MICHAEL RUDERER

BELIEF PROPAGATION

[July 8, 2018 at 22:03 – classicthesis version 1.0]

BELIEF PROPAGATION

MICHAEL RUDERER

Seminararbeit

Juli 2018 – version 1.0

[July 8, 2018 at 22:03 – classicthesis version 1.0]

ABSTRACT

Belief Propagation is a probabilistic method for approximating marginal distributions in factor graphs. For satisfiable SAT formulas this method can be used as a heuristic for finding satisfying assignments that significantly outperforms conventional algorithms.

CONTENTS

1 INTRODUCTION 1

2 FACTOR GRAPHS AND MESSAGE PASSING 3

2.1 Factor Graphs 3

2.1.1 Factor Graph of a SAT Formula 4

2.2 Message Passing Algorithms on Trees 5

2.3 Message Passing on General Graphs 6

3 BELIEF PROPAGATION FOR SAT FORMULAS 9

3.1 Warning Propagation 9

3.1.1 Propagation Algorithm 9

3.1.2 Decimation Algorithm 11

3.2 Belief Propagation 12

3.2.1 Propagation Algorithm 12

3.2.2 Marginal Probabilities and Valid Assignments 16

4 CONCLUSION 19

BIBLIOGRAPHY 21

1 INTRODUCTION

As many common problems naturally reduce to finding solutions of boolean formulas, SAT solvers are frequently used in practice and there is a need for finding fast and precise algorithms. The main part of this seminar thesis is Section 3 where two probabilistic methods for heuristically finding assignments of satisfiable SAT formulas are described. These methods are called *Warning Propagation* and *Belief Propagation*. Both are so called *message passing* algorithms which are explained in section 2 together with basic terms and notations.

The belief propagation algorithm is a general method for probabilistic inference problems and was proposed by Judea Pearl in 1982. The generic algorithm is used in many different contexts like coding theory or image processing. It can be applied to SAT formulas by approximating marginal probabilities over the formula's solutions. These probabilities then yield valid variable assignments.

A belief propagation approach was used in the SAT solver "Dimetheus", which won gold medals in 2014 and 2016 in the international SAT competition (see [3]) for efficiently solving randomly generated satisfiable instances.

Section 2 starts with defining the factor graph representation of a SAT formula and describes the general procedure of message passing on factor graphs.
In section 3 the two algorithms are presented and demonstrated on an example formula.

2

FACTOR GRAPHS AND MESSAGE PASSING

2.1 FACTOR GRAPHS

For problems that include complex relations among many variables it is useful to have an abstract representation of how those variables interact with each other. So called *factor graphs* are such representations.

In general, a factor graph represents the structure of a function's factorization into smaller functions.
If a function $f(X_1, \ldots, X_n)$ can be written as a product $\prod_{j=1}^{m} f_j(S_j)$ where the functions f_j have smaller inputs $S_j \subset X$, its factorization can be expressed by a factor graph: The graph has two types of nodes: *Variable nodes* that correspond to the variables X_i and *factor nodes* corresponding to the functions f_j. An edge connects a variable node X_i to a factor node f_j if and only if X_i is part of f_j's input. This means the graph is an undirected bipartite graph with the node set $V = \{X_1, \ldots, X_n\} \cup \{f_1, \ldots, f_n\}$ and edge set $E = \{(X_i, f_j) \mid X_i \in S_j\}$.
In many applications the global function f is a joined probability distribution that can be factorized by using information about independence between the variables. Typical tasks on factor graphs are computing variable assignments that maximize or minimize f or computing marginal distributions if f is a probability distribution. Both of these will be done in Section 3.

Example. *For random variables X_1, X_2 and X_3 their joined probability distribution f is defined as $f(x_1, x_2, x_3) := P(X_1 = x_1 \wedge X_2 = x_2 \wedge X_3 = x_3)$. f can be factorized using the definition of conditional probabilities:*

$$f(x_1, x_2, x_3) = P(X_1 = x_1) * P(X_2 = x_2 \wedge X_3 = x_3 \mid X_1 = x_1)$$

This factorization still contains a function depending on all 3 variables and is not useful. If X_2 and X_3 are known to be conditionally independent given X_1 this unwanted function can itself be factorized:

$$f(x_1, x_2, x_3) = \underbrace{P(X_1 = x_1)}_{f_1(x_1)} * \underbrace{P(X_2 = x_2 \mid X_1 = x_1)}_{f_2(x_1, x_2)} * \underbrace{P(X_3 = x_3 \mid X_1 = x_1)}_{f_3(x_1, x_3)}$$

In this factorization of f each factor's input is smaller than the original one.

