Loop Calculus for Satisfiability

Lukas Kroc
Cornell University
Ithaca, NY 14853
kroc@cs.cornell.edu

Michael Chertkov
Los Alamos National Laboratory
MS B213, T-13, Los Alamos, NM 87545
chertkov@lanl.gov

Abstract

Loop Calculus, introduced by Chertkov and Chernyak, is a new technique to incrementally improve approximations computed by Loopy Belief Propagation (LBP), with the ability to eventually make them exact. In this extended abstract, we give a brief overview of this technique, and show its relevance to the AI community. We consider the problem of Boolean Satisfiability (SAT) and use LBP with Loop Calculus corrections to perform probabilistic inference about the problem. In this preliminary work, we focus on identifying the main issues encountered when applying Loop Calculus, and include initial empirical results in the SAT domain.

Introduction

Belief Propagation (BP), as a message-passing algorithm for probabilistic inference in graphical models, has been used in various disciplines including artificial intelligence, coding theory, or statistical mechanics (Yedidia, Freeman, and Weiss 2005). In its basic form, BP provides exact results only on relatively simple problems, namely when the factor graph of the problem contains no loops. This, of course, severely limits is applicability. Nevertheless, it has been observed that BP applied to problems from certain domains where the underlying factor graph does contain loops (it is then called Loopy Belief Propagation, LBP) yields sufficiently accurate approximations (Murphy, Weiss, and Jordan 1999). Making the LBP algorithm more accurate in general is a topic of ongoing research (Mooij and Kappen 2007).

In this extended abstract, we focus on a new technique called Loop Calculus (Chertkov and Chernyak 2006b). Introduced recently in the coding-theoretic framework, the technique provides a way to correct for *all* inaccuracies of the LBP algorithm, in an incremental fashion. That is, Loop Calculus allows to tune the trade-off between accuracy of solutions and computational cost, to a level appropriate for a particular application. For the purpose of this abstract, the application is probabilistic inference about Boolean Satisfiability (SAT). But the Loop Calculus, if proven successful, would be applicable in other domains of probabilistic inference. A technical report with full details accompanying this abstract is available (Chertkov and Kroc 2008).

Overview of Loop Calculus

The basic idea behind Loop Calculus is relatively simple: since the inaccuracies of LBP are caused by the presence of loops in the factor graph, can one write an expression involving some terms for each loop that would correct it? Somewhat surprisingly, this is possible to do in a relatively elegant way, which is what Loop Calculus is about.

Let Z denote the *partition function*, that is the sum of weights of all configurations in a given graphical model: $Z = \sum_{\vec{x}} w(\vec{x})$, where $\vec{x}$ is a (any) setting of variables, and $w(\vec{x})$ the associated weight. The partition function can be used to express all inference measures (such as marginal probabilities of variables), and we will thus focus on it. The LBP algorithm computes an estimates of Z, which we denote by Z_0. The Loop Calculus correction to Z_0 can then be elegantly expressed as

$$Z = Z_0 \left(1 + \sum_{c \in \mathcal{C}} r_c \right) \tag{1}$$

where $\mathcal{C}$ is a set of all *generalized loops*, which are subgraphs of the factor graph with no nodes of degree 1, and r_c is a *loop weight* associated with each generalized loop $c \in \mathcal{C}$. We call loops with large $|r_c|$ *heavy loops*. The loop weights are almost always in the range $[-1, 1]$ (provably so for simple loops discussed later), but their form depends on the factor graph. Nevertheless, they can always be expressed as products of certain terms derived from values provided by LBP. This means that Loop Calculus expresses the *exact* value of Z solely based on values provided by LBP, regardless of the structure of the factor graph.[1]

There is one obvious issue with the correction: the number of generalized loops is typically exponential. The hope here is that for some problems, only a limited number of generalized loops will be necessary to provide sufficient accuracy, as been shown for example in the coding domain (Chertkov and Chernyak 2006a). The number of generalized loops considered in the sum is the point where fine tradeoff between accuracy and computational cost is achieved: including more loops leads in general to higher accuracy (but as will be discussed later, this property is not monotone). Designing an efficient algorithm for summing the loop series is discussed in the following section.

[1] Convergence, another serious issue of LBP besides its accuracy, can be addressed by other means, see e.g. (Yuille 2002).

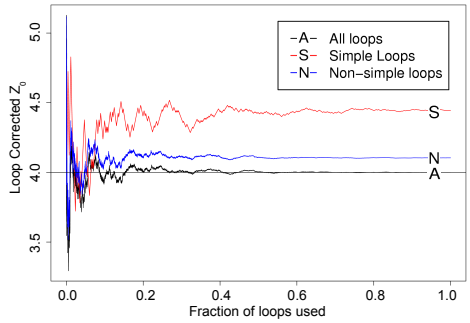
Summing the Loop Series

The simplest way to make the exponentially long sum in (1) feasible for evaluation is to truncate it at a certain point. If enough heavy loops have been summed till then, the resulting correction might already be sufficiently accurate. This turns out to be the case in coding, where considering just one (heaviest) loop is already an improvement in the decoding algorithm (Chertkov and Chernyak 2006a). An important question in a particular problem is thus whether there are "reasonably few" generalized loops that allow for "sufficient accuracy", and if so, whether one can find them efficiently (Gómez, Mooij, and Kappen 2007).

