PAPER • OPEN ACCESS

Learning from survey propagation: a neural network for MAX-E-3-SAT

To cite this article: Raffaele Marino 2021 *Mach. Learn.: Sci. Technol.* **2** 035032

View the article online for updates and enhancements.

MACHINE
LEARNING
Science and Technology

PAPER

Learning from survey propagation: a neural network for MAX-E-3-SAT

Raffaele Marino[1,2]

[1] Laboratoire de Théorie des Communications, Faculté Informatique et Communications, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland
[2] Dipartimento di Fisica, Università di Roma "La Sapienza", Piazzale Aldo Moro, 5, Rome 00185, Italy

E-mail: raffaele.marino@uniroma1.it

Keywords: maximum satisfiability, message passing, combinatorial optimization, deep learning

Abstract

Many natural optimization problems are NP-hard, which implies that they are probably hard to solve exactly in the worst-case. However, it suffices to get reasonably good solutions for all (or even most) instances in practice. This paper presents a new algorithm for computing approximate solutions in $\Theta(N)$ for the maximum exact 3-satisfiability (MAX-E-3-SAT) problem by using supervised learning methodology. This methodology allows us to create a learning algorithm able to fix Boolean variables by using local information obtained by the Survey Propagation algorithm. By performing an accurate analysis, on random conjunctive normal form instances of the MAX-E-3-SAT with several Boolean variables, we show that this new algorithm, avoiding any decimation strategy, can build assignments better than a random one, even if the convergence of the messages is not found. Although this algorithm is not competitive with state-of-the-art maximum satisfiability solvers, it can solve substantially larger and more complicated problems than it ever saw during training.

1. Introduction

The Boolean Satisfiability (SAT) Problem [1, 2] is the issue of finding an assignment that satisfies a given Boolean formula. A Boolean formula is any operation made with Boolean variables, where each variable can take the value *TRUE* or *FALSE*, $\{1, 0\}$ respectively. For example, a CNF (conjunctive normal form) [3] formula is a conjunction of one or more clauses, where a clause is a disjunction of literals. A CNF formula is satisfiable if and only if there is a configuration of the Boolean variables that simultaneously satisfy all the clauses.

In our work, N defines the number of Boolean variables and M the number of clauses so that the CNF formula has the following form:

$$\bigwedge_{1 \leqslant c \leqslant M} \left(\bigvee_{1 \leqslant i \leqslant l_c} p_{ci} \right), \tag{1}$$

where l_c is the size of clause, i.e. the number of literals in clause c for $1 \leqslant c \leqslant M$, and p_{ci} is a literal, thus a proposal variable x_i or its negation $\bar{x}_i$, for $1 \leqslant i \leqslant N$.

The maximization problem associated with SAT is called maximum satisfiability (MAX-SAT). In this case, a solver tries to satisfy the maximum number of clauses given a CNF formula [4–6]. If a CNF formula has in each clause at least k literals, then the problem is called MAX-k-SAT. If there are exactly k literals for each clause in a CNF formula, then the problem is named MAX-E-k-SAT [7].

The MAX-SAT is of considerable interest not only from the theoretical side but also for applications. For instance, many mathematical logic and artificial intelligence issues can be expressed in the form of SAT or some of its variants, like constraint satisfaction [8]. Examples are in probabilistic inference [9], data analysis [10], Maximum Clique and Maximum Independent Set [11–13], software analysis [14–16], reasoning over

bio networks and Bayesian network structure learning [17], minimization of visibly pushdown automata [18], compressive sensing [19], community detection [20] and much more [21–24]. For instance, in physics, the MAX-SAT problem is used for providing a provable periodically constrained ground state of a complex lattice [25]. Physicists also study the global landscape structure of the MAX-SAT problem for understanding phase transitions that appear into the solution space [26], trying to connect them to computational complexity limits. An interesting example of this research was developed in [27] and references therein. In these papers, continuous-time deterministic systems based on ordinary differential equations were proposed as SAT and MAX-SAT solvers. These works are based on the observation that the continuous-time deterministic systems have dynamics attracted by fixed points that identify solutions with minimum energy.

From the theoretical point of view [28–30], the MAX-SAT problem is studied for giving optimal inapproximability results. Inapproximability results help to understand the computational complexity of hard problems [7, 31]. Many natural optimization problems, indeed, are NP-hard. This implies that they are probably hard to solve exactly in the worst-case. The worst-case complexity measures the maximum amount of resources that an algorithm requires, given an input of arbitrary size [32].

However, it suffices to get reasonably good solutions for all (or even most) instances in practice. Examples of these results have been studied since 1973 when Johnson [31] analyzed the worst-case behavior of simple, polynomial-time, random algorithms for finding approximate solutions to various combinatorial optimization problems. He measured the worst solution value ratio, which can be reached by an algorithm, to the optimal one. For a maximization problem, an algorithm is a ρ-approximation algorithm, $\rho < 1$, if it produces a solution whose objective value is at least $\rho \cdot OPT$ where OPT is the global optimum, for each instance. A similar definition applies to minimization problems. The approximation algorithms' important property relates the size of the solution produced directly to a lower bound on the optimal solution. Instead of telling us how well we might do, they will tell us about the worst-case, i.e. how badly we might perform.

Following this research topic, many computer scientists have proven rigorous results over optimal inapproximability. The first result proving hardness for the problem we are discussing here was obtained in the fundamental paper by Arora *et al* [33]. He established the probabilistically checkable proofs (PCP) theorem. The theorem states that every decision problem in the NP complexity class has probabilistically checkable proofs, where a verifier reads only a constant number of bits and uses logarithmic random of bits. Many results have been obtained since then, as in [34–36] and references therein, but the most famous was given by Håstad in 1997 [7]. He proved optimal inapproximability results, up to an arbitrary $\epsilon > 0$, for MAX-E-k-SAT for $k \geqslant 3$, by maximizing the number of satisfied linear equations in an over-determined system of linear equations modulo a prime p. More precisely, the author showed that for the maximum exact 3-satisfiability (MAX-E-3-SAT), no approximate algorithm could outperform the random assignment threshold, which is set to be 7/8 the optimal one, unless $P = NP$. He also stated that the maximum exact 4-satisfiability (MAX-E-4-SAT) is not approximable beyond the random assignment threshold on satisfiable instances. The random assignment threshold is set to be 15/16 the optimal one, unless $P = NP$. These results are only valid for approximate algorithms.

In contrast to this kind of algorithms, heuristics [32, 37] can find better approximate solutions. However, their worst-case performance can be tough to analyze and, therefore, heuristic methods may be considered approximate and not accurate algorithms. Although they have this negative reputation, heuristics are the only viable option for various optimization problems that need to be routinely solved in real-world applications.

Heuristic algorithms for solving MAX-SAT problems can be roughly classified into two main categories. The first category of algorithms searches for a solution by performing a biased random walk in the space of configurations. Instead, the second one tries to build a solution assigning variables, according to some estimated marginals. MaxWalkSAT, focused Metropolis search or diffusion Monte Carlo algorithms, genetic algorithms with local search, and many other SAT-Solvers belong to the former category [38–55]. In contrast, in the second category, we find algorithms deriving from the class of message passing algorithms. Examples are warning propagation, belief propagation guided decimation, survey inspired decimation (SID), backtracking survey propagation (BSP), and SP-y, a generalization of SP algorithm [56–60]. The last algorithms use a decimation move for building a solution to the problem. This decimation procedure modifies the underlying graph's structure and makes these algorithms hard to be analytically analyzable.

This paper introduces a new heuristic-learning algorithm that collects local information from the SP algorithm, avoids any decimation strategy, and fixes the local variables by using supervised learning methodology [61].

The supervised learning methodology is part of a broader family of machine learning methods and plays a key role in the fields of statistics, data mining and artificial intelligence, with many useful applications [62–66]. In this paper we are interested in learning patterns from the solutions of MAX-E-3-SAT for understanding how to fix a single Boolean variable.

In combinatorial optimization, learning methodology has been used for solving SAT problems or problems related to graphical models. For instance, Selsam *et al* [67] obtained exciting results for SAT problems. More precisely, the authors presented a message passing neural network that learns to solve SAT problems after only being trained as a classifier to predict SAT. Instead, for problems related to graphical models, Dai *et al* [68] presented a learning algorithm that uses a unique combination of reinforcement learning and graph embedding. Many other results have been reached during this period, and we refer to an interesting survey on machine learning for combinatorial optimization of Bengio *et al* [69] and references therein. Regarding the MAX-SAT Problem, instead, an exciting work based on statistical learning theory has been presented recently in [70]. They introduce a novel setting for learning combinatorial optimization problems from contextual examples.

These works deal with small instances because the neural networks are fed with the whole instance of the problem. In this paper, in contrast, we present, for the first time, as far as we know, a linear algorithm that takes information locally on the graph and uses a multi-layers neural network for assigning a single variable. This strategy allows us to analyze CNF formulae composed of 10^6 variables, larger and more difficult than the neural network ever saw during training.

The motivation that guides us in simplifying heuristic methods is the following: although making a heuristic more complicated does not necessarily make it better in the worst-case, maybe making it simpler does not necessarily make it worse in the worst-case. The worst-case scenario for heuristic message passing algorithms is due to the fact that a full convergence of the messages is not found. This can appear even for a single message. When such a scenario appears, we cannot say anything about the instance we are looking at. We lose all the local information that is correctly obtained by the message passing procedure.

For this reason, we deal with SP equations. Although they are not the best for the MAX-SAT problem, in contrast to the SP-y equations, they are simpler and much more suitable for meeting the worst-case scenario. They may make this heuristic-learning algorithm analytically analyzable. Indeed, decimation and backtracking moves are avoided.

