Trajectories in Phase Diagrams, Growth Processes, and Computational Complexity: How Search Algorithms Solve the 3-Satisfiability Problem

Simona Cocco* and Rémi Monasson†
CNRS-Laboratoire de Physique Théorique de l'ENS, 24 rue Lhomond, 75005 Paris, France
(Received 15 September 2000)

Decision and optimization problems typically fall into one of two categories for any particular solving algorithm. The problem is either solved quickly (easy) or demands an impractically long computational effort (hard). Here we show that some characteristic parameters of problems can be tracked during a run of the algorithm defining a trajectory through the parameter space. Focusing on 3-satisfiability, a recognized representative of hard problems, we analyze trajectories generated by search algorithms. These trajectories can cross well-defined phases, corresponding to domains of easy or hard instances, and allow one to successfully predict the times of resolution.

DOI: 10.1103/PhysRevLett.86.1654 PACS numbers: 89.20.Ff, 02.50.–r, 05.20.–y, 89.75.Hc

Consider a set of N Boolean variables and a set of $M = \alpha N$ constraint clauses. Each clause is the logical OR of three variables or of their negations; see Fig. 1. Then, try to figure out whether or not there exists an assignment of variables satisfying all clauses (called solution) [1].

This problem is termed 3-satisfiability (3-SAT), and is among the most difficult ones to solve as its size N becomes large. A fundamental conjecture of computer science is that no method exists to solve 3-SAT efficiently [2], i.e., in time growing at most polynomially with N. In practice, one therefore resorts to methods that need, *a priori,* exponentially large computational resources. One of these algorithms, the ubiquitous Davis-Putnam-Loveland-Logemann (DPLL) solving procedure [1,3,4], is illustrated in Fig. 1. DPLL operates by trial and error, the sequence of which can be graphically represented as a search tree made of nodes connected through edges (Fig. 1). Examples of search trees for satisfiable (sat) or unsatisfiable (unsat) instances are shown in Fig. 2. Computational complexity is the amount of operations performed by the solving algorithm. We follow the convention that it is measured by the size of the search tree, i.e., the number of nodes.

Complexity may vary enormously with the instance, the set of clauses, under consideration. To understand why instances are easy or hard to solve, computer scientists have focused on model classes of 3-SAT instances. Probabilistic models that define distributions of random instances controlled by few parameters are particularly useful in shedding light on the onset of complexity. An example that has attracted a lot of attention over the past years is random 3-SAT: all clauses are drawn randomly and each variable negated or left unchanged with equal probabilities. Experiments [4,5] and theory [6] indicate that clauses can almost surely always (respectively, never) be simultaneously satisfied if α is smaller (respectively, larger) than a critical threshold $\alpha_C \simeq 4.3$ as soon as M, N go to infinity at fixed ratio α. This phase transition [6,7] is accompanied by a drastic peak in hardness at threshold [4,5]; see Fig. 3. A complete understanding of this pattern of complexity is lacking so far.

In this Letter, we argue that search algorithms induce a dynamical evolution of the computational problem to be solved. We shall show how this complex, non-Markovian dynamics is closely related to growth processes. Concepts and tools from statistical mechanics allow one to understand and predict analytically computational complexity. For the sake of clarity, we shall report technical details in another, extended paper [8].

As shown in Fig. 1, the action of DPLL on an instance of 3-SAT causes the reduction of 3-clauses to 2-clauses. We use a mixed $2 + p$-SAT distribution [7], where p is the fraction of 3-clauses, to model what remains of the input instance at a node of the search tree. Using experiments and methods from statistical mechanics [7], the threshold line $\alpha_C(p)$, separating sat from unsat phases, may be obtained with the results shown in Fig. 4. The phase diagram of $2 + p$-SAT is the natural space in which the DPLL dynamic takes place. An input 3-SAT instance with ratio α shows up on the right vertical boundary of Fig. 4 as a point of coordinates $(p = 1, \alpha)$. Under the action of DPLL, the representative point moves aside from the 3-SAT axis and follows a trajectory. The location of this trajectory in the phase diagram allows a precise understanding of the search tree structure and of complexity. We consider the trajectories generated by DPLL using the generalized unit clause (GUC) heuristics [9] (Fig. 1); however, our calculation could be repeated for other rules [4,5,9,10].

