An Investigation into Concurrent Expectation Propagation

David Hall and Alex Kantchelian
EECS CS Division
University of California, Berkeley
{dlwh, akant}@cs.berkeley.edu

Abstract

As statistical machine learning becomes more and more prevalent and models become more complicated and fit to larger amounts of data, approximate inference mechanisms become more and more crucial to their success. Expectation propagation (EP) is one such algorithm for inference in probabilistic graphical models. In this work, we introduce a robustified version of EP which helps ensure convergence under a relaxed memory consistency model. The resulting algorithm can be efficiently implemented on a GPU in a straightforward way. Using a 2D Ising spin glass model, we evaluate both the original EP algorithm and our robustified version in terms of convergence, if any, and precision on a classic single core processor. We also compare the naive parallelized version of the original EP algorithm against the parallelized robustified EP on both a multicore CPU and a GPU.

1 Introduction

Probabilistic graphical models (PGMs) are one of the cornerstones of modern AI, with applications ranging from medical diagnosis, to social network analysis, to robotics. PGMs provide the flexibility to model almost any real world phenomena in a coherent, simple way. However, exact computation in all but the most trivial PGMs is impossible. In the discrete case, inference in PGMs is #P-complete [3], and for continuous variables it is possible to come up with incomputable models, even for "well behaved" distributions [1].

Therefore, approximate inference regimes are one of the main cottage industries within the AI community, with several variants proposed each year. Broadly, they fall into three categories. The first is based on random sampling, especially Markov Chain Monte Carlo. MCMC is a randomized algorithm that is perhaps the easiest to implement, though it almost always the slowest in practice. Second are relaxation techniques based on linear or convex programming, but these are complex to implement and require significantly different implementations for each new PGM considered. Finally, there are message passing algorithms like mean field inference, belief propagation, and expectation propagation. We focus on message passing algorithms.

Unfortunately, all of these algorithms are tricky to parallelize, especially on arbitrarily structured graphs. Most of these algorithms can be parallelized to a certain extent given a graph coloring, but there is a limit to how much parallelism can be extracted from graph partitions in arbitrary graphs.

Message passing algorithms such as Expectation Propagation [4] optimize an approximation to the true PGM that is "as close as possible" to the original graph while remaining within a tractable set of decomposed distributions. Operationally, these algorithms typically work by decomposing the graph structure into subgraphs, and then (potentially) further approximating those parts with simpler distributions. Then, one component is updated at a time, optimizing it locally using the other components as a guide. That is, these algorithms are basically a kind of coordinate ascent. This implies that these updates are inherently sequential: coordinate ascent only makes sense if one coordinate is updated at a time.

However, it is possible to just perform these updates in parallel, using "old information" when making decisions [2]. However, this update schedule can exacerbate the non-convergence problems already observed in practice. Because there are multiple local optima, different components of the graph might be attracted to different optima, leading to oscillations.

We propose a new message passing algorithm called Convex Expectation Propagation (CEP) that allows for non-serialized concurrent updates without introducing the same pathologies possible in the naïve algorithm. One class of approximations—convex approximations—ensures that there is only one optimum.[7] Thus, different pieces of the structure of the graph (which are operating largely independently on different processors) cannot update towards different local optima, which is one of the primary sources of

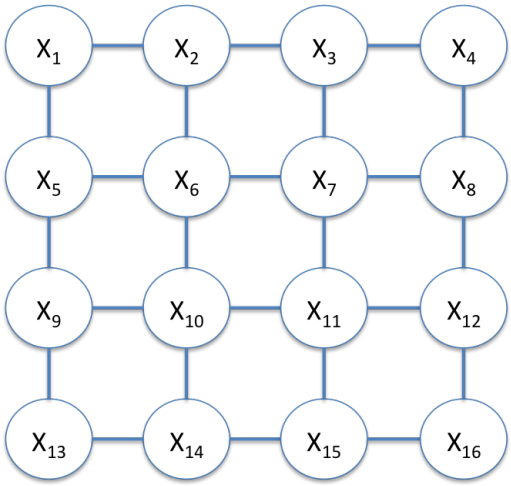


Figure 1: A 2-D Ising model, a commonly employed synthetic class of graphical model. Ising models are pairwise graphical models over binary-valued random variables with two kind of potentials: singleton and pair. Singleton potentials govern the probability a variable is "on" in isolation, while the pairwise potentials govern the extent to which variables agree or disagree with their neighbors.

oscillation.

