Survey Propagation for Random K-sat problems

Area I Seminar, 6.454

December 6, 2006

1 Introduction

This report will attempt to summarize some recent research on the random K-sat problem. After a brief introduction, we will describe some recently proposed algorithms for random K-sat, in particular the "survey propagation" algorithm. We will explain the equivalence of survey propagation to an appropriately defined belief propagation iteration, which a well-known iterative technique for estimation problems.

2 K-sat

2.1 Introduction to the problem

A clause with d variables is a Boolean expression that can be written as the logical OR of the variables $x_1, \ldots, x_d$, or their negations. For example, if $d = 3$, the following are clauses:

$$x_1 \vee x_2 \vee x_3$$
$$x_1 \vee \bar{x}_2 \vee x_3$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$$

where "$\vee$" is the logical OR symbol and $\bar{x}$ is the negation of x. The Boolean satisfiability problem is to determine whether an expression written as

$$P = C_1 \wedge C_2 \wedge \cdots \wedge C_n,$$

(where "$\wedge$" is the logical AND symbol and each C_i is a clause) is satisfiable for some assignment of the input variables x_i, i.e. whether we can assign each x_i the value 0 or 1 so that $P = 1$. If each C_i has exactly K variables, the problem is known as the K-sat problem.

A random K-sat problem is generated by choosing each C_i, $i = 1, \ldots, n$ via uniform sampling from the space of all $\binom{n}{K}$ subsets size of K of the set $\{x_1, \ldots, x_n\}$, and negating each variable with probability 1/2.

2.2 Motivation

2.2.1 In computer science

K-sat is one of the NP-complete problems: any problem in the class NP may be converted to a 3-sat problem with at most a polynomial time slowdown. An algorithm for solving K-sat problems would thus immediately solve many other problems (e.g. integer programming, traveling salesman problems) [11].

A random instance of K-sat, however, does not immediately translate into natural random instances of other problems. Nevertheless, there is a relationship between the complexity of random K-sat and hardness of approximation for various combinatorial problems - see [4].

2.2.2 In physics

A spin glass is a disordered magnetic system, i.e. a collection of molecules without a regular magnetic pattern. The disorder can arise because the molecules are arranged in an irregular order, or due to deliberate contamination of a regular lattice of molecules. The contaminated sites may have magnetic properties differing from the properties of the rest of the material, leading to irregularity.

Here is a model designed to encompass spin glass systems. Consider the space $\Sigma_N = \{0,1\}^N$ of n-letter words from the alphabet $\{0,1\}$. Each $\sigma \in \Sigma_N$ is a representation of N spins which are either $+1$ or -1. Let $H_N(\sigma)$ be a real valued function[1] representing the potential of the configuration σ. Different spins may attract or repel each other, and $H_N(\sigma)$ measures the energy of the system in configuration σ. Consider the probability measure on Σ_N defined by

$$p(\sigma) = \frac{1}{Z} e^{-H_N(\sigma)/T}$$

where Z is a normalizing constant and T is a parameter called the temperature of the system. As $T \to 0$, we have that $p(\sigma)$ approaches a uniform measure over all configurations minimizing H_N.

Much of the study of spin glasses is dedicated to characterizing the properties of the distribution $p(\sigma)$, for example: the constant Z, the average spin, whether $p(\sigma)$ can be iteratively generated, and so on.

If we consider Σ_N to be the set of assignments to a K-sat problem, and let $H_n(\sigma)$ be the number of violated constraints in assignment σ, we have that as $T \to 0$, $p(\sigma)$ is a uniform measure over all the solutions of the K-sat problem. Thus, K-sat can be cast as a zero-temperature limit of an appropriately defined spin glass system.

2.3 Experimental results for random K-sat

There are strong experimental (but non-rigorous) results on the behavior of random K-sat ensembles. The following summary is based on [10]; see the references therein for more information.

[1] In some models of this type, H is a random variable.

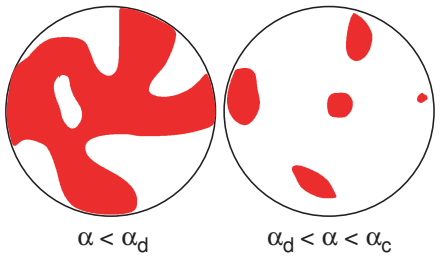

Figure 1: The solution clusters of random K-sat problems. Figure is taken from [10].

Let $\alpha = k/n$ be fixed, and let n be large. Then:

1). There exists a constant α_c, such that for $\alpha < \alpha_c$, the problem is almost always satisfiable and for $\alpha > \alpha_c$, the problem is almost always unsatisfiable.

2). There exists a constant α_d, smaller than α_c, such that for $\alpha \in (0, \alpha_d)$, the set of solutions almost always forms a connected cluster - see left hand side of Figure [10].

On the other hand, for $\alpha \in (\alpha_d, \alpha_c)$, the set of solutions almost always consists of disconnected clusters - see right hand side of Figure [10].