This paper is divided into the following sections: the first one recalls the MAX-E-3-SAT problem and its factor graph representation; the second one recalls the SP algorithm; the third one introduces the neural network and also presents the numerical analysis. We conclude our manuscript with a discussion on the future research directions that this new method gives rise to.
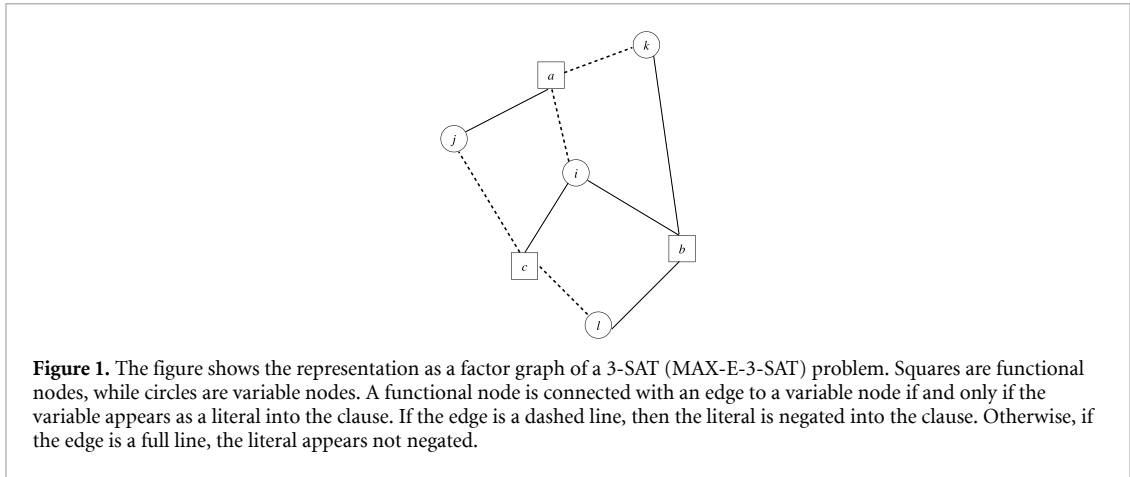
2. The MAX-E-3-SAT problem and its factor graph representation

As explained in the section 1, the MAX-SAT is the maximization problem associated with the SAT problem. It is asked to find an assignment of the Boolean variables such that the maximum number of clauses, in a CNF formula, is satisfied. The maximization version of the k-SAT problem is called MAX-E-k-SAT, in this case, each clause contains exactly k literals. With $k = 3$ a 3-SAT problem with $N = 9$ variables and $M = 4$ clauses is of the form:

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4 \vee x_5) \wedge (\bar{x}_2 \vee x_6 \vee x_7) \wedge (\bar{x}_3 \vee x_8 \vee x_9). \tag{2}$$

Clearly an assignment that satisfies all the clauses is $\text{Sol}_{3\text{-}SAT} : (x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0, x_6 = 1, x_7 = 0, x_8 = 1, x_9 = 0)$. The MAX-E-3-SAT looks for an approximate solution of the problem in (2). For example, a solution of the MAX-E-3-SAT, outputted by a random algorithm, could be $\text{Sol}_{MAX\text{-}E\text{-}3\text{-}SAT} : (x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0, x_6 = 1, x_7 = 0, x_8 = 1, x_9 = 0)$. $\text{Sol}_{MAX\text{-}E\text{-}3\text{-}SAT}$ does not satisfy all the clauses in (2), but it is an approximate solution where just a clause is unsatisfied, i.e. $(x_1 \vee x_2 \vee x_3) = FALSE$. For a general instance of the MAX-E-3-SAT problem that contains N variables and M clauses, it is easy to see that a random assignment satisfies each clause with probability 7/8. Indeed, if there are M clauses, it is not hard to find an assignment that satisfies $7M/8$ clauses. Since we can never satisfy more than all the clauses this gives a 7/8-approximation algorithm [7, 31]. In this paper, as stated in the section 1, we are interested in heuristic methods that use message passing procedure. For this reason, we recall the factor graph representation of SAT problems. The SAT problem, and thus the MAX-SAT, is represented as a factor graph where clauses are identified as functional nodes and variables as variable nodes. A factor graph is a bipartite graph representing the factorization of a function.

In figure 1, it is shown a cartoon of the factor graph associated with a 3-SAT (MAX-E-3-SAT) problem. The variables, i, j, l and k, are variable nodes (circles) while clauses, a, b and c, are functional nodes (squares). Each variable enters a clause as a literal, e.g. p_{ci}, if and only if it is connected with an edge. Dashed edges identify literals where variables are negated, while full edges identify literals where variables are not negated.

Figure 1. The figure shows the representation as a factor graph of a 3-SAT (MAX-E-3-SAT) problem. Squares are functional nodes, while circles are variable nodes. A functional node is connected with an edge to a variable node if and only if the variable appears as a literal into the clause. If the edge is a dashed line, then the literal is negated into the clause. Otherwise, if the edge is a full line, the literal appears not negated.

With the symbol ∂_a, we define the set of variables nodes that are connected with the functional node a, i.e. the literals of clause a. In contrast, with the symbol ∂_i, we define the set of functional nodes connected with the variable node i, i.e. the set of clauses where the literal indexed i appears. The cardinality of the set ∂_i is the degree of a variable node i, i.e. the number of links connected to a circle, and is defined with n_i. The set ∂_i is also composed of two other sets, namely ∂_i^+ that contains the functional nodes where the variable node i appears not negated, and ∂_i^- that contains the functional nodes where the variable node i appears negated. Obviously, the relation $\partial_i = \partial_i^+ \cup \partial_i^-$ holds. We also defined two more quantities, $n_i^\pm$, for the number of dashed and full edges of a variable node i. More precisely, n_i^+ defines the cardinality of the set ∂_i^+, while n_i^- defines the cardinality of the set ∂_i^-.

With the symbol ∂_{ia}^+ (respectively ∂_{ia}^-) we define the set of functional nodes containing the variable node i, excluding the functional node a itself, satisfied (respectively not satisfied) when the variable x_i is assigned to satisfy clause a. In other words, if the variable x_i is not negated in the clause a, then the ∂_{ia}^+ is the set of functional nodes containing the variable node i, excluding the functional node a itself, where the variable node i is connected with a full edge, thus where the variable x_i appears not negated, while ∂_{ia}^- is the set of functional nodes containing the variable node i, where the variable node is connected with dashed edges, thus where the variable $\bar{x}_i$ appears negated. In contrast, if the variable $\bar{x}_i$ is negated in the clause a, then the ∂_{ia}^+ is the set of functional nodes containing the variable node i, excluding the functional node a itself, where the variable node i is connected with a dashed edge, thus where the variable $\bar{x}_i$ appears negated, while ∂_{ia}^- is the set of functional nodes containing the variable node i, where the variable node is connected with full edges, thus where the variable x_i appears not negated.

3. The SP algorithm

The SP algorithm is a heuristic message passing algorithm. A detailed description of the SP algorithm can be found in [57, 58], here we recall it naively. Mezard, Parisi, and Zecchina developed it in [57] from the assumption of one-step replica symmetry breaking and the cavity method of spin glasses. SP has been applied to different combinatorial optimization problems, like random K-SAT, MAX-E-k-SAT, q-coloring, Maximum Independent Set, etc [57–59, 71–76], always showing the best performance for solving these problems. It works on a factor graph. For $N \to \infty$, SP is conjectured to work better and better because it runs over locally-tree like factor graphs, and cycles into the graph are at least O(log N).

Broadly speaking, SP exchanges messages between variables and clauses for guessing the value that each variable needs to be set. More precisely, a message of SP, called a survey, passed from one function node a to a variable node i (connected by an edge) is a real number $\eta_{a \to i} \in [0, 1]$. Under the assumption that SP runs over a tree-like factor graph, the messages have a full probabilistic interpretation. In particular, the message $\eta_{a \to i}$ corresponds to the probability that the clause a sends a warning to variable i, telling which value the variable i should adopt to satisfy itself [77].

The updating rules of a single message $\eta_{a \to i}$ are presented in algorithm 1.

SP is a local algorithm that extracts information on the underlying graph of a CNF formula. As input, it takes a CNF formula of a Boolean SAT Problem, and it performs a message passing procedure to obtain convergence of the messages. More precisely, we are given a random initialization of all messages, and at each iteration, each message is updated following the SP-UPDATE rule described in algorithm 1. SP runs until all messages would satisfy a convergence criterion. This convergence criterion is defined as a small number ε such that the iteration is halted at the first time t^* when no message has changed by more than ε over the last

---

Algorithm 1: Subroutine SP-UPDATE($\eta_{a\to i}$)

Input: set of all messages arriving onto each variable node $j \in \partial_a \setminus i$

Output: new value for the message $\eta_{a\to i}$.

for $j \in \partial_a \setminus i$ **do**

$\quad s_{j\to a}^- = \left[1 - \prod_{b\in\partial_{ja}^-}(1-\eta_{b\to j})\right] \prod_{b\in\partial_{ja}^+}(1-\eta_{b\to j})$

$\quad s_{j\to a}^+ = \left[1 - \prod_{b\in\partial_{ja}^+}(1-\eta_{b\to j})\right] \prod_{b\in\partial_{ja}^-}(1-\eta_{b\to j})]$

$\quad s_{j\to a}^0 = \left[\prod_{b\in\partial_j\setminus a}(1-\eta_{b\to j})\right]$

$\quad$ if a set $\partial_{ja}^{\pm}$ is empty, the corresponding product takes value 1 by definition

end

return $\eta_{a\to i} = \prod_{j\in\partial_a\setminus i}\left[\dfrac{s_{j\to a}^-}{s_{j\to a}^- + s_{j\to a}^+ + s_{j\to a}^0}\right]$

