of solutions divided by the mean number of solutions. It is interesting that this ratio increases with increasing N , regardless of the value of α , which contrasts with the usual self-averaging observable quantities of thermodynamics for which that ratio vanishes with $N^{-1/2}$ in the thermodynamic limit.³⁷ The results show that, for the range of N considered, the standard deviations are of the same order of magnitude as the means. The conspicuous increase of the standard deviation in the region around α_c is the mark of the threshold phenomenon. In fact, we expect that the height of the peak will diverge at α_c for $N \rightarrow \infty$. This means that random CNF instances at the critical region may exhibit very few and very large number of solutions.

Finally, we note that the *exhaustive search*, the approach used in this chapter so far, is a simple brute force method: we visited every possible assignment in the solution space and checked whether the formula evaluates to *True* or *False*. While this method guarantees an answer, it is only feasible for small values of N , as the cardinality of the set of assignments 2^N grows exponentially with increasing N . Even with modern computers, this strategy is viable for $N < 30$ only. Figure 4 below shows the time it takes to explore the entire solution space and evaluate the formula.

Figure 3 – Ratio between the standard deviation and the mean number of solutions of satisfiable random 3-CNF problems. The peak in the region around $\alpha_c \approx 4$ is the mark of the threshold phenomenon and we expect that its height will diverge in the limit $N \rightarrow \infty$.

Source: By the author.

For $N = 5$, the exhaustive search solves the problem in about $65ns$. Doubling the number of variables, it requires approximately $2\mu s$. For $N = 20$, it takes $\approx 2ms$, and it is still under a second for $N = 25$. For small values of N , this method is fast enough since it takes about 2^{N+1} nanoseconds to explore the entire solution space for N variables. Extrapolating these figures, we see that the exhaustive search would take about 35 minutes to solve the problem for $N = 40$, 26 days for $N = 50$ and 73 years (almost the orbital period of Uranus) for $N = 60$. For $N = 88$, the exhaustive search would require 1.4 times the age of the Universe to solve the problem. It is clear then that even problems of modest size, say $N = 100$, cannot be tackled using this method. In the next chapter we will study a very efficient algorithm that can tackle satisfiable problems of that size, but that never halts in the case of unsatisfiable problems.

Figure 4 – Time needed to explore the entire solution space for N variables and check the resulting value of the formula. Code was executed in a modern laptop with an Intel® Core™ i5-4210H CPU running at $3.5GHz$. Both panels show the same data using a linear plot in panel (a) and a semi-log plot in panel (b). Each point represents an average over at least 10^5 and at most 10^7 instances. Note that the horizontal scales varies from 5 to 25, while the vertical scale spans about 6 orders of magnitude: from around $100ns$ to $0.1s$. The exhaustive search is viable for small solution spaces only. For $N = 50$ it would take 26 days to explore the whole solution space and for $N = 60$, 73 years.

Source: By the author.

3 THE WalkSAT ALGORITHM

In contrast to exact algorithms, heuristic algorithms are designed to find solutions to optimization problems without any guarantee that those solutions are the optimal ones or that they will eventually find the optimal solution if allowed to run forever. In addition, the issue of finding all the solutions to a given problem (if there are more than one) is completely outside the scope of the heuristic algorithms. Thus, this kind of algorithm sacrifices accuracy and completeness for speed and are commonly used to produce near-optimal solutions to NP problems within an acceptable running time.¹

A local search heuristic is a general and flexible strategy to explore the solution space of an optimization problem in a sequential order. Starting from a given assignment, it moves to a neighbor assignment iteratively. Coupling this strategy with a metric to describe the quality of an assignment and a ‘memory’ of the last position, local search heuristics are likely to generate assignments of higher qualities as the number of iterations increases. It is important to note, however, that there is no guarantee of finding a global optimum, as the local search heuristic may be trapped in local optima.⁴

The definition of the neighbors of a given assignment and the rules to choose the next assignment are key factors for the performance of a local search. In the context of the Satisfiability Problem, we say that the assignment S' is a neighbor of the assignment S if they differ by one variable only. For example, $S = (True, True, True)$ and $S' = (True, True, False)$ are neighbors, but $S = (True, True, True)$ and $S' = (True, False, False)$ are not. Of course, this is the usual neighborhood of binary strings of length N when they are thought of as points of an N -dimensional space.

This chapter focuses on the *WalkSAT* (or WSAT) algorithm, a very simple and surprisingly efficient local search heuristic for the Satisfiability Problem.^{25, 26}

3.1 Outline of Algorithm

The WalkSAT algorithm uses a stochastic rule to decide whether to accept or not a new assignment. Thus, this algorithm can be seen as a stochastic local search. In particular, WalkSAT uses a random walk based strategy (hence its name) to explore the space of assignments. The search proceeds as follows. We start with a randomly generated assignment S . If this assignment satisfies the given Boolean formula F , then we return it and the search is halted. Otherwise, we select uniformly at random one of the unsatisfied clauses in the assignment. Let us denote C the selected clause. Then with probability P we randomly select a variable x_l in C and flip it (that is, if $x_l = True$ we set $x_l = False$ and if $x_l = False$ we set $x_l = True$) and with probability $1 - P$, we select and flip the

variable x_l that minimizes the number of clauses that are satisfied in the assignment S but would become unsatisfied if the variable is flipped. Of course, in the case of ties, we select one of the tied variables at random and flip it. In the case the new assignment S' satisfies F , we return it and stop the search. Otherwise, we pick a random unsatisfied clause in S' and repeat the process.^{10, 25, 26, 38, 39} Note that the assignments S and S' are neighbors since they differ only by the variable x_l . This procedure is illustrated below in algorithm 3.

The parameter P is the *noise parameter* and the main idea behind it is to try to prevent the algorithm from getting trapped in sub-optimal regions of the space of assignments. The parameter P is also known as the greed-avoidance parameter, as it avoids the greedy choice of flipping the variable that minimizes the number of unsatisfied clauses.

Because the randomly selected variable x_l , which is chosen with probability P , must belong to an unsatisfied clause, flipping it will satisfy the clause. However, this variable might appear in another clause that was *True* only because of the previous value of x_l , and this other clause is now unsatisfied. This means that the algorithm may never find a solution to the Boolean formula F , even if a solution is known to exist. In this case, the WalkSAT algorithm will enter an infinite loop as it does in the case that F is unsatisfiable. To avoid these scenarios, it is necessary to define a maximum number of steps T_{max} for the WalkSAT. If the algorithm fails to find a solution to F within the prescribed number of iterations, the procedure returns the message ‘No assignment found’.

```

input  :  $F, P, T_{max}$ 
output: A solution to  $F$ , or "No assignment found"

1 Assignment  $S \leftarrow$  random assignment;
2 for  $t \leftarrow 1$  to  $T_{max}$  do
3   if  $F(S)$  is True then
4     return  $S$ ;
5   end
6    $C \leftarrow$  random unsatisfied clause from  $S$ ;
7    $u \leftarrow \text{UniformRandomNumber}(0, 1)$ ;
8   if  $u < P$  then
9      $x_l \leftarrow$  random variable in  $C$ ;
10  else
11     $x_l \leftarrow$  best variable to flip in  $C$ ;
12  end
13   $S[l] \leftarrow \neg x_l$ ;
14 end
15 return No assignment found;
```

Algorithm 3: The WalkSAT algorithm returns a solution to the Boolean formula F or a message ‘No assignment found’ if the number of iterations exceeds the parameter T_{max} . The noise parameter P allows the algorithm to perform non-optimal moves to avoid getting stuck in sub-optimal regions of the space of assignments.