We implemented CEP on both a CPU and a commodity GPU, via the new OpenCL standard. We compare our algorithm to a serialized version of the algorithm, as well as a non-robustified traditional version of EP. We consider three metrics: accuracy, speed, and convergence. We show that while EP performs surprisingly well in many circumstances even when parallelized, it does for some conditions exhibit the oscillations and slower convergence we hypothesized. Moreover, our new algorithm does not exhibit these problems, though it usually less accurate. We therefore consider an unprincipled compromise between the two algorithms, finding that it combines the benefits of both algorithms.

This paper is organized as follows. First, we describe necessary background on probabilistic graphical models. Second, we describe Expectation Propagation. Third, we motivate and present Convex Expectation Propagation. Then, we conduct our experiments on synthetic models in a Ising spin-glass model under a variety of parameter conditions. Finally, we discuss potential future avenues for research.

2 Graphical Models

Probabilistic graphical models are defined on a graph structure as in Figure 1.[1] Each PGM defines a class of probability distributed that share certain independence assumptions. The nodes in a graph are the underlying random variables, and a variable is independent of all other variables conditioned on observations about all of its neighbors. For instance, in Figure 1, X_1 is independent of all other variables given X_2 and X_5.

A graphical model does not describe a single probability distribution. Instead, we need a set of *potential functions* ψ, possibly one for each clique $\vec{x}_{\{k\}}$ in the underlying graph:

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_k \psi_k(\vec{x}_{\{k\}})$$
$$Z = \sum_{x'_1, \ldots, x'_n} \prod_k \psi_k(\vec{x}'_{\{k\}}) \tag{1}$$

Z is called the partition function and ensures the distribution normalizes to unity. Note that it involves a sum over all possible assignments to all variables, which can take exponential time in an arbitrary graph. Most of our attention will be focused on approximating this term.

Usually, for both theory and practice, it is convenient to represent these distributions in the exponential family of distributions as follows:

$$p(x_1, \ldots, x_n) = \exp\left(\sum_k \langle \theta_k, \phi^k(\vec{x}_{\{k\}}) \rangle - A(\theta)\right)$$
$$A(\theta) = \log \sum_{x'_1, \ldots, x'_n} \exp\left(\sum_k \langle \theta_k, \phi^k(\vec{x}'_{\{k\}}) \rangle\right) \tag{2}$$

Each $\phi^k(\cdot)$ is a "feature" or "projection" function that lifts the underlying variables in a space where the potential function can be computed as the exponentiation of an inner product. $A(\theta)$ is called the log-partition function.

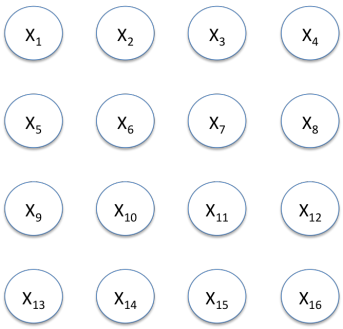
In this paper, we actually restrict ourselves to pairwise graphical models: models where the potentials are constrained to at most 2 variables. That is, we consider models of the form:

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_i \psi_i(x_i) \prod_{i,j} \psi_{ij}(x_i, x_j) \tag{3}$$

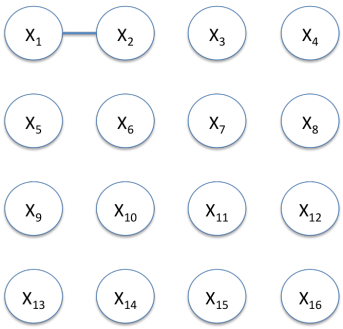
Arbitrary graphical models can be made pairwise via a simple transform, and so it is appropriate for simulation studies. In particular, we use the Ising Model for our experiments. Everything we find should generalize to higher arity models.

