23 Belief Propagation and Survey Propagation

23.1 Belief Propagation, Message Passing, and Cavities

Belief propagation [162] is a fairly old strategy, popular in artificial intelligence, for using the calculus of probabilities to estimate where solutions to complicated discrete problems such as constraint satisfaction are most likely to be found. The mathematics behind it is usually not rigorous, but it offers the promise of replacing a difficult integer program with a more tractable linear or quadratic evaluation, which may often give a solution in which all the variables are in fact integers as desired, or may be accurately rounded off to the nearest integers. Belief propagation is generally performed as an iterative algorithm in which each probability, of "belief", is updated in the light of the information currently available for the beliefs of the other variables with which the variable on which the belief is based directly interact. To physicists, this sort of iterative update procedure is reminiscent of the "cavity models", of mean field theory used in early treatments of magnetic ordering. Finally, because the iterative evaluation of beliefs is carried out by passing information in messages that flow from one variable to another along the graph that is defined by the interactions in a problem, it is natural to think of distributing the calculation so that it can proceed asynchronously and in parallel without central coordination.

To make these ideas concrete, we shall show how they can be applied to constraint satisfaction, in particular to K-SAT. A recent series of papers drawing upon some general ideas from the statistical mechanics of disordered materials have given deep insight into the nature of the SAT-UNSAT phase transition [138, 139, 27, 161]. This work relies on the concept of "replicas" of the random system being studied, identical copies of that system that are studied together. The basic insight, going back to Edwards and Anderson [56], is that ordering in random systems is a matter of stability, rather than regular structure evident by its symmetry. Thus an ordered structure is observed because it forms repeatedly in the different replicas of the system. The simplest forms of order in random systems are called "replica symmetric".

The ground states of the XOR-SAT problem discussed in the previous chapter were first understood using the replica analysis. More complicated structures with additional hierarchy are now known. The clustered solutions found in the XOR-SAT problem in the hard-SAT region are one example of these. Most of the formal methods that follow this line of investigation are beyond the scope of this book, but practical methods of obtaining some of the results using message passing and iterative "cavity" evaluation of quantities called "surveys" that are similar in spirit to the standard beliefs have become available, and will be explored next, following the derivation in [12].

An iterative "belief propagation" (BP) [162] algorithm for K-SAT can be derived to evaluate the probability, or "belief", that a variable will take the value TRUE in the set of configurations that satisfy the formula considered. To calculate this, we first define a message ("transport") sent from a variable to a clause:

- $t_{i\to a}$ is *the probability that variable x_i satisfies clause a.*

In the other direction, we define a message ("influence") sent from a clause to a variable:

- $i_{a\to i}$ is *the probability that clause a is satisfied by another variable than x_i.*

In 3-SAT, where clause a depends on variables x_i, x_j, and x_k, BP gives the following iterative update equation for its influence:

$$i^{(l)}_{a\to i} = t^{(l)}_{j\to a} + t^{(l)}_{k\to a} - t^{(l)}_{j\to a} t^{(l)}_{k\to a}\,. \tag{23.1}$$

The BP update equations for the transport $t_{i\to a}$ involve the products of influences acting on a variable from the clauses that surround x_i, forming its "cavity", V_i, sorted by which literal (x_i or $\neg x_i$) appears in the clause:

$$A^0_i = \prod_{b\in V_i,\, y_{i,b}=\neg x_i} i_{b\to i} \qquad \text{and} \qquad A^1_i = \prod_{b\in V_i,\, y_{i,b}=x_i} i_{b\to i}\,, \tag{23.2}$$

with

$$y_{i,b} = \begin{cases} x_i & \text{if } x_i \text{ is part of clause } b \\ \\ \neg x_i & \text{if } \neg x_i \text{ is part of clause } b \end{cases}. \tag{23.3}$$