Clearly, the crucial ingredient of the WalkSAT is the procedure to find the best variable to flip. It has to incorporate the notion that when trying to fix a particular unsatisfied clause, we should avoid to turn too many satisfied clauses into unsatisfied ones. The procedure we adopted is shown in algorithm 4.

```

input  :  $F, S, C$ 
output : The label  $l$  of the literal to flip
1   $fc \leftarrow \infty$ ;
2  for  $i$  such  $x_i$  in  $C$  do
3       $S' \leftarrow S$ ;
4      flip  $x_i$  in  $S'$ ;
5       $counter \leftarrow$  number of clauses that were True with  $S$  but are False with  $S'$ ;
6      if  $counter < fc$  then
7           $l \leftarrow i$ ;
8           $fc \leftarrow counter$ ;
9      end
10 end
11 return  $l$ ;

```

Algorithm 4: Procedure to find the best variable x_l to flip in an unsatisfied clause C of a Boolean formula F . It identifies the label l of the variable that minimizes the number of clauses that are satisfied with S but would become unsatisfied if the variable is flipped.

3.1.1 Implementation Tips

Although the implementation of the WalkSAT algorithm is rather straightforward, we offer now some optimizations we adopted in order to speed up the running time of the algorithm.^{10, 40}

An useful artifice is to map all the clauses each literal appears. More pointedly, for every literal x_l in the formula F , we add an entry in this map with all the clauses it appears. For example, in the formula $F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$ this map would have four entries: $x_1 \rightarrow (C_0, C_1, C_2)$, $x_2 \rightarrow (C_0, C_1)$, $x_3 \rightarrow (C_0, C_2)$, and $x_4 \rightarrow (C_1, C_2)$, where C_i is the index of each clause in the formula. This map reduces the cost to check the number of clauses that are broken when we flip one variable, as we now need to check only a subset of the clauses of the formula, namely, those clauses where the flipped variable appears.

Another useful scheme is to cache all the unsatisfied clauses in an array. Before entering the main loop of the algorithm, we parse the formula with the randomly generated assignment and store the indices of the unsatisfied clauses in an array. This procedure has two advantages. First, checking the length of an array is faster than evaluating a formula (that is, if the array is empty, the assignment S satisfies the formula: $F(S) = \text{True}$).

Second, it is easy to select randomly an unsatisfied clause (just pick a random element of this array). The downside is to keep this cache up to date since, at the end of the loop, we need to check if we need to insert or remove entries. As we know exactly which clauses to check, this step is not too computationally expensive.

3.2 Analysis of the search times

In order to study the efficiency of the WalkSAT to solve random Boolean formulas, we focus on the number of steps t^* it requires to find a solution. Henceforth we will refer to t^* as the search time of the algorithm. We discard the runs for which $t^* > T_{max}$. As already pointed out, these events may indicate that the Boolean formula is unsatisfiable or simply that the WalkSAT failed to find a solution within the prescribed running time.

We begin our analysis of the WalkSAT by emphasizing the stochastic nature of this algorithm. In figure 5, we show the number of unsatisfied clauses as function of the number of steps t for two different runs on the same randomly generated 3-SAT problem with $N = 100$ variables and $M = 420$ clauses, which happens to be satisfiable. Of course, the reason we know the Boolean formula is solvable is because WalkSAT found a solution to it, since it is impossible to use the exhaustive enumeration of the assignments to verify the satisfiability of a formula with $N = 100$ variables.

Figure 5 – Number of unsatisfied clauses as a function of the number of steps t for two different runs of the WalkSAT on the same solvable Boolean formula with $N = 100$ variables and $M = 420$ clauses. The noise parameter is $P = 0.5$. Notice the enormous discrepancy between the search times t^* for the runs illustrated in the panels (a) and (b).

Source: By the author.

The first point to note in figure 5 is the enormous variability of the search time t^* of the WalkSAT, which is the value of t in which the number of unsatisfied clauses reaches zero. More pointedly, we have $t^* = 104$ for the run illustrated in panel (a) and $t^* = 2444$

for the run in panel (b). In both runs the algorithm keeps the number of unsatisfied clauses below 10 after about 10^2 iterations. Most of the search time is spent trying to satisfy a few clauses in the run shown in panel (b). In addition, both runs start with the majority of the clauses already satisfied. This is a consequence of initializing the algorithm with a randomly generated assignment, which guarantees that only about $1/8$ of the clauses are unsatisfied at the outset, on average.⁴ We present the proof of this result in appendix B.

In the following sections we study how the search time t^* of the WalkSAT depends on the noise parameter P and on the search time upper bound T_{max} , as well as on the parameters N and M of the random Boolean formulas. In addition, we show that the probability for the WalkSAT to succeed to find a solution to random Boolean formulas (that is, the probability that $t^* \leq T_{max}$) exhibits a threshold phenomenon at some value of the ratio $\alpha = M/N$, similarly to the observed for the exhaustive search in the previous chapter.

3.2.1 Probability distribution of the search times

In this section we present the conditional probability distributions of the search times t^* for the WalkSAT. For a given random 3-Conjunctive Normal Form (3-CNF) formula we run the WalkSAT algorithm until it finds a solution at t^* or the upper bound T_{max} is exceeded. We repeat this procedure 10^5 to 10^7 times, generating a random formula for each run, that is, we run the WalkSAT only once for each formula. The reason we choose this strategy, rather than running the WalkSAT several times for the same formula as illustrated in figure 5, is to avoid the risk of running many searches in a same unsatisfiable formula. The next chapter focus on running the WalkSAT multiple times on the same random formula.

We note that since we record t^* only for the runs such that $t^* \leq T_{max}$, the probability distributions of the search times are conditional on $t^* \leq T_{max}$. For simplicity, in what follows we will often omit the adjective conditional when referring to the conditional probability distribution of the search times. In addition, since $t^* = 1, 2, \dots, T_{max}$ is a discrete variable, a histogram with a bin of size 1 is a faithful representation of the probability distribution $P(t^*)$.

Before embarking on the analysis of the search time probability distributions, a word is in order about the choice of the parameters used in our study. We consider four values of the noise parameter, namely, $P = 0.2$, $P = 0.4$, $P = 0.5$ and $P = 0.8$. The optimum value of the noise parameter is around $P = 0.5$, hence our choice of $P = 0.4$ and $P = 0.5$.³⁸ The extreme values $P = 0.2$ and $P = 0.8$ are instructive since they illustrate the cases where the WalkSAT makes too many or too few greedy choices. In addition, we consider three values of the ratio between the number of clauses M and the number of variables N , as follows, $\alpha = 3$, $\alpha = 4.1$ and $\alpha = 5$. The reason for this choice is that

we expect a phase transition at $\alpha \approx 4.1$ separating the regime where the algorithm finds a solution and the regime where it fails to find a solution (see subsection 3.2.2). Hence WalkSAT is very likely to succeed for $\alpha = 3$ and very likely to fail for $\alpha = 5$. The failure to find a solution may be because the formula is unsatisfiable or because the WalkSAT cannot find the solution even if a solution exists. Finally, we set $T_{max} = 10^6$ unless stated otherwise.

Figure 6 shows the probability distribution of the search time t^* for $\alpha = 3$ so that the WalkSAT is very likely to find a solution for a random 3-CNF formula. These unimodal distributions are very asymmetric exhibiting a fat tail for large t^* (note the logarithmic scale of the x-axis). As N increases, the left tail moves to the right as expected, since the minimum time t_{min}^* the WalkSAT needs to solve a formula increases linearly with N , but the right tail seems to be very little affected by the change in the number of variables. As a result, $P(t^*)$ becomes more concentrated around its maximum as N increases. We discuss the relationship of t_{min}^* with N in appendix C.

Figure 6 – Probability that the WalkSAT solves a random 3-CNF formula at time t^* for $\alpha = 3$. The values of the noise parameter P , as well as the number of variables N , are as indicated. The search time upper bound is $T_{max} = 10^6$.

Source: By the author.