At sufficiently small ratios $\alpha < \alpha_L \simeq 3.003$, DPLL easily finds a solution [9,10]. The search tree has a unique branch, i.e., a sequence of edges joining the top node to the extremity. This branch grows as the fraction t of variables assigned by DPLL increases and terminates with a solution (Fig. 2A). Each node along the branch carries a $2 + p$-SAT instance with characteristic parameters p and α. The knowledge of p and α as functions of the "depth" t of the node, first established by Chao and Franco [9], allows us to draw the trajectory followed by the instance

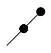
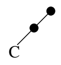
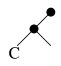
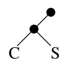
step	clauses	search tree
0	$w \vee \bar{x} \vee y$ $\bar{w} \vee x \vee z$ $\bar{w} \vee \bar{x} \vee \bar{y}$ $\bar{w} \vee \bar{x} \vee y$ $x \vee y \vee \bar{z}$	
1	split : w = T	
2	$x \vee z$ $\bar{x} \vee \bar{y}$ $\bar{x} \vee y$ $x \vee y \vee \bar{z}$	
3	split : x = T	
4	$\bar{y}$ y	
5	propagation : y = F, y = T contradiction	
6	backtracking to stage 1 : x = F	
7	z $y \vee \bar{z}$	
8	propagation : z = T, y = T solution : w = T, x = F, y = T, z = T	

FIG. 1. Example of 3-SAT instance and Davis-Putnam-Loveland-Logemann resolution. Step 0: The instance consists of $M = 5$ clauses involving $N = 4$ variables x, y, w, and z, which can be assigned to true (T) or false (F). $\bar{w}$ means (NOT w) and $\vee$ denotes the logical OR. The search tree is empty. Step 1: DPLL randomly selects a variable among the shortest clauses and assigns it to satisfy the clause it belongs to, e.g., $w = T$ (splitting with the generalized unit clause—GUC—heuristic) [9]. A node and an edge symbolizing, respectively, the variable chosen (w) and its value (T) are added to the tree. Step 2: The logical implications of the last choice are extracted: clauses containing w are satisfied and eliminated, clauses including $\bar{w}$ are simplified, and the remaining ones are left unchanged. If no unitary clause (i.e., with a single variable) is present, a new choice of variable has to be made. Step 3: Splitting takes over. Another node and another edge are added to the tree. Step 4: Same as step 2 but now unitary clauses are present. The variables they contain have to be fixed accordingly. Step 5: The propagation of the unitary clauses results in a contradiction. The current branch dies out and gets marked with C. Step 6: DPLL backtracks to the last split variable (x), inverts it (F) and creates a new edge. Step 7: Same as step 4. Step 8: The propagation of the unitary clauses eliminates all the clauses. A solution S is found and the instance is satisfiable. For an unsatisfiable instance, unsatisfiability is proven when backtracking (see step 6) is not possible anymore, since all split variables have already been inverted. In this case, all the nodes in the final search tree have two descendent edges and all branches terminate by a contradiction C.

under the action of DPLL in Fig. 4. The trajectory, indicated by a light dashed line, first heads to the left and then reverses to the right until reaching a point on the 3-SAT axis at a small ratio without ever leaving the sat region. Further action of DPLL leads to a rapid elimination of the remaining clauses and the trajectory ends up at the lower right corner S, where a solution is achieved. Thus, in the range of ratios $\alpha < \alpha_L \simeq 3.003$, 3-SAT is easy to solve: the computational complexity scales linearly with the size N (Fig. 3).