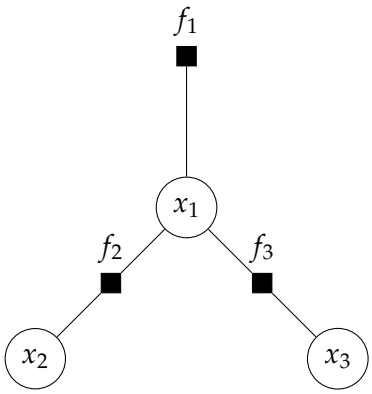


Figure 2.1: Factor graph of f's factorization into f_1, f_2, f_3
Usually, variable nodes are drawn as circles whereas constraint nodes are drawn as rectangles to easily distinguish the types of nodes

Factor graphs can also be used for describing constraint satisfaction problems. A factor corresponds to a constraint on its neighbour vertices, it evaluates to 1 if the constraint is satisfied and to 0 if not. For the global function f - the product of all factors - to be 1, every constraint has to be is satisfied. The special case of SAT problems is discussed in the following chapter.

2.1.1 *Factor Graph of a SAT Formula*

Any SAT formula over the variables $x_1, \ldots, x_n$ in CNF form can be interpreted as a function that factorizes into the formulas clauses.
$x_1, \ldots, x_n$ are boolean variables which can be either 0 or 1. A SAT formula $\mathcal{F}$ is a conjunction / product of special constraints, the formula's *clauses*. A clause is a disjunction of variables and their negations $(x_i \vee \overline{x_j} \vee \ldots)$ and is satisfied if at least one variable x_i that appears unnegated is set to 1 or at least one x_j that appears negated is set to 0.
In the factor graph of a formula each factor node a represents the local function defined by a single clause. If the variable x_i or its negation $\overline{x_i}$ appears in this clause the factor graph contains an edge between a and the variable node i.

In following descriptions the letters a and b always describe factor nodes, i and j are used for variable nodes. Also often no distinction is made between the nodes of the factor graph and the corresponding elements of the formula. Especially from here on variables are denoted as single letters i, j whereas the value of a variable i is written as $x_i \in \{0, 1\}$

In [1] some additional notation is defined to simplify the description of the algorithms in section 3:

Definition. *Let a be a factor node and i one of its variables.*

- *The value J_i^a is 1 if i appears negated in a and -1 if not*
- *$V(a)$ is the set of all variables in a*
  $V(i)$ is the set of all clauses containing i.
  Alternatively $V(a)$ and $V(i)$ are the neighbourhoods of a and i in the factor graph.
- *$V_+(i)$ is the set of clauses in which i appears unnegated*
  $V_-(i)$ is the set of clauses that contain i in negated form.
- *$V_a^u(i)$ is the set of i's neighbours $b \neq a$ with $J_i^b \neq J_i^a$*
  $V_a^s(i)$ is the set of neighbours $b \neq a$ with $J_i^b = J_i^a$
  The letters u, b stand for (**un**)**s**atisfied *meaning that if i satisfies a clause $b \in V_a^s(i)$, it also satisfies a.*

Example. *Let*

$$\mathcal{F} = \underbrace{(x_1 \vee x_2 \vee x_3)}_{a} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee x_4)}_{b} \wedge \underbrace{(\overline{x_3} \vee x_4)}_{c} \wedge \underbrace{(\overline{x_1} \vee \overline{x_2})}_{d}$$

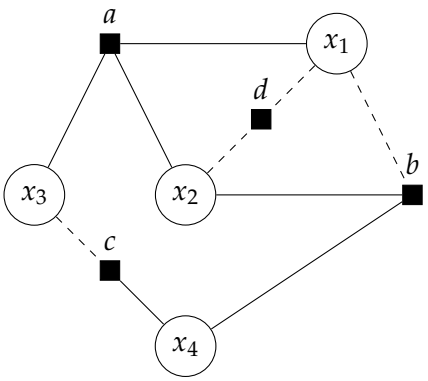


Figure 2.2: Factor graph of a $\mathcal{F}$

Again the circles are variable nodes, the rectangles are constraint nodes. If x_i appears negated in the clause a (if $J_a^i = 1$), the edge is drawn dashed. With the addition of dashed lines the factor graph completely describes the formula $\mathcal{F}$.

The variable set of b is for example $V(b) = \{1, 2, 4\}$. The clause sets of x_1 are $V(1) = \{a, b, d\}$, $V_+(1) = \{a\}$, $V_-(1) = \{b, d\}$ and $V_a^u(1) = \{b, d\}$, $V_a^s(1) = \emptyset$.