In addition, figure 6 reveals the nontrivial effect of the noise parameter P on the performance of the WalkSAT. Interestingly, making too many greedy choices (that is, attempting to minimize the number of unsatisfied clauses), as done for $P = 0.2$, increases the odds of solving the problem quickly, regardless of the value of N . The trade-off is that the WalkSAT takes a very long time to solve those problems that were not solved quickly. Hence, for $P = 0.2$ the probability is the highest for both small and large t^* , but not for intermediate values. As expected, making too few greedy choices, as done for $P = 0.8$, makes it very unlikely to find the solution for short times. It is necessary to emphasize that the (discrete) distributions shown in this figure have area equal to 1 and so we can guarantee that, for instance, there is a range of t^* for which the probability for $P = 0.8$ is larger than the probability for $P = 0.2$.

We can obtain more quantitative information on the influence of the noise parameter P , and the number of variables N , on the performance of the WalkSAT by looking at the (conditional) mean time $\langle t^* \rangle$ to find a solution of a random 3-CNF formula, as shown in figure 7. Here the notation $\langle \dots \rangle$ stands for the average over the random Boolean formulas.

Figure 7 – Mean time $\langle t^* \rangle$ for the WalkSAT to solve a random 3-CNF formula as function of the number of variables N , for $\alpha = 3$ and for the noise parameter $P = 0.2, 0.4, 0.5$, and 0.8 as indicated. The search time upper bound is $T_{max} = 10^6$.

Source: By the author.

The results for $P = 0.4$ and $P = 0.5$ are practically indistinguishable in the scale of the figure. It is clear from these results that the optimum value of the noise parameter is around $P = 0.5$. This figure indicates that the mean search time does not increase exponentially with increasing N , this being the reason we can use the WalkSAT to tackle large 3-CNF formulas. (We recall that the mean search time of the exhaustive search scales with 2^N .) The apparent tendency of $\langle t^* \rangle$ to converge to a finite value for large N is consistent with our observation that the right tail of the distributions shown in figure 6 are practically insensitive to N . We note that this tendency is unlikely to be an artifice of the search time upper bound since $\langle t^* \rangle \ll T_{max} = 10^6$ for the range of N shown in figure 7.

Figure 8 summarizes the effect of the ratio clauses-to-variables $\alpha = M/N$ on the probability distribution of t^* .

Figure 8 – Probability that the WalkSAT solves a random 3-CNF formula at time t^* for $N = 50$ and $\alpha = 3, 4.1$, and 5 . The values of the noise parameter P are as indicated. The search time upper bound is $T_{max} = 10^6$.

Source: By the author.

As pointed out before, for $\alpha = 3$ the WalkSAT is almost certain to find a solution to the Boolean formula, whereas for $\alpha = 5$ it is almost certain to fail. The choice $\alpha = 4.1$ is close to the transition between these two regimes. The variation of α does not produce any qualitative change on the probability distributions of the search times. The most significant effect is the fattening of the right tails of those distributions for increasing α . The WalkSAT takes much longer to find a solution of a formula for $\alpha = 5$ than for $\alpha = 3$. The methodological difficulty here is the increasing number of runs with t^* close to T_{max} observed for $\alpha = 5$, indicating that the upper bound may be affecting the probability distribution in that case.

The mean search time $\langle t^* \rangle$ shown in figure 9 for $\alpha = 4.1$ and $\alpha = 5$ illustrates more clearly the points discussed above. In particular, these results are qualitatively very similar to those for $\alpha = 3$ shown in figure 7. Note the difference on the range of N in these figures. The optimum value of the noise parameter is close to $P = 0.5$ regardless of the value of α . It is interesting that for small N , the WalkSAT with $P = 0.8$ performs better, on average, than with $P = 0.2$ for the three values of α considered. However, we think this result is an artifact of the upper bound T_{max} , as we will see next. Our conjecture is that, considering $\langle t^* \rangle$ as the performance measure, the WalkSAT with $P = 0.8$ performs better than the WalkSAT with $P = 0.2$ for all N if we let $T_{max} \rightarrow \infty$.

Accordingly, figure 10 illustrates the effects of the choice of T_{max} on the mean search time $\langle t^* \rangle$. Since $\langle t^* \rangle \leq T_{max}$, small values of this upper bound distort completely the estimate of the mean search time. Fortunately, for $P = 0.4$ and $P = 0.5$ the choices $T_{max} = 10^5$ and $T_{max} = 10^6$ give essentially the same results, so in the next section we will set $P = 0.5$ and $T_{max} = 10^6$.

An interesting feature shown in figure 10 is the dependence of the mean search time

Figure 9 – Mean time $\langle t^* \rangle$ for the WalkSAT solve a random 3-CNF formula as function of the number of variables N for $\alpha = 4.1$ and $\alpha = 5$. The noise parameter is $P = 0.2, 0.4, 0.5$, and 0.8 as indicated. The search time upper bound is $T_{max} = 10^6$.

Source: By the author.

for $P = 0.2$ and $P = 0.8$ on T_{max} . It is clear that increasing T_{max} increases the range of N for which the choice $P = 0.8$ is superior to the choice $P = 0.2$. We think the reason is that for $P = 0.2$ the algorithm can frequently reach a situation where it enters a loop because it makes many greedy choices and requires a very long time until the right sequence of random flips occurs to leave the cycle, because P is small. In fact, figures 6 and 8 reveal that the distributions for $P = 0.2$ exhibit the fattest right tails. This event will produce a very large search time, say, $t^* = 10^{10}$, which would strongly affect the estimate of $\langle t^* \rangle$ if allowed to enter the statistics. This is the reason that the increase of T_{max} affects so strongly the mean search time for $P = 0.2$. We think a similar phenomenon happens for $P = 0.8$ as well, since flipping random bits of an unsatisfied clause is not helpful if finding the solution requires flipping a bit of a satisfied clause. But it is clear that the effect of increasing T_{max} is stronger for $P = 0.2$ than for $P = 0.8$, since the tendency revealed in figure 10 is that the mean search time for $P = 0.2$ becomes larger than the mean search time for $P = 0.8$ for all N .

A simple way to circumvent these difficulties is to consider the median search time $\tilde{t}^*$ instead of the mean search time $\langle t^* \rangle$. In fact, figure 11 shows that the median is much less sensitive to the presence of outliers than the mean. By outliers we mean values of t^* that lies very far away from $\langle t^* \rangle$. This figure shows that the estimate of $\tilde{t}^*$ for $P = 0.4$ and $P = 0.5$ is already stable for $T_{max} = 10^4$. In addition, these results show that the median for $P = 0.8$ is always greater than the median for $P = 0.2$, which confirms that the outliers are responsible for the abnormal increase of the mean search times with T_{max} observed in figure 10.

However, since the choice $T_{max} = 10^6$ seems to have no effect on the mean search

Figure 10 – Mean time $\langle t^* \rangle$ for the WalkSAT solve a random 3-CNF formula as function of the number of variables N for $\alpha = 4.1$ and search time upper bound $T_{max} = 10^3, 10^4, 10^5$, and 10^6 . The noise parameter is $P = 0.2, 0.4, 0.5$ and 0.8 as indicated.

Source: By the author.

time for the optimal value of the noise parameter, namely $P = 0.5$, in the remaining of this dissertation we will keep $\langle t^* \rangle$ as the characteristic measure of the distribution of the search times.

Figure 11 – Median $\tilde{t}^*$ of the probability distribution of the search times of the WalkSAT as function of the number of variables N for $\alpha = 4.1$ and search time upper bound $T_{max} = 10^3, 10^4, 10^5$, and 10^6 . The noise parameter is $P = 0.2, 0.4, 0.5$, and 0.8 as indicated.

Source: By the author.

3.2.2 Phase transition

The study of the previous section shows that for the noise parameter $P = 0.5$ the choice of the search time upper bound $T_{max} = 10^6$ seems to affect very little the distribution of the search times. We interpret this fact, somewhat optimistically, by saying that if the WalkSAT failed to find a solution within that time bound it is because the random 3-CNF formula is unsatisfiable. Hence we can estimate the probability that a random formula is satisfiable as the fraction ρ of the formulas for which the WalkSAT found a solution. Typically, we generate 10^6 to 10^7 random formulas to estimate ρ .