---

iteration. If this convergence criterion is not satisfied after t_{max} iterations, SP stops and returns a failure output. Once a convergence of all messages $\eta_{a\to i}$ is found, SP's goal is to minimize the number of violated clauses. For doing that, a new strategy for fixing the value of the variables must be introduced. This strategy is called decimation and transforms the SP into the SID Algorithm. For using decimation, however, one needs to compute the SP marginals for each variable i:

$$S_i^- = \frac{\pi_i^-(1-\pi_i^+)}{1-\pi_i^+\pi_i^-}, \qquad S_i^+ = \frac{\pi_i^+(1-\pi_i^-)}{1-\pi_i^+\pi_i^-}, \qquad S_i^0 = 1 - S_i^- - S_i^+, \tag{3}$$

where:

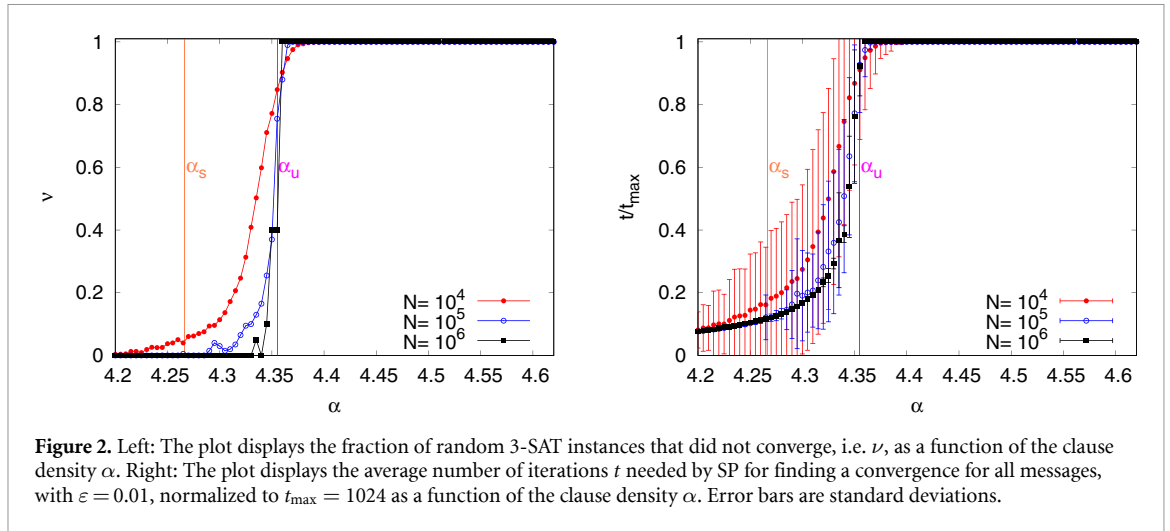$$\pi_i^{\pm} = 1 - \prod_{b\in\partial_i^{\pm}}(1-\eta_{b\to i}).$$

The SP marginal S_i^+ (S_i^-) tells the probability that the variable i must be forced to take the value $x_i = 1(x_i = 0)$, conditional on the fact that it does not receive a contradictory message, while S_i^0 provides the information that the variable i is not forced to take a particular value.

Once all the SP marginals have been computed, the decimation strategy can be applied. Decimating a variable node i means fixing the variable to *TRUE* or *FALSE* depending on the SP marginals, removing all satisfied functional nodes and the variable node i from the factor graph, and removing all the literals into the clauses that have not been satisfied by the fixing. However, how to choose the variable node i to decimate? The answer is simple, just selecting a variable with the maximum bias $S_{C_i} = 1 - \min(S_i^-, S_i^+)$. Decimated the variable node i, the SID iteratively runs the SP algorithm and uses decimation again. The decimation procedure continues till one of these three different outcomes appears: (i) a contradiction is found, then SID returns exit failure; (ii) SP does not find a convergence, then SID returns exit failure; (iii) all the messages converge to a trivial fixed point, i.e. all the messages are equal to 0, in this case, SID calls WalkSAT, which solves the residual formula and builds the complete solution of the problem.

The SID has extremely low complexity. Each SP iteration requires $O(N)$ operations, which yields $O(Nt_{max})$, where t_{max} is the maximum time allowed for finding a convergence, i.e. a big constant. In the implementation described above, the SID has a computational complexity of $O(t_{max}N^2 \log N)$, where the $N\log N$ comes from the sorting of the biases. This can be reduced to $O(Nt_{max}(\log N)^2)$ by noticing that fixing a single variable does not affect the SP messages significantly. Consequently, SP can be called every $N\delta$ decimation step by fixing a fraction of variables at each decimation step. The efficiency of SID can be improved by introducing a backtracking strategy or a reinforcement strategy. We refer to [58], and references therein for a complete explanation of these strategies.

4. The neural network and a new heuristic-learning algorithm

This manuscript aims to present a new heuristic-learning algorithm that can find an assignment for a set of Boolean variables that maximizes the number of satisfied clauses of a given CNF formula. Although this new heuristic-learning algorithm does not reach state-of-the-art algorithms for the MAX-SAT problem, it can solve substantially larger and more difficult problems than it ever saw during training. The code was developed in C++ using mlpack, a fast and flexible C++ machine learning library [78]. The experiments were performed on a cluster with 128 cores and 512 GB of RAM. The code, the training dataset, and the validation/test dataset can be downloaded from [79]. The experimental results were obtained by analyzing

Figure 2. Left: The plot displays the fraction of random 3-SAT instances that did not converge, i.e. ν, as a function of the clause density α. Right: The plot displays the average number of iterations t needed by SP for finding a convergence for all messages, with $\varepsilon = 0.01$, normalized to $t_{max} = 1024$ as a function of the clause density α. Error bars are standard deviations.

10^3 instances for $N = 10^4$, 10^2 instances for $N = 10^5$, and 10 instances for $N = 10^6$. In all figures, they will be indicated with red circle points, blue empty circle points, and black square points respectively.
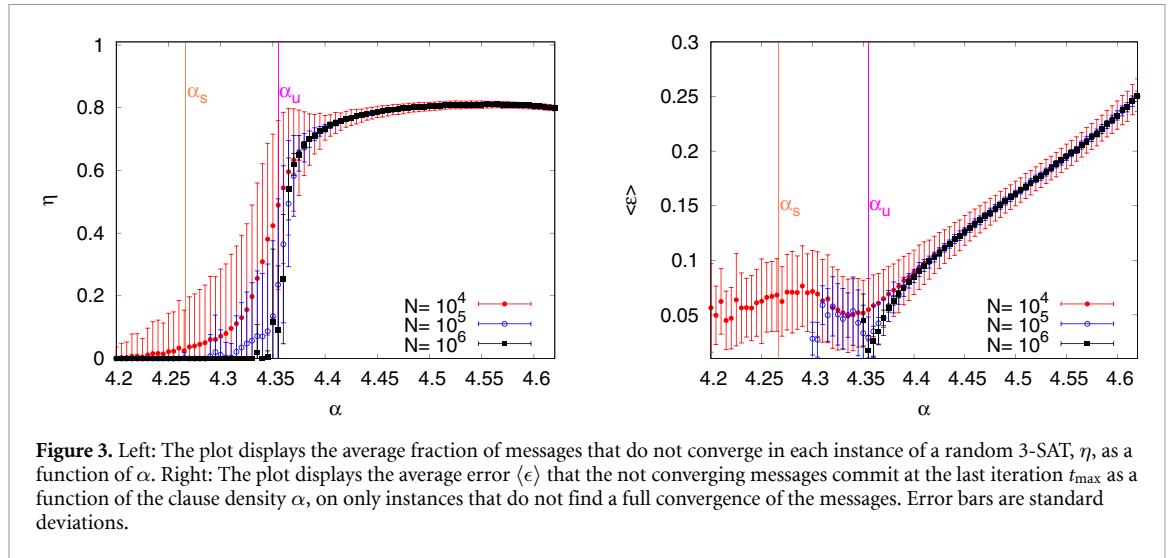
4.1. Empirical analysis of SP equations

Before presenting the whole algorithm, we start to analyze the SP algorithm. It is known that the SP algorithm collects information locally and by using equations in (3) predicts the marginal probabilities that allow fixing a Boolean variable. This information, however, can be achieved only by a full convergence of the messages. Without a full convergence of the messages, algorithms based on SP equations always return a failure output. Therefore, for understanding the limits of SP, we perform an accurate analysis on a set of random 3-SAT instances for different values of the clause density (the ratio of the number of clauses to the number of variables) $\alpha = M/N$, with large M, large N, and keeping constant α. Instances are generated by considering N variables and $M = \alpha N$ clauses, where each clause contains exactly $k = 3$ distinct variables, and is picked up with uniform probability distribution from the set of $N!(k!(N-k)!)^{-1} 2^k$ possible clauses [80].

We choose random 3-SAT instances for two reasons. The first one is just for the sake of simplicity. We use the same instances to analyze the new algorithm's performance in approximating the solutions of the MAX-E-3-SAT problem associated with them. The second one, instead, is given by the fact that many theoretical results are well known. For example, it is known that the SAT-UNSAT threshold for the random 3-SAT is at $\alpha_s = 4.267$ (vertical coral line) [72, 81]. The SAT-UNSAT threshold defines two regions sharply: for $N \to \infty$, the region before the threshold contains all the instances of random k-SAT problems that have at least an assignment that satisfies all the clauses (SAT region), while beyond the threshold, no assignment that satisfies all the clauses exists (UNSAT region). It is also known that when $\alpha = \alpha_U \approx 4.36$, the SP equations do not have a unique solution. This fact is not of direct importance for the random 3-SAT problem because we are beyond the SAT-UNSAT threshold. No exact solution exists, i.e. not all the clauses of an instance can be satisfied simultaneously. However, for the MAX-E-3-SAT problem, this point is interesting. Indeed, from there, we expect that SP equations will not converge, and therefore the worst-case scenario for the SP algorithm appears.