Some progress towards making the above statements rigorous was made in [8]. One way of restating the first point is that the probability of satisfiability goes to 1 if

$$k \geq (\alpha_c + \epsilon)n$$

and to zero if

$$k \leq (\alpha_c - \epsilon)n$$

as $n \to \infty$. In [8], it was proved that the probability of satisfiability goes to 1 if

$$k \geq (a(n) + \epsilon)n$$

and to zero if

$$k \leq (a(n) - \epsilon)n$$

where $a(n)$ is some function of n. It has not been proved that $a(n)$ can be taken to be a constant. This appears to be best of what has been proved as far as phase transitions are concerned. Some bounds on the constant α_c are known - see [3].

3 Factor graphs and belief propagation

3.1 Introduction

In this section we introduce factor graphs and belief propagation. We indicate how these apply to the K-sat problem. The exposition here is is based on [7].

Let $g(z_1, \ldots, z_n)$ be a function of the variables $z_1, \ldots, z_n$, which take values in some discrete set. We will use the "not-sum" $\sum_{\sim\{z_i\}}$ to indicate that the sum is over every variable except z_i, i.e.

$$\sum_{\sim\{z_i\}} g(z_1, \ldots, z_n) = \sum_{z_1, z_2, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n} g(z_1, \ldots, z_n)$$

Suppose $g(z_1, \ldots, z_n)$ factors into a product of "local functions" f_j, where each f_j is defined on some subset of $\{z_1, \ldots, z_n\}$:

$$g(z_1, \ldots, z_n) = \prod_j f_j(Z_j), \tag{3.1}$$

Z_j is a subset of $\{z_1, \ldots, z_n\}$, and $f_j(Z_j)$ is a function of the variables in the set Z_j. Our goal in this section is to efficiently compute the "marginal functions"

$$g_i(z_i) = \sum_{\sim\{z_i\}} g(z_1, \ldots, z_n)$$

One way to do this is by straightforwardly going through every possible value of $z_1, \ldots, z_n$ and adding the corresponding evaluation of g. We will see that it is possible to use Eq. (3.1) to compute g_i more efficiently.

3.2 Factor graphs

We introduce the factor graph representation of the function g. A factor graph has a vertex for every variable z_i and a vertex for every function f_j. There is an edge between the vertex corresponding to the variable z_i and the vertex corresponding to the function f_j if and only if $z_i \in Z_j$, i.e. if and only if variable z_i appears in function f_j.

This defines a bipartite graph (i.e. none of the variable vertices are connected to each other, and neither are the function vertices).

Example: Consider the function g defined by

$$g(x_1, x_2, \ldots, x_5) = f_{\mathrm{A}}(x_1) f_{\mathrm{B}}(x_2) f_{\mathrm{C}}(x_1, x_2, x_3) f_{\mathrm{D}}(x_2, x_4) f_{\mathrm{E}}(x_2, x_5)$$

The corresponding factor graph is shown in Figure 2.

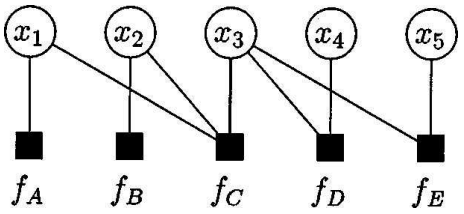

Figure 2: Factor graph for function $g(x_1, x_2, x_3, x_4, x_5)$. The figure is from [7].

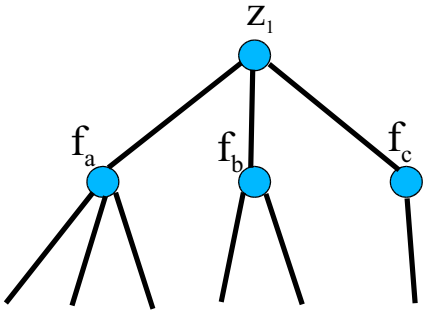

Figure 3: Neighborhood of the root in a sample tree factor graph.

3.3 Belief propagation

Now if the factor graph of $g(x_1, x_2, \ldots)$ is a tree, then we can write a certain recursive factorization for the functions $g_i(x_i)$. To show this, we need the so-called cartesian product lemma, which states that for any two disjoint sets X,Y and for any two functions f, h, we have that

$$\sum_{x \in X, y \in Y} f(x)h(y) = (\sum_{x \in X} f(x))(\sum_{y \in Y} h(y))$$

With this in mind, consider the factor graph shown in Figure 3. Let Z_a, Z_b, Z_c represent the variable sets used by the functions f_a, f_b, f_c. Let f_{ai}, $i = 1, 2, \ldots$ be all the functions in the branch of the tree under f_a, and let their variable sets be Z_{ai}; similarly for f_{bi}, f_{ci} and Z_{bi}, Z_{ci}. Then,

$$\sum_{\sim z_1} g(z_1, z_2, \ldots) = \sum_{\sim z_1} f_a(Z_a)(\prod f_{ai}(Z_{ai}))f_b(Z_b)(\prod f_{bi}(Z_{bi}))f_c(Z_c)(\prod f_{ci}(Z_{ci}))$$

which is simply writing out the factor graph representations for $g(z_1, z_2, \ldots)$. Now fix z_1 to some real number and delete it from all the variables sets Z_i; note that now the variable sets Z_a, $Z_{ai}, i = 1, 2, \ldots$, do not intersect with any of the sets Z_b, Z_{bi} or Z_c, Z_{ci}. Applying the cartesian product lemma, we have that for

all z_1,

$$\sum_{\sim z_1} g(z_1, z_2, \ldots) = [\sum_{\sim z_1} f_a(Z_a) \prod f_{ai}(Z_{ai})] \cdot [\sum_{\sim z_1} f_b(Z_b) \prod f_{bi}(Z_{bi})] \cdot [\sum_{\sim z_1} f_c(Z_c) \prod f_{ci}(Z_{ci})] \tag{3.2}$$