Figure 12 shows the dependence of ρ on the ratio of clauses-to-variables α as the number of variables N increases. The fact that ρ increases with increasing N for $\alpha < \alpha_c$ and decreases with increasing N for $\alpha > \alpha_c$ suggests the occurrence of a threshold phenomenon or phase transition at the critical point $\alpha = \alpha_c$ in the limit $N \rightarrow \infty$. In this limit we have $\rho \rightarrow 1$ for $\alpha < \alpha_c$ and $\rho \rightarrow 0$ for $\alpha > \alpha_c$ so that α_c determines the

regimes where the random 3-CNF formulas are satisfiable or unsatisfiable. In that sense, ρ can be interpreted as the order parameter⁴¹ of the satisfiability problem. Accordingly, we can estimate α_c by selecting the value of α at which the curves for the largest values of N (namely $N = 60$ and $N = 80$) intersect in figure 12. Keeping the first and second significant figures only we obtain $\alpha_c \approx 4.1$. Next we will see how to evaluate visually the quality of this estimate.

Figure 12 – Fraction of random 3-CNF formulas for which the WalkSAT found a solution within the search time upper bound $T_{max} = 10^6$ as function of ratio of clauses-to-variables α . The number of variables is $N = 20, 30, 40, 50, 60$, and 80 as indicated. The noise parameter is $P = 0.5$.

Source: By the author.

We focus now on the characterization of the sharpness of the threshold, that is, the range of α about α_c where the threshold features persist. The threshold features persist when the order parameter ρ is neither too close to 1 nor to 0. In other words, we want to understand how the steepness of the transition between the two regimes varies with N . To achieve that we will assume that the critical region shrinks to zero like $N^{-1/\nu}$ as $N \rightarrow \infty$, in this case, we will assume that the order parameter is described by the expression²⁷

$$\rho = f \left[N^{1/\nu} (\alpha - \alpha_c) \right] \quad (3.1)$$

in the critical region. Here the scaling function $f(y)$ is a continuous function such that $f(y) \rightarrow 1$ for $y \rightarrow -\infty$ and $f(y) \rightarrow 0$ for $y \rightarrow \infty$. We note that, according to the finite size scaling assumption, the order parameter should not depend on N at $\alpha = \alpha_c$. Hence our prescription to estimate α_c as the intersection of the curves of ρ vs. α for different N . It is important to stress that the scaling assumption holds for large N only.

The way to verify if our order parameter ρ obeys the finite size scaling assumption 3.1 is through the data collapse method.²⁷ We vary the exponent ν with $\alpha_c = 4.1$ fixed until the curves ρ vs. y , with $y = N^{1/\nu} (\alpha - \alpha_c)$, for different N collapse into a single curve, namely, the scaling function $f(y)$. Figure 13 shows the final data collapse obtained with the choice $\nu = 1.5$.

Figure 13 – Order parameter ρ against the scaled distance to the critical point $y = N^{1/\nu}(\alpha - \alpha_c)$ for the number of variables $N = 20, 30, 40, 50, 60$, and 80 as indicated. The noise parameter is $P = 0.5$. The critical point is $\alpha_c = 4.1$ and the critical exponent is $\nu = 1.5$.

Source: By the author.

It is interesting that the values of the critical point α_c and of the exponent ν agree with the findings of a seminal study that used a highly optimized implementation⁴² of the Davis-Putnam procedure⁴³ to find solutions of random 3-CNF formulas.²⁴ This procedure performs a backtrack search through the space of possible truth assignments and is the fastest known complete procedure for satisfiability testing. This agreement lends credence to our optimistic view that figures 12 and 13 describe genuine properties of the satisfiability of random 3-CNF formulas, rather than peculiarities of the algorithm we used to solve those formulas.

In that sense, the threshold phenomenon described here should not be surprising, since properties of randomly generated combinatorial structures often exhibit sharp threshold phenomenon.^{44, 45} For instance, if we consider a random graph with N vertices and M edges, and make N large, but with the ratio $\alpha = M/N$ kept fixed, then for $\alpha \geq \alpha_c = 1/2$ the resulting graph will exhibit a single giant component whose size is proportional to N . Whereas for $\alpha < 1/2$ there will be many components whose maximum size is proportional to $\ln N$. In addition, in this case the exponent that determines the sharpness of the threshold is $\nu = 3$.⁴⁶ This is an example of the classic percolation transition.⁴⁷ We have intentionally used the same notation in both the random graph problem and the satisfiability problem to stress the similarity between the two problems, which are examples of randomly generated combinatorial structures.

An important consequence of the phase transition from the perspective of computer science is that the algorithms take much longer to find the solution of (satisfiable) random formulas if the clauses-to-variables ratio is in the critical region, so that those formulas could have some value to evaluate the efficiency of new algorithms for the SAT. This conclusion was derived originally using the Davis-Putnam algorithm to solve the random

formulas.³⁴ Figure 14, which shows the dependence of the median search time $\tilde{t}^*$ on the clauses-to-variables ratio α , confirms that this conclusion holds true for the WalkSAT as well. This figure reveals that it is not necessarily the case that generating larger formulas provides harder formulas. For instance, on average it is harder to solve a formula with $N = 50$ and $M = 225$ (that gives $\alpha = 4.5$) than a formula with $N = 80$ and $M = 480$ (that gives $\alpha = 6$).

Figure 14 – Median search time $\tilde{t}^*$ as function of ratio of clauses-to-variables α . The number of variables is $N = 20, 30, 40, 50, 60$, and 80 as indicated. The noise parameter is $P = 0.5$ and the search time upper bound is $T_{max} = 10^6$.

Source: By the author.

Finally, to quantify the increase of the median search time with the problem size, we present in figure 15 log-log plots of $\tilde{t}^*$ against N for different values of α . The data is well described by the power law $\tilde{t}^* \sim /N^{\tilde{b}}$, where the power $\tilde{b}$ depends on the value of α and is maximum for $\alpha \approx 4.1$. The difficulty to obtain results for $\alpha = 5$ is that there are very few satisfiable formulas for large N , so in this case we have restricted the analysis to $N \leq 80$ only in order to have reliable statistics. For $\alpha = 4.1$, the median time is approaching the search time upper bound T_{max} as N grows, so in this case we restricted the analysis to $N \leq 400$.

Figure 15 – Median search time $\tilde{t}^*$ as function of the number of logical variables N for $\alpha = 3, 4.1$ and 5 as indicated. The noise parameter is $P = 0.5$ and the search time upper bound is $T_{max} = 10^6$. The solid lines are the fittings $\tilde{t}^* = \tilde{a}/N^{\tilde{b}}$. For $\alpha = 3$ we find $\tilde{b} \approx 1.09$; for $\alpha = 4.1$ we find $\tilde{b} \approx 2.53$; and for $\alpha = 5$ we find $\tilde{b} \approx 1.98$.

Source: By the author.

4 COLLECTIVE SEARCH

The motivation of this chapter is given by figure 5 that illustrates the changes on the number of unsatisfied clauses for two runs of the WalkSAT on a same satisfiable formula. The observation that the search times t^* can differ by one or more orders of magnitude for different runs, leads us to wonder whether it would be advantageous to run L WalkSATs in parallel and then stop the search when one of them finds the solution. This is the problem we address in this chapter.

We note that the collective search performed by the L WalkSATs is not cooperative since they do not exchange information, except when the solution is found and a halting signal is broadcasted to all agents. We recall that if we perform L blind searches where each logical variable flips with probability $1/2$ then the mean search time is⁴⁸

$$\langle t^* \rangle = \frac{1}{1 - (1 - S/2^N)^L} \approx \frac{2^N}{S} \frac{1}{L}, \quad (4.1)$$

where the approximation is the first order expansion of the denominator and is valid for $LS/2^N \ll 1$, and S is the number of solutions of the Boolean formula. We note that figure 2 shows that $S/2^N \ll 1$ for large N regardless of the value of α . In the appendix D we offer a derivation of this interesting result. We expect that the mean search time of the L independent WalkSATs would exhibit a behavior similar to the blind search. A truly cooperative collective search should result in a superlinear decrease of the mean search time with the number of agents.⁴⁹

Figure 16 shows the probability $P(t^*)$ that a group of L independent WalkSATs finds the solution at time t^* . For $L = 1$, the simulation procedure is the same as that described in the previous chapter. In particular, each group of L agents solves 10^5 different random 3-CNF formulas and the frequencies of the search times t^* are our estimate of the probability distribution $P(t^*)$.

