MICHAEL RUDERER

BELIEF PROPAGATION

[June 27, 2018 at 19:18 – classicthesis version 0.0]

BELIEF PROPAGATION

MICHAEL RUDERER

Seminararbeit

Juni 2018 – version 0.0

Michael Ruderer: *Belief Propagation,* Seminararbeit, © Juni 2018

[June 27, 2018 at 19:18 – 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

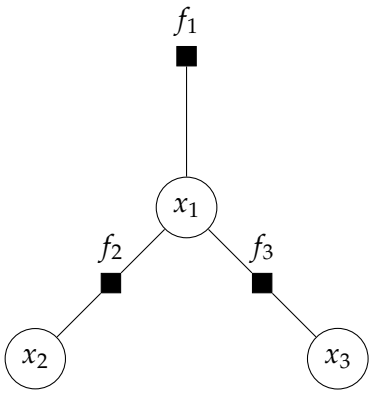

Figure 2.1: Factor graph of fs factorization into f_1, f_2, f_3

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.*

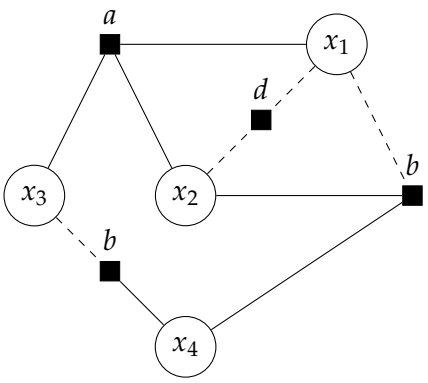

Figure 2.2: Factor graph of a SAT formula

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 $m_{i\to a}$ and messages passed from a to i, denoted as $m_{a\to i}$.
The messages must be defined so that a message $m_{a\to i}$ is determined by the messages $m_{j\to a}$ that a received from neighbour variables $j \neq i$. The same must hold for $m_{i\to a}$.
Usually the massages $m_{i\to a}$ are obtained by summing over $m_{b\to i}$ and $m_{a\to i}$ by multiplying the messages $m_{j\to a}$. The fundamental equation usually has the form $m_{i\to a} = \sum_{j\in V(a)\setminus i} \prod_{b\in V(j)\setminus a} m_{j\to b}$. Therefore these types of algorithms are called *sum-product*-algorithms.
For tree factor graphs which do not contain cycles the value of $m_{i\to a}$ 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 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 update the provisional values. The update steps are repeatedly performed for each edge $i \to a$ in random order. 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.

On trees this heuristical approach still returns correct results, on general graphs experience shows that convergence happens rather frequently.

Lemma. *Loopy message passing converges on trees.*

- "Loopy" belief propagation
- Convergence for trees
- Heuristic for general graphs
- Examples only in next chapter

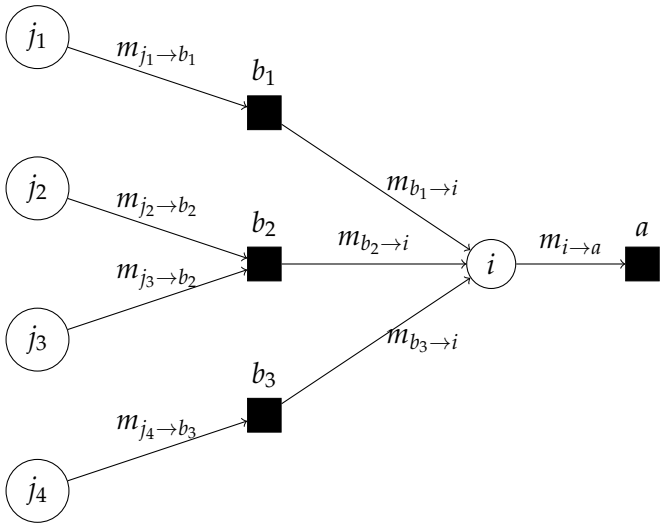

Figure 2.3: Factor graph of a SAT formula

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 [[**dummyzitat**]] 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:

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

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 prefered 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(−J_j^a (Σ_{b∈V(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}
```

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:

3.1.2 *Decimation Algorithm*

- Contradiction Numbers
- Lemma: $u^\star_{a\to i} = 1 \Rightarrow$ i has a fixed value
- Decimation Algorithm
- Example on graph from last chapter (Je nachdem wie viel Platz in Tabellenform oder grafisch)

3.2 BELIEF PROPAGATION

Warning Propagation braucht bisher 2 Seiten, in der fertigen Version also etwa 4-5. Belief Propagation sollte in etwa den selben Umfang haben, etwas weniger Beschreibung nötig weil vieles gleich ist, dafür aber komplizierter.

Algorithmus In 3.2.1 Algorithmus beschreiben. Grundsätzliche Vorgehensweise ist dieselbe wie WP, also "nur" unterschiedliche Arten von Nachrichten und neue Update-Regel beschreiben

Lösungen Hier zeigen, wie die Anzahl der erfüllenden Belegungen und die Wahrscheinlichkeit für $x_i = 1$ aus den konvergierten Nachrichten berechnet wird. Wenn noch genug Seiten frei mit ausführlichem Beispiel

3.2.1 *Propagation Algorithm*

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.

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.