MICHAEL RUDERER

BELIEF PROPAGATION

[July 3, 2018 at 18:05 – classicthesis version 0.0]

BELIEF PROPAGATION

MICHAEL RUDERER

Seminararbeit

Juni 2018 – version 0.0

[July 3, 2018 at 18:05 – classicthesis version 0.0]

ABSTRACT

Short summary of the contents in English... a great guide by Kent Beck how to write good abstracts can be found here:

https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache...

CONTENTS

1 EINLEITUNG

Das ist die Einleitung

- Allgemeines über Message Passing, Belief Propagation
- Praktische Anwendungen (**TODO**: suchen)
- eventuell Motivation aus Quelle (hat mehr mit SAT als mit BP zu tun)
-

2

BELIEF PROPAGATION

2.1 FACTOR GRAPHS

For problems that include many variables influencing each other it is useful to have an abstract representation of how those variables are related to each other. So called factor graphs are such representations.

In general, factor graphs represent 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 X_i is part of f_j's input. This means the factor 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 $f(x_1, x_2, x_3) := P(X_1 = x_1 \wedge X_2 = x_2 \wedge X_3 = x_3)$. If X_2 and X_3 are conditionally independent given X_1 this function can be factorized:*

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

This factorization still contains a factor depending on all variables. If X_2 and X_3 are known to be conditionally independent given X_1 this factor can again be factorized:

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

the variable nodes are drawn here as circles whereas constraint nodes are drawn as rectangles to easily distinguish the types of nodes

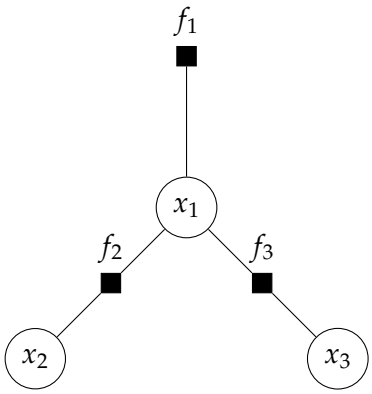


Figure 2.1: Factor graph of fs factorization into f_1, f_2, f_3

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 Problem*

A SAT formula in CNF form can be interpreted as a boolean function that factorizes to the formulas clauses.
In the corresponding factor graph each factor node a represents the local function defined by a single clause of the original formula. The clause is a disjunction of variables and negated variables $(x_i \vee \overline{x_j} \vee \ldots)$. 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 [**survprop**] some additional notation is defined to simplify the description of the algorithms in section **??**:

Definition. *Let a be a factor node and i a variable node*

- *The value* J_i^a

The constraints can directly be viewed as functions ...

Example.

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

The defined values of clause n are $V_+(b) = \{x_2, x_4\}, V_-(b) = \{x_1\}$

Again the circles are variable nodes, the rectangles are constraint nodes. If if x_i *appears negated in the clause a (if* $J_a^i = 1$*), the edge is drawn dotted.*

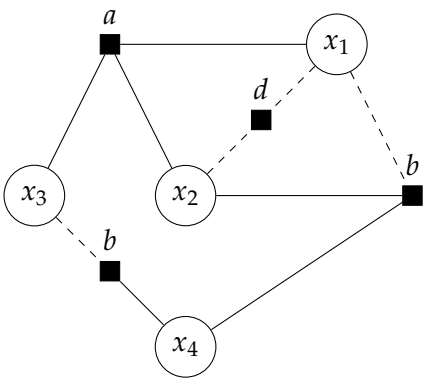


Figure 2.2: Factor graph of a SAT formula

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 $\mu_{i\to a}$.
Usually the massages $\mu_{i\to a}$ are obtained by summing over $\mu_{b\to i}$ and $\mu_{a\to i}$ by multiplying the messages $\mu_{j\to a}$. The fundamental equation form

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

Therefore these types of algorithms are called *sum-product*-algorithms. The 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 $\mu_{i\to a}$ does not influence its predecessors $m_{b\to i}$. The messages can be computed sequentially starting with the factor graphs'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 initial-

ization 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 $i \to a$. 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*. Since there is no guarantee for convergence it is also possible for the update process to never terminate. In this case the computation has to be terminated after a fixed number of steps.
In practice the update rules have to be slightly modified when applying message passing on loopy graphs. The messages are computed from random starting values and often do not fulfil properties which the original defined messages do.

On trees this heuristic approach still returns correct results, on general graphs experience shows that convergence happens rather frequently. Also, loopy message passing may converge at incorrect states.
Convergence for trees, like other properties of message passing algorithms, can be shown by induction on the *level* of a message.

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

The level can be thought as the length of the longest path 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 is a leaf since there are no ingoing messages $j \to a$ so its message 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 on the next update step.