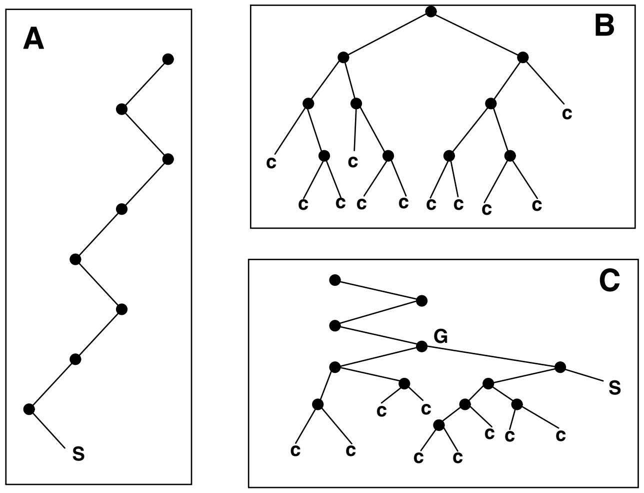


FIG. 2. Types of search trees generated by the DPLL solving procedure. (a) *Simple branch:* the algorithm easily finds a solution without ever backtracking. (b) *Dense tree:* in the absence of solution, the algorithm builds a "bushy" tree, with many branches of various lengths, before stopping. (c) *Mixed case, branch + tree:* if many contradictions arise before reaching a solution, the resulting search tree can be decomposed in a single branch followed by a dense tree. The junction G is the highest backtracking node reached back by DPLL.

For ratios above threshold ($\alpha > \alpha_C \simeq 4.3$), instances almost never have a solution but a considerable amount of backtracking is necessary before proving that clauses are incompatible. As shown in Fig. 2B, a generic unsat tree includes many branches. The number of branches, B, is related to the number of nodes, Q, through the relation $Q = B - 1$ valid for any complete tree; check Fig. 2B.

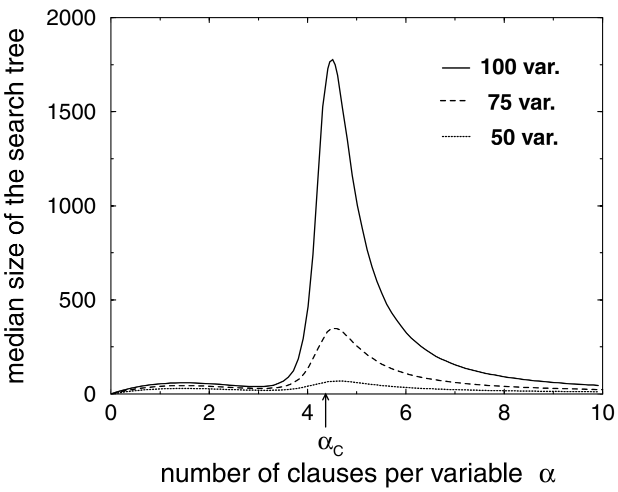


FIG. 3. Complexity of 3-SAT solving for three problem sizes and averaged over 10 000 randomly drawn samples.