The update equations are then

$$t^{(l)}_{i\to a} = \begin{cases} \dfrac{i^{(l-1)}_{a\to i} A^1_i}{i^{(l-1)}_{a\to i} A^1_i + A^0_i} & \text{if } y_{i,a} = \neg x_i\,, \\ \\ \dfrac{i^{(l-1)}_{a\to i} A^0_i}{i^{(l-1)}_{a\to i} A^0_i + A^1_i} & \text{if } y_{i,a} = x_i\ . \end{cases} \tag{23.4}$$

The superscripts (l) and $(l-1)$ denote iteration. The probabilistic interpretation is as follows: suppose we have $i^{(l)}_{b\to i}$ for all clauses b connected to variable i. Each of these clauses can either be satisfied by another variable (with probability $i^{(l)}_{b\to i}$) or not be satisfied by another variable (with probability $\left(1-i^{(l)}_{b\to i}\right)$) and also be satisfied by variable i itself. If we set variable x_i to 0, then some clauses are satisfied by x_i, and some must be satisfied by other variables. The probability that they will all be *satisfied* is $\prod_{b\neq a, y_{i,b}=x_i} i^{(l)}_{b\to i}$. Similarly, if x_i is set to 1, then all these clauses b are satisfied with probability $\prod_{b\neq a, y_{i,b}=\neg x_i} i^{(l)}_{b\to i}$. The products in Eq. (23.4) can therefore be interpreted as joint probabilities of independent events. Variable x_i can be 0 or 1 in a solution if the clauses in which x_i appears are satisfied either directly by x_i itself or by other variables. Hence

$$\mathrm{Prob}(x_i) = \frac{A_i^0}{A_i^0 + A_i^1} \qquad \text{and} \qquad \mathrm{Prob}(\neg x_i) = \frac{A_i^1}{A_i^0 + A_i^1}\,. \tag{23.5}$$

23.2 Message Passing as Side Information for Decimation

To use BP for decimation, we select the variables with the largest probability to be either true or false. We then assign them their likely value, recalculate the beliefs for the reduced formula, and repeat. As with removal of 1-variables in K-XOR-SAT, decimation using BP proceeds until there are no clauses left and leaves the remaining variables untouched. The entropy is therefore given by the number of remaining variables, or 1—the "depth of decimation" plotted with the dashed line in Fig. 23.1. This method succeeds in finding satisfying configurations (ground states) of large 3-SAT formulas up to nearly $\alpha = 3.9$, which is believed to be the beginning of a Hard-SAT regime for this problem. Since we studied only large formulas, and the SAT-UNSAT transition is now understood to occur at $\alpha = 4.267\ldots$ [139], the formulas studied were almost certainly all satisfiable. On the rising part of the BP depth of decimation curve in Fig. 23.1, the decimation stops when the BP equations fail to converge. At this point, use of the full WalkSAT algorithm in the form described in the previous chapter finds a satisfying configuration in every case studied.

In the hard-SAT region, a hierarchical decomposition into clusters of solutions that are more separated from each other, as occurs in XOR-SAT, is plausible. A more complicated set of messages, designed for use in a single cluster of solutions, has been called survey propagation (SP) [27] and shown to provide a viable decimation scheme in this region. To arrive at SP we introduce a modified system of beliefs: every variable falls into one of three classes: TRUE in all solutions (1), FALSE in all solutions (0), and TRUE in some and FALSE in other solutions ($free$). Thus a decimation scheme will