2.2 MESSAGE PASSING ALGORITHMS ON TREES

If the factor graph is a tree, many problems can be solved efficiently using a form of dynamic programming called *message passing*.
In general, message passing algorithms compute values for each edge of the factor graph. These values can be interpreted as *messages* that

are sent between the nodes. Since all edges connect factor nodes to variable nodes there can be two types of messages: messages passed from a variable i to a factor a, denoted as $\mu_{i\to a}$ and messages passed from a to i, denoted as $\mu_{a\to i}$.
The messages must be defined so that a message $\mu_{a\to i}$ is determined by the messages $\mu_{j\to a}$ that a received from neighbour variables $j \neq i$. The same must hold for $\mu_{i\to a}$.
In the probability context the massages $\mu_{i\to a}$ are often functions and are obtained by summing over configurations of variables and multiplying the affiliated messages. The fundamental equation often has the general form

$$\mu_{a\to i} = \sum \prod_{j\in V(a)\setminus i} \underbrace{\prod_{b\in V(j)\setminus a} \mu_{b\to j}}_{\mu_{j\to a}}$$

Therefore these types of algorithms are called *sum-product*-algorithms. The reasoning behind this form and an exact formula is explained in the description of the BP algorithm. The rule is formulated so that messages sent from factors to variables only appear as intermediate results in the computation of messages sent from variables to factors. For tree factor graphs which do not contain cycles the value of $\mu_{a\to i}$ does not influence its predecessors $\mu_{j\to a}$. The messages can therefore be computed sequentially starting with the factor graph's leaves.

2.3 MESSAGE PASSING ON GENERAL GRAPHS

If a graph contains cycles, the above described messages are in general not well-defined. However, message passing algorithms which are correct for trees can be used as a heuristic for general graphs.
This *loopy* form of message passing consists of two parts. In the initialization step each message is provisionally assigned a random value. Now that each message is set the sum-product equation can be used not to compute the absolute result but to repeatedly update the provisional values. In each update step the update rule is applied to all edges $a \to i$. There are several variants how to schedule these update tasks. For example the messages could be updated in parallel (*synchronous*) or step by step (*asynchronous*) in a fixed order. Here, the order of updates is chosen uniformly at random in each update step. If a message changed its value in the beginning of the step the following updates can instantly access and use its new value.

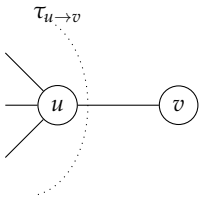
The goal is to reach a point where no message would significantly change when applying the update rule to it. If this is the case, the messages are said to have *converged*. A point of convergence is also called a *fixed point*. Wether a fixed point is found does not only depend on the input graph but also on the chosen starting values and the update schedule. Since there is no guarantee that the messages

will converge it is possible for the update process to never terminate. In this case the computation has to be terminated after a fixed number of steps. Also an instance can have multiple fixed points that are not the desired solution. The converged messages are therefore only a heuristic solution if the factor graph contains cycles.

On trees this heuristic approach always converges to a unique set of fixed point warnings. The convergence for trees, like other properties of message passing algorithms, can be shown by induction on the *level* of a message.

Definition. *Let (u, v) be an edge of a tree τ.*
The tree $\tau_{u\to v}$ is the component of $\tau \setminus (u, v)$ that still contains u. The level *of the directed edge $u \to v$ is the height of u in $\tau_{u\to v}$.*

The level can be thought as the length of the longest path (the longest possible chain of messages) in τ that ends in u and does not pass v.



Lemma. *[1] On a tree factor graph the message on an edge $a \to i$ at level r converges after at most $r/2 + 1$ update steps.*

Proof. Induction on r:

If $r = 0$, a must be a leaf. Since there are no incoming messages $j \to a$ the message $j \to i$ is constant at every step.
Similarly for $r = 1$ all ingoing edges have level 0 and $m_{a\to i}$ is constant from step 0.

If $r \geq 2$ the message $m_{a\to i}$ is determined by the set of messages $m_{b\to j}$ sent to adjacent variables j. These messages are sent on edges with level $\leq r - 2$ and have by induction converged on step $\frac{r-2}{2} + 1 = r/2$. $m_{a\to i}$ converges in the next update step. □

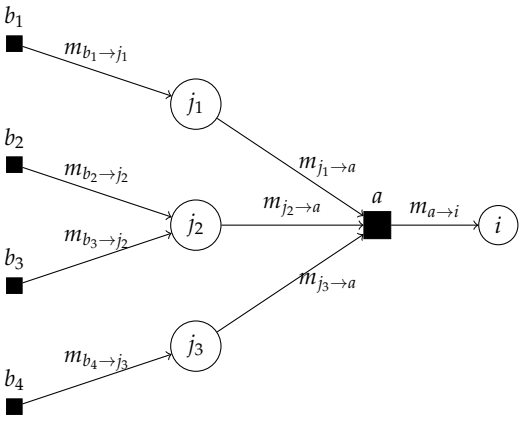


Figure 2.3: Subgraph of a factor tree with all messages required for computing $m_{a\to i}$

3

BELIEF PROPAGATION FOR SAT FORMULAS

In the following section two algorithms for SAT will be presented that both use the factor graph representation of a SAT formula. The description of both algorithms is based on [1].

3.1 WARNING PROPAGATION

The messages used in the *Warning Propagation* Algorithm (WP) presented in [1] are called *warnings*. A warning $u_{a\to i} \in \{0,1\}$ is passed from clause a to variable i. A converged warning $u^\star_{a\to i}$ with value 1 should indicate, that to satisfy the clause a, the variable i has to take the value 1 if $j \in V_+(a)$ or 0 if $j \in V_-(a)$. Equivalently if i does not satisfy a, the entire formula can not be satisfied. The warning $u^\star_{a\to i}$ will *fix* the variable i.