The results indicate a clear improvement of the performance with respect to the single agent ($L = 1$) case. In fact, the widths of the distributions and consequently their right tails are reduced significantly by increasing L , in accord with our initial expectations. One way to smooth out the noise observed in the direct estimate of $P(t^*)$ is to consider instead the cumulative distribution $\Pi(t^*) = P(t \leq t^*)$ as shown in figure 17. We recall that the median $\tilde{t}^*$ is defined as $\Pi(\tilde{t}^*) = 1/2$ so we can clearly see from this figure that $\tilde{t}^*$ decreases with increasing L .

Of course, this qualitative analysis is not enough to attest the advantage of the (independent) collective search. Since there is a cost to run the WalkSATs, which we assume grows linearly with L , the search time t^* should decrease faster than L for the

Figure 16 – Probability that L independent WalkSATs running in parallel solve a random 3-CNF formula at time t^* for $N = 50$, $P = 0.5$ and $\alpha = 3, 4.1$ and 5 as indicated. The search time upper bound is $T_{max} = 10^6$.

Source: By the author.

Figure 17 – Cumulative distribution that L independent WalkSATs running in parallel solve a random 3-CNF formula before or at time t^* for $N = 50$, $P = 0.5$ and $\alpha = 3, 4.1$ and 5 as indicated. The search time upper bound is $T_{max} = 10^6$.

Source: By the author.

collective search be the winner strategy. In fact, equation (4.1) shows that for the blind search, there is no net gain of using a collective search ($L > 1$) in comparison with a single search ($L = 1$) in the realistic case $LS \ll 2^N$.

Accordingly, to check whether there is an advantage of running L independent WalkSATs on the same formula, we show in figure 18 log-log plots of the mean $\langle t^* \rangle$ and the median $\tilde{t}^*$ search times against L .

The results for the median $\tilde{t}^*$ indicate that for $\alpha = 4.1$ and $\alpha = 5$ this quantity decreases as $L^{-0.29}$ whereas for $\alpha = 3$ it decreases as $L^{-0.2}$. The power-law exponents for the mean $\langle t^* \rangle$ are slightly different for $\alpha = 4.1$ and $\alpha = 5$ and indicate that the mean decays faster than the median as the group size increases. However, as pointed out before, the mean search time is very sensitive to the choices of T_{max} and of the number of samples used to estimate the mean, especially near the threshold point $\alpha = \alpha_c$, so we conjecture that the power-law exponents of the median should govern the decay of the mean as well. This means that it is actually disadvantageous to use more than one WalkSAT to solve a

Figure 18 – Mean time $\langle t^* \rangle$ (panel a) and median time $\tilde{t}^*$ (panel b) for L independent WalkSATs running in parallel to solve a random 3-CNF formula as function of the number of agents L for $\alpha = 3, 4.1$ and 5 as indicated. The other parameters are $N = 50$, $P = 0.5$ and $T_{max} = 10^6$. The dashed lines are the fittings $\langle t^* \rangle = a/L^b$ and $\tilde{t}^* = \tilde{a}/L^{\tilde{b}}$ for large L . For $\alpha = 3$ we find $b = \tilde{b} \approx 0.2$; for $\alpha = 4.1$ we find $b \approx 0.41$ and $\tilde{b} \approx 0.29$; and for $\alpha = 5$ we find $b \approx 0.32$ and $\tilde{b} \approx 0.29$.

Source: By the author.

3-CNF formula. The reason for this is that, in contrast to the blind search, the WalkSAT has a greedy procedure, that is, the update of the variable that will minimize the number of unsatisfied clauses. In addition, since only variables within unsatisfied clauses are updated, the search space is reduced greatly. This is all to say that there is a great chance that two or more WalkSATs explore the same assignments resulting in duplication of work and the consequent inefficiency of the WalkSAT collective search. Of course, if there is no cost to run the WalkSATs in parallel, because, for instance, there are plenty of CPUs available, then the results shown in figure 18 yield the performance gain of the collective search.

To conclude this chapter, we offer an explanation for the finding that the collective search becomes more inefficient as α decreases or, since we are keeping N fixed, as M decreases. We note that the performance of the collective blind search, equation (4.1), is not affected by the choice of M . In fact, as pointed out before, since the WalkSAT updates only variables within unsatisfied clauses, the decrease of M increases the chances that different WalkSATs are updating the same variables and so that work is being duplicated.

5 CONCLUSIONS

A characteristic of the Boolean Satisfiability Problem (SAT) that makes it a particularly difficult problem to tackle using general purpose algorithms such as evolutionary,⁵⁰ cultural,⁵¹ and physics⁵² based algorithms is that there is no natural cost function to minimize. In fact, using the number of unsatisfied clauses as the cost function yields a poor performance because completely distinct assignments may have the same cost so that the cost landscape exhibits large plateaus where the search carried out by those algorithms is essentially a blind search.⁵³ Hence the need to develop algorithms tailored specifically for the SAT. This dissertation studies one of these algorithms, the WalkSAT.^{25, 26}

We note that according to the exponential time hypothesis, SAT, being an NP-Complete problem, cannot be solved in sub-exponential time in the worst case by any algorithm.¹² This means that there is at least one particular Boolean formula with N variables that would require a time $\mathcal{O}(2^{\delta N})$ for some $\delta > 0$ to be solved by any known algorithm.¹⁰ However, it is sometimes argued that this analysis is irrelevant in practice since it is based on a worst-case scenario and a more useful approach would be to consider the performance of the algorithms on “typical” or “average” Boolean formulas.³⁴

The simplest approach to generate “typical” Boolean formulas with fixed number of variables N and number of clauses M is to generate random formulas by choosing randomly the variables that appear in each clause and negating them with probability $1/2$. It turns out that these random objects have interesting properties on their own, namely properties that have nothing to do with the algorithms used to solve them. In particular, for 3-CNF formulas, which have exactly three distinct logical variables in each clause, and are the only problems studied in this dissertation, there is a phase transition at $\alpha = M/N \approx 4.1$ that separates the satisfiable formulas ($\alpha < 4.1$) from the unsatisfiable ones ($\alpha > 4.1$).^{23, 24} Although this is a pure property of the ensemble of random 3-CNF formulas, its numerical verification requires the use of efficient algorithms to solve those formulas. The papers^{23, 24} that discovered this phase transition used a highly optimized implementation⁴² of the Davis-Putnam procedure,⁴³ which performs a backtrack search through the space of possible truth assignments. In this dissertation we showed that the same results can be obtained using the much simpler local algorithm WalkSAT.

An important result regarding the use of random Boolean formulas to generate SAT instances is the possibility of producing hard formulas that may be used to test new algorithms for the SAT.³⁴ In fact, we observed that near the critical region the WalkSAT takes much longer to find the solution of satisfiable random formulas than away from that region. This observation was quantified by determining how the median search time $\tilde{t}^*$ scales with N for different values of the clauses-to-variables ratio α . We find that the

scaling is a power law $\tilde{t}^* \sim N^{\tilde{b}}$, where the power $\tilde{b}$ depends on the value of α and is maximum for $\alpha \approx 4.1$.