There are two *inference* tasks commonly associated with graphical models. The first is the identification of the maximum probability assignment to all variables. The second is the task of determining the partition function Z or marginal distributions $\mu(x_i, x_j, \ldots)$. For general discrete distributions, this first task is NP-hard, while the second task is #P-hard in the general case.

[1] We focus on the undirected case. These are sometimes called Markov Random Fields.



(a) Fully disconnected "core" graph



(b) An "augmented" graph with one edge

Figure 2: EP functions by using a "core" graph and a collection of "augmented" graphs. The goal is to approximate the original connected graph (like in Figure 1) with the fully disconnected graph, by augmenting the graph one at a time, and iteratively "projecting" this augmented graph down to the original core graph.

Many papers have been written on methods for approximately solving both of these problems, as well as identifying subclasses of PGMs where one or the other task is tractable. (See, e.g., [6]) In this paper, we focus on the second, harder, task, choosing to study in particular on Expectation Propagation[4].

3 Expectation Propagation

Recall in our discussion of Equation 1 that the exponential number of assignments to the variables $x_1, \ldots, x_k$ was the source of the intractability associated with inference in graphical models. If, however, our graph is suitably structured—say, a fully disconnected graph or a tree—then computing this sum would be possible in polynomial time via dynamic programming. Moreover, if a graph structure is just "a little intractable" then inference is still not so bad. For instance, a graph with a single loop can be decomposed into just a few trees by summing over the loop in the outermost layer of the sum. More generally, if we could find a way to efficiently approximate a full graph by a "simple graph" and a collection of "slightly intractable" graphs, we could perform the sum efficiently.

Expectation Propagation[4] seeks to operationalize this intuition precisely. Define a core graph by a feature function ϕ^0 that collects some subset of the original feature functions. For instance, in the Ising model in Figure 1, we might choose the potentials on the fully disconnected graph in Figure 2a. Then, we cluster the remaining potentials into "augmented" graphs. For instance, we might keep each edge in its own cluster, as in Figure 2b. Denote these clusters by their clustered feature functions ϕ^i.

EP works by taking the potentials from an augmented graph and combining them with an "adjusted" version of the core graph. Specifically, we maintain a distribution q based on the core graph that approximates p as closely as possible. At each step, we choose an augmented graph with features ϕ^k, multiply it in with q (after removing cluster k's contribution to q), and then create a new q that looks as much like the product as possible. Write the potentials associated with the augmented graph by $f_k(x) = \exp(\langle \theta^k, \phi^k(x) \rangle)$, and let $\tilde{f}_k(x) = \exp(\langle \lambda_k, \phi^0(x) \rangle)$ be f_k's contribution to the q. The algorithm is as follows:

1. Initialize the $\tilde{f}_k$.
2. Set $q(x) \propto \prod_k \tilde{f}_k$.
3. Until convergence, choose k:
4. Create $q^{\backslash k}(x) = q(x)/\tilde{f}_k(x)$
5. Create the augmented distribution: $q_k(x) = q^{\backslash k}(x) \cdot f_k(x)$.
6. Project $\mathrm{KL}(q_k(x)||q^{\text{new}}(x))$
7. Store $\tilde{f}_k(x) = q^{\text{new}}(x)/q(x)$

Step 6 simply says that one should adjust q to have the same expectations as q_k for all of the features in ϕ^0. For instance, if we use a fully disconnected graph as our q, then we will update q so that the marginal distributions $\mu_i(x_i)$ will be the same in both graphs.

From a practical perspective, this algorithm is much easier to implement than it looks. Note that all of these functions are just the exponentiation of dot products. Therefore, the multiplication of two terms just the addition of two vectors (say θ^0 and λ_k), while division is just subtraction.

Finally, EP is defined sequentially, but one could perform the updates in parallel, simply computing the $\tilde{f}_k$ in parallel and reconstituting q in a reduce-scatter operation.