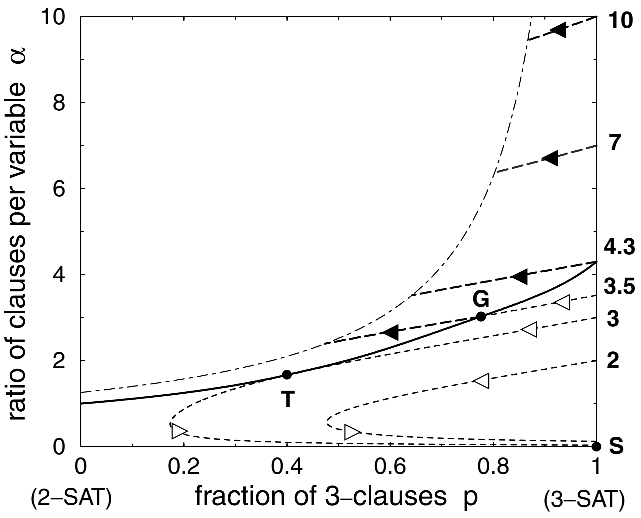


FIG. 4. Phase diagram of 2 + p-SAT and dynamical trajectories of DPLL. The threshold line $\alpha_C(p)$ (bold full line) separates sat (lower part of the plane) from unsat (upper part) phases. Extremities lie on the vertical 2-SAT (left) and 3-SAT (right) axes at coordinates ($p = 0, \alpha_C = 1$) and ($p = 1, \alpha_C \simeq 4.3$), respectively. Departure points for DPLL trajectories are located on the 3-SAT vertical axis and the corresponding values of α are explicitly given. Dashed curves represent tree trajectories in the unsat region (thick lines, black arrows) and branch trajectories [9] in the sat phase (thin lines, empty arrows). Arrows indicate the direction of "motion" along trajectories parametrized by the fraction t of variables set by DPLL. For small ratios $\alpha < \alpha_L$, branch trajectories remain confined in the sat phase and end in S of coordinates $(1, 0)$, where a solution is found. At $\alpha_L \simeq 3.003$, the single branch trajectory hits tangentially the threshold line in T of coordinates $(2/5, 5/3)$ (T depends *a priori* on the heuristics used, but lies very close to the tricritical point of Ref. [7]; see [8]). In the intermediate range $\alpha_L < \alpha < \alpha_C$, the branch trajectory intersects the threshold line at some point G (that depends on α). A dense tree then grows in the unsat phase, as happens when 3-SAT departure ratios are above threshold $\alpha > \alpha_C \simeq 4.3$. The tree trajectory halts on the dot-dashed curve $\alpha \simeq 1.259/(1 - p)$ where the tree growth process stops [14]. At this point, DPLL has reached back the highest backtracking node in the search tree, that is, the first node when $\alpha > \alpha_C$, or node G for $\alpha_L < \alpha < \alpha_C$. In the latter case, a solution can be reached from a new descending branch while, in the former case, unsatisfiability is proven; see Fig. 2.

Complexity grows exponentially with N [11], so it is convenient to define its logarithm ω through $Q = 2^{N\omega}$. We experimentally directly counted Q or, alternatively, B, and averaged the corresponding logarithms ω over a large number of instances. Results have then be extrapolated to the $N \to \infty$ limit [8] and are reported in Table I.

We have analytically computed ω as a function of α, extending to the unsat region the probabilistic analysis of DPLL. The search tree of Fig. 2B is the output of a sequential process: nodes and edges are added by DPLL through successive descents and backtrackings. We have imagined a different building up that results in the same complete tree but can be mathematically analyzed: the tree grows in parallel, layer after layer. A new layer is added by assigning, according to DPLL heuristic, one more variable along each living branch. As a result, a branch may split (case 1), keep growing (case 2), or carry a contradiction and die out (case 3). Cases 1, 2, and 3 are stochastic events, the probabilities of which depend essentially on the characteristic parameters (p, α) defining the 2 + p-SAT instance carried by the branch, and on the depth (fraction of assigned variables) t in the tree. This Markovian approximation permits one to write an evolution equation for the logarithm $\omega(p, \alpha; t)$ of the average number of branches with parameters p, α as the depth t increases,

$$\frac{\partial \omega}{\partial t} = \mathcal{H}\left[p, \alpha, \frac{\partial \omega}{\partial p}, \frac{\partial \omega}{\partial \alpha}, t\right], \qquad (1)$$

and $\mathcal{H}$ incorporates the details of the splitting heuristics [8,12]. Partial differential equation (1) is analogous to growth processes encountered in statistical physics [13]. The surface ω, growing with "time" t above the bidimensional plane p, α, describes the whole distribution of branches. The average number of branches at depth t in the tree equals $B(t) = \int dp\, d\alpha\, 2^{N\omega(p,\alpha;t)} \simeq 2^{N\omega^*(t)}$, where $\omega^*(t)$ is the maximum over p, α of $\omega(p, \alpha; t)$ reached in $p^*(t)$, $\alpha^*(t)$. In other words, the exponentially dominant contribution to $B(t)$ comes from branches carrying 2 + p-SAT instances with parameters $p^*(t)$, $\alpha^*(t)$, which define the tree trajectories in Fig. 4. The hyperbolic line indicates the halt points, where contradictions prevent dominant branches from further growing [14]. Along the tree trajectory, $\omega^*(t)$ grows from 0, on the right vertical axis, up to some final positive value, $\hat{\omega}$, on the halt line. $\hat{\omega}$ is our theoretical prediction for the logarithm of the complexity (divided by N). Values of $\hat{\omega}$, obtained for $4.3 < \alpha < 20$ by solving Eq. (1) compare very well with numerical results (Table I).