For stochastic algorithms, such as WalkSAT, a natural strategy to speed up the search is to execute several runs, say L , of the algorithm in parallel using different initial conditions and halt the collective search when one of the algorithms finds the solution. In principle, we should expect the search time to decrease linearly with L , as in the case of a blind collective search. However, in this dissertation we show that the gain in the case of the WalkSAT is sub-linear, that is, the median search time decreases as $L^{-\tilde{b}}$ with $\tilde{b} \approx 0.3$. We stress that this finding is for the case of L independent WalkSATs. An interesting possibility we may pursue in the future is to allow the WalkSATs to exchange information among them using, for instance, a blackboard⁵⁴ to post the assignments of the satisfied clauses that may be viewed as hints to the solution of the problem, as done in the distributed cooperative problem-solving approach to solve crypt-arithmetic puzzles.^{49, 55, 56}

REFERENCES

- 1 GAREY, M. R.; JOHNSON, D. R. *Computers and intractability: a guide to the theory of NP-completeness*. San Francisco: W.H. Freeman, 1979. ISBN 0716710455.
- 2 COOK, S. A. The complexity of theorem-proving procedures. *In: ACM SYMPOSIUM ON THEORY OF COMPUTING*, 3., 1971, New York. *Proceedings [...]*. New York: Association for Computing Machinery, 1971. p. 151–158. ISBN 9781450374644. DOI 10.1145/800157.805047.
- 3 LEVIN, L. A. Universal sequential search problems. *Problemy Peredachi Informatsii*, Russian Academy of Sciences, v. 9, n. 3, p. 115–116, 1973.
- 4 KLEINBERG, J.; TARDOS, É. *Algorithm design*. Boston: Pearson Education, 2005. ISBN 0321295358.
- 5 KARP, R. M. Reducibility among combinatorial problems. *In: MILLER, R. E.; THATCHER, J. W.; BOHLINGER, J. D. (ed.). Complexity of computer computations*. Boston: Springer, 1972. p. 85–103. DOI 10.1007/978-1-4684-2001-2_9.
- 6 DAVIS, M. *Computability and unsolvability*. New York: Dover Publications, 1985. ISBN 9780486614717.
- 7 TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, v. 42, n. 1, p. 230–265, 1936. DOI 10.1112/plms/s2-42.1.230.
- 8 GREENLAW, R.; HOOVER, H. J.; RUZZO, W. L. *Limits of parallel computation: P-completeness theory*. Oxford: Oxford University Press, 1995. ISBN 978-0195085914.
- 9 ALSUWAIYEL, M. H. *Algorithms: design techniques and analysis*. Singapore: World Scientific, 2016. ISBN 978-9814723640.
- 10 KNUTH, D. E. *The art of computer programming*, v. 4, fascicle 6: satisfiability. New York: Addison-Wesley Professional, 2015. ISBN 9780134397603.
- 11 CLAY MATHEMATICS INSTITUTE. *Millenium problems*. 2000. Available from: <https://www.claymath.org/millennium-problems>. Accessible at: 23 Jan. 2020.
- 12 IMPAGLIAZZO, R.; PATURI, R. Complexity of k-sat. *In: ANNUAL IEEE CONFERENCE ON COMPUTATIONAL COMPLEXITY (Formerly: Structure in Complexity Theory Conference)*, 14., 1999, Atlanta. *Proceedings [...]*. Atlanta: IEEE, 1999. p. 237–240. DOI 10.1109/CCC.1999.766282.
- 13 BIERE, A.; HEULE, M.; MAAREN, H. van; WALSH, T. *Handbook of satisfiability*. Amsterdam: IOS Press, 2009. (Frontiers in artificial intelligence and applications, v. 185). ISBN 1586039296.
- 14 VIZEL, Y.; WEISSENBACHER, G.; MALIK, S. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, v. 103, n. 11, p. 2021–2035, 2015. DOI 10.1109/JPROC.2015.2455034.

- 15 GUPTA, A.; GANAY, M. K.; WANG, C. Sat-based verification methods and applications in hardware verification. *In: BERNARDO, M.; CIMATTI, A. (ed.). Formal methods for hardware verification.* Berlin, Heidelberg: Springer, 2006. p. 108–143. ISBN 978-3-540-34305-9.
- 16 NAM, G.-J.; SAKALLAH, K. A.; RUTENBAR, R. A. A new FPGA detailed routing approach via search-based boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 21, n. 6, p. 674–684, 2002. DOI 10.1109/TCAD.2002.1004311.
- 17 MIRONOV, I.; ZHANG, L. Applications of sat solvers to cryptanalysis of hash functions. *In: BIERE, A.; GOMES, C. P. (ed.). Theory and applications of satisfiability testing - SAT 2006.* Berlin, Heidelberg: Springer, 2006. p. 102–115. ISBN 978-3-540-37207-3. DOI 10.1007/11814948_13.
- 18 INTERNATIONAL CONFERENCES ON THEORY AND APPLICATIONS OF SATISFIABILITY TESTING (SAT), 23., 2020, Alghero, Italy. *Proceedings [...]*. Alghero, Italy: SAT, 2020. Available from: <http://www.satisfiability.org/>. Accessible at: 30 Oct. 2020.
- 19 THE International SAT Competition Web Page. 2020. Available from: <http://www.satcompetition.org/>. Accessible at: 30 Oct. 2020.
- 20 BARAHONA, F. On the computational complexity of ising spin glass models. *Journal of Physics A: mathematical and general*, v. 15, n. 10, p. 3241–3253, 1982. DOI 10.1088/0305-4470/15/10/028.
- 21 MEZARD, M.; PARISI, G.; VIRASORO, M. *Spin glass theory and beyond*. Singapore: World Scientific, 1986. ISBN 978-9971-5-0116-7.
- 22 LUCAS, A. Ising formulations of many np problems. *Frontiers in Physics*, v. 2, p. 5, 2014. DOI 10.3389/fphy.2014.00005.
- 23 KIRKPATRICK, S.; GYORGYI, G.; TISHBY, N.; TROYANSKY, L. The statistical mechanics of k-satisfaction. *In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, 6, 1993, Denver. *Proceedings [...]*. Denver: Morgan Kaufmann, 1993. p. 439–446.
- 24 KIRKPATRICK, S.; SELMAN, B. Critical behavior in the satisfiability of random boolean expressions. *Science*, v. 264, n. 5163, p. 1297–1301, 1994.
- 25 SELMAN, B.; KAUTZ, H. A.; COHEN, B. Noise strategies for improving local search. *In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI-94)*. 12., 1994, Seattle. *Proceedings [...]*. Seattle: AAAI Press, 1994. p. 337–343. ISBN 978-0-262-51078-3.
- 26 SELMAN, B.; KAUTZ, H.; COHEN, B. Local search strategies for satisfiability testing. *In: DIMACS: series in discrete mathematics and theoretical computer science*. Providence, Rhode Island: American Mathematical Society, 1996. p. 521–532. Available from: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.2207&rep=rep1&type=pdf>. Accessible at: 30 Oct. 2020.