This factorization gives rise to a "bottom-up" algorithm for computing $g_1(z_1)$, which computes the quantities $\sum_{\sim z_1} f_k(Z_k) \prod f_{ki}(Z_{ki})$ recursively for all subbranches of the tree. This can be done with the following steps:

1). Each leaf variable node sends an identity message to its parent; each leaf function node sends a description of itself to its parent

2). When a variable node receives messages from all its children, it forms the product of them and sends them to its parent (where product is the simple pointwise product of functions).

3). When a function node f_i receives messages from all its children, it forms the product of these messages with $f_i(Z_i)$ and operates on the result with the $\sum_{\sim z_i}$ operator.

Applying Eq. (3.2) recursively, it follows that when these message-passing rules are executed, the product of the final messages to the root node z_1 will be the function $g_1(z_1)$.

The update rules may be summarized as:

$$\mu_{z_i \to f_j} = \prod_{h \in n(z_i) - \{f_j\}} \mu_{h \to z_i} \tag{3.3}$$

and

$$\mu_{f_j \to z_i} = \sum_{\sim z_i} (f(Z_j) \prod_{y \in n(f_j) - \{z_i\}} \mu_{y \to f_j}) \tag{3.4}$$

where $\mu_{a \to b}$ means the message sent by node a to node b.

The execution of Eq. (3.3) and Eq. (3.4) is known as belief propagation or the sum-product algorithm. If the factor graph is not a tree, Eq. (3.3) and Eq. (3.4) may still be applied, though now there are no guarantees on the correctness of the final solution. In practice, however, one often obtains good results by using belief propagation on graphs with cycles [6].

3.4 Parallelization

If we want to compute all $g_i(z_i)$ simultaneously, it is possible to parallelize belief propagation. All messages from node a to node b are computed as before using Eq. (3.3) and Eq. (3.4) and **assuming b is the parent of a**. As before, message passing is initiated in the leaves.

We claim that if the factor graph is a tree, then after n iterations the message sent to variable node z_j will be $g_j(z_j)$. To see this, note that we have shown it for the case when Eq. (3.3) and Eq. (3.4) are executed in a bottom-up schedule after having designated z_j as the root. We then show by induction that the relevant messages in a parallelized schedule are the same as the messages in a bottom-up schedule.

Clearly, the messages from the leafs of the tree at step 1 of the algorithm are the same, by definition. Defining the distance of a node to be the the length of the shortest path to the closest leaf, our inductive hypothesis is that the messages of the vertices at distance i at time i is the same in both schedules. To show that messages of nodes at distance $i+1$ at time $i+1$ are the same, simply note that these messages are defined using the same Eq. (3.3) and Eq. (3.4) on the same inputs. Indeed, a given node will receive a message from its parent on the path to z_j, but it will ignore this message when computing its own message to that parent according to Eq. (3.3) and Eq. (3.4).

This proves that all the functions g_j may be computed in parallel by iterating the equations Eq. (3.3) and Eq. (3.4).

3.5 K-sat

For a given K-sat problem, we may write $f_i = C_i$, i.e. $f_i = 1$ if clause i is satisfied, and 0 if it is not. Then, $g(x_1, \ldots, x_n) = \prod_i f_i$; we have that $g(x_1, \ldots, x_n) = 1$ if the configuration $(x_1, \ldots, x_n)$ satisfies the K-sat problem, and 0 if it does not.

If the factor graph is a tree, then belief propagation will correctly compute the functions $g_i(x_i)$, which map $a \in \{0, 1\}$ to the number of solutions to the K-sat problem having $x_i = a$.

4 Survey propagation

In this section, we describe the survey propagation algorithm for K-sat problems. To this end, we first introduce a related algorithm for the K-sat problem known as "warning propagation."

The material here is based on [1] and [9].

4.1 Warning propagation

First, we introduce some notation. Given constraint a and variable i, we will define $C_a^s(i)$ to be those constraints whose preferred assignment for i is the same as that of constraint a. For example, if variable i appears in constraint a negated as $\bar{x}_i$, then $C_a^s(i)$ consists of all those constraints except a in which $\bar{x}_i$ appears. Similarly, $C_a^u(i)$ consists of all those constraints whose preferred assignment for i is not the same as that for constraint a.

Warning propagation is a message passing algorithm defined as follows. Messages from variables to constraints are either "s," "u," "*," (intuitively corre-

sponding to, respectively, the variable being inclined to adopt a satisfying value for the constraint, an unsatisfying value for the constraint, and indifference towards which value to adopt). Messages from constraints to variables are either empty or "w" (intuitively corresponding to a constraint warning a variable that it must take a satisfying value for that constraint).