We start analyzing the empirical convergence of the SP algorithm (initialized with uniformly random messages) as a function of the clause density α, for different values of N. We fix, as described in the previous section, the value of t_{max} to 1024 and ε to 10^{-2}.

In figure 2, we plot, for random 3-SAT, the fraction of instances that did not converge, ν, (left panel), and the number of iterations t that SP needs to make for reaching a convergence of all messages (right panel), as a function of the clause density α. The analysis shows that the SP algorithm returns a failure output for random 3-SAT at $\alpha_{3\text{-}SAT}^{conv} = 4.355$ because a full convergence is not found. In both cases, we observe a step function form of the fraction of instances that did not converge for $N \to \infty$. For $N \to \infty$, therefore, SP always converges before the specific value of the clause density $\alpha_{3\text{-}SAT}^{conv}$, because the solutions of the SP equations are unique, while does not converge beyond the $\alpha_{3\text{-}SAT}^{conv}$, because the SP equations have many solutions [72]. This property shows that no algorithm, based on SP equations presented in Algorithm 1, can build any solution beyond $\alpha_U = \alpha_{3\text{-}SAT}^{conv}$ (vertical magenta line). Analyzing the average fraction of messages that do not converge, η, as a function of α, however, it seems that beyond the convergence threshold $\alpha_{3\text{-}SAT}^{conv}$, a fraction of messages always converges. More precisely, a fraction of converging messages, which is almost $\sim 20\%$ of the messages in each random 3-SAT instance (see figure 3, left panel), exists. This fact inspires us to look at the average

Figure 3. Left: The plot displays the average fraction of messages that do not converge in each instance of a random 3-SAT, η, as a function of α. Right: The plot displays the average error $\langle \epsilon \rangle$ that the not converging messages commit at the last iteration $t_{\max}$ as a function of the clause density α, on only instances that do not find a full convergence of the messages. Error bars are standard deviations.

error of convergence. We define the average error of convergence as the quantity $\langle \epsilon \rangle$ such that:

$$\langle \epsilon \rangle = \frac{1}{\mathcal{M}} \sum_{s=1}^{\mathcal{M}} \frac{1}{kM - n_s^{conv}} \sum_{i=1}^{N} \sum_{b \in \partial_i} \Delta_{b \to i}(t_{\max}), \tag{5}$$

where $k = 3$, because we have three messages for each clause, n_s^{conv} is the number of messages that converge in the sth instance over the $\mathcal{M}$ analyzed, and

$$\Delta_{b \to i}(t_{\max}) = \begin{cases} |\eta_{b \to i}^{t_{\max}} - \eta_{b \to i}^{t_{\max}-1}| & \text{for} |\eta_{b \to i}^{t_{\max}} - \eta_{b \to i}^{t_{\max}-1}| \geqslant \epsilon \\ 0 & \text{for} |\eta_{b \to i}^{t_{\max}} - \eta_{b \to i}^{t_{\max}-1}| < \epsilon. \end{cases} \tag{6}$$

In the case where $n_s^{conv} = kM$, we define that $\frac{1}{kM - n_s^{conv}} \sum_{i=1}^{N} \sum_{b \in \partial_i} \Delta_{b \to i}(t_{\max}) = 0$.

In the right panel of figure 3, we plot the quantity $\langle \epsilon \rangle$ as a function of the clause density α. For $N \to \infty$, the probability of finding a full convergence in the region $\alpha \in [4.200, 4.355)$ is equal to one because SP equations run over a factor graph that is locally tree-like. When N is finite and small, for instance, $N = 10^4$, the property of having a tree-like structure is not always preserved, and, therefore, we can meet before the threshold $\alpha_{3\text{-}SAT}^{conv}$, instances of the random 3-SAT problem that are not able to find a full convergence of the messages. This property allows us to understand the worst-case scenario of the SP algorithm in this region. Therefore, we analyzed a set of instances, with cardinality $2\,10^3$, with $N = 10^4$, and we looked at the instances where the average error of convergence was not trivial, i.e. different from 0. The plot shows that in the region $\alpha \in [4.200, 4.355)$ the average error of convergence $\langle \epsilon \rangle$ is bounded, i.e. $\langle \epsilon \rangle \leqslant 0.15$, while beyond the threshold $\alpha_{3\text{-}SAT}^{conv}$ the quantity $\langle \epsilon \rangle$ grows linearly with α. This fact suggests that some information can also be extracted by those messages, although it is not completely correct. However, if we use the local information obtained by those messages, could we find an assignment of the Boolean variables better than a random one? To answer this question, we create a simple neural network described in the next subsection.

4.2. The neural network and numerical analysis

Algorithms based on supervised learning methodology have become essential in a wide variety of scientific disciplines. The prime example is deep learning. Deep learning is a class of machine learning algorithms that uses multiple layers to extract higher-level features from the raw input. It is based on artificial neural networks (ANNs), a series of functional transformations. These functional transformations can be obtained by fixing a set of basis functions in advance and allowing them to be adaptive during training. In this manuscript, we use an ANN for fixing a single Boolean variable by building a function learned by solutions coming from SID Algorithm.

In our case, the ANN is a feed-forward neural network that is trained as a classifier to predict the conditional probability that a variable i must be set to *TRUE* or not, given a piece of local information expressed into a vector of input data $\vec{\mathbf{x}}(i)$. This vector $\vec{\mathbf{x}}(i)$ has four dimensions and it is composed by $\vec{\mathbf{x}}(i) = [1 - \pi_i^+, 1 - \pi_i^-, n_i^+, n_i^-]^T$. $1 - \pi_i^{\mp}$, under the assumption that the factor graph is locally tree-like, may be interpreted as the probability that the variable i does not receive warnings from the set of clauses where it

appears negated (n^-) or not negated (n^+). The components of the vector $\vec{\mathbf{x}}(i)$ could be interpreted as the features used for feeding a neural network in the general framework of machine learning.

The neural network $\mathcal{NN}(\vec{\mathbf{x}}(i), \vec{\theta})$ has five layers. The input and output layers have four and one neuron respectively, where sigmoidal activation function acts element-wise on each neuron, i.e. $\sigma(a) = (1 + \exp(-a))^{-1}$. The hidden layers are sigmoidal layers composed by 40 neurons each. The total set of parameters to be optimized is defined with $\vec{\theta}$. We define as loss function the following cross-entropy error function:

$$E(\vec{\theta}) = -\sum_{i=1}^{\mathcal{N}} \{ \mathcal{T}_i \ln y_i(\vec{\mathbf{x}}(i), \vec{\theta}) + (1 - \mathcal{T}_i) \ln(1 - y_i(\vec{\mathbf{x}}(i), \vec{\theta})) \}, \tag{7}$$

where $\mathcal{T}$ is the target variable, and $\mathcal{N}$ is the total batch size. $y(\vec{\mathbf{x}}(i), \vec{\theta})$ is a sigmoidal function of the output layer. In our case, the target variables $\mathcal{T}_i$ are the variables into a satisfiable assignment of a random 3-SAT problem. We chose the cross-entropy error function as a loss function because we are dealing with a classification problem. Classification is the problem of identifying to which of a set of categories a new observation belongs, based on a training set of data containing observations whose category membership is known. In our case, the training set is composed by vectors $\vec{\mathbf{x}}(i)$ and targets that are Boolean variables. We can assume that each of these targets has a Bernoulli distribution. Considering them composed by independent observations, the loss function that arises naturally by taking the negative log-likelihood is the cross-entropy error function.

The output $y(\vec{\mathbf{x}}(i), \vec{\theta})$ can be interpreted as the conditional probability $p(x_i = 1 | \vec{\mathbf{x}}(i))$, with $p(x_i = 0 | \vec{\mathbf{x}}(i))$ given by $1 - y(\vec{\mathbf{x}}(i), \vec{\theta})$, where $\vec{\theta}$ is the set of parameters that has to be optimized.

For optimizing these parameters, we need to train our neural network. For doing that, we solved 400 random 3-SAT problems at $\alpha = 4.2$ and $N = 10^4$ using SID (see section 3). For each of this 400 instances we stored one solution and the $\vec{\mathbf{x}}(i)$ for each variable i. In other words, for each instance random 3-SAT we have 10^4 vectors $\vec{\mathbf{x}}(i)$, and to each of these vectors is associated the target variable $\mathcal{T}_i$, which is the Boolean variable associated with the satisfiable assignment. The training of the neural network was performed by giving a batch of 20 random vectors $\vec{\mathbf{x}}(i)$ and the respective target variables $\mathcal{T}_i$, without replacement, to the neural network. The optimal assignment of $\vec{\theta}$, i.e. $\vec{\theta}^*$, to which the right-hand side of equation (7) vanishes, can be found by running an SGD algorithm.

