A Mathematical Foundation of Survey Propagation

Abstract

In a recent breakthrough, Mezard et al. [2] have developed an extremely efficient algorithm for finding satisfying assignments of random 3-CNF formulas in the satisfiable regime. For example, their algorithm typically finds a satisfying truth assignment of a random 3-CNF formula with $n = 10^6$ variables and $4.25n$ clauses in minutes. No other algorithm is known that can solve formulas of such density for $n = 10^5$.

The core of the algorithm is a procedure, called Survey Propagation (SP), in which messages are iteratively passed back and forth between variables and clauses, such that upon convergence each variable holds a distribution over the three values $\{0, 1, *\}$. Intuitively, the more mass a variable has on $*$, the less “constrained” it is. Each step of the algorithm amounts to: computing statistics for the current formula using SP, setting the “most constrained” variable to its favorite value, and simplifying the formula.

The derivation of the SP messages was motivated by heuristic techniques of statistical physics and relied on certain physical hypotheses regarding the organization of the space of satisfying assignments. Here we give precise, mathematical definitions of the quantities computed by SP without any assumptions. We then proceed to give a rigorous derivation of the SP messages.

1 Introduction

It is widely believed that the probability of satisfiability for random $k$-SAT formulae exhibits a sharp threshold as the ratio of clauses to variables is increased. In [3], Selman, Mitchell and Levesque gave extensive experimental evidence suggesting that for $k \geq 3$ there is a range of the clauses-to-variables ratio, $r$, within which it seems hard even to decide if a randomly chosen $k$-SAT instance is satisfiable or not (as opposed to finding all satisfying truth assignments or giving a proof of unsatisfiability). For example, for $k = 3$ their experiments draw the following remarkable picture. For $r < 4$, a satisfying truth assignment can be easily found for almost all formulas; for $r > 4.5$, almost all formulas are unsatisfiable; for $r \approx 4.2$, a satisfying truth assignment can be found for roughly half the formulas and around this point the computational effort to find a satisfying truth assignment, whenever one exists, is maximized.

In a recent breakthrough, Mezard, Parisi, and Zecchina [2] have developed an extremely efficient algorithm for finding satisfying assignments of random 3-CNF formulas in the satisfiable regime. For example, their algorithm typically finds a satisfying truth assignment of a random 3-CNF formula with $n = 10^6$ variables and $4.25n$ clauses in minutes. No other algorithm practically solves formulas of such density for $n = 10^4$.

The core of the algorithm is a procedure, called Survey Propagation (SP), in which messages are iteratively passed back and forth between variables and clauses, such that upon convergence each variable holds a distribution over the three values $\{0, 1, *\}$. Intuitively, the more mass a variable has on $*$, the less “constrained” it is. Each step of the algorithm amounts to: computing statistics for the current formula using SP, setting the “most constrained” variable to its favorite value, and simplifying the formula.

For any CNF formula $F$, let us say that a string $x \in \{0, 1, *\}^n$ is a cover of $F$, if for every clause the following is true under $x$: at least one literal is satisfied, or at least two literals are assigned $*$.

Clearly, being a cover is a “locally checkable” property, akin to being a satisfying assignment, or a codeword of an LDPC code. We give a novel, explicit formulation of this checking as a factor (Tanner) graph $G_F$, so that if $F$ is locally tree-like, so is $G_F$. We then show that the messages exchanged by SP on input $F$ are precisely the messages exchanged by the standard Belief Propagation (BP) algorithm [1] for computing the variable marginals using $G_F$. 
2 The Belief Propagation Formalism

Let $X = \{x_1, \ldots, x_n\}$ be a set of variables. We will be interested in functions with domain $X$ that can be expressed as the product of $m$ functions $f_j$, each with domain $\{x\}_j \subseteq X$, i.e.,

$$F(x_1, \ldots, x_n) = \prod_{j=1}^{m} f_j(\{x\}_j),$$

where $|\{x\}_j| \leq k$ for some $k \ll n$. Two examples of such functions are SAT problems, and LDPC codes. In the former, each (clause) $f_j$ takes the value 1 for all but one inputs; $F(x) = 1$ iff $x$ is a satisfying assignment. In the latter, each (parity check) $f_j$ takes the value 1 when the parity of the variables $\{x\}_j$ is 0; $F(x) = 1$ iff $x$ is a valid codeword.

As we focus on discrete domains, the integrals above will reduce to sums.