The message updating rules for constraints are simple: a constraint sends a warning "w" to a variable if that variable is the only variable in the constraint; or if the constraint has received "u"s from all the other variables. Else it sends an empty message.

For the variables, we have the following rules.

- If a variable i receives a warning from constraints in $C_a^s(i)$ and no warnings from $C_a^u(i)$, it sends an "s" to a.
- If a variable i receives a warning from constraints in $C_a^u(i)$ and no warnings from $C_a^s(i)$, it sends an "u" to a.
- If a variable i receives no warning from any nodes in $C_a^u(i)$ or $C_a^s(i)$, then it sends a $*$ to a.
- If a variable i receives warnings from both nodes in $C_a^s(i)$ and $C_a^u(i)$ it sends an s if it has received more warnings from $C_a^s(i)$ and a u otherwise.

We next introduce the "warning numbers" c_i:

c_i = 1 if at the fixed point variable i receives warnings from two constraints with different preferred values for i

= 0 if at the fixed point variable i receives no warnings, or all the warnings it receives are from constraints with the same preferred value for i

We then have the following theorem.

Theorem 4.1 *Consider an instance of the K-sat problem on N variables for which the factor graph is a tree. Then at time N the warning propagation algorithm converges to a fixed point. Moreover, if any c_i are nonzero, the problem is unsatisfiable; if all the c_i are zero, the problem is satisfiable.*

Proof:

1). First, we prove that the algorithm converges to a fixed point in N steps. Given a tree factor graph, let $T_{a\to i}$ be the subgraph defined by removing the edge (a, i) and picking the remaining component containing a. We will define the **largest** distance between constraint a and a leaf node in $T_{a\to i}$ to be the "level" of the edge (a, i). We will prove convergence by induction on the level.

If an edge has level zero (i.e. the constraint node is a leaf), then the message from the constraint to the variable is "w" at time 1. Moreover, this message does not change.

If an edge has level one, then the constraint node has a variable child which is a leaf. That node will always send a "*" to the constraint node. It follows that the message from the constraint to the variable on a level 1 node will be empty for $t \geq 2$.

Our inductive hypothesis is that the message from constraint to variable on an edge with level r is constant after time $1 + r$. We have proven this for $r = 0, 1$. To complete the induction, note that messages on an edge with level r are completely determined by the messages on edges with level at most $r - 2$. Indeed, after deleting the edge (a, i), consider edges (b, i) in the component containing a, where b is some constraint which is not a. This edge is on a path from a to *some* leaf, and hence must have level strictly less than r - at most $r - 2$ in fact, since the distance between a and b is at least 2 since a variable vertex must be traversed. Thus the messages sent by all other constraints b appearing in $T_{a \to i}$ are constant after time $1 + (r - 2) = r - 1$ by the inductive hypothesis. Thus, all the variable neighbors of a except i must send the same message to a after time r. This means that the message from a to i is constant after time $r + 1$. This proves the claim about convergence.

2). Next, we prove a useful lemma. We will refer to message from constraint a to variable i as $u_{a \to i}$, and similarly $u_{i \to a}$ will denote the message from variable i to constraint a. We will add stars to messages at the fixed point, i.e. $u^*_{a \to i}$ means the message from a to i at the fixed point of warning propagation.

Note also that subgraphs of our factor graphs also correspond to K-sat problems, obtained by simple omitting the constraints and variables not in the subgraph.

Lemma 4.2 *If $u^*_{a \to i} = w$, then clause a is violated in the K-sat problem defined by the subgraph $T_{a \to i}$.*

Proof: Obviously true if the level of the edge (a, i) is 0. Its easy to see that the assumptions of the lemma can never happen if the level of (a, i) is 1.

For higher levels, we prove the statement by induction. Suppose the assumptions of the lemma hold for some edge (a, i) with level at least 2. Consider a variable node j which is (i) a neighbor of the constraint node a (ii) not i. At the fixed point, j must have received a warning from some constraint c_j, leading it to send "u" to constraint a. But then, by the inductive hypothesis, the K-sat problem corresponding to the subgraph $T_{c_j \to j}$ is unsatisfiable! This means that either the problem corresponding to $T_{a \to i}$ is unsatisfiable, in which case we are done; or the value of node

j is fixed by constraint c_j to satisfy the problem. This value must not be the value constraint a prefers for node j, as j sent a "u" to a. It follows that all of the neighbors j except i of a are fixed to values which do not satisfy a. This proves that the problem defined by the subgraph $T_{a\to i}$ is unsatisfiable.

3). Having shown that $u^*_{a\to i}$ = "w'' implies that constraint a fixes the value of i, it now immediately follows that the formula is unsatisfiable if there is a conflict represented by $c_i = 1$ for some i.

4). If all c_i are 0, we use the fixed point messages of warning propagation to produce a satisfactory assignment. We fix the variables which have received a warning (since all c_i are zero, there are no conflicts). One then "cleans" the graph by removing the satisfied constraints and removing the fixed variables from all other constraints. Note that the only variables which are left are those which did not receive any warnings and consequently picked "*" values.

Next, pick randomly a variable i, fix it to any value, and clean the graph once again.