For our simulations, we used the default SGD (Adam [82]) given by mlpack [78]. For testing the performance of the neural network, we calculated the AC in computing the conditional probability $p(x_i = 1 | \vec{\mathbf{x}}(i))$ on a validation data set. The validation data set was obtained by solving 36 random 3-SAT problems at $\alpha = 4.23$ and $N = 10^4$ using SID. We define the AC of the neural network as:
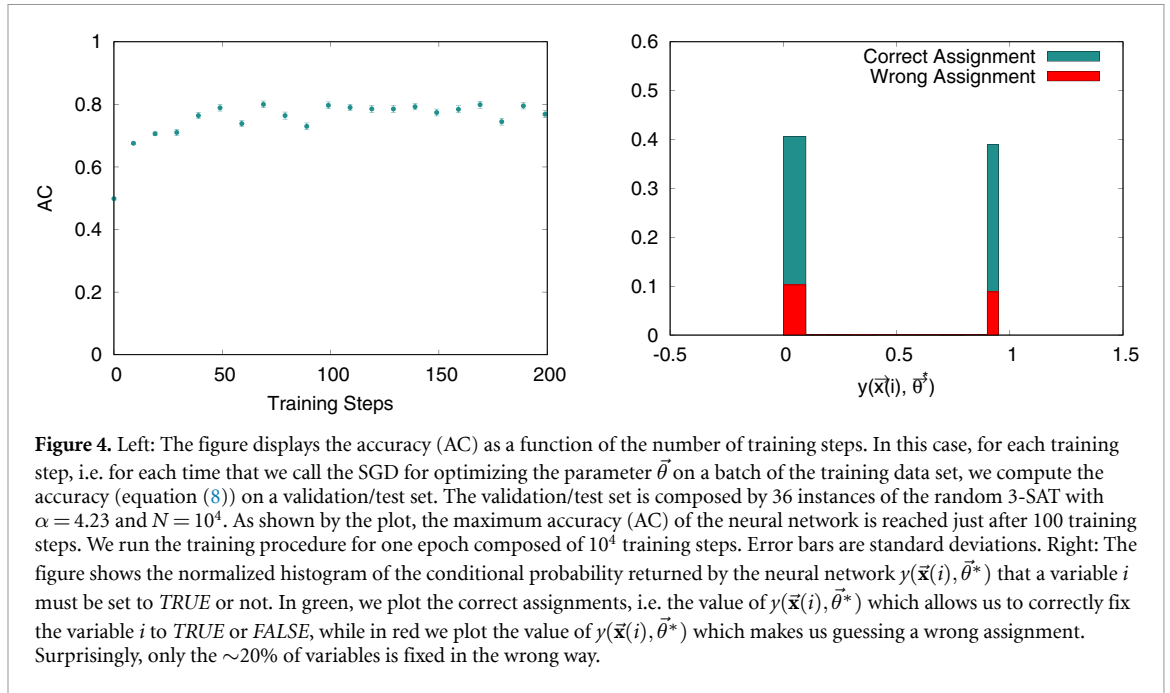
$$AC = \frac{1}{\mathcal{M}} \sum_{s=1}^{\mathcal{M}} \mathcal{NH}(\mathcal{S}_s, \mathcal{Y}_s), \tag{8}$$

where, $\mathcal{M}$ is the total number of test solutions, i.e. the 36 solutions of random 3-SAT problems at $\alpha = 4.23$ and $N = 10^4$; $\mathcal{NH}(a, b)$ is the normalized Hamming distance between two strings a and b with the same length; $\mathcal{S}_s$ is the exact solution obtained by SID; $\mathcal{Y}_s$ is the approximate solution obtained from the deep neural network. As the reader can see, we tested the network on solutions with a different value of α to which the neural network was trained. It was possible because we assumed that the local information obtained by SP equations should be independent of the clause density α.

In the left panel of figure 4, we present the AC of the neural network on the validation data set of $36 \, 10^4$ elements as a function of the number of training steps, i.e. the number of times that we called the SGD for optimizing the set of parameters $\vec{\theta}$. The neural network starts by giving a random assignment to the Boolean variables, AC $\sim$50%, and after 100 training steps, it learns how to assign the Boolean variables. We also tested the AC of the neural network on a validation data set of $17 \, 10^4$ elements coming from 17 solutions of 3-SAT at $\alpha = 4.24$ and $N = 10^4$, obtaining the same AC. However, as shown on the right panel of figure 4, the approximation of the conditional probability $y(\vec{\mathbf{x}}(i), \vec{\theta}^*)$ completely fails for $\sim$20% of the variables. This failure is not bad for our purpose. Indeed, we are not interested in building a solution that satisfies all the clauses. Still, we are interested in finding an approximation of a solution that minimizes the number of unsatisfied clauses into an instance of MAX-E-3-SAT.

The whole algorithm, which we name **NeurNet-SP**, is presented in algorithm 2.

For speeding up the algorithm, we introduced the convergence criterion explained in section 3. The computational complexity of **NeurNet-SP** algorithm is, therefore, $\Theta(N)$. Indeed, the maximum number of operation that it takes for outputting a result, after the training procedure for optimizing the parameters $\vec{\theta}$ of the deep neural network, is $\sim t_{\max} k M + O(|\vec{\theta}|^2) N$, where $|\vec{\theta}| \ll N$ and $M = \alpha N$. Moreover, once the neural
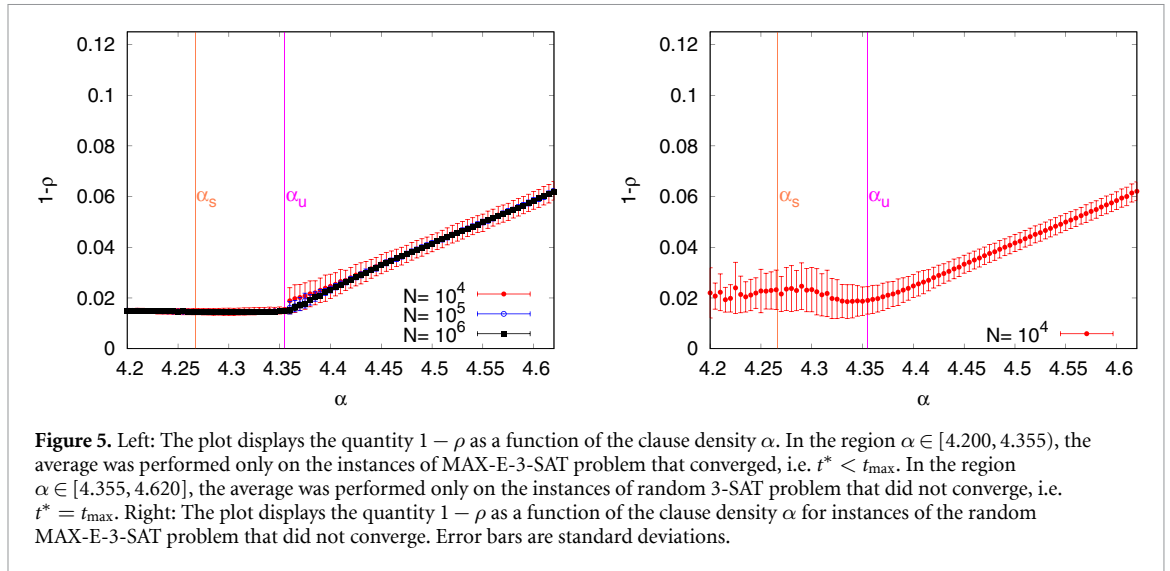
Figure 4. Left: The figure displays the accuracy (AC) as a function of the number of training steps. In this case, for each training step, i.e. for each time that we call the SGD for optimizing the parameter $\vec{\theta}$ on a batch of the training data set, we compute the accuracy (equation (8)) on a validation/test set. The validation/test set is composed by 36 instances of the random 3-SAT with $\alpha = 4.23$ and $N = 10^4$. As shown by the plot, the maximum accuracy (AC) of the neural network is reached just after 100 training steps. We run the training procedure for one epoch composed of 10^4 training steps. Error bars are standard deviations. Right: The figure shows the normalized histogram of the conditional probability returned by the neural network $y(\vec{\mathbf{x}}(i), \vec{\theta}^*)$ that a variable i must be set to *TRUE* or not. In green, we plot the correct assignments, i.e. the value of $y(\vec{\mathbf{x}}(i), \vec{\theta}^*)$ which allows us to correctly fix the variable i to *TRUE* or *FALSE*, while in red we plot the value of $y(\vec{\mathbf{x}}(i), \vec{\theta}^*)$ which makes us guessing a wrong assignment. Surprisingly, only the $\sim$20% of variables is fixed in the wrong way.

Algorithm 2: NeurNet-SP

Input: A CNF formula for MAX-E-3-SAT.
Output: An assignment $\text{Sol}_{MAX-E-3-SAT}$ for MAX-E-3-SAT.
Learn the parameters $\vec{\theta}^*$ of the neural network $\mathcal{NN}(\vec{\mathbf{x}}(i), \vec{\theta})$
Random initialization of all messages $\eta_{a \to i}$ in the clauses.
for $t < t_{max}$ **do**
     RUN SP on the factor graph underlying the CNF formula, i.e.
     Algorithm 1
end
for $1 \leqslant i \leqslant N$ **do**
     Compute $\vec{\mathbf{x}}(i)$
     Use the output of $\mathcal{NN}(\vec{\mathbf{x}}(i), \vec{\theta}^*)$ for fixing the variable i, i.e. if
     $y(\vec{\mathbf{x}}(i), \vec{\theta}) \geqslant 0.5$ then i is set to 1, else i is set to 0
     Save the value of i into $\text{Sol}_{MAX-E-3-SAT}$
end
return $\text{Sol}_{MAX-E-3-SAT}$

---

network's training procedure is performed, one can save the parameters' value and upload them instead of re-training the neural network each time. We also release the parameters' value, which can be downloaded from [79].

For performing an analysis on the performance of this heuristic-learning algorithm, we need, therefore, to check the ratio of the number of satisfied clauses to the total number of clauses, i.e. ρ. This result is described in figure 5. Both plots describe the quantity of $1 - \rho$ as a function of α. In each plot, having $1 - \rho$ on the y-axis, the top end of the plot coincides with the estimate of the random assignment threshold, i.e. $1 - \rho_{\text{rand}} = 1 - 7/8 = 0.125$, to provide an immediate indication of the performance of the **NeurNet-SP**. In the left panel of the figure 5, in the region $\alpha \in [4.200, 4.355)$ the average of $1 - \rho$ is performed only on the instances of random 3-SAT problem that converged, i.e. $t^* < t_{\max}$. In other words, we are looking at the average-case performance of the heuristic-learning algorithm.