3.1.1 *Propagation Algorithm*

Like the general algorithm described in section 3.2.1 warning propagation is correct on trees and can be used as a heuristic for cyclic graphs by randomly initializing the warnings and hoping for convergence.

The algorithm starts by assigning each warning $u_{a\to i}$ a random starting value and updates these provisional warnings until their values have converged to a set of fixed point warnings $u^\star_{a\to i}$ or until the number of iterations has exceeded some limit t_{max}.

The general idea is that the clause a has to fix the variable i only if all of its other variables $j \in V(a) \setminus i$ are already fixed to values that do not satisfy the clause a.
The first step in the update procedure is to compute for each $j \in V(a) \setminus i$ the so called *cavity field* $h_{j\to a}$ that indicates what value j should take in the subproblem defined by $\tau_{j\to a}$. To compute $h_{j\to a}$ one has to count how many of the clauses $b \neq a$ fix j to 1 and how many fix j to 0:

$$h_{j\to a} = \sum_{b\in V_+(j)\setminus a} u_{b\to j} - \sum_{b\in V_-(j)\setminus a} u_{b\to j} = -\sum_{b\in V(j)} J_j^b u_{b\to j}$$

The clauses $b \in V_+(j)$ are the ones that would fix j to 1 if their warnings are active, the clauses $b \in V_-(j)$ would fix j to 0. So if $h_{j\to a}$ is positive the variable j tends to the value 1. If the cavity field is negative it tends to 0. If $h_{j\to a} = 0$ which includes the case $V(j) \setminus a = \emptyset$ no conclusion can be made.

When all cavity fields are computed, each variable j with $h_{j\to a} \neq 0$ has a preferred value. This preferred value either makes the clause a satisfied or does not contribute to its satisfaction. If all variables $j \in V(a) \setminus i$ prefer a non satisfying value, the clause a sends a warning to i, meaning that i should take the satisfying value.

This warning can be computed by

$$u_{a\to i} = \prod_{j\in V(a)\setminus i} \theta(h_{j\to a} J_j^a) \quad \text{with } \theta(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{otherwise} \end{cases}$$

The factor $\theta(h_{j\to a} J_j^a)$ is 1 if j prefers to violate a and 0 if not:

If j has no preferred value its cavity field is 0 and $\theta(h_{j\to a} J_j^a) = 0$ meaning no warning will be sent.

If the preferred value of j satisfies a, J_j^a and $h_{j\to a}$ have different signs and
$\theta(h_{j\to a} J_j^a) = \theta(-1)$ is again 0.

If the preferred value of j *violates* a, J_j^a and $h_{j\to a}$ have the same sign and $\theta(h_{j\to a} J_j^a) = 1$. If this is the case for all $j \neq i$ the product evaluates to 1 and a sends a warning to i.

```
Warning Propagation Algorithm
  Input: A CNF formula and its factor graph

0. Randomly initialize all warnings u_{a→i} ∈_R {0,1}

1. For t = 0 to t = t_max
        1.1 Compute in random order for all edges (a, i)
            u_{a→i} := ∏_{j∈V(a)\i} θ( −Σ_{b∈V(j)\a} (J_j^a J_j^b) u_{b→j} )
        1.2 If no message has changed goto 2.
2. If t = t_max return UN-CONVERGED, else return the
    generated warnings u*_{a→i}
```

The following lemma shows that - on trees - the computed messages indeed serve the purpose described in the first paragraph: If the algorithm successfully returns a set of converged warnings, each warning $u^\star_{a\to i} = 1$ fixes the variable i to the value satisfying a:

Lemma. *[1] Let $a \to i$ be an edge of level r.*
If $u^*_{a\to i} = 1$ the SAT formula defined by $\tau_{a\to i}$ is not satisfiable.

Proof. Induction on r.

- If $r = 0$, a is a leaf and $\tau_{a\to i}$ defines a formula that consists of one empty clause. By definition, an empty disjunction evaluates to 0.
- If $r = 1$ the message $u^*_{a\to i}$ cannot be 1: a contains one or more variables $j \neq i$ which all are leaf nodes that pass the message $m_{j\to a} = 0$ to a. The update equation yields $m_{a\to i} = \prod_j \theta(0) = 0$.

- Let $r \geq 2$ and $u^*_{a\to i} = 1$. For each variable $j \neq i$ of a the value $J^a_j h_{j\to a}$ must be positive. $h_{j\to a}$ was defined as $-\sum_{b\in V(j)\setminus a} J^b_j u^*_{b\to j}$. This means there is at least one b with $J^a_j(-J^b_j u^\star_{b\to j}) = 1$ or equivalently $J^b_j \neq J^a_j$ and $u^\star_{b\to j} = 1$.
  By induction the subproblem $\tau_{b\to j}$ is not satisfiable in any assignment in which j supports a. To satisfy a one must violate at least one clause in $\tau_{b\to j}$.