4 Convex Expectation Propagation

Like almost all message-passing-based approximate inference algorithms, EP is known to have multiple local

optima. (The main exception to our knowledge is Tree-Reweighted Belief Propagation [7].) In practice, sequential EP seems to have little issue with these local optima, as the errors found with EP are typically much lower than with other algorithms. Moreover, the algorithm always seems to converge.

However, when doing parallel updates, EP is basically using "stale" information. That is, when doing each update, the different components have no knowledge of the changes about to be performed by the other graphs. Because different subgraphs might be locally attracted to different optima, we claim that parallel-update EP is less likely to converge, because it might alternate between these optima.

The reason that Tree-Reweighted Belief propagation is exempted from this critique is that it attempts to optimize a *convex* relaxation of the objective function. Convexity assures, among many other things, that there is at most one optimum.[2] Thus, even if the updates in the Tree-Reweighted case are done in parallel, they cannot be attracted to different local optima because they simply do not exist.

We propose, therefore, to modify EP by "convexifying" its updates. Here, we derive the algorithm starting from the modified objective function. Our proof—given sufficient background—is fairly straightforward, and follows the derivation of standard EP given in [8] closely.

4.1 The EP Objective

We begin by motivating the EP objective function. First, we use standard results to rewrite the log-partition function $A(\theta)$:

$$\begin{aligned} A(\theta) &= \log \sum_x \langle \theta, \phi(x) \rangle \\ &= \max_{\mu \in \mathcal{M}} \{ \langle \theta, \mu \rangle + H(\mu) \} \end{aligned} \tag{4}$$

with summation replaced by integration as appropriate. Here, μ is a function defining marginal distributions and H is the standard entropy functional in base e. There are two problems to computing this function directly. First, there are potentially exponentially many constraints defining the set $\mathcal{M}$. Second, the term $H(\mu)$ might not decouple in a way that admits efficient dynamic programming. We will relax both of these conditions.

First, we split the graphical model with features ϕ into a number of components. Specifically, there are the *core* features ϕ^0, and then a set of *augmented* features ϕ^i. In the Ising model, the core features would be the singleton potentials, while the ϕ^i correspond to non-overlapping subsets of the various edges. Clearly, taking all of these features together yields the original model, while taking just the core features and one of the ϕ^i yields a relaxed approximation to the full model, which we will call an *augmented* model.

The key insight behind standard EP is to approximate the entropy of the full model as follows:

$$\begin{aligned} H(\mu) &\approx H(\mu^0) + \sum_i \left(H((\mu^0, \mu^i)) - H(\mu^0) \right) \\ &\triangleq F(\mu^0, \ldots, \mu^n) \end{aligned} \tag{5}$$

Intuitively, this expression says that the full entropy function H can be approximated as the entropy of the core model, plus a set of "corrections" from each of the augmented models.

What remains is to relax the set $\mathcal{M}$ to one with fewer constraints. This can be achieved by specifying that each of the marginals (μ^0, μ^0) must live in a set $\mathcal{M}^i$ that specifies constraints that affect only the variables and potentials used in that augmented model. All together, we obtain a new objective:

$$A(\theta) \approx \max_{(\mu^0, \mu^i) \in \mathcal{M}^i} \{ \langle \theta^0, \mu^0 \rangle + \sum_i \langle \theta^i, \mu^i \rangle + F(\mu^0, \ldots, \mu^n) \} \tag{6}$$

By following a derivation similar to what we present in the next section, we can arrive at the EP updates described previously.

4.2 Convex EP

The entropy functional is well-known to be convex, but the approximate entropy functional F defined in Equation 5 is in general not convex[8]. However, if we were to restrict ourselves to just one augmented model, then F would be convex, because

$$\begin{aligned} F(\mu^0, \mu^i) &= H(\mu^0) + \left(H((\mu^0, \mu^i)) - H(\mu^0) \right) \\ &= (H((\mu^0, \mu^i)) \end{aligned} \tag{7}$$