The behavior of $1 - \rho$ is constant, showing, therefore, that the **NeurNet-SP** algorithm can find approximate solutions such that only 1.46% of clauses are unsatisfied by the assignment found, more precisely $(1 - \rho) = 0.0146 \pm 0.0002$. In the region $\alpha \in [4.355, 4.620]$ the average was performed only on the instances of random 3-SAT problem that did not converge, i.e. $t^* = t_{\max}$. This is obvious because no convergence is possible beyond α_{3-SAT}^{conv}. The behavior of $1 - \rho$, in this case, is not constant anymore, but it is linear with the clause density α.

Figure 5. Left: The plot displays the quantity $1 - \rho$ as a function of the clause density α. In the region $\alpha \in [4.200, 4.355)$, the average was performed only on the instances of MAX-E-3-SAT problem that converged, i.e. $t^* < t_{\max}$. In the region $\alpha \in [4.355, 4.620]$, the average was performed only on the instances of random 3-SAT problem that did not converge, i.e. $t^* = t_{\max}$. Right: The plot displays the quantity $1 - \rho$ as a function of the clause density α for instances of the random MAX-E-3-SAT problem that did not converge. Error bars are standard deviations.

In the right panel, we plot, instead, the worst-case scenario for the SP equations, i.e. only the instances that did not find a convergence of all messages, i.e. the time $t^* = t_{\max}$ and $\langle \epsilon \rangle \neq 0$. For showing the worst-case scenario, we run $2 \, 10^3$ the heuristic-learning algorithm on instances random 3-SAT with $N = 10^4$, and we analyzed only those where a full convergence of the messages was absent. We observe that the local nature of the SP algorithm helps us to find an approximate solution much better than the one outputted by the Johnson algorithm [31]. In the region $\alpha \in [4.200, 4.355)$ the average error of the convergence, i.e $\langle \epsilon \rangle$, in the right panel of figure 3 is bounded and the **NeurNet-SP** seems to follow the same behavior. This behavior shows that the solutions we found using **NeurNet-SP** are not affected by the loss of convergence. In contrast, in the region $\alpha \in [4.360, 4.620]$ the algorithm, following the behavior of the average error of convergence $\langle \epsilon \rangle$ defined in (5), is affected to the linear growth, and, therefore, the behavior of the quantity $1 - \rho$ grows linearly with α. As test, we computed the sample Pearson correlation between two sets of variables:

$$r_{\text{corr}} = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - \mu_x)^2 \, \sum_{i=1}^{n}(y_i - \mu_y)^2}}, \tag{9}$$

where n is the sample size, x_i, y_i are the individual sample points indexed with i, and $\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i$ the sample mean (and analogously for μ_y), between the set of data $1 - \rho$ and $\langle \epsilon \rangle$. The sample Pearson correlation is equal to $r_{\text{corr}} = 0.9959$, confirming that the two quantities are dependent on each other.

As stated in section 1, **NeurNet-SP** is not competitive with state-of-the-art of MAX-SAT solvers. For showing this, we compared our results with the results obtained by two different established methods: MaxWalkSat [80], and SP-y [59]. MaxWalkSat searches for a solution by performing a biased random walk in the solution space. At the same time, SP-y is a message passing algorithm that tries to build a solution assigning variables according to some estimated marginals.
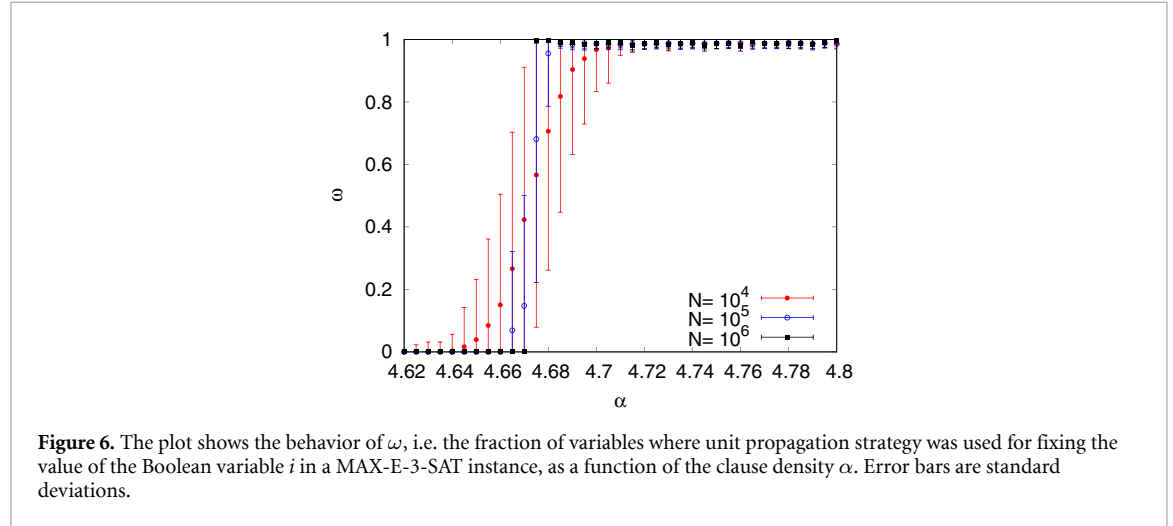
We start with comparing **NeurNet-SP** and MaxWalkSat, by analyzing the performance of the two algorithms for instances of MAX-E-3-SAT composed by $N = 10^6$ variables. In table 1, we show the values of $1 - \rho$, the fraction of unsatisfied clauses, at three different values of α, i.e. 4.2, 4.3, 4.5. These three points are in three distinct regions of the solution space. The first one, i.e. $\alpha = 4.2$, is in the region where solutions always exist. The second one, i.e. $\alpha = 4.3$, is in the region where no solution exists, but SP equations always converge. Instead, the third one is in the region where no solution exists, and SP equations do not converge.

The fraction of unsatisfied clauses obtained by MaxWalkSat is strongly dependent on the cutoff parameter chosen. The *cutoff* parameter in MaxWalkSat identifies the number of flips performed by the algorithm for leading to the greatest decrease in the total number of unsatisfied clauses. We chose the values $10^4, 10^5, 10^6, 10^7, 10^8$ for our comparison. When the cutoff is smaller than or equal to 10^7, the performance of **NeurNet-SP** is better than the performance of MaxWalkSat in satisfying the maximum number of clauses. However, from a cutoff of 10^8, or bigger, the MaxWalkSat performance is the best.

When the cutoff increases, the time for searching for an optimal solution also increases. This situation makes the MaxWalkSat really hard to analyze analytically. In other words, we do not know when $N \to \infty$ how big the cutoff should be. It is just a parameter that is chosen *a priori*. In contrast, **NeurNet-SP** overcomes this issue. No parameter increases its efficiency in finding an optimal approximate solution.

Table 1. MaxWalkSat results at $N = 10^6$ for different values of α and different cutoffs. We stop increasing the cutoff as soon as we find that MaxWalkSat results are better than the one found by **NeurNet-SP**.

Cutoff	N	$(1 - \rho)_{\alpha=4.2} \pm \sigma_{(1-\rho)_{\alpha=4.2}}$	$(1 - \rho)_{\alpha=4.3} \pm \sigma_{(1-\rho)_{\alpha=4.3}}$	$(1 - \rho)_{\alpha=4.5} \pm \sigma_{(1-\rho)_{\alpha=4.5}}$
10^4	10^6	0.5074 (6)	0.5197 (7)	0.5456 (8)
10^5	10^6	0.3760 (6)	0.3880 (6)	0.4137 (6)
10^6	10^6	0.0872 (3)	0.0971 (3)	0.1176 (4)
10^7	10^6	0.0240 (3)	0.0320 (4)	0.0497 (3)
10^8	10^6	0.0062 (1)	0.0137 (3)	0.0340 (4)

Figure 6. The plot shows the behavior of ω, i.e. the fraction of variables where unit propagation strategy was used for fixing the value of the Boolean variable i in a MAX-E-3-SAT instance, as a function of the clause density α. Error bars are standard deviations.

SP-y is a message passing algorithm ideated for minimizing the number of violated clauses in the MAX-E-k-SAT. It takes as INPUT a Boolean formula $\mathcal{F}$ in CNF and outputs a simplified Boolean formula $\mathcal{F}'$ in CNF and a partial truth-value assignment for the variables. If $\mathcal{F}' \neq \emptyset$ is given to a heuristic MAX-SAT Solver (as MaxWalkSat, Simulated Annealing) for building the complete assignment of $\mathcal{F}$. It uses decimation and backtracking strategies, which means that iteratively fixes and un-fixes variables for building up a partial (or complete) solution of $\mathcal{F}$ according to some estimated marginals.

Its performance is extraordinary. For $\alpha = 4.24$ and $N = 10^5$, SP-y can find a completely satisfiable assignment of the formula $\mathcal{F}$, or, in its worst performance, a value of $(1 - \rho) \approx 10^{-5}$. Above the $SAT - UNSAT$ threshold, for example, when $\alpha = 4.29$, the best performance of the algorithm, on a single sample of a MAX-E-3-SAT instance, reached a value of $(1 - \rho) \approx 2 \, 10^{-4}$. These performances can be obtained only when backtracking and decimation moves are performed, implying a very long run-time. Indeed, the SP-y algorithm has a computational complexity of order $O(N^2)$, making the established method unfeasible for huge values of N.

Without any decimation or backtracking moves, as in **NeurNet-SP**, the algorithm has the same performances of the heuristic MAX-SAT Solver used (indeed $\mathcal{F} = \mathcal{F}'$), and the performance of the algorithm is strongly dependent on the parameters of the heuristic.