The factor graph of $F$, denoted by $G_F$, is a bipartite graph with $n$ variable-nodes and $m$ kernel-nodes. It contains an edge $i \sim j$, iff $x_i \in \{x\}_j$. If $G_F$ is a tree $T$, then $F_i(z)$ can be computed by rooting $T$ at $x_i$ and passing messages bottom-up along the edges of $T$. More precisely, messages are exchanged in rounds so that in each round every variable/kernel sends a message to its parent node. If a variable [node] $x_i$ and a kernel [node] $f_j$ are adjacent in $T$, the possible messages are:

$$[x_i \rightarrow f_j](z) = \prod_{\ell \neq j} [f_{\ell} \rightarrow x_i](z) \quad \text{and} \quad [f_j \rightarrow x_i](z) = \sum_{X \setminus x_i} \left\{ f_j \left( X \mid x_i = z \right) \prod_{\ell \neq i} [x_{\ell} \rightarrow f_j](x_{\ell}) \right\}.$$

Clearly, since messages are passed bottom-up, only one of the two possible messages will travel on each edge. On the other hand, observe that the messages defined above are independent of the rooting and, therefore, the proper ones for every possible rooting of $T$. As a result, if in each step both possible messages are passed along each edge, after $\text{diam}(T)$ rounds, we simultaneously get all the marginals $F_i$ as the messages passed to the variables $x_i$. Moreover, this process can be, alternatively, phrased as follows:

In each step, each variable/kernel sends a message along each edge incident to it, by taking the product/sum of the messages received in the previous round along all its other edges.

With this phrasing, it is clear that the message-passing process is, in fact, well-defined for any factor graph $G$. Obviously, for non-trees the outcome of the process (and, indeed, the existence of an outcome) is not a priori clear. Nevertheless, such “loopy Belief Propagation” (loopy-BP) has been an extremely successful algorithm in a number of domains, perhaps most notably in decoding LDPC codes. We note, in passing, that a key common element between random $k$-CNF formulas and LDPC codes is that the factor graph is a sparse random bipartite graph on a prescribed pair of degree sequences, i.e., (essentially) an expander.

To simplify notation we will maintain the following convention: given a CNF formula $F$, we will also let $F$ denote the function taking the value 1 when the formula is satisfied. Thus, $G_F$ will denote the factor graph of $F$. 


3 Clusters and Covers

We introduce a sequence of progressively weaker definitions of cluster-like objects for arbitrary CNF formulas. In going from each definition to the next we try to motivate the tradeoff between the loss in accuracy and the gain in tractability. The SP messages are recovered by applying the Belief Propagation formalism above to the weakest definition. Throughout we assume that we are dealing with a CNF formula \( F \) defined over variables \( X = x_1, \ldots, x_n \).

We will say that \( x, y \in \{0, 1\}^n \) are adjacent if their Hamming distance is 1. We let \( S(F) \subseteq \{0, 1\}^n \) denote the set of satisfying assignments of an arbitrary CNF formula \( F \).

Definition 1 The clusters of a formula \( F \) are the connected components of \( S(F) \).

The obvious problem with clusters is their potentially enormous descriptive complexity (we use the term here in the non-technical sense). To reduce this complexity, we must throw away some information about the cluster. A very useful notion for this is the cluster’s projection onto each variable \( x_i \), i.e., the union of the values taken by each \( x_i \) over the truth assignments in the cluster. As each variable \( x_i \in \{0, 1\} \), the projection of a cluster on each variable is either \( \{0\}, \{1\}, \) or \( \{0, 1\} \). We give these values the shorthand 0, 1, and \( * \) respectively. So, for any subset of \( \{0, 1\}^n \) its projection is a string from \( \{0, 1, *\}^n \). For \( x, y \in \{0, 1, *\}^n \) we will say that \( x \) is dominated by \( y \), written \( x \preceq y \), if for every \( i \), either \( x_i = y_i \), or \( x_i \in \{0, 1\} \) and \( y_i = * \).

Definition 2 The cluster-labels of a formula \( F \) are the projections of its clusters.

Remark 3 Distinct clusters of a formula can have the same cluster-label.

While cluster-labels are far more compact than clusters, determining the cluster-label of a cluster \( C \) remains a hard problem even if we are given a satisfying assignment \( x \in C \). To remedy this problem, we introduce an upper bound on the cluster-label of the satisfying assignment \( x \), defined as follows.

Definition 4 A variable \( x_i \) is free in a string \( x \in \{0, 1, *\}^n \), if in every clause containing \( x_i \) there is some literal other than \( x_i \) assigned true or *.