This process has split our original tree into a set of disjoint trees. We run warning propagation on each of these trees, initialized with the fixed point messages of the previous iteration of warning propagation. Since no warnings were sent on any edges of the new trees at this fixed point, the only constraints which will send a warning are those who are adjacent to the variable i that was fixed. However, each variable can receive only one warning, coming from the unique constraint on the path to i. Thus, c_i must be all 0 after running warning propagation on the strictly smaller graph.

Thus we've produced a strictly smaller graph with all c_i zero. Since it is trivial that all c_i being zero is sufficient for satisfiability for graphs of size 1 and 2, by induction this completes the proof for an arbitrary graph.

4.2 Survey propagation

Now consider the statistics of warning propagation, i.e. the probabilities that a given constraint will send a "w" at time t. How do these evolve?

Let us denote the probabilities of sending a "u," "s," and "*" on link (i, a) by $\Pi^u_{i\to a}$,$\Pi^s_{i\to a}$, and $\Pi^*_{i\to a}$; and let us denote the probability of warning on link (i, a) as $\eta_{a\to i}$.

Assuming all warning events are independent, we have that a warning is sent on a link if and only if the constraint receives "u"s from all other links. Thus:

$$\eta_{a\to i} = \prod_{j\in N(a)-\{i\}} \frac{\Pi^u_{j\to a}}{\Pi^u_{j\to a} + \Pi^s_{j\to a} + \Pi^*_{j\to a}} \tag{4.5}$$

where $N(a)$ is the set of neighbors of a in the factor graph.

A link will send a $*$ message if and only if it will get no warnings at all:

$$\Pi^*_{i\to a} = \prod_{b\in N(i)-\{a\}} (1-\eta_{b\to i}) \tag{4.6}$$

Now we make another assumption: that no node gets contradictory warnings. Intuitively, we assume that warning propagation stays within the cluster of satisfying assignments which dampens the likelihood of contradictory warnings.

Then, we will have that a node i sends a "u" to constrain a if and only if it receives no warnings from constraints that agree with a, and at least one warning from a constraint that disagrees with a:

$$\Pi^u_{i\to a} = [\prod_{b\in C^s_a(i)} (1-\eta_{a\to i})][1-\prod_{b\in C^u_a(i)} (1-\eta_{a\to i})] \tag{4.7}$$

Similarly,

$$\Pi^s_{i\to a} = [\prod_{b\in C^u_a(i)} (1-\eta_{a\to i})][1-\prod_{b\in C^s_a(i)} (1-\eta_{a\to i})] \tag{4.8}$$

Equations 4.5,4.6, 4.7, 4.8 are the survey propagation algorithm. Node i sends the triple $\Pi^u_{i\to a}$,$\Pi^s_{i\to a}$,$\Pi^*_{i\to a}$ to constraint a, which sends back $\eta_{a\to i}$. The above equations constitute the update rules.

5 Belief propagation on generalized state realizations

In this section, we summarize an extension of factor graphs and belief propagation. The exposition here follows [5].

A generalized state realization has a set of state variables in addition to a set of symbol variables and function variables. Each state variable can take values in some discrete set. Functions will now involve both state variables and symbol variables. There is an edge from (i,j) if one of i,j corresponds to a function, and the other corresponds to a variable or state used by the function.

From now on we will restrict our functions f_j to have range in $\{0,1\}$. Intuitively, we will think of functions as *constraints*. A value of $f_j = 1$ means constraint C_j is satisfied; a value of $f_j = 0$ means it is not.

We will further assume that our factor graph is normal, which means that all symbol variables have degree 1 and all state variables have degree 2. All factor graphs may be made normal by replication; see [5] for details.

A configuration $(z_1,\ldots,z_n,s_1,\ldots,s_m)$, where z_i are variables, and s_i are states, will be called admissible if satisfies all the constraints. A configuration of symbol variables only $(z_1,\ldots,z_n)$ will be called admissible if it satisfies all the constraint for some state assignment $(s_1,\ldots,s_m)$.

Let $\mathbf{z} = (z_1,\ldots,z_n)$ be an admissible configuration. We assume that we have some weight function which factorizes as $w(z) = \prod_j w^j(z_j)$ mapping admissible

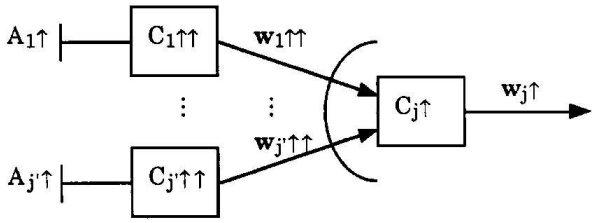

Figure 4: Local configuration for a sum-product update rule. Figure is from [5].

configurations to real numbers; we assume w maps inadmissible configurations to 0. We are interested in computing

$$w_k(s_k) = \sum_{\mathbf{z} \in C_k(s_k)} w(\mathbf{z})$$

where $C_k(s_k)$ is the set of all admissible configurations consistent with s_k.