□

3.1.2 *Decimation Algorithm*

The algorithm that actually computes satisfying assignments based on the WP results is called *Warning Inspired Decimation* (CID). CID uses WP as a subprocedure. For a satisfiable formula $\mathcal{F}$ after running WP, a partial assignment is computed that can be applied to $\mathcal{F}$ to obtain a smaller formula. Running CID recursively on that smaller formula gives a complete satisfying assignment. CID detects after the first run of WP if $\mathcal{F}$ is not satisfiable.

To determine if a formula is satisfiable one has to compute certain values using the converged WP messages.

The *local field* of a variable i is defined as $H_i := -\sum_{b\in V(i)} J^b_i u^*_{b\to i}$.
H_i is computed similar to $h_{i\to a}$ in the WP algorithm. While $h_{i\to a}$ gave the current tendency of i when ignoring a, the local field of i uses the converged messages and counts how often i is fixed to 1 or 0 by all of its neighbours (including a).
The *contradiction number* c_i of a variable is 1 if i gets fixed to different values and 0, if not. c_i can be obtained from the converged WP messages. If after running WP a variable has a contradiction number of 1 the formula is unsatisfiable.
The opposite direction is also true: If no contradiction appears, the formula is satisfiable and CID returns a valid assignment. After one run of WP a partial assignment can be obtained by setting all variables with active warnings to the value they were fixed to: 1 if $H_i > 0$ and 0 if $H_i < 0$. If i received at least one warning, H_i can not be 0 because all incoming warnings fix i to the same value. If no variable received a warning (all H_i are 0), CID chooses a random variable and sets it to a random value.
Now the factor graph is modified (*cleaned*) according to the partial assignment. The graph is cleaned by removing all fixed variables and all clauses that are satisfied by them. The modified formula will be satisfiable and cause no contradictions in the following WP runs.

```
Warning Inspired Decimation
  Input: A CNF formula and its factor graph
1. While there are unfixed variables
        1.1 Run WP
        1.2 If WP does not converge, return UN-CONVERGED.
             Else compute the numbers H_i and c_i
        1.3 If at least one c_i is 1 return UNSAT.
        1.4 Choose a partial assignment and clean the
            graph
2. Return the complete assignment
```

3.2 BELIEF PROPAGATION

The Warning Propagation algorithm is able to solve the decision problem and to compute valid assignments of SAT formulas. A different message passing method called the *Belief Propagation* algorithm is able to determine the actual number of satisfying assignments and the fraction of those assignments where variables are restricted to fixed values.

The messages sent in the BP algorithm are conditional probabilities $\mu \in [0,1]$. The corresponding events are taken from the probability space built by all configurations X of the boolean variables $x_1, \ldots, x_n$. The used probability measure is a uniform distribution so for a SAT formula over n variables each configuration has probability $P(X) = 2^{-n}$.
This probability space can be restricted to only those configurations that satisfy a boolean formula $\mathcal{F}$. If there are $\mathcal{N} > 0$ satisfying assignments this conditioned probability measure factorizes to

$$P(X \mid X \text{ satisfies } \mathcal{F}) = \mathcal{N}^{-1} \prod_{a \in A} f_a(X)$$

where A is the set of clauses in $\mathcal{F}$ and $f_a(X)$ the characteristic function of a clause a which is 1 if X satisfies a and 0 if not.
The fraction of satisfying assignments where for example $x_1 = 1$ or $x_4 = 0$ can now be viewed as a marginal of the factorized function.

In general, belief propagation is a message passing technique for approximating marginals of factorized probability distributions. In this chapter the BP equations and their interpretations in the SAT context are explained and demonstrated on an example formula.

3.2.1 *Propagation Algorithm*

The basic procedure of sending and updating messages between nodes of the function's factor graph stays the same as in WP. Instead of warnings, the BP messages are conditional probabilities. From the

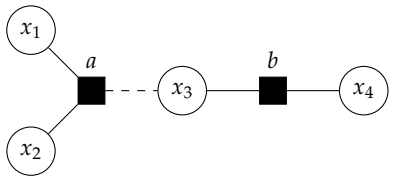
converged probabilities one can obtain marginals and the function's normalization constant $\mathcal{N}$. With this information one gets the total number of satisfying assignments as well as the number of assignments where a single variable has a fixed value.

3.2.1.1 *Messages*

The message $\mu_{a\to i}(x_i)$ sent from a factor a to a variable i is meant to be the probability that a and all the factors behind a are satisfied conditioned on i taking the value x_i.

The message $\mu_{i\to a}(x_i)$ sent in the opposite direction is the probability that i takes the value x_i in an assignment that satisfies $\tau_{i\to a}$.

Example. *Let* $\mathcal{F} = \underbrace{(x_1 \vee x_2 \vee \overline{x_3})}_{a} \wedge \underbrace{(x_3 \vee x_4)}_{b}$.