Algorithm \( * \)-propagation: if \( x \in \{0, 1, *\}^n \) has a free variable, pick any such variable and set it to *.

Lemma 5 For every formula \( F \), the fixed points of \( * \)-propagation reachable from \( S(F) \) are in 1:1 correspondence with the clusters of \( F \). Moreover, the fixed point corresponding to each cluster dominates that cluster’s label.

Proof. Trivially, applying \( * \)-propagation to a string \( x \) produces a string \( y \) such that \( x \preceq y \). Moreover, if \( x_i \) was free in \( x \), then \( y_i \) will be free in \( y \). As a result, if both \( y, z \in \{0, 1, *\}^n \) are reachable from \( x \in \{0, 1, *\}^n \) by \( * \)-propagation operations, so is the string that results by starting at \( x \) and concatenating the two sequence of \( * \)-propagation operations. This implies that there is a unique fixed point \( T(x) \) for each \( x \in \{0, 1, *\}^n \). Observe now that if \( x, x' \in S(F) \) differ only in the \( i \)-th coordinate, then \( * \)-propagation can be applied to both \( x \) and \( x' \), yielding the same string \( z \). By our earlier argument, \( T(x) = T(x') = T_C \), where \( C \subseteq S(F) \) is the cluster containing \( x, x' \). Considering all adjacent pairs in a cluster \( C \), we see that \( T_C \) dominates the label of \( C \). ■

Definition 6 The cluster-covers of a formula \( F \) are the \( * \)-propagation fixed points of its clusters.

Lemma 7 If the factor graph of \( F \) is a tree, then \( F \) has a unique cluster \( C \). The label of \( C \) equals its cover.

Proof. Observe that if the factor graph of \( F \) is a forest, then there is always a literal that appears in \( F \) whose complement does not appear in \( F \), i.e., a “pure” literal. Therefore, we can always satisfy \( F \) by repeatedly satisfying its lowest-numbered remaining pure literal and let \( p \in S(F) \) be the result of assigning 0 to any variables remaining unassigned at the end of this process. Observe now that if \( x \in S(F) \), then \( x' \) results by changing the \( i \)-th coordinate of \( x \) to \( p_i \), then \( x' \in S(F) \). Therefore, \( F \) has precisely one cluster.

Note, further, that if variable \( x_i \) does not appear in a unit-clause in \( F \), then assigning any value to \( x_i \) leaves a satisfiable formula (since the factor graph is still a forest). Therefore, the \( i \)-th coordinate of \( F \)’s cluster-label will be *. On the other hand, if \( x_i \) does appear in a unit-clause, it will never become free under the action of \( * \)-propagation. ■

Observe that every cluster-cover \( x \) of a formula \( F \) satisfies:
I. Each clause in $F$ has either a satisfied literal or at least two literals assigned $\ast$ under $x$.

II. No variable in $x$ is free.

**Definition 8** A **cover** of a formula $F$ is any $x \in \{0, 1, \ast\}^n$ satisfying Properties (I) and (II).

**Remark 9** Not every cover is a cluster-cover. For example, $\{\ast\}^n$ is a cover of unsatisfiable formulas.

We have arrived at our central objects: we will prove that SP computes marginals over the covers of the input formula.

More precisely, we will derive the messages exchanged under SP by applying the BP formalism of Section 2 to a specific factor graph representation of the indicator function “$x$ is a cover of $F$”, thus establishing SP as a loopy-BP approximation of the true marginals over covers. That said, there is one more twist. As we will see shortly, the natural attempt to express “$x$ is a cover of $F$” as a product of kernels over the variables in $X$ fails in the following sense: even if $F$ is locally tree-like, the factor graph for checking whether a string is a cover of $F$ is not.

Instead, given a $k$-CNF formula $F$ with $n$ variables and $m$ clauses we will first establish a bijection between the covers of $F$ and a certain set $T \subset \{0, 1\}^{2kn}$. Having done so, we then give a product-form representation of the indicator function for “$t \in T$”, i.e., we derive kernels $f_j$ such that $L(t) = L_F(t) = \prod_j f_j(t_j) = 1$ iff $t \in T$. Applying the BP formalism of Section 2 to the factor graph $G_L$ yields exactly the SP message-passing rules of [2]. Thus, the statistic computed by SP for each variable $x_i$ is precisely the loopy-BP approximation to $x_i$’s true marginal over the covers, when the indicator function for “$x$ is a cover of $F$” is expressed in the particular form that we give.