-
- 27 PRIVMAN, V. *Finite-size scaling and numerical simulations of statistical systems*. Singapore: World Scientific, 1990. ISBN 9789810201081.
 - 28 BÜNING, H. K.; LETTMANN, T. *Propositional logic: deduction and algorithms*. Cambridge: Cambridge University Press, 1999. (Cambridge tracts in theoretical computer science). ISBN 9780521630177.
 - 29 SKIENA, S. S. *The algorithm design manual*. 2nd ed. London: Springer, 2009. ISBN 9781848000704.
 - 30 ASPVALL, B.; PLASS, M. F.; TARJAN, R. E. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, v. 8, n. 3, p. 121 – 123, 1979. ISSN 0020-0190. DOI 10.1016/0020-0190(79)90002-4.
 - 31 CALABRO, C.; IMPAGLIAZZO, R.; PATURI, R. The complexity of satisfiability of small depth circuits. In: CHEN, J.; FOMIN, F. V. (ed.). *Parameterized and exact computation*. Berlin, Heidelberg: Springer, 2009. p. 75–85. ISBN 978-3-642-11269-0.
 - 32 VALIANT, L. The complexity of computing the permanent. *Theoretical Computer Science*, v. 8, n. 2, p. 189 – 201, 1979. ISSN 0304-3975. DOI 10.1016/0304-3975(79)90044-6.
 - 33 ARORA, S.; BARAK, B. *Computational complexity: a modern approach*. Cambridge, UK: Cambridge University Press, 2009. ISBN 9780521424264.
 - 34 SELMAN, B.; MITCHELL, D. G.; LEVESQUE, H. J. Generating hard satisfiability problems. *Artificial Intelligence*, v. 81, n. 1, p. 17 – 29, 1996. ISSN 0004-3702.
 - 35 SAT COMPETITION 2009: benchmark submission guidelines. 2009. Available from: <http://www.satcompetition.org/2009/format-benchmarks2009.html>. Accessible at: 28 Apr. 2020.
 - 36 MONASSON, R.; ZECCHINA, R.; KIRKPATRICK, S.; SELMAN, B.; TROYANSKY, L. Determining computational complexity from characteristic phase transitions. *Nature*, v. 400, n. 6740, p. 133–137, 1999.
 - 37 REIF, F. *Fundamentals of statistical and thermal physics*. Long Grove, Illinois: Waveland Press, 2009. ISBN 9781478610052.
 - 38 MCALLESTER, D.; BART, B. S.; KAUTZ, H. Evidence for invariants in local search. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 14., 1997; CONFERENCE ON INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE, 9., 1997, Providence. *Proceedings [...]*. Providence, Rhode Island: AAAI Press, 1997. p. 321–326. ISBN 0262510952.
 - 39 GOTTLIEB, J.; MARCHIORI, E.; ROSSI, C. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, v. 10, n. 1, p. 35–50, 2002. DOI 10.1162/106365602317301763.
 - 40 MCDONALD, A.; GORDON, A. G. *Parallel WalkSAT with clause learning*. 2009. Available from: https://www.ml.cmu.edu/research/dap-papers/dap_mcdonald.pdf. Accessible at: 23 Jan. 2020.
 - 41 MARRO, J.; DICKMAN, R. *Nonequilibrium phase transitions in lattice models*. Cambridge, UK: Cambridge University Press, 1999. ISBN 9780511524288.

- 42 CRAWFORD, J. M.; AUTON, L. D. Experimental results on the crossover point in satisfiability problems. *In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI-93)*. 11., 1993, Seattle. *Proceedings [...]*. Seattle: AAAI Press, 1993. p. 21–27. ISBN 9780262510714.
- 43 DAVIS, M.; PUTNAM, H. A computing procedure for quantification theory. *Journal of the ACM*, v. 7, n. 3, p. 201–215, 1960.
- 44 ERDÖS, P.; RÉNYI, A. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, v. 5, p. 17–61, 1960.
- 45 NEWMAN, M. E. J.; STROGATZ, S. H.; WATTS, D. J. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, v. 64, n. 2, p. 26118, 2001.
- 46 JANSON, S.; KNUTH, D. E.; ŁUCZAK, T.; PITTEL, B. The birth of the giant component. *Random Structures Algorithms*, v. 4, n. 3, p. 231–358, 1993.
- 47 STAUFFER, D.; AHARONY, A. *Introduction to percolation theory*. London: Taylor & Francis, 1985. ISBN 978-0748402533.
- 48 REIA, S. M.; AQUINO, L. F.; FONTANARI, J. F. The surprising little effectiveness of cooperative algorithms in parallel problem solving. *European Physical Journal B*, v. 93, n. 7, p. 140, 2020.
- 49 REIA, S. M.; AMADO, A. C.; FONTANARI, J. F. Agent-based models of collective intelligence. *Physics of Life Reviews*, v. 31, n. 3, p. 320–331, 2019. DOI 10.1016/j.plrev.2018.10.004.
- 50 BÄCK, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford: Oxford University Press, 1996.
- 51 FONTANARI, J. F. Imitative learning as a connector of collective brains. *PLoS ONE*, v. 9, n. 10, p. e110517, 2014.
- 52 KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983.
- 53 GOTTLIEB, J.; ROSSI, E. M. and C. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, v. 10, n. 1, p. 35–50, 2002.
- 54 CORKILL, D. D. Blackboard systems. *Artificial Intelligence Review*, v. 100, n. 6, p. 40–47, 1988.
- 55 CLEARWATER, S.; HOGG, T.; HUBERMAN, B. Cooperative problem solving. *Science*, v. 254, n. 5035, p. 1181–1183, 1991.
- 56 FONTANARI, J. F. Reputation blackboard systems. *Cognitive Systems Research*, v. 50, p. 29–35, 2018. DOI 10.1016/j.cogsys.2018.03.008.
- 57 ESFAHBOD, B. *Euler diagram for P, NP, NP-Complete, and NP-Hard*. 2007. Available from: https://en.wikipedia.org/w/index.php?title=File:P_np_np-complete_np-hard.svg. Accessible at: 23 Jan. 2020.

Appendix

APPENDIX A – TURING MACHINES

A Deterministic one-tape Turing Machine (also written as *DTM* or simply Turing Machine) is a mathematical model of computation, developed in 1936 by Alan Turing, that defines a mechanical machine.⁷ This machine has three elements:

- An infinite tape that contains the input, output, and internal values of symbols used by the machine. This tape represents the memory of the machine and is composed of discrete cells. Each cell can be empty or contain a symbol (data).
- A read/write head that operates on a single cell each time step and is capable of reading its value or writing into it, as well as moving the tape one cell to the left or to the right.
- A state control register that stores the current state of the machine.

For this machine to operate, it needs a *program* that specifies:¹

- A finite set of symbols for the tape, which contains at least the input data and a blank symbol.
- A finite set of machine states, which includes at least three special states: the initial state and two result states.
- A finite table of instructions, also known as transition function, that specifies what the machine will do next.

The machine then proceeds as follows. It starts in the supplied initial state with some input symbols written on the tape. The head then reads the symbol beneath it. Based on this symbol, the current state of the machine, and the table of instructions, the machine then erases or writes a symbol under the head, moves the tape one cell to the left or to the right, and changes to a new state based on the transition function. This new state could be the same as the previous one, a different one, or one of the two halting states, representing the end of the computation. These two halting states represent the answer of that program for that input, and it can either be ‘yes’ or ‘no’.

In this context, we say this machine is *deterministic*, as we always know what its next state will be.

If a Turing Machine completes the execution of a program in a polynomial number of steps on the length of the input, we say that problem belongs to the complexity class

P.²⁹ But not all problems can be solved in a polynomial number of steps. If a DTM can verify if a given solution to a decision problem is indeed correct in polynomial time, this problem is said to be in the complexity class NP.⁴ This clearly shows that $P \subseteq NP$, as if a problem can be solved in polynomial time it can also be verified in polynomial time.

Alternatively, the complexity class NP can be defined as the set of all decision problems that can be solved in polynomial time using a non-deterministic Turing Machine (NTM). In this machine, the transition function might specify more than one state for the next action. In this case, the machine works as an oracle, always picking the state that leads to the solution. Since the next state is not fully determined by the current state and the symbol it reads, this machine is called *non-deterministic*. But note that the non-determinism is only about its internal state, an NTM always gives the same result for the same input.⁹

Decision problems can be more difficult to solve than those in the NP class. There is a class of problems called *NP-hard* defined as follows. A decision problem X is in NP-hard if for every problem $Y \in NP$, there is a polynomial time reduction from Y to X . This class contains all problems that are at least as hard as any NP problem. A problem is said to be NP-complete if it is both in NP and in NP-hard.²⁹

An interesting visualization of the four complexity classes P, NP, NP-complete and NP-hard is presented in the figure 19. One should take care to distinguish between two distinct scenarios: the case $P = NP$ and the case $P \neq NP$. If $P = NP$ then $P = NP$ -complete as well, since NP-complete is a subset of NP.

Figure 19 – Euler diagrams for P, NP, NP-complete, and NP-hard computational complexity classes. The left panel is valid under the assumption that $P \neq NP$, whereas the right panel is valid under the assumption that $P=NP$.