Moreover, if we exploit the fact that any convex combination of convex functions is convex, we obtain:

$$\begin{aligned} G(\mu^0, \mu^1, \ldots, \mu^n) &= \sum_i \rho_i \left(H(\mu^0) + \left(H((\mu^0, \mu^i)) - H(\mu^0) \right) \right) \\ &= H(\mu^0) + \sum_i \rho_i \left(H((\mu^0, \mu^i)) - H(\mu^0) \right) \end{aligned} \tag{8}$$

is convex. Here ρ_i parameterizes the convex combination. In general, we require that $\sum_i \rho_i = 1$, though for specific cases (such as the Ising model) we can employ other choices that maintain convexity and might be more accurate. Hence,

[2] In principle the optimum could be at $\pm\infty$.

we have a new approximate objective:

$$A(\theta) \approx \max_{(\mu^0,\mu^i)\in\mathcal{M}^i} \{\langle\theta^0,\mu^0\rangle + \sum_i \langle\theta^i,\mu^i\rangle + G(\mu^0,\ldots,\mu^n)\} \tag{9}$$

What remains is to actually derive the algorithm. To do that, we first employ a mathematical sleight-of-hand. We define new marginals η^i defined over the same space as μ^0 and constrain them to equal μ^0. We modify G as follows:

$$\begin{aligned} &G(\mu^0,(\eta^0,\mu^1),\ldots,(\eta^n,\mu^n)) \\ &\qquad = H(\mu^0) + \sum_i \rho_i \left(H((\eta^i,\mu^i)) - H(\eta^i)\right) \end{aligned}$$

This trick allows us to relax the constraint that $\eta_i = \mu_i$ and enforce it with Lagrange multipliers.

Given this modified G, we define the Lagrangian:

$$\begin{aligned} &L(\mu^0,(\eta^i,\mu^i),\lambda^i) \\ &\quad = \langle\theta^0,\mu^0\rangle + \sum_i \langle\theta^i,\mu^i\rangle + G(\mu^0,(\eta^1,\mu^1),\ldots,(\eta^n,\mu^n)) \\ &\qquad + \sum_i \langle\lambda^i,\mu^0-\eta^i\rangle\rangle \end{aligned} \tag{10}$$

Taking gradients and setting them to 0 we obtain:

$$\begin{aligned} 0 &= \nabla_{\mu^0} L(\cdot) \\ &= \theta^0 + \nabla H(\mu^0) + \sum_i \lambda_i \\ &= \theta^0 - \log\mu^0 + \sum_i \lambda_i + \text{const} \\ \log\mu^0 &= \theta^0 + \sum_i \lambda_i + \text{const} \\ \mu^0(x) &\propto \exp(\left\langle \theta^0 + \sum_i \lambda_i, \phi^0(x) \right\rangle) \\ &\triangleq q(x) \end{aligned}$$

Because μ^0 is a probability distribution, μ^0 is a distribution that has the same form as the base distribution.

Taking derivatives with respect to each (η^i,μ^i) we have:

$$\begin{aligned} 0 &= \nabla_{(\eta^i,\mu^i)} L(\cdot) \\ &= \theta^i + \rho_i \nabla H\left((\eta^i,\mu^i)\right) - \rho_i \nabla H(\eta^i) - \lambda_i \\ &= \theta^i - \rho_i \log\binom{\mu^i}{\eta^i} + \rho_i \log\eta^i - \lambda_i \\ \rho_i \log\binom{\mu^i}{\eta^i} &= \theta^i - \lambda_i + \rho_i \log\eta^i \end{aligned}$$

Using the fact that $\eta^i = \mu^0$ at the optimum and doing some algebra, we have:

$$\begin{aligned} \rho_i \log\binom{\mu^i}{\eta^i} &= \theta^i - \lambda_i + \rho_i \log\mu^0 \\ \rho_i \log\binom{\mu^i}{\eta^i} &= \theta^i - \lambda_i + \rho_i \cdot (\theta^0 + \sum_\ell \lambda_\ell) \\ \log\binom{\mu^i}{\eta^i} &= \frac{1}{\rho_i}\theta^i + \theta^0 + \sum_{\ell\neq i}\lambda_\ell + (1-\frac{1}{\rho_i})\lambda_i \\ q_i(x) &\propto \exp(\left\langle \theta^0 + \sum_{\ell\neq i}\lambda_\ell + (1-\frac{1}{\rho_i})\lambda_i, \phi^0(x) \right\rangle \\ &\qquad + \frac{1}{\rho_i}\langle\theta^i,\phi^i(x)\rangle) \end{aligned}$$