For concluding the analysis of the algorithm, we looked at the performance of the algorithm beyond $\alpha = 4.620$. In the region $\alpha \in (4.620, +\infty)$, we meet a point at $\alpha = 4.67$, for $N \to \infty$, where the SP equations return at least one message $\eta_{a \to i} = 1$, also if $n_i > 1$. When such an issue happens, numerical instability into the SP equations appears. For avoiding this issue, therefore, we introduce a simple strategy, i.e. *unit propagation* strategy, on the variable i where at least one message $\eta_{a \to i} = 1$ appears in the set of functional nodes associated with it, and we fix such a variable by using the rule: i is *TRUE* if $n_i^+ > n_i^-$, *FALSE* otherwise. In figure 6 we present the fraction of variables where unit propagation strategy was used for fixing the value of the Boolean variable i in a sample of MAX-E-3-SAT instances, i.e. ω, as a function of α. At $\alpha = 4.67$, for $N = 10^6$, we pass from a region where **NeurNet-SP** uses the neural network to a region where the unit propagation rule fixes all the variables. Beyond the threshold at $\alpha = 4.67$ we can claim, also if numerically we meet the random assignment threshold $\rho_{\text{rand}} = 7/8$ at $\alpha = 10$ for each value of N analyzed, that the worst-case performance of the **NeurNet-SP** is equal to the random assignment outputted by the Johnson algorithm [31].

5. Conclusion

This paper has presented a new heuristic-learning algorithm, namely **NeurNet-SP** algorithm, that finds approximate solutions for the MAX-E-3-SAT problem. This algorithm runs SP equations on the random factor graph associated with the MAX-E-3-SAT problem. It gives the local information computed to a neural network $\mathcal{NN}(\vec{x}(i), \vec{\theta}^*)$. The set of parameters $\vec{\theta}$ are optimized following a supervised learning approach (by using target values obtained by SID on a sample of random 3-SAT problems) and outputs an assignment. We have displayed an accurate analysis to explain the algorithm's average and worst-case behavior as a function of the clause density α. We have started with presenting the limits of the SP equations and the neural network's performance in learning and inferring the conditional probability that a variable i should take to *TRUE* or *FALSE*. Then, we have shown that this algorithm can find approximate solutions that outperform the random assignment threshold value in the region where the SP equations do not present any numerical instability, and we have identified its algorithmic threshold, which is the ultimate limit of the algorithm, at $\alpha_a = 4.67$. Moreover, we have observed that the algorithm's output is strongly related to the average error of convergence $\langle \epsilon \rangle$ that the SP equations commit if they do not find a unique set of fixed points. Although this algorithm is not competitive with state-of-the-art MAX-SAT solvers, it can solve substantially larger and more difficult problems than it ever saw during training.

As future research directions, we propose to analyze the performance of the algorithm on MAX-E-4-SAT problem and verify if a Belief Propagation algorithm version could perform as well as our **NeurNet-SP** on that particular problem. We also suggest using SP-y equations, instead of SP equations, for improving the performance of this new heuristic-learning algorithm. Moreover, we suggest analyzing with a probabilistic approach the properties of the maximum error of convergence $\epsilon_{\max}$ that the SP equations can perform on random instances of the MAX-E-k-SAT problem. This maximum error should be related to the algorithmic performance, as numerically shown in our analysis.

Data availability statement

The data that support the findings of this study are available from the corresponding author upon reasonable request. The code that supports the findings of this study is openly available at [79].

Acknowledgments

ORCID iD

Raffaele Marino ⓘ https://orcid.org/0000-0002-2311-4380

References

[1] Pulina L and Seidl M 2020 *Theory and Applications of Satisfiability Testing-SAT* ed P Luca and S Martina Lecture Notes in Computer Science (Berlin: Springer) XI, p 538

[2] Yolcu E and Póczos B 2019 Learning local search heuristics for boolean satisfiability *NeurIPS* vol 32 ed H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox and R Garnett (*Vancouver, Canada*) pp 7990–8001 (available at: https://papers.nips.cc/paper/2019/file/12e59a33dea1bf0630f46edfe13d6ea2-Paper.pdf)

[3] Chang W L and Vasilakos A V 2021 Boolean algebra and its applications *Fundamentals of Quantum Programming in IBM's Quantum Computers* 1st edn vol 81 Studies in Big Data (Berlin: Springer) pp 53–108

[4] Berend D and Twitto Y 2021 Probabilistic characterization of random Max r-Sat *Discrete Optim.* **40** 100630

[5] Lourenço H R, Martin O C and Stützle T 2019 Iterated local search: framework and applications *Handbook of Metaheuristics* vol 272 ed M Gendreau and J Y Potvi (Berlin: Springer) pp 129–68

[6] Pei Y R, Manukian H and Di Ventra M 2020 Generating weighted MAX-2-SAT instances with frustrated loops: an RBM case dtudy *J. Mach. Learn. Res.* **21** 1–55

[7] Håstad J 2001 Some optimal inapproximability results *J. ACM (JACM)* **48** 798–859

[8] Dlask T and Werner T 2020 Bounding linear programs by constraint propagation: application to Max-SAT *Int. Conf. on Principles and Practice of Constraint Programming* (Springer) pp 177–93

[9] Walter R, Felfernig A and Küchlin W 2017 Constraint-based and SAT-based diagnosis of automotive configuration problems *J. Intell. Inf. Syst.* **49** 87–118

[10] Berg O J *et al* 2019 Applications of MaxSAT in data analysis EPiC Series in Computin vol 59 ed D Le Berre and M Järvisalo *Proc. Pragmatics of SAT 2015 and 2018* pp 50–64

[11] San Segundo P, Coniglio S, Furini F and Ljubić I 2019 A new branch-and-bound algorithm for the maximum edge-weighted clique problem *Eur. J. Oper. Res.* **278** 76–90

[12] Marino R and Kirkpatrick S 2018 Revisiting the challenges of MaxClique (arXiv:1807.09091)

[13] Marino R and Kirkpatrick S 2020 Large independent sets on random d-regular graphs with fixed degree d (arXiv:2003.12293)

[14] Si X, Zhang X, Grigore R and Naik M 2017 Maximum satisfiability in software analysis: applications and techniques *Int. Conf. on Computer Aided Verification* (Springer) pp 68–94

[15] Paxian T, Raiola P and Becker B 2021 On preprocessing for weighted MaxSAT *Int. Conf. on Verification, Model Checking and Abstract Interpretation* (Springer) pp 556–77

[16] Paxian T, Reimer S and Becker B 2018 Dynamic polynomial watchdog encoding for solving weighted MaxSAT *Int. Conf. on Theory and Applications of Satisfiability Testing* (Springer) pp 37–53

[17] Gouveia F, Lynce I and Monteiro P T 2020 Revision of Boolean models of regulatory networks using stable state observations *J. Comput. Biol.* **27** 144–55

[18] Heizmann M, Schilling C and Tischner D 2017 Minimization of visibly pushdown automata using partial Max-SAT *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems* (Springer) pp 461–78

[19] Ayanzadeh R, Halem M and Finin T 2019 SAT-based compressive sensing (arXiv:1903.03650)

[20] Jabbour S, Mhadhbi N, Raddaoui B and Sais L 2020 SAT-based models for overlapping community detection in networks *Computing* **102** 1–25

[21] Benedetti M and Mori M 2018 Parametric RBAC maintenance via Max-SAT *Proc. 23nd ACM Symp. on Access Control Models and Technologies* pp 15–25

[22] Urbonas M, Bundy A, Casanova J and Li X 2020 The use of Max-SAT for optimal choice of automated theory repairs *Int. Conf. on Innovative Techniques and Applications of Artificial Intelligence* (Springer) pp 49–63

[23] Chandriah K K and Naraganahalli R V 2021 Maximizing a deep submodular function optimization with a weighted MAX-SAT problem for trajectory clustering and motion segmentation *Appl. Intell.* **51** 1–20

[24] Li Y L, Lin S T, Nishizawa S and Onodera H 2020 Mcell: multi-row cell layout synthesis with resource constrained max-sat based detailed routing *Proc. 39th Int. Conf. on Computer-Aided Design* pp 1–8

[25] Huang W, Kitchaev D A, Dacek S T, Rong Z, Urban A, Cao S, Luo C and Ceder G 2016 Finding and proving the exact ground state of a generalized Ising model by convex optimization and MAX-SAT *Phys. Rev.* B **94** 134424

[26] Ochoa G, Chicano F and Tomassini M 2020 Global landscape structure and the random Max-SAT phase transition vol 12270 *Int. Conf. on Parallel Problem Solving from Nature* (Springer) pp 125–38

[27] Molnár B, Molnár F, Varga M, Toroczkai Z and Ercsey-Ravasz M 2018 A continuous-time MaxSAT solver with high analog performance *Nat. Commun.* **9** 1–12

[28] Py M, Cherif M S and Habet D 2020 Towards bridging the gap between SAT and Max-SAT refutations *2020 IEEE 32nd Int. Conf. on Tools With Artificial Intelligence (ICTAI)* (IEEE) pp 137–44

[29] Joshi S, Kumar P, Martins R and Rao S 2018 Approximation strategies for incomplete MaxSAT vol 11008 *Int. Conf. on Principles and Practice of Constraint Programming* (Springer) pp 219–28

[30] Li C M, Manya F and Soler J R 2019 Clausal form transformation in MaxSAT *2019 IEEE 49th Int. Symp. on Multiple-Valued Logic (ISMVL)* (IEEE) pp 132–7

[31] Johnson D S 1974 Approximation algorithms for combinatorial problems *J. Comput. Syst. Sci.* **9** 256–78