TABLE I. Logarithm of the complexity ω from measures of search tree sizes (number of nodes, Q, and of branches, B) and theory [12]. Experimental results for $\alpha = 3.5$ (sat phase) are compared to the complexity of the unsat tree built from G; $\hat{\omega} = 0.035$ is obtained from $\hat{\omega} = \hat{\omega}_G(1 - t_G)$, where $t_G \simeq 0.19$ is the fraction of variables assigned by DPLL at point G; see text. At large α, each variable appears in many clauses and contradictory clauses are detected earlier by DPLL. Tree trajectories are short, and refutation trees, small. Complexity is found to scale asymptotically as $\hat{\omega} = (3 + \sqrt{5})\{\ln[(1 + \sqrt{5})/2]\}^2/(6 \ln 2)/\alpha \simeq 0.292/\alpha$; this result seems to be exact [8,16].

Initial	Experiments		Theory
Ratio α	$\log_2 Q$	$\log_2 B$	$\hat{\omega}$
20	0.0153 ± 0.0002	0.0151 ± 0.0001	0.0152
15	0.0207 ± 0.0002	0.0206 ± 0.0001	0.0206
10	0.0320 ± 0.0005	0.0317 ± 0.0002	0.0319
7	0.0482 ± 0.0005	0.0477 ± 0.0005	0.0477
4.3	0.089 ± 0.001	0.0895 ± 0.001	0.0875
3.5	0.034 ± 0.003		0.035
G	0.040 ± 0.002	0.041 ± 0.003	0.044

The intermediate region $\alpha_L < \alpha < \alpha_C$ juxtaposes the two previous behaviors; see tree Fig. 2C. The branch trajectory, started from the point $(p = 1, \alpha)$ corresponding to the initial 3-SAT instance, hits the critical line $\alpha_c(p)$ at some point G with coordinates (p_G, α_G) after Nt_G variables have been assigned by DPLL; see Fig. 4. The algorithm then enters the unsat phase and generates $2 + p$-SAT instances with no solution. A dense subtree, that DPLL has to go through entirely, forms beyond G till the halt line (Fig. 4). The size of this subtree, $2^{N(1-t_G)\hat{\omega}_G}$, can be analytically predicted from our theory. G is the highest backtracking node in the tree (Fig. 2C) reached back by DPLL, since nodes above G are located in the sat phase and carry $2 + p$-SAT instances with solutions. DPLL will eventually reach a solution. The corresponding branch (rightmost path in Fig. 2C) is highly nontypical and does not contribute to the complexity, since almost all branches in the search tree are described by the tree trajectory issued from G (Fig. 4). We have checked experimentally this scenario for $\alpha = 3.5$. The coordinates of the average highest backtracking node, $(p_G \simeq 0.78, \alpha_G \simeq 3.02)$, coincide with the analytically computed intersection of the single branch trajectory and the critical line $\alpha_c(p)$. As for complexity, experimental measures of ω from 3-SAT instances at $\alpha = 3.5$, and of ω_G from $2 + 0.78$-SAT instances at $\alpha_G = 3.02$, obey the expected identity $\omega = \omega_G(1 - t_G)$ and are in very good agreement with theory (Table I). Therefore, the structure of search trees for 3-SAT reflects the existence of a critical line for $2 + p$-SAT instances.