This implies a procedure very similar to EP that is actually a special instance of Power EP[5]. Identifying $\tilde{f}_i(x) = \exp(\langle\lambda_i,\phi^0(x)\rangle)$ and $f_i(x) = \exp(\langle\theta^i,\phi^i(x)\rangle)$ we can define convex EP as:

1. Initialize $q(x) \propto f_0(x)\prod_i \tilde{f}_i(x)$
2. Create $q^{\backslash i}(x) = q(x)f(x)^{1/\rho_i - 1}$
3. Minimize KL $\left(q^{\backslash i}(x)f(x)^{\frac{1}{\rho_i}}||q(x)\right)$
4. Update $\tilde{f}_i(x) = q(x)/q^{\backslash i}(x)$

Note that this algorithm has a kind of hysteresis: powers of the f_i are subtracted out rather than all of them. This is an interesting effect that helps explain why this algorithm is expected to perform better in parallel: by keeping some "old" information, the convex algorithm is less likely to get stuck in a fixed loop. As we shall show in our experiments, this seems to be true in practice.

5 Evaluation

We now turn to a practical evaluation of the derived robustified EP. We chose to evaluate the robustified EP on Ising model instances because of the model universality, ease of parametrization and relatively straightforward implementation for the GPU. We start by a short introduction to the Ising model and give an operational definition of the inference task we are solving. After rapidly going through the actual implementation, we present a complete set of measurements which describe the algorithm performance with respect to convergence, accuracy and speed.

5.1 Ising model

The two dimensional Ising model is defined by the quartet $(n,m,(\theta_i)_i,(\theta_{i,j})_{i,j})$ where $n,m \in \mathbb{N}$ represent the

number of rows and columns of the model grid, $\theta_{\cdot} \in \mathbb{R}^{nm}$ define the singleton potentials and $\theta_{\cdot,\cdot} \in \mathbb{R}^{2nm-n-m}$ the pair potentials. For 0/1 valued variables, the corresponding joint probability writes:

$$p(x_1, \ldots, x_{nm}) \propto \prod_i e^{\theta_i x_i} \prod_{i,j} e^{\theta_{i,j}(x_i x_j + (1-x_i)(1-x_j))}$$

which we usually express in the log domain:

$$\log p \propto \sum_i \theta_i x_i + \sum_{i,j} \theta_{i,j}(x_i x_j + (1-x_i)(1-x_j))$$

There are mainly two inference tasks one is interested in. The obvious one is determining the mode of $\log p$, e.g. the configuration of x variables which maximize the probability. A second one is computing the nm marginals $p(x_i)$. This is precisely what EP as previously defined achieves. In particular, the goal is to approximate p by a simpler distribution $\tilde{p}$ where each variable is decoupled.

$$\log \tilde{p} \propto \sum_i \tilde{\theta}_i x_i$$

It is clear that computing the $\tilde{\theta}$ parameters directly yields the marginals we want, by namely $p(x_i) \approx \frac{1}{1+e^{-\tilde{\theta}_i}}$. Thus, to evaluate our algorithm, we compute the $\tilde{\theta}$ parameters exactly by explicit marginalization when possible (in practice, the problem becomes intractable for sizes larger than 5×6 with our naive exact implementation), and approximately using our algorithm.

Ising models exhibit a rich behavior which depends on how the different parameters are chosen. Table 1 defines the typology of Ising models we use in the evaluation.