Source: ESFAHBOD.⁵⁷

APPENDIX B – RANDOM ASSIGNMENTS

In section 3.2, we pointed the rather curious fact that the WalkSAT algorithm already starts with a small number of unsatisfied clauses. Of course, this has nothing to do with the algorithm, but with the fact that by simply assigning the values *True* and *False* at random to the literals is enough to satisfy a large number of clauses. In fact, there is a simple proof of this result⁴ that we will present in this appendix.

We begin by noting that for a k -SAT, a clause is not satisfied if and only if each of its k literals evaluate to *False*. Hence, for a formula generated at random the probability that a clause is not satisfied is simply $1/2^k$ and the probability that it is satisfied is $1 - 1/2^k$. Since the random formula has M independent clauses, the probability that exactly m of them are satisfied is given by the binomial distribution

$$P(m) = \binom{M}{m} \left(1 - 1/2^k\right)^m \left(1/2^k\right)^{M-m} \quad (\text{B.1})$$

so that the mean or expected number of satisfied clauses is

$$\langle m \rangle = M \left(1 - 1/2^k\right). \quad (\text{B.2})$$

We note that the probability that a random formula is satisfied is $P(M) = \left(1 - 1/2^k\right)^M$, which vanishes exponentially fast as the number M of clauses increases.

For the 3-SAT, the fraction of satisfied clauses by a random assignment of the N literals is $7/8$, which means that only 12.5% of the clauses are unsatisfied.⁴

To complement the previous analytical argument, we measured the fraction of unsatisfied clauses as a function of the ratio α of clauses-to-variables for different choices of the number N of variables. The result is shown in figure 20 and, as expected, agrees very well with the predicted value $1 - \langle m \rangle / M = 1/8$. In particular, for each set of N and M we generated 100000 random instances of the 3-SAT and for each instance, we generated a single random assignment of the N literals.

Hence, our simulations confirm the theoretical prediction for the expected number of satisfied clauses when the assignment is generated at random. It is important to emphasize that this result is valid for random SAT instances only. In fact, the binomial distribution (B.1) holds true provided that the M clauses are generated independently of one another.

Figure 20 – Fraction of unsatisfied clauses for a randomly generated assignment of many random 3-SAT formulas. Our experiment confirms that a random assignment of the literals leaves on average only 12.5% of the clauses unsatisfied. This result does not depend on the numbers of variables and clauses. Note the small range of the vertical axis, which varies from 12.46% to 12.54%.

Source: By the author.

APPENDIX C – WalkSAT MINIMUM SEARCH TIME

In section 3.2, we pointed that the minimum time needed by the WalkSAT to find a solution grows linear with the number N of variables. This minimum search time t_{min}^* is the leftmost search time of the distribution $P(t^*)$ (see figures 6 and 8). We will discuss this point in more detail in this appendix.

As a general argument, let us consider an assignment where all N variables must be flipped in order to produce the (assumed unique) solution of the formula. This is the worst possible initial scenario for the WalkSAT. Clearly, there is a run of the WalkSAT that flips the N variables in the correct sequence so that $t_{min}^* = N$. Of course, a typical run has $t^* \gg t_{min}^*$ as illustrated in figures 6 and 8.

In figure 21 we show t_{min}^* as a function of N using random initial assignments. The linear fitting indicates that $t_{min}^* \approx 0.44N$ for large N . This is so because in the random initial assignments about 7/8 of the initial clauses are already satisfied, as explained in appendix B, so there are less than N variables to negate in order to hit the solution.

Figure 21 – Minimum search time t_{min}^* for the WalkSAT to solve a random 3-CNF as a function of the number N of variables. The dashed straight line is the linear fitting $t_{min}^* = -32 + 0.44N$. The parameters are $\alpha = 3$, $P = 0.5$ and $T_{max} = 10^6$.

Source: By the author.

APPENDIX D – COLLECTIVE BLIND SEARCH

In this appendix we derive equation (4.1) that gives the mean search time of a group of L independent agents in the case that each agent explores the solution space of a given Boolean formula by randomly negating all its variables. Let us start with the case of a single agent ($L=1$). Recalling that at each time step this agent produces a new random assignment, the probability that this assignment is one of the solutions of the Boolean formula is $q = S/2^N$, where S is the number of solutions and 2^N is the number of possible assignments. Hence the probability that the agent finds the solution at time t^* is

$$P_1(t^*) = (1 - q)^{t^*-1} q, \quad (\text{D.1})$$

which is a geometric distribution with success probability equal to q . We can interpret this equation as the probability of the agent to search for a solution, but not find one, in $t^* - 1$ steps, times the probability of finding a solution. Here we have used the subscript 1 in $P_1(t^*)$ to indicate that $L = 1$.

In the case of L agents, the time t^* when the solution is found is the random variable

$$t^* = \min(t_1^*, t_2^*, \dots, t_L^*) \quad (\text{D.2})$$

where t_i^* are independent random variables distributed by the geometric distribution (D.1). Our task is to derive the probability distribution of the variable t^* defined in equation (D.2). For $L = 2$ agents (a and b), we have

$$\begin{aligned} P_2(t^*) &= P_a(t^*)P_b(t \geq t^*) + P_a(t > t^*)P_b(t^*) \\ &= 2P_1(t^*) \sum_{t=t^*+1}^{\infty} P_1(t) + [P_1(t^*)]^2 \\ &= [1 - (1 - q)^2](1 - q)^{2(t^*-1)}, \end{aligned} \quad (\text{D.3})$$

in which $P_a(t)$ and $P_b(t)$ are independent and identically distributed random variables, so that we can change their labels to $P_1(t)$. For $L = 3$ agents (a , b , and c), we note that the we can reuse the previous equation:

$$\begin{aligned} P_3(t^*) &= P_2(t^*)P_c(t \geq t^*) + P_2(t > t^*)P_c(t^*) \\ &= P_1(t^*) \sum_{t=t^*+1}^{\infty} P_2(t) + P_2(t^*) \sum_{t=t^*+1}^{\infty} P_1(t) + P_1(t^*)P_2(t^*) \\ &= [1 - (1 - q)^3](1 - q)^{3(t^*-1)}, \end{aligned} \quad (\text{D.4})$$

so that we can write

$$P_L(t^*) = (1 - q)^{L(t^*-1)} [1 - (1 - q)^L]. \quad (\text{D.5})$$

The interpretation of (D.5) is straightforward: $[(1-q)^L]^{(t^*-1)}$ is the probability that the L agents do not find a solution in the first $t^* - 1$ attempts, and $[1 - (1-q)^L]$ is the probability that at least one of the L agents finds a solution in the subsequent attempt. Hence the probability distribution of the random variable t^* defined in equation (D.2) is a geometric distribution with success probability $1 - (1-q)^L$ and the average search time is

$$\begin{aligned}
\langle t^* \rangle &= \sum_{t^*=1}^{\infty} t^* P_L(t^*) \\
&= \frac{[1 - (1-q)^L]}{(1-q)^L} \sum_{t^*=1}^{\infty} t^* (1-q)^{Lt^*} \\
&= \frac{[1 - (1-q)^L]}{(1-q)^L} \sum_{t^*=1}^{\infty} \frac{(1-q)}{L} \frac{\partial}{\partial(1-q)} (1-q)^{Lt^*} \\
&= \frac{[1 - (1-q)^L]}{(1-q)^L} \frac{(1-q)}{L} \frac{\partial}{\partial(1-q)} \sum_{t^*=1}^{\infty} (1-q)^{Lt^*} \\
&= \frac{[1 - (1-q)^L]}{(1-q)^{L-1}L} \frac{\partial}{\partial(1-q)} \frac{(1-q)^L}{1 - (1-q)^L} \\
&= \frac{[1 - (1-q)^L]}{(1-q)^{L-1}L} \left[\frac{L(1-q)^{L-1}}{1 - (1-q)^L} + \frac{(1-q)^L L(1-q)^{L-1}}{[1 - (1-q)^L]^2} \right] \\
&= \frac{1}{1 - (1-q)^L}, \tag{D.6}
\end{aligned}$$

which is equation (4.1).