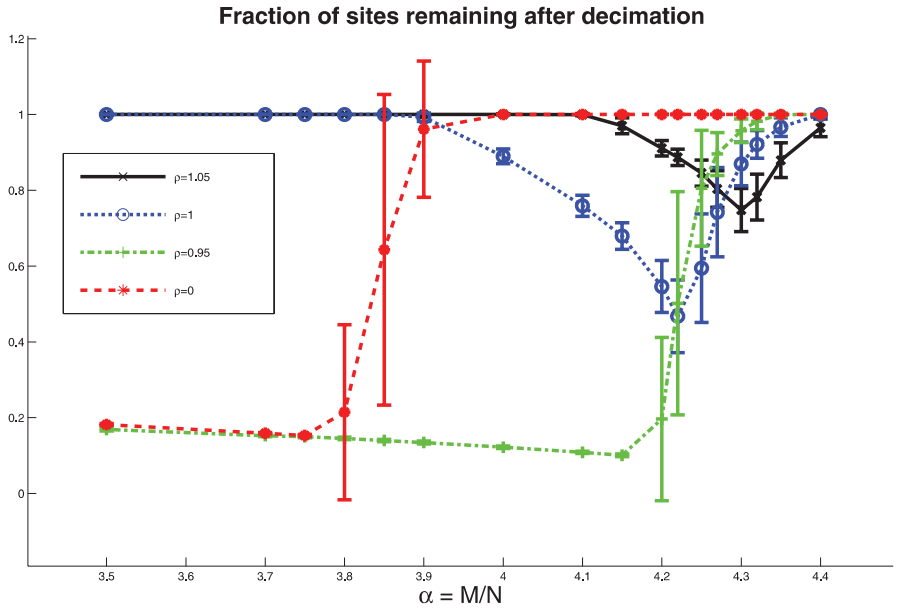

Fig. 23.1. Depth of decimation achieved by BP, SP, and two mixed cases ($\rho = 0.95$ and $\rho = 1.05$) as a function of α, the ratio of the number of clauses to the number of variables in 3-SAT (from [12])

attempt to identify the most frozen variables, those that are constant over a cluster of solutions. The message from a clause to a variable (an influence) is the the same as in BP above. Although we will again only need to keep track of one message from a variable to a clause (a transport), it is convenient to first introduce three ancillary messages:

- $\hat{T}_{i\to a}(1)$ is *the probability that variable x_i is true in clause a in all solutions*,
- $\hat{T}_{i\to a}(0)$ is *the probability that variable x_i is false in clause a in all solutions*,
- $\hat{T}_{i\to a}(free)$ is *the probability that variable x_i is true in clause a in some solutions and false in others.*

Note that there are here three transports for each directed link $i \to a$, from a variable to a clause, in the graph. As in BP, these numbers will be functions of the influences from clauses to variables in the preceeding update step. Taking again the incoming influences independently, we have

$$\begin{aligned}
\hat{T}^{(l)}_{i\to a}(free) &\propto \prod_{b\in V_i\setminus a} i^{(l-1)}_{b\to i}\,, \\
\hat{T}^{(l)}_{i\to a}(0) + \hat{T}^{(l)}_{i\to a}(free) &\propto \prod_{b\in V_i\setminus a, y_{i,b}=x_i} i^{(l-1)}_{b\to i}\,, \\
\hat{T}^{(l)}_{i\to a}(1) + \hat{T}^{(l)}_{i\to a}(free) &\propto \prod_{b\in V_i\setminus a, y_{i,b}=\neg x_i} i^{(l-1)}_{b\to i}\,.
\end{aligned} \tag{23.6}$$

The proportionality indicates that the probabilities are to be normalized. We see that the structure is quite similar to that in BP. But we can make it closer still by introducing $t_{i\to a}$ with the same meaning as in BP. In SP it will then, as the case might be, be equal to $T_{i\to a}(free)+T_{i\to a}(0)$ or $T_{i\to a}(free)+T_{i\to a}(1)$. That gives [cf. Eq. (23.4)]:

$$t_{i\to a}^{(l)} = \begin{cases} \dfrac{i_{a\to i}^{(l-1)} A_i^1}{i_{a\to i}^{(l-1)} A_i^1 + A_i^0 - A_i^1 A_i^0} & \text{if } y_{i,a} = \neg x_i \,, \\[2ex] \dfrac{i_{a\to i}^{(l-1)} A_i^0}{i_{a\to i}^{(l-1)} A_i^0 + A_i^1 - A_i^1 A_i^0} & \text{if } y_{i,a} = x_i \,. \end{cases} \tag{23.7}$$