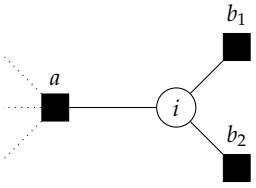


The messages exchanged between b and x_3 can be computed using only the definition:
For example $\mu_{b\to 3}(1)$ is the probability that $(x_3 \vee x_4)$ is satisfied by an assignment where $x_3 = 1$. Its value is 1 because each such assignment satisfies b. If $x_3 = 0$ however, only those assignments where $x_4 = 1$ will satisfy b so $\mu_{b\to 3}(0) = 0.5$.
Out of all 14 configurations satisfying $(x_1 \vee x_2 \vee \overline{x_3})$ there are 8 where $x_3 = 0$ and 6 where $x_3 = 1$, so $\mu_{3\to b}(0) = \frac{4}{7}$, $\mu_{3\to b}(1) = \frac{3}{7}$.

This method of counting desired configurations in the whole configuration space is infeasible for larger graphs. Instead the probabilities are obtained by message passing.

3.2.1.2 *Update Rules*

$\mu_{i\to a}(x)$, the message passed to a, tells the probability that i is $x_i = x$ considering only the factors on i's side of the graph.
The node i receives information about these factors through its neighbour vertices $b_i \neq a$. The incoming messages $\mu_{b_i\to i}(x)$ tell, if the factors are satisfied conditioned on $x_i = x$.



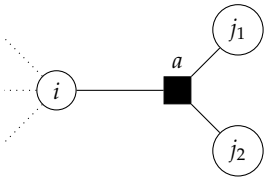
Using Bayes' theorem and the fact that the messages $\mu_{b_i\to i}$ are condi-

tionally independent, one can write the outgoing message as a product of the incoming ones:

$$\mu_{i\to a}(x) = P(x_i = x \mid \tau_{i\to a} \text{ is sat.}) = \frac{P(x_i = x)P(x_i = x \mid \tau_{i\to a} \text{ is sat.})}{P(\tau_{i\to a} \text{ is sat.})}$$
$$= \underbrace{\frac{P(x_i = x)}{P(\tau_{i\to a} \text{ is sat.})}}_{C_{i\to a}} \prod_{b\in V(i)\setminus a} \mu_{b\to i}(x)$$

The factor $C_{i\to a}$ is the same for $x = 0$ and $x = 1$ so it suffices to compute the products for $x_i = 0$ and $x_i = 1$ and then to normalize so that $\mu_{i\to a}(0) + \mu_{i\to a}(1) = 1$. The usual notation to indicate the implicit existence of a normalization factor is $\mu_{i\to a}(x) \propto \prod \mu_{b\to i}(x)$ ([4]).

To compute $\mu_{a\to i}(x)$ one has to sum over all configurations X of the variables in a where $x_i = x$. If a configuration does not satisfy a, it contributes nothing to $\mu_{a\to i}(x_i)$. If it does, one has to compute the probability that it also satisfies the other factors in $\tau_{a\to i}$. The probability that a configuration where j takes the value x_j satisfies the factors behind the neighbour variable j is $\mu_{j\to a}(x_j)$. Again these probabilities are independent and the probability for all factors is simply the product.



$$\mu_{a\to i}(x_i) = \sum_{\sim\{x_i\}} f_a(X) \prod_{j\in V(a)\setminus i} \mu_{j\to a}(x_j)$$

The expression $\sim \{x_i\}$ means that the sum goes over all possible configurations X of a's variables with the exception of the value of x_i which is always the same.

Example. *Now the messages from the first example can be computed with belief propagation. Since the formula's factor graph is a tree, the updates can be scheduled so that each message has to be computed only once.*

- *Level 0 (Leaves)*

  The leaves are all variable nodes. As they receive no incoming messages they will all send the same constant message. The empty product in the update rule evaluates to 1*, after normalization the values are* $\mu_{1\to a}(0) = \mu_{1\to a}(1) = 1/2$*. The same holds for* $2 \to a$ *and* $4 \to b$*.*

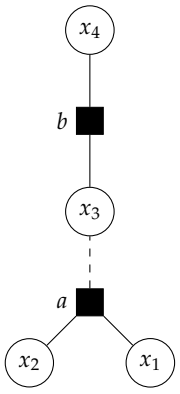
- *Level 1. The message of level* 1 *are both sent from factors to variables:*

$$\begin{aligned}\mu_{a\to 3}(0) =& \mu_{1\to a}(0)\mu_{2\to a}(0) + \mu_{1\to a}(0)\mu_{2\to a}(1) + \mu_{1\to a}(1)\mu_{2\to a}(0)\\ &+\mu_{1\to a}(1)\mu_{2\to a}(1) = 1\\ \mu_{a\to 3}(1) =& \mu_{1\to a}(0)\mu_{2\to a}(1) + \mu_{1\to a}(1)\mu_{2\to a}(0) + \mu_{1\to a}(1)\mu_{2\to a}(1)\\ =& 3/4\\ \mu_{b\to 3}(0) =& \mu_{4\to b}(1) = 1/2\\ \mu_{b\to 3}(1) =& \mu_{4\to b}(0) + \mu_{4\to b}(1) = 1\end{aligned}$$