We define $\mathbf{w}_k$ to be the vector which stacks up $w_k(s_k)$. As before, we are assuming that the factor graph is a tree; thus we assume all configurations can be broken up into "upstream" and "downstream" parts. Define $\mathbf{w}_{k\uparrow}(s_k)$ to be the sum of the weights of all upstream configurations consistent with s_k; and $\mathbf{w}_{k\downarrow}$ to be the sum of the weights of all the downstream configurations consistent with s_k. We can similarly define $\mathbf{w}_{k\uparrow\uparrow}$ and $\mathbf{w}_{k\downarrow\downarrow}$. See Figure 4 for an illustration, though note that state variable are shown as edges (without a dongle sign) in the picture.

Now the weight of an upstream configuration consistent with s_j is the product of the weights of subconfigurations along each branch $w_{k\uparrow\uparrow}$ in the picture, times the weight of any symbol variable connected to $C_{j\uparrow}$, which we will denote as $w_{j\uparrow}$. Summing over all possible choices for the configurations and applying the cartesian product lemma, we have:

$$\mathbf{w}_{k\uparrow} = \sum_{C_j(s_j)} \prod w_{k\uparrow\uparrow}(s_{j'}) \tag{5.9}$$

where $C_j(s_j)$ is the set of neighboring upstream states consistent with s_j.

Equation (5.9) is the equation corresponding to belief propagation on the general state realization. As before, if executed sequentially from the leaf to the root, the final iteration will compute $w_k(s_k)$ for the root node. Parallel execution is possible as well.

6 Survey propagation as belief propagation

In this section, we show how survey propagation is equivalent to belief propagation on a normal realization of the K-sat problem. The material is taken from [12]. See [2] and [9] for similar results.

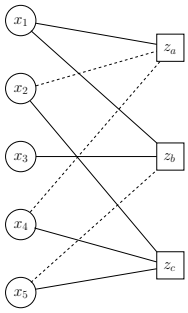

Figure 5: Factor graph for a K-sat problem. The figure is from [12].

6.1 State realization of the K-sat problems

We will construct a normal realization of the K-sat problem as follows. Each variable i will be connected to a new constraint node g_i, and to no other nodes. If before, the link (i, a) was in the factor graph, we add a state $s_{i,a}$ and connect g_i to $s_{i,a}$ and in turn $s_{i,a}$ to the constraint node a. See for example Figures 5 transformed into the graph in Figure 6.

Note that the new state realization is normal as all the state nodes have degree two and all the symbol variables have degree one.

Each variable node will take on a possible value of $\{0, 1, *\}$. Each state node will take a value from the alphabet $\{F, \bar{F}, *\} \times \{F, \bar{F}, *\}$, where $\times$ represents the cartesian product. We will represent the state as an ordered pair, $s_i = (l_i, r_i)$, where l_i will be called the left state, and r_i will be called the right state. Alternatively, we will sometimes write $l(s)$ and $r(s)$ for the left and right components of state s. Intuitively, left or right state values of "F" will denote that the variable is inclined to take on a value that satisfies the constraint, and conversely with "$\bar{F}$".

We next explicitly state the constraints. For simplicity, let $[X]$ be the indicator function of the event X, i.e. $[X] = 1$ if X is true, and $[X] = 0$ if X is not true. We will write $u = v$ to indicate that both u and v must take on the same value, and $u \sim v$ to indicate that either $u = v$ or $v = *$. Moreover, $A_{i,a}$ will denote the map which maps $\{0, 1\}$ into $F, \bar{F}$ by outputing F if the input is the preferred assignment of constraint a for node i and $\bar{F}$ otherwise.

The constraint associated with a constraint node a carried over from the original factor graph is a function of the state nodes $s_i = (l(s), r(s)$ incident to the constraint:

$$f_a = \prod_{s \in N(a)} ([r(s) = F \text{ and all neighboring states } \hat{s} \neq s \text{ have } l(\hat{s}) = \bar{F}] + [r(s) = * \text{ and at least one state } \hat{s} \neq s \text{ has } r(\hat{s}) \neq \bar{F}])$$

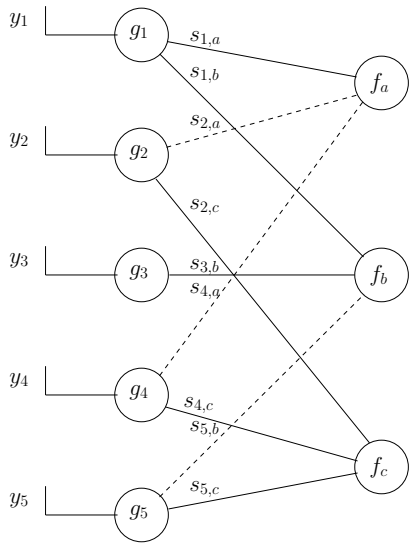

Figure 6: Normal realization of the K-sat problem from Figure 5. The figure is from [12].

In words, each state node must have that some other neighboring state node has right state thats not $\bar{F}$, or all the other state nodes have right state $\bar{F}$ and it has state F.

The constraint associated with the new constraint nodes g_i is a function of the the state values $s_i = (l_i, r_i)$ and the values of the variable nodes, which we will denote as y_i:

$$g_i = \prod_{s \in N(g_i)} [A_{i,a}(y_i) \sim l(s_i)] \prod_{s_i \in N(g_i)} [l_i \sim r_i]$$

In words, variable node values must be compatible with all left states, and each left state must be compatible with each right state, where a is compatible with b means $a = b$ or $b = *$.