The update equations for $t_{i\to a}$ are the same in SP as in BP, i. e., one uses Eq. (23.1) in SP as well. Similarly to Eq. (23.5), decimation now removes the most fixed variable, i. e., the one with the largest absolute value of $(A_i^0 - A_i^1)/(A_i^0 + A_i^1 - A_i^1 A_i^0)$. Given the complexity of the original derivation of SP [138, 139], it is remarkable that the SP scheme can be interpreted as a type of belief propagation in another belief system. And even more remarkable is the fact that the final iteration formulas differ so little.

A modification of SP that we will consider in what follows is to interpolate between BP ($\rho = 0$) and SP ($\rho = 1$)[1] by considering

$$t_{i\to a}^{(l)} \propto \begin{cases} \dfrac{i_{a\to i}^{(l-1)} A_i^1}{i_{a\to i}^{(l-1)} A_i^1 + A_i^0 - \rho A_i^1 A_i^0} & \text{if } y_{i,a} = \neg x_i \,, \\[2ex] \dfrac{i_{a\to i}^{(l-1)} A_i^0}{i_{a\to i}^{(l-1)} A_i^0 + A_i^1 - \rho A_i^1 A_i^0} & \text{if } y_{i,a} = x_i \,. \end{cases} \tag{23.8}$$

We do not have an interpretation of the intermediate cases of ρ as belief systems.

Figure 23.1 shows the depth of decimation resulting from SP as well as the variants that are shifted by use of the ρ parameter slightly in the direction of BP and slightly away from BP ("overrelaxed"). We see that the SP decimation is effective only in the hard-SAT region, since below $\alpha = 3.9$ it considers all variables as "free" to take either value in the configurations that can be constructed. In the hard-SAT region, it has the effect of removing the core variables, leaving WalkSAT with a problem to solve that is no more difficult than finding ground states at $\alpha = 3.9\ldots$, the entry to the hard-SAT

[1] This interpolation has also been considered and implemented by Zecchina and coworkers.

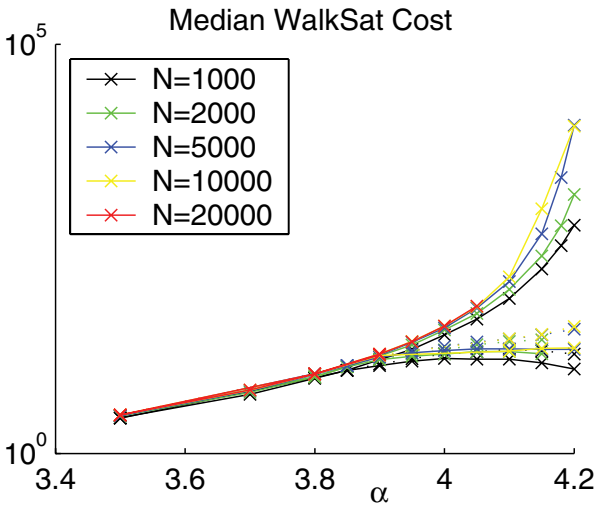

Fig. 23.2. Median cost of a WalkSAT solution close to the SAT-UNSAT transition, with and without first performing the SP-induced decimation to reduce the complexity of the formula (from [12])

region. This is suggested by the cost data contained in Fig. 23.2. Setting the parameter ρ to 0.95 produces an interesting result. The decimation proceeds to completion, just as in BP, but the method continues working in the hard-SAT region of the parameter space. Overrelaxed SP ($\rho = 1.05$) gives what may be reasonable recommendations even above the SAT–UNSAT boundary, where these methods can be used to arrive at configurations with the smallest number of unsatisfied clauses.