- *Level 2. The unnormalized values for* $3 \to b$ *are* 0.5 *and* 1. *After normalization one gets* $\mu_{3\to b}(0) = \frac{0.5}{0.5+1} = \frac{1}{3}$ *and* $\mu_{3\to b}(1) = \frac{2}{3}$. *The values* $\mu_{3\to a}(0) = \frac{1}{1+0.75} = \frac{4}{7}$ *and* $\mu_{4\to a}(1) = \frac{3}{7}$ *are obtained in the same way.*

The values for the remaining 3 *edges of level* 4 *can be looked up in the tabular*

Edge	$\mu(0)$	$\mu(1)$
$1 \to a$	$1/2$	$1/2$
$2 \to a$	$1/2$	$1/2$
$4 \to b$	$1/2$	$1/2$
$a \to 3$	1	$3/4$
$b \to 3$	0.5	1
$3 \to a$	$1/3$	$2/3$
$3 \to b$	$3/7$	$4/7$
$a \to 2$	$5/6$	1
$a \to 1$	1	$5/6$
$b \to 4$	$4/7$	1



The equations described above are the general BP update rules that can be applied not only to SAT problems but to any problem that is solved by belief propagation. In [1] and [4] these standard rules are expressed in a different, but equivalent way to make computation easier.
Instead of computing functions where $\mu_{i\to a}(0)$ and $\mu_{i\to a}(1)$ add up to 1, one can also compute the value $\gamma_{i\to a}$, the probability that x_i violates a. It can be expressed as $\gamma_{i\to a} = \begin{cases}\mu_{i\to a}(0) & \text{if } J_a^i = -1\\ \mu_{i\to a}(1), & \text{if } J_a^i = 1\end{cases}$

In the original BP equation for $\mu_{a\to i}(x_i)$ one has to sum over all satisfying configurations of a's variables $j \neq i$. Since there is only one configuration that does not satisfy a it is easier to just compute the probability of this single configuration. The message $\delta_{a\to i}$ is the probability that all of a's neighbours are in a state that violates a (conditioned on $\tau_{a\to i}$ still being satisfiable). The probabilities for each

j to be in such a state are exactly the values of the incoming messages $\gamma_{j\to a}$ which are conditionally independent. $\delta_{a\to i} = \prod_{j\neq i}\gamma_{j\to a}$ is the first update equation for the modified BP version.

The more complicated part is to compute $\gamma_{i\to a}$ from incoming messages $\delta_{b\to i}$. The probability that i violates a conditioned on $\tau_{i\to a}$ being satisfied is again proportional to the probability that $\tau_{i\to a}$ is satisfied conditioned on i violating a. This is the case exactly if no $b \neq a$ fixes i to the value that would satisfy a.

$$\gamma_{i\to a} \propto P^u_{i\to a} := \prod_{b\in V^s_a(i)} (1-\delta_{b\to i})$$

To get the real value of $\gamma_{i\to a}$ one has to normalize so that $P^u_{i\to a}$ and the probability for i to satisfy a, $P^s_{i\to a}$, add up to 1.

With this this new formulation the messages can be obtained without summing over all possible configurations like in the last example and without summing over all possible configurations of a single variable's clauses like in the standard BP rules.
The overall update procedure can now be described as

```
Belief Propagation Algorithm
  Input: A CNF formula and its factor graph
0. Randomly initialize all message δ_{a→i} ∈_R [0,1]

1. For t = 0 to t = t_max
        1.1 Compute in random order for all edges (a, i)
          δ_{a→i} := ∏_{j∈V(a)\i} γ_{j→a}
        where
            γ_{j→a} := P^u_{j→a} / (P^u_{j→a} + P^s_{j→a}) = ∏_{b∈V^s_a(j)}(1−δ_{b→j}) / (∏_{b∈V^s_a(j)}(1−δ_{b→j}) + ∏_{b∈V^u_a(j)}(1−δ_{b→j}))

        1.2 If no δ_{a→i} has changed more than ε goto 2.
2. If t = t_max return UN-CONVERGED, else return the
    generated warnings δ*_{a→i}
```

3.2.2 *Marginal Probabilities and Valid Assignments*

If the BP-Algorithm returns a set of converged messages one can obtain the single variable marginal $\mu_i = P(x_i = 1 \,|\mathcal{F} \text{ is satisfied})$ in the same way, the values $P_{i\to a}$ where defined: The normalized probability that $\mathcal{F}$ is satisfied conditioned on $x_i = 1$.

$$\mu_i := \frac{\prod_{b\in V_-(i)}(1-\delta^\star_{b\to i})}{\prod_{b\in V_-(i)}(1-\delta^\star_{b\to i}) + \prod_{b\in V_+(i)}(1-\delta^\star_{b\to i})}$$

At each step of the algorithm one can obtain in the same way the current *belief* that $x_i = 1$ which has a changing value until the messages converge. μ_i is the final belief induced by the converged messages.