Finding heaviest loops is a difficult problem in general, but a slight restriction yields a practical approach for finding at least very heavy loops. The restriction is that we will only consider *simple loops*, consisting of only one cycle. The expression for the weights of generalized loops in case of binary variables (being a product of terms mostly from $[-1, 1]$) suggests that the more nodes and edges in a loop, the lower its weight $|r_c|$. This justifies our focus on simple loops. The problem of finding the heaviest simple loop can be viewed as searching for the minimal-weight cycle in a certain graph with labeled and weighted edges, such that each edge label appears at most once. We have developed an A^*-style search algorithm which, while not polynomial in the worst case, is very efficient even on instances with thousands of variables. Moreover, besides finding the heaviest simple loop, it can also output all simple loops ordered by decreasing $|r_c|$, which is what we ultimately need to approximate the loop series. One way to perform the series truncation is to sum up only simple loops above a certain weight threshold. We include preliminary results for such approach in the Empirical Validation section below.

Empirical Validation

We first apply Loop Calculus to a toy SAT instance with 4 solutions, which allows to consider all generalized loops and thus fully observe the behavior of the method.



The above figure shows estimated $Z = Z_0(1+\sum_{c\in Loops} r_c)$ on the y-axis, where $Z_0 = 4.55$ and $Loops$ is a set of, respectively, all loops, only simple loops and non overlapping simple loops (multi-simple loops) and only non-simple loops (all the others). X-axis is the fraction of the respective loops used in the sum, ordered by decreasing $|r_c|$. There are 682 (multi-)simple loops, and $329,664$ non-simple loops. The figure confirms that simple loops are very important, even though there is only a small fraction of them overall. Omitting them results in a large error (the N curve), despite the effort spent on summing 99.8% of all generalized loops.

Increasing the size of the problems, we look at random 3-SAT instances with 100 variables. The table below shows results for two instances with different clause-to-variable ratio α. The loop correction series is summed across all (multi-)simple loops with $|r_c| \geq 0.001$.

α	exact count Z	LBP's Z_0	$Z_0\,(1+\sum r_c)$
3.0	1.97×10^{10}	4.34×10^{10}	2.09×10^{10}
3.5	6.66×10^{8}	2.60×10^{8}	2.46×10^{8}

Some observations can be readily made: Loop Calculus *can* provide a significant improvement in the accuracy (the first instance), although as the LBP's estimate Z_0 gets more inaccurate, Loop Calculus requires more fine terms in the series, and can sometimes worsen the estimate (second instance). This points out the non-monotonic behavior also observable in the figure above. Additional empirical results can be found in the technical report (Chertkov and Kroc 2008).

Summary and Future Work

Loop Calculus, while not yet a fully developed technique, might provide a very useful tool for improving accuracy of standard LBP in various domains, including many in AI. This work discusses application of Loop Calculus to the problem of reasoning about SAT. While the preliminary results show that in general, there are many issues making the application to SAT difficult, it also shows that improvement *is* possible, given a suitable problem domain. The most important future work will look into alternative ways of summing the loop series, as well as identifying more domains where Loop Calculus is efficient, and significantly increases the accuracy of LBP.

References

Chertkov, M., and Chernyak, V. Y. 2006a. Loop calculus helps to improve belief propagation and linear programming decodings of low-density-parity-check codes. In *Proc. 44th Allerton Conf.*

Chertkov, M., and Chernyak, V. Y. 2006b. Loop series for discrete statistical models on graphs. *Journal of Statistical Mechanics: Theory and Experiment* 2006(06):P06009.

Chertkov, M., and Kroc, L. 2008. Loop calculus for satisfiability, LAUR 08-0321, `http://www.cs.cornell.edu/w8/~kroc/misc/pub/loop_sat.pdf`.

Gómez, V.; Mooij, J. M.; and Kappen, H. J. 2007. Truncating the loop series expansion for belief propagation. *J. Mach. Learn. Res.* 8:1987–2016.

Mooij, J. M., and Kappen, H. J. 2007. Loop corrections for approximate inference on factor graphs. *J. Mach. Learn. Res.* 8:1113–1143.

Murphy, K.; Weiss, Y.; and Jordan, M. I. 1999. Loopy belief-propagation for approximate inference: An empirical study. In *Proc. of UAI'99*, 467–475.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2005. Constructing free-energy approximations and generalized belief propagation algorithms. *Inf. Theory, IEEE Trans. on* 51(7):2282–2312.

Yuille, A. L. 2002. CCCP algorithms to minimize the bethe and kikuchi free energies: convergent alternatives to belief propagation. *Neural Comput.* 14(7):1691–1722.