Name	θ_i	Name	$\theta_{i,j}$
negative	$\mathrm{Unif}[-1;0]$	strongly repulsive	$\mathrm{Unif}[-3;0]$
zero	0	repulsive	$\mathrm{Unif}[-1;0]$
mixed	$\mathrm{Unif}[-1;1]$	mixed	$\mathrm{Unif}[-1;1]$
positive	$\mathrm{Unif}[0;1]$	strongly mixed	$\mathrm{Unif}[-3;3]$
		attractive	$\mathrm{Unif}[0;1]$
		strongly attractive	$\mathrm{Unif}[0;3]$

Table 1: Singleton and pair potential typology. Unif is the uniform distribution.

5.2 Implementation

We have implemented an exact marginalization code, a vanilla sequential EP and an OpenCL parallel version of EP. As the difference between the naive parallelization of EP and the convexified parallelization of EP is only a parameter (the power exponent ρ), there is no separate implementation for the robustified EP. In what follows, we call *convex EP* a parallel EP with $\rho = 2$, and *pseudo-convex EP* a parallel EP with $\rho = 1.5$. For all of the implementations, we ran into underflow issues which were successfully solved by performing all the computations in the log domain, namely, we only work with log values. This is not a problem where the original algorithm contains multiplication and exponentiation, but requires an log-exp operation and a conditional[3] whenever we add two log values. This significantly reduces performance.

Following OpenCL terminology, the heart of the parallel implementation is a kernel function (75 lines of C code) which is executed by the workers of the OpenCL target device. The parallelization is such that each worker is identified to an edge of the model. Two in device-memory tables contain the computed messages (one for the messages of the previous iteration, one for the current messages being computed) and the corresponding input pointers are swapped between each iteration. In sum, each worker updates its portion of the messages table, and a corresponding one full sequential iteration of EP is achieved when all the workers on the edges finished their updates. The only conditionals the kernel contains are these introduced by the previously mentioned log add operation.

In effect, the OpenCL implementation is quite concise if one ignores the overhead of the setup code (specifying the device, creating the command queues, compiling the kernel for the device, adjusting parameters, moving data around, ...).

As for any optimization algorithm, we are interested in evaluating three characteristics: convergence, defined as the number of iterations before the output changes by less than a given threshold; accuracy defined as the error between the approximate and the exact solution once convergence occurs; and last but not least speed.

5.3 4×4 models

We start by determining challenging types of Ising models both in terms of convergence and accuracy for our algorithm. We use 4×4 Ising instances for a complete exploration of the space. Figure 3 presents how convergence (if any) occurs on the $4 \times 6 = 24$ different types of Ising instances. Figure 4 presents the corresponding results in terms of accuracy.

On 4×4 instances, instability rarely occurs. In effect, only positive strongly attractive instances result in an exploding number of iterations. As expected, the vanilla parallel EP usually results in longer convergence times. The convex EP does significantly reduce the convergence time in the strongly repulsive case and does converge in the positive strongly attractive case, but introduces more iterations

[3] $\log(e^a + e^b) = a + \log(1 + \exp(b-a))$ if $a > b$; $\log(e^a + e^b) = b + \log(1 + \exp(a-b))$ else

on strongly mixed cases. The pseudo-convex EP has a beneficial effect on these but tends to increase instability for the positive strongly attractive case.

The price we pay for the improved convergence is depicted in figure 4. Both strongly repulsive and strongly mixed instances are generally hard for the algorithms. However, there is an order of magnitude more error for the convex EP when compared to the vanilla EP. Interestingly, there appears to be no significant accuracy difference between the sequential EP and the naive parallel EP. Finally, we can observe that pseudo-convex EP has a significantly better accuracy than convex EP.

Based on this exploration of the convergence-accuracy space, we want to focus on three interesting Ising instances: negative strongly mixed, positive strongly repulsive and positive strongly repulsive.

5.4 Increasing model size

Using these three families of instances, we explore how convergence and accuracy is affected by the model size. Figure 5 shows the convergence and accuracy data for these three types across 3 increasing model sizes. For negative strongly mixed and positive strongly repulsive, there is no significant difference in the number of iterations before convergence. On the contrary, increasing size tends to overall lead to more instability with size on positive strongly attractive cases. On the accuracy side, there is a global degradation for all the algorithms on all instances, with the noticeable exception of the positive strongly attractive case which always leads to a very accurate solution, and the positive strongly repulsive case which turns out to be significantly more difficult on 25 nodes for the vanilla EP.