Next, we introduce the concept of constrained and unconstrained variable. Given some assignment of state and variable values, a variable i is called constrained if for some a, $l(s_{i,a}) \in \{F, \bar{F}\}$, i.e. not all associated left-states are "*."[2] A variable is called unconstrained if it is not constrained, but its value is not "*." Define the function W_i as,

$$W_i \quad = \quad \omega_* \text{ if } y_i = *$$

[2]Of course, since constraint g_i must hold, this would also imply that all the left states associated with i are equal to the same element of $\{F, \bar{F}\}$.

$$
\begin{aligned}
&= \omega_u \text{ if } y_i \text{ is unconstrained} \\
&= 1 \text{ if } y_i \text{ is constrained}
\end{aligned}
$$

We define the weight of an assignment of values to state and variable nodes as

$$W = \prod_{i,a} W_i g_i f_a$$

where the sum is taken over all variable and constraint nodes. Clearly, for assignments of variables which satisfy all the constraints,

$$W = (\text{ number of unconstrained variables })^{\omega_u} (\text{ number of star variables })^{\omega_*}$$

6.2 Survey propagation as belief propagation

Let us apply the belief propagation formula

$$w_{j\uparrow} = \sum_{C_i(s_j)} \prod w_{j'\uparrow\uparrow}(s_{j'})$$

to the normal realization just created. We denote by $\rho_{a\to i}$ to be the message from constraint a to constraint g_i and $\lambda_{i\to a}$ to be the message from constraint g_i to constraint a.

We next write down the message iteration rules.

The state FF for example will appear in $s_{i,a}$ only if all the other state variables adjacent to a have $(l(s), r(s)) = (\bar{F}, *)$. Thus,

$$\rho_{a\to i}(FF) = \prod_{j\in N(a)-\{i\}} \lambda_{j\to a}(\bar{F}*) \tag{6.10}$$

The state $**$ will appear in $s_{i,a}$ only if every other right state adjacent to a is a $*$ (there can't be any F right states: sending an F to a node implies all the other nodes take left states $\bar{F}$. But our node has a $*$ state, so this is not the case). But though all right states are $*$, not all the left states can be $\bar{F}$, because then the state of $s_{i,a}$ would be FF. Thus, the set of compatible states $C_i(s_i)$ is $\bar{F}*$ for every other neighbor of a:

$$\rho_{a\to i}(**) = \prod_{j\in N(a)-\{i\}} (\lambda_{j\to a}(F*)+\lambda_{j\to a}(\bar{F}*)+\lambda_{j\to a}(**)) - \prod_{j\in N(a)-\{i\}} \lambda_{j\to a}(\bar{F}*)$$

The state $\bar{F}*$ will appear in $s_{i,a}$ if all the other nodes get $*$ right states but not all of them take the left state $\bar{F}$ (because that would imply we get an F right state). Alternatively, there may be a single FF state among the other nodes; but not more than one because that would violate constraint f_a. Moreover, the existence of an FF state automatically implies all other states must be $\bar{F}*$. So:

$$\rho_{a\to i}(\bar{F}*) = \prod_{j\in N(a)-\{i\}} (\lambda_{j\to a}(F*) + \lambda_{j\to a}(\bar{F}*) + \lambda_{j\to a}(**)) - \prod_{j\in N(a)-\{i\}} \lambda_{j\to a}(\bar{F}*)$$
$$+ \sum_{k\in N(a)-\{i\}} (\lambda_{k\to a}(FF) - \lambda_{k\to a}(F*) - \lambda_{k\to a}(**)) \prod_{m\in N(a)-\{i,k\}} \lambda_{m\to a}(\bar{F}*)$$

Finally, the state $F*$ is valid with exactly the same combination of states as $**$, so that their update equations are the same.

For the messages from constraints g_i to a we have the following update rules.

The state $s_{i,a}$ will take the value FF if it gets no Fs from those constraints which disagree with a and Fs or $*$s from those constraints which agree. In all these possibilities, i will be a constrained variable. Thus:

$$\lambda_{i\to a} = \prod_{v\in C_a^u(i)} \rho_{b\to i}(\bar{F}*) \prod_{v\in C_a^s(i)} (\rho_{b\to i}(FF) + \rho_{b\to i}(F*))$$

The state $s_{i,a}$ will take the value $**$ only if all of its incoming values are $*$s. Since i is a star, this assignment has weight ω_*. Thus:

$$\lambda_{i\to a}(**) = \omega_* \prod_{b\in N(i)-\{a\}} \rho(b\to i)(**) \tag{6.11}$$

The state $s_{i,a}$ will take the value $\bar{F}*$ if it receives only $*$s from those constraints which agree with a, and F and $*$s from those which do not. However, we must also take note that the assignment where i receives all $*$s and takes the value of $\bar{F}$ must be weighed by ω_u, because here i is an unconstrained variable. Thus:

$$\lambda_{i\to a}(\bar{F}*) = \prod_{b\in C_a^s(i)} \rho_{i\to a}(\bar{F}*)(\prod_{b\in C_a^u(i)} (\rho_{i\to a}(F*)+\rho_{i\to a}(FF))-(1-\omega_u) \prod_{b\in C_a^u(i)} \rho_{i\to a}(F*)) \tag{6.12}$$

Similarly, we have,

$$\lambda_{i\to a}(F*) = \prod_{b\in C_a^u(i)} \rho_{i\to a}(\bar{F}*)(\prod_{b\in C_a^s(i)} (\rho_{i\to a}(F*)+\rho_{i\to a}(FF))-(1-\omega_u) \prod_{b\in C_a^s(i)} \rho_{i\to a}(F*)) \tag{6.13}$$

These are the updates of belief propagation on the generalized state realization. Can we recover survey propagation from these updates?