These marginals are used to obtain satisfying assignments in the so called *Belief Propagation Guided Decimation* algorithm. Like CID, the algorithm stepwise assigns values to a formula's variables.

For each variable i the BP algorithm is run and x_i is set to 1 with probability μ_i and to 0 with probability $1 - \mu_i$. Then the graph is cleaned like in the CID algorithm and the next variable can be assigned. After assigning a value to x_i the formula must still be satisfiable because x_i will be set to a value with positive marginal which means there exist assignments in which x_i takes exactly the chosen value. By induction the returned assignment is valid.

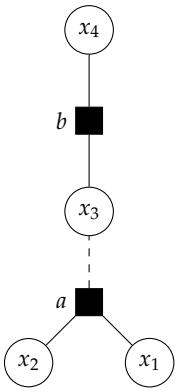
```
Belief Propagation Guided Decimation [2]
  Input: A CNF formula and its factor graph

1. For i = 1,...,n do
        1.1 Run BP and compute μ_i
        1.2 Assign ρ(x_i) = { 1  with probability μ_i
                            { 0, with probability 1 − μ_i
2. Return the complete assignment ρ
```

For loopy graphs where the beliefs may be incorrect, usually in each step the variable with the strongest belief (the variable i that maximizes $|\mu_i|$) is assigned a value.

Example. *In a final example, BPGD can be used to compute an assignment for the formula* $\mathcal{F} = \underbrace{(x_1 \vee x_2 \vee \overline{x_3})}_{a} \wedge \underbrace{(x_3 \vee x_4)}_{b}$

Edge	γ^*	δ^*
$1 \to a$	$1/2$	
$2 \to a$	$1/2$	
$4 \to b$	$1/2$	
$a \to 3$		$1/4$
$b \to 3$		$1/2$
$3 \to a$	$2/3$	
$3 \to b$	$4/7$	
$a \to 2$		$1/3$
$a \to 1$		$1/3$
$b \to 4$		$4/7$



The beliefs μ_i *are* $\mu_1 = \frac{1}{1+(1-\delta^*_{a\to 1})} = 3/5$, $\mu_2 = \frac{1}{1+(1-\delta^*_{a\to 2})} = 3/5$, $\mu_3 = \frac{1-\delta^*_{a\to 3}}{(1-\delta^*_{a\to 3})+(1-\delta^*_{b\to 3})} = 3/5$ *and* $\mu_4 = \frac{1}{1+(1-\delta^*_{b\to 4})} = 7/10$.

With probability $3/5$, x_1 *is set to* 1 *the and the cleaned formula is* $(x_3 \vee x_4)$ *which can be solved recursively.*

4

CONCLUSION

The two message passing algorithms described in section 3 are able to to compute assignments and some other parameters of a formula if its factor graph is a tree. For general graphs both algorithms may not converge or even yield wrong answers. There is not much known about what conditions a graph must fulfil to assure convergence of the passed messages.

In general the possibility for a random formula to be satisfiable is connected to its clause density, the ratio r between the number of clauses and variables. The higher this ratio, the more likely a formula is to be unsatisfiable. For 4-SAT this threshold is conjectured to be around 9.93. In [2] it is stated that belief propagation is able to efficiently find assignments for 4-SAT fomulas with clause densities up to 9.2. Conventional SAT solvers need exponential time to find solutions for $r > 5.54$.

Possible approaches to improve convergence are damping the message updates or modifying the factor graph to make it more tree-like. Possible drawbacks are increased computation time and a higher risk of incorrect results.

Even though both algorithms empirically converge rather often and are even used in praxis there are better techniques based to the general sum-product algorithm. In [1] a variant called *survey propagation* is presented which is tied to methods used in statistical physics. On trees this new method behaves like BP but on experiments on random graphs it performs better than standard belief propagation.

BIBLIOGRAPHY

[1] Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. "Survey propagation: an algorithm for satisfiability." In: *CoRR* cs.CC/0212002 (2002). URL: http://arxiv.org/abs/cs.CC/0212002.

[2] A. Coja-Oghlan. "On belief propagation guided decimation for random k-SAT." In: *ArXiv e-prints* (July 2010). arXiv: 1007.1328 [math.CO].

[3] MultiMedia LLC. *International SAT Competitions web page*. 2018. URL: http://www.satcompetition.org (visited on 07/03/2018).

[4] Rudiger Urbanke. "Lecture Notes 8: BP-Guided Decimation for SAT Problems." In: (2011). URL: https://documents.epfl.ch/groups/i/ip/ipg/www/2010-2011/Statistical_Physics_for_Communication_and_Computer_Science/lecture_8.pdf.