[32] Knebl H 2020 *Algorithms and Data Structures: Foundations and Probabilistic Methods for Design and Analysis* 1st edn (Switzerland: Springer) XI, p 349

[33] Arora S, Lund C, Motwani R, Sudan M and Szegedy M 1998 Proof verification and the hardness of approximation problems *J. ACM (JACM)* **45** 501–55

[34] Poloczek M, Schnitger G, Williamson D P and Van Zuylen A 2017 Greedy algorithms for the maximum satisfiability problem: simple algorithms and inapproximability bounds *SIAM J. Comput.* **46** 1029–61

[35] Chou C N, Golovnev A and Velusamy S 2020 Optimal streaming approximations for all boolean Max-2CSPs and Max-kSAT *2020 IEEE 61st Symp. on Foundations of Computer Science (FOCS)* (IEEE) pp 330–41

[36] Brakensiek J, Huang N, Potechin A and Zwick U 2021 On the mysteries of MAX NAE-SAT *Proc. 2021 ACM- Symp. on Discrete Algorithms (SODA)* (SIAM) pp 484–503

[37] Demirović E and Stuckey P J 2019 Techniques inspired by local search for incomplete MaxSAT and the linear algorithm: varying resolution and solution-guided search vol 11802 *Int. Conf. on Principles and Practice of Constraint Programming* (Springer) pp 177–94

[38] Bouhmala N, Oseland M and Brådland Ø 2016 Walksat based-learning automata for Max-SAT *Int. Conf. on Soft Computing-MENDEL* (Springer) pp 98–110

[39] Liu S and De Melo G 2017 Should algorithms for random SAT and Max-SAT be different? *Proc. Conf. on Artificial Intelligence* vol 31 pp 3915–21

[40] Bouhmala N 2019 Combining simulated annealing with local search heuristic for MAX-SAT *J. Heuristics* **25** 47–69

[41] Djenouri Y, Habbas Z and Djenouri D 2017 Data mining-based decomposition for solving the MAXSAT problem: toward a new approach *IEEE Intell. Syst.* **32** 48–58

[42] Brandts-Longtin A 2019 *Smoothed Analysis of Max-k-SAT* (Canada: McGill University)

[43] Jarret M, Jordan S P and Lackey B 2016 Adiabatic optimization versus diffusion Monte Carlo methods *Phys. Rev.* A **94** 042318

[44] Traversa F L, Cicotti P, Sheldon F and Di Ventra M 2018 Evidence of exponential speed-up in the solution of hard optimization problems *Complexity* **2018** 1–13

[45] Ali H M, Ejaz W, Al Taei M and Iqbal F 2019 Solving Max-SAT problem by binary biogeograph-based optimization algorithm *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Conf. (IEMCON)* (IEEE) pp 1092–7

[46] Bouhmala N and Øvergård K I 2018 Combining genetic algorithm with variable neighborhood search for max-sat *Innovative Computing, Optimization and Its Applications* 1st edn vol 741 (Berlin: Springer) XVIII, p 336

[47] Berend D and Twitto Y 2020 Effect of initial assignment on local search performance for Max SAT *18th Int. Symp. on Experimental Algorithms (SEA 2020)* (Schloss Dagstuhl-Leibniz-Zentrum für Informatik) (available at: http://www.sea2020.dmi.unict.it/LIPIcs160/LIPIcs-SEA-2020-8.pdf)

[48] Xu Z, He K and Li C M 2019 An iterative Path-Breaking approach with mutation and restart strategies for the MAX-SAT problem *Comput. Oper. Res.* **104** 49–58

[49] Bacchus F, Berg J, Järvisalo M and Martins R 2020 *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions* (Helsinki: University of Helsinki, Department of Computer Science) (available at: https://helda.helsinki.fi/bitstream/handle/10138/318451/mse20proc.pdf?sequence=1)

[50] AlKasem H H and Menai M E B 2021 Stochastic local search for partial Max-SAT: an experimental evaluation *Artif. Intell. Rev.* **54** 2525–66

[51] Lei Z and Cai S 2018 Solving (weighted) partial MaxSAT by dynamic local search for SAT *IJCAI* **7** 1346–52

[52] Ignatiev A, Morgado A and Marques-Silva J 2019 RC2: an efficient MaxSAT solver *J. Satisfiabil. Boolean Model. Comput.* **11** 53–64

[53] Berg J, Demirović E and Stuckey P J 2019 Core-boosted linear search for incomplete MaxSAT vol 11494 *Int. Conf. on Integration of Constraint Programming, Artificial Intelligence and Operations Research* (Springer) pp 39–56

[54] Luo C, Cai S, Su K and Huang W 2017 An efficient local search algorithm for weighted partial maximum satisfiability *Artif. Intell.* **243** 26–44

[55] Nadel A 2018 Solving MaxSAT with bit-vector optimization vol 10929 *Int. Conf. on Theory and Applications of Satisfiability Testing* (Springer) pp 54–72

[56] Zhao C Y and Fu Y R 2021 Belief propagation guided decimation algorithms for random constraint satisfaction problems with growing domainsBelief propagation guided decimation algorithms for random constraint satisfaction problems with growing domains *J. Stat. Mech.: Theory Expl.* **2021** 033408

[57] Mézard M, Parisi G and Zecchina R 2002 Analytic and algorithmic solution of random satisfiability problems *Science* **297** 812–5

[58] Marino R, Parisi G and Ricci-Tersenghi F 2016 The backtracking survey propagation algorithm for solving random K-SAT problems *Nat. Commun.* **7** 1–8

[59] Battaglia D, Kolář M and Zecchina R 2004 Minimizing energy below the glass thresholds *Phys. Rev.* E **70** 036107

[60] Wang X and Jiang J 2017 Propagation algorithm for the MAX-3-SAT problem *IEEE Trans. Emerg. Top. Comput.* **7** 578–84

[61] Berry M W, Mohamed A and Yap B W 2019 *Supervised and Unsupervised Learning for Data Science* Unsupervised and Semi-Supervised Learning (Berlin: Springer) VIII, p 187

[62] Shinde P P and Shah S 2018 A review of machine learning and deep learning applications *2018 Fourth Int. Conf. on Computing Communication Control and Automation (ICCUBEA)* (IEEE) pp 1–6

[63] Ker J, Wang L, Rao J and Lim T 2017 Deep learning applications in medical image analysis *IEEE Access* **6** 9375–89

[64] Zhang T, Gao C, Ma L, Lyu M and Kim M 2019 An empirical study of common challenges in developing deep learning applications *2019 IEEE 30th Int. Symp. on Software Reliability Engineering (ISSRE)* (IEEE) pp 104–15

[65] Balas V E, Roy S S, Sharma D and Samui P 2019 *Handbook of Deep Learning Applications* vol 136 Smart Innovation, Systems and Technologies (Berlin: Springer) (https://doi.org/10.1007/978-3-030-11479-4)

[66] Sarmadi H and Entezami A 2021 Application of supervised learning to validation of damage detection *Arch. Appl. Mech.* **91** 393–410

[67] Selsam D *et al* 2018 Learning a sat solver from single-bit supervision *Int. Conf. on Learning Representations*

[68] Dai H, Khalil E, Zhang Y, Dilkina B and Song L 2020 Learning combinatorial optimization algorithms over graphs *Proc. 31st Int. Conf. on Neural Information Processing Systems*

[69] Bengio Y, Lodi A and Prouvost A 2020 Machine learning for combinatorial optimization: A methodological tour d'horizon *Eur. J. Oper. Res.* **290** 405–21

[70] Kumar M, Kolb S, Teso S and De Raedt L 2020 Learning Max-SAT from contextual examples for combinatorial optimisation *Proc. Conf. Artif. Intell.* **34** 4493–500

[71] Budzynski L, Ricci-Tersenghi F and Semerjian G 2019 Biased landscapes for random constraint satisfaction problems *J. Stat. Mech.: Theory Exp.* **2019** 023302

[72] Bartha Z, Sun N and Zhang Y 2019 Breaking of 1RSB in random regular MAX-NAE-SAT *2019 IEEE 60th Symp. on Foundations of Computer Science (FOCS)* (IEEE) pp 1405–16

[73] Gabrié M, Dani V, Semerjian G and Zdeborová L 2017 Phase transitions in the q-coloring of random hypergraphs *J. Phys.* A **50** 505002

[74] Budzynski L and Semerjian G 2020 The asymptotics of the clustering transition for random constraint satisfaction problems *J. Stat. Phys.* **181** 1490–522

[75] Bartha Z 2020 Replica symmetry breaking in the random regular k-NAE-SAT problem PhD Thesis UC Berkeley

[76] Moore C 2017 The computer science and physics of community detection: landscapes, phase transitions, and hardness (arXiv:1702.00467)

[77] Hetterich S 2016 Analysing survey propagation guided decimationon random formulas *43rd Int. Colloquium on Automata, Languages, and Programming (ICALP 2016)* vol 55 pp 1–12 (arXiv:1602.08519)

[78] Curtin R R, Cline J R, Slagle N P, March W B, Ram P, Mehta N A and Gray A G 2013 MLPACK: a scalable C++ machine learning library *J. Mach. Learn. Res.* **14** 801–5

[79] Marino R 2020 (available at: https://github.com/RaffaeleMarino/DeepSP)

[80] Selman B *et al* 1996 Local search strategies for satisfiability testing *Cliques Coloring Satisfiabil.* **26** 521–32

[81] Ding J, Sly A and Sun N 2015 Proof of the satisfiability conjecture for large k *Proc. Forty-Seventh Annual ACM Symp. on Theory of Computing* pp 59–68

[82] Kingma D and Ba J 2015 Adam: A method for stochastic optimization *3rd Int. Conf. Learning Representations (San Diego, CA, USA, 7–9, May 2015)* (arXiv:1412.6980)