□

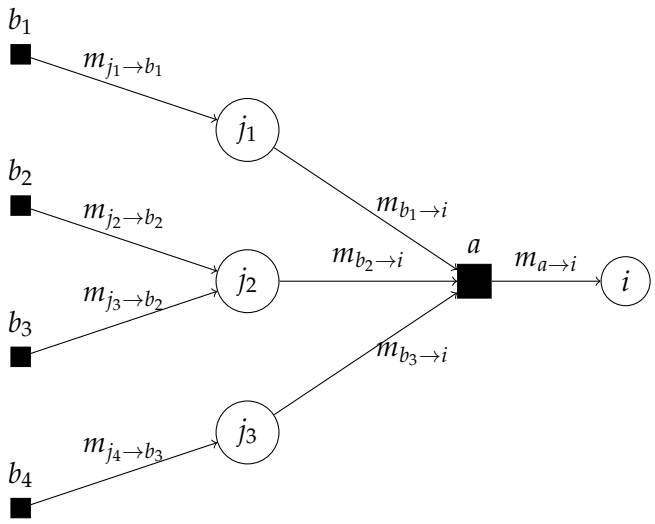


Figure 2.3: Subgraph of a factor tree with all messages required for computing $m_{a \to i}$

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 [**dummizitat**].

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 $j \in V_-(a)$. 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 the 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 i to 1 if their warnings are active, the clauses $b \in V_-(j)$ would fix i to 0. So if $h_{j\to a}$ is positive, the variable i 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 the clause. 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 $\theta(h_{j\to a} J_j^a)$ 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

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 on 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.
- 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 there is at least one b with $J_j^b J_j^a u^\star_{b\to j} = 1$ or equivalently $J_j^b \neq J_j^a$ and $u^\star_{b\to j} = 1$. By induction the subproblem $\tau_{b\to j}$ is not satisfiable and any assignment in which j supports a must violate at least one clause on $\tau_{b\to j}$.

□

3.1.2 *Decimation Algorithm*

The algorithm that computes satisfying assignments based on the WP results is called *Warning Inspired Decimation*. 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 assignments. 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 $H_i := -\sum_{b \in V(i)} J_i^b u_{b \to i}^*$.
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.
If a variable is both fixed to 1 and 0, the formula must be unsatisfiable. If no such contradiction appears a partial assignment can be obtained by setting each variable to the value it was fixed to: 1 if $H_i > 0$ and 0 if $H_i < 0$. If no variable is fixed (all H_i are 0), CID chooses a random variable and sets it to a random value.

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 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 assignment has probability 2^{-n}.
This probability space can be restricted to only those configurations that satisfy a boolean formula $\mathcal{F}$. If there are $\mathcal{N}$ satisfying assignments the probability measure P factorizes to

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

where A is the set of clauses in $\mathcal{F}$ and $f_a(X)$ the characteristic function of a clause a that is 1 if X satisfies a and 0 if not.

The fraction of satisfying assignments where for example $x_1 = 1$ and $x_4 = 0$ can now be viewed as a marginal of $P_{\mathcal{F}}$ and computed using message passing techniques.

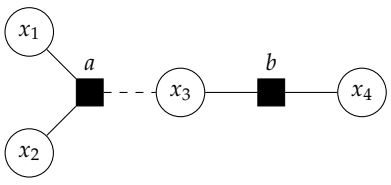
3.2.1 *Propagation Algorithm*

3.2.1.1 *Messages*

The message $\mu_{a \to i}(x_i)$ sent from a factor a to a variable i is 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* $\mathcal{F} = \underbrace{(x_1 \vee x_2 \vee \overline{x_3})}_{a} \wedge \underbrace{(x_3 \vee x_4)}_{b}$.

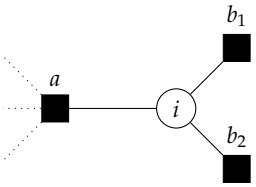


The messages sent from b to x_3 can be computed using only the definition:
$\mu_{b \to 3}(1)$ is the probability that $(x_3 \vee x_4)$ is satisfied by an assignment where $x_3 = 1$ which is 1. If $x_3 = 0$, only those assignments with

$x_4 = 1$ 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 $\mu_{3\to b}(0) = \frac{4}{7}$, $\mu_{3\to b}(1) = \frac{3}{7}$.

3.2.1.2 *Update Rules*

$\mu_{b\to 3}(1)$ is the message that is passed to a and tells the probability i is x_i considering only the factors on i's side of the graph.
Through the tree structure of $\mathcal{F}$ these factors can be grouped into the subtrees $\tau_{b\to i}$ rooted in i's neighbour factors $b_i \neq a$. These subgraphs are not connected and therefore independent to each other given the value of i.



$$\mu_{i\to a}(x_i) = P(x_i \mid \tau_{b_1\to i} \wedge \ldots \wedge \tau_{b_k\to i}) = \prod_b P(x_i \mid \tau_{b\to i}) = \prod_b \mu_{b\to i}$$

3.2.2 *Marginal Propabilities*

3.2.3 *Number of satisfying assignments*

Part I

APPENDIX

You can put some informational part preamble text here. Illo principalmente su nos. Non message *occidental* anglo-romanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

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.

DECLARATION

Put your declaration here.

, Juni 2018

Michael Ruderer

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Thank you very much for your feedback and contribution.