In conclusion, we have shown that statistical physics is useful to study the solving complexity of branch and bound algorithms [1,2] applied to hard combinatorial problems. The phase diagram of Fig. 4 affords a qualitative understanding of the probabilistic complexity of DPLL variants on random instances. This view may reveal the nature of the complexity of search algorithms for SAT and related NP-complete problems [2,15]. In the sat phase, branch trajectories are related to polynomial time computations while in the unsat region, tree trajectories lead to exponential calculations. Depending on the starting point (ratio α of the 3-SAT instance), one or a mixture of these behaviors is observed. Figure 4 furthermore gives some insights to improve the search algorithm. In the unsat region, trajectories must be as horizontal as possible to minimize their length but resolution is necessarily exponential [11]. In the sat domain, heuristics making trajectories steeper could avoid the critical line $\alpha_C(p)$ and solve 3-SAT polynomially up to threshold.

We are grateful to J. Franco for his encouragement and numerous suggestions in the writing of this paper. We also thank A. Hartmann and M. Weigt for their interest in our work [15], and M. Poirier for a careful reading of our manuscript.

*Current address: Department of Physics, The University of Chicago, 845 W. Taylor Street, Chicago, IL 60607.

†Current address: The James Franck Institute, The University of Chicago, 5640 S. Ellis Avenue, Chicago, IL 60637.

[1] B. Hayes, Am. Sci. **85**, 108–112 (1996).

[2] M. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[3] M. Davis and H. Putnam, J. Assoc. Comput. Mach. **7**, 201–215 (1960).

[4] J. Crawford and L. Auton, *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)* (The AAAI Press/MIT Press, Cambridge, MA, 1993), pp. 21–27.

[5] *Frontiers in Problem Solving: Phase Transitions and Complexity,* edited by T. Hogg, B. A. Huberman, and C. Williams, Artificial Intelligence Vol. 81 (I and II) (Elsevier, Amsterdam, 1996).

[6] R. Monasson and R. Zecchina, Phys. Rev. E **56**, 1357–1370 (1997).

[7] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, Nature (London) **400**, 133–137 (1999).

[8] S. Cocco and R. Monasson (to be published).

[9] M. T. Chao and J. Franco, Inf. Sci. **51**, 289–314 (1990).

[10] A. Frieze and S. Suen, J. Algorithms **20**, 312–335 (1996).

[11] V. Chvàtal and E. Szmeredi, J. ACM **35**, 759–768 (1988).

[12] In terms of the densities of 2- and 3-clauses, $c_2 = \alpha(1 - t)(1 - p)$, $c_3 = \alpha(1 - t)p$, and of the partial derivatives $z_2 = \partial\omega/\partial c_2$, $z_3 = \partial\omega/\partial c_3$, we have for the GUC heuristics $\mathcal{H}(c_2, c_3, z_2, z_3; t) = \log_2 \nu(z_2) + \{3c_3[e^{z_3}(1 + e^{-z_2})/2 - 1] + c_2[\nu(z_2) - 2]\}/(1 - t)/\log 2$ where $\nu(z_2) = e^{z_2}(1 + \sqrt{4e^{-z_2} + 1})/2$ [8].

[13] *Scale Invariance, Interfaces, and Non-Equilibrium Dynamics,* edited by A. McKane, M. Droz, J. Vannimenus, and D. Wolf, Nato ASI, Ser. B, Vol. 344 (Plenum Press, New York, 1995).

[14] Each time DPLL assigns a variable through unit propagation, an average number $u(p, \alpha)$ of new 1-clauses is produced, resulting in a net rate of $u - 1$ additional 1-clauses. As long as $u < 1$, 1-clauses are quickly eliminated and do not accumulate. Conversely, if $u > 1$, 1-clauses tend to accumulate. Opposite 1-clauses x and $\bar{x}$ are likely to appear, leading to a contradiction. The halt line is defined through $u(p, \alpha) = 1$ [9].

[15] The approach exposed in this article has been recently applied to the vertex cover problem, see A. Hartmann and M. Weigt, following Letter, Phys. Rev. Lett. **86**, 1658 (2001).

[16] P. Beame, R. Karp, T. Pitassi, and M. Saks, in *Proceedings of the ACM Symposium on Theory of Computing (STOC98)* (Association for Computing Machinery, New York, 1998), pp. 561–571.