The answer is yes. The following is the main theorem of [12]. Let $\omega_u+\omega_* = 1$, and let us initialize the algorithm with $\rho_{a\to i}(F*) = \rho_{a\to i}(**) = \rho_{a\to i}(\bar{F}*)$ and $\rho_{a\to i}(\bar{F}*) + \rho_{a\to i}(FF) = 1$ for all messages $\rho_{a\to i}$. Then, these relations persist throughout the execution of the algorithm, and

$$\rho_{a\to i}(FF) = \eta_{a\to i} \tag{6.14}$$

$$\lambda_{j\to a}(F*) + \lambda_{j\to a}(**) = \Pi^s_{j\to a} + \Pi^*_{j\to a} \tag{6.15}$$

$$\lambda_{i\to a}(\bar{F}*) = \Pi^u_{i\to a} \tag{6.16}$$

We show how to prove the last two equations.

1). To see that Eq. (6.16) is correct, note that because $\rho_{b\to i}(\bar{F}*) = 1 - \rho_{b\to i}(FF)$ and because $\rho_{b\to i}(F*) + \rho_{b\to i}(FF) = 1$, we have

$$\lambda_{i\to a}(\bar{F}*) = \prod_{b\in C^s_a(i)} (1-\rho_{i\to a}(FF))(\prod_{b\in C^u_a(i)} 1-(1-\omega_u) \prod_{b\in C^u_a(i)} (1-\rho_{i\to a}(FF))$$

And using the fact that $\rho_{v\to i}(FF) = \eta_{v\to i}$, we immediately have the update rule of Eq. (4.7) with $\omega_u = 0$.

2). To see that Eq. (6.15) is correct, make the substitutions $\rho_{i\to a}(F*) + \rho_{i\to a}(FF) = 1$, and $\rho_{i\to a}(**) = \rho_{i\to a}(\bar{F}*)$ into Eq. (6.13) and Eq. (6.11) and add them. Using $w^* + w_u = 1$, the result is:

$$\lambda_{i\to a}(F*) + \lambda_{i\to a}(**) = \prod_{b\in C^u_a(j)} \rho_{b\to i}(\bar{F}*) \tag{6.17}$$

On the other hand, from Eq. (4.6) and Eq. (4.8), we have

$$\Pi^s_{i\to a} + \Pi^u_{i\to a} = \prod_{b\in C^u_a(i)} (1 - \eta_{a\to i})$$

Since $1 - \eta_{a\to i} = 1 - \rho_{a\to i}(FF)$ by Eq. (6.14) and $1 - \rho_{a\to i}(FF) = \rho_{a\to i}(\bar{F}*)$ due to the initializations, it follows that:

$$\lambda_{i\to a}(F*) + \lambda_{i\to a}(**) = \Pi^s_{i\to a} + \Pi^u_{i\to a}$$

References

[1] A. Braunstein, M. Mezard, R. Zecchina, "Survey propagation: an algorithm for satisfiability," *Random Structures and Algorithms,* vol. 27, no. 2, 201–226, 2005.

[2] A. Braunstein, R. Zecchina, "Survey propagation as local equilibrium equations," *Journal of Statistical Mechanics,* 2004.

[3] O. Dubois, Y. Boufkhad, J. Mandler, "Typical random 3-SAT formulae and the satisfiability threshold," *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA '00),* 2000.

[4] U. Feige, "Relations between average case complexity and approximation complexity," *Proceedings of 34th STOC,* 2002, 534–543.

[5] D. Forney, "Codes on graphs: Normal realizations," *IEEE Transactions on Information Theory,* Vol. 47, No. 2, February 2001.

[6] B. Frey, D. MacKay, "A revolution: belief propagation on graphs with cycles," *NIPS,* 1997.

[7] F.R. Kschischang, B.J. Frey, H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory,* Vol. 47, No. 2, February 2001.

[8] E. Friedgut, "Sharp thresholds of graph properties and the k-SAT problem," Journal of the American Mathematical Society, 12:4, pp. 1017-1054, 1999.

[9] E. Maneva, E. Mossel, M.J. Wainwright, "A new look at survey propagation and its generalizations," *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms 2005 (SODA '05)*, cs.CC/0409012

[10] M. Mezard, "Passing messages between disciplines," *Science,* 301, 2003.

[11] C. Papadimitriou, *Computational Complexity,* Addison Wesley, 1994.

[12] R. Tu, Y. Mao, J. Zhao, "On generalized survey propagation: normal realization and sum-product interpretation," *Proceedings of ISIT '06.*