23.3 Belief Propagation and Sudoku

The Sudoku puzzle that we showed in Fig. 20.1 poses a difficult challenge for stochastic methods of resolving constraint problems. The puzzles are generated in apparently limitless quantities on popular Web sites. In our study, we made use of puzzles from `www.websudoku.com`. Programs that automatically solve the puzzles exist and are used in the filtering process that leads to automatic generation of good puzzles, but to the best of our knowledge all employ a long list of complex logic rules, followed by exhaustive search and backtracking to explore the hardest cases.

The puzzles are graded by level of difficulty. “Easy” puzzles typically have about 35 squares already filled in. “Medium” puzzles will have 32 clues. One can solve puzzles at these levels by repeatedly using the rule that no square can have a value that is already taken by another square in the same row, column, or 3×3 square. As soon as you discover a square that has only

one possible value available to it, you insert that value and see what other discoveries the assignment may enable. A second rule is essential. If a square is the only place in a given row, column, or 3×3 square that can take on a particular value, then it must be assigned that value. These two rules can solve any "easy" or "medium" Sudoku puzzle, and many "hard" puzzles as well. "Hard" puzzles typically have 28 to 30 squares filled in initially. "Evil", sometimes called "diabolical", puzzles start with 24 to 26 squares filled in, and these two simple rules seldom provide more than one or two additional assignments.

A plausible way to develop a stochastic Sudoku solver is to use belief propagation to replace a hard discrete optimization problem with a softer problem using real numbers. For each square, we might calculate the nine probabilities that the square contains each one of the nine allowed numbers. If we let $P(i, j)$ be the probability of square j taking value i, then we can evaluate this probability as the product of the probabilities that no other square in the same row, column, or 3×3 square will take value i:

$$P(i,j) = \prod_{k \text{ in ngbhd of } j} (1 - P(i,k)) \,. \tag{23.9}$$

After evaluating Eq. (23.9) for each of the nine choices of i, we normalize the results so that the probabilities in a single square sum to unity. Unfortunately, this takes advantage of only one of the two basic rules. After iterating a while, we may find that the total probability of finding a "3" somewhere in a given row does not sum to unity but can take almost any value. Since we have already normalized the beliefs after iterating with Eq. (23.9), there is no convenient place left to incorporate the second rule in the evolution of the probabilities. Use of decimation, however, gets around some of this difficulty [22].

After iterating the beliefs to convergence, one selects the square and the value that are most strongly indicated and asserts that value in that square. One can augment this procedure with rules to prevent making assignments that are manifestly wrong (e. g., assigning the same number to two squares in the same row, column, or 3×3 square). The result is much stronger than simply applying the two basic rules to eliminate choices and solves nearly all "easy" to "hard" puzzles, and some "evil" puzzles as well. Probably the fact that the use of only nine numbers for the squares is now built in adds some of the effects of the second logical rule, which forces values when they cannot be assigned elsewhere in their local neighborhood.

6	7	8	2	5	1	3	9	4
3	1	4	7	9	8	2	6	5
2	5	9	3	4	6	7	1	8
9	2	1	4	8	7	5	3	6
7	3	6	1	2	5	8	4	9
8	4	5	6	3	9	1	2	7
5	8	2	9	6	3	4	7	1
4	9	7	5	1	2	6	8	3
1	6	3	8	7	4	9	5	2

Fig. 23.3. Solution to the puzzle posed in Fig. 20.1. A human puzzle solver solved this "evil" puzzle in only 12 min. It appears to be at about the limit for automatic solution (using belief propagation) today (March 2006)

The puzzle in Fig. 20.1 was solved by a practiced human Sudoku solver in 12 min (Fig. 23.3) and by belief propagation in seconds, but the belief propagation program needed several tries to find a successful solution. The program solved another "evil" puzzle on the first try, while the human solver required 14 min to solve it. This work is still in progress, but it appears likely that these extremely complicated logical inference puzzles will yield to stochastic optimization with modest further effort.