Figure 6 shows how the convergence evolves with larger instance sizes of positive strongly attractive instances. As expected, convex EP is the most stable algorithm, followed by pseudo-convex EP. In comparison, vanilla parallel EP is strongly divergent.

Last but not least, figure 7 shows how the iteration time evolves for plain sequential EP, CPU-parallelized EP and GPU-parallelized EP. The CPU parallelization becomes competitive with the sequential EP around 200 nodes instances (15×15 Ising instances), while the larger overhead of the GPU parallelization makes it a quite effective competitor around 2000 nodes (45×45 nodes). The bi-modal behavior of the GPU implementation for large instances is most probably due to the limited amount of on-chip memory available (256Mb). As for the smaller instances, we suspect some interference between the OS and the GPU, such as GUI rendering.

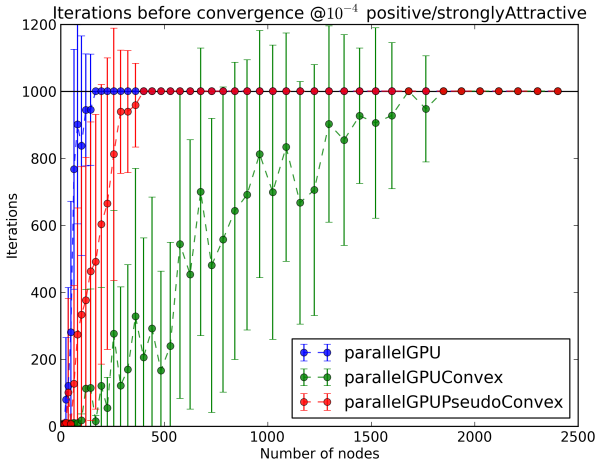


Figure 6: Eventually, both convex and pseudo-convex EP diverge on large positive strongly attractive instances. The maximum number of iterations was set to 1000 in this measurement.

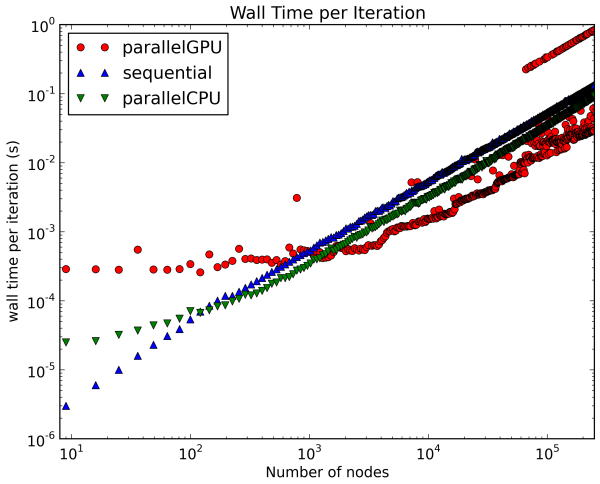


Figure 7: Log-log plot of the wall time per iteration of sequential EP and parallel EPs (naive, convex and pseudo-convex) ran on a 4 cores Intel i7 CPU and a low end laptop GPU (AMD Radeon HD 6490M). Asymptotic slopes are unitary as the algorithm is linear in the number of nodes.

6 Conclusion

We have derived a robustified version of expectation propagation and implemented it on an OpenCL platform. The evaluation shows an improved stability for both convex and pseudo-convex expectation propagation at the cost of a decreased accuracy. The GPU implementation is only faster than sequential expectation propagation for Ising instances larger than 45×45 nodes.

References

[1] N. L. Ackerman, C. E. Freer, and D. M. Roy. Noncomputable conditional distributions. In *Proc. of the 26th Ann. Symp. on Logic in Comp. Sci.* IEEE Press, 2011.

[2] B. Cseke and T. Heskes. Improving posterior marginal approximations in latent gaussian models. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 121–128, 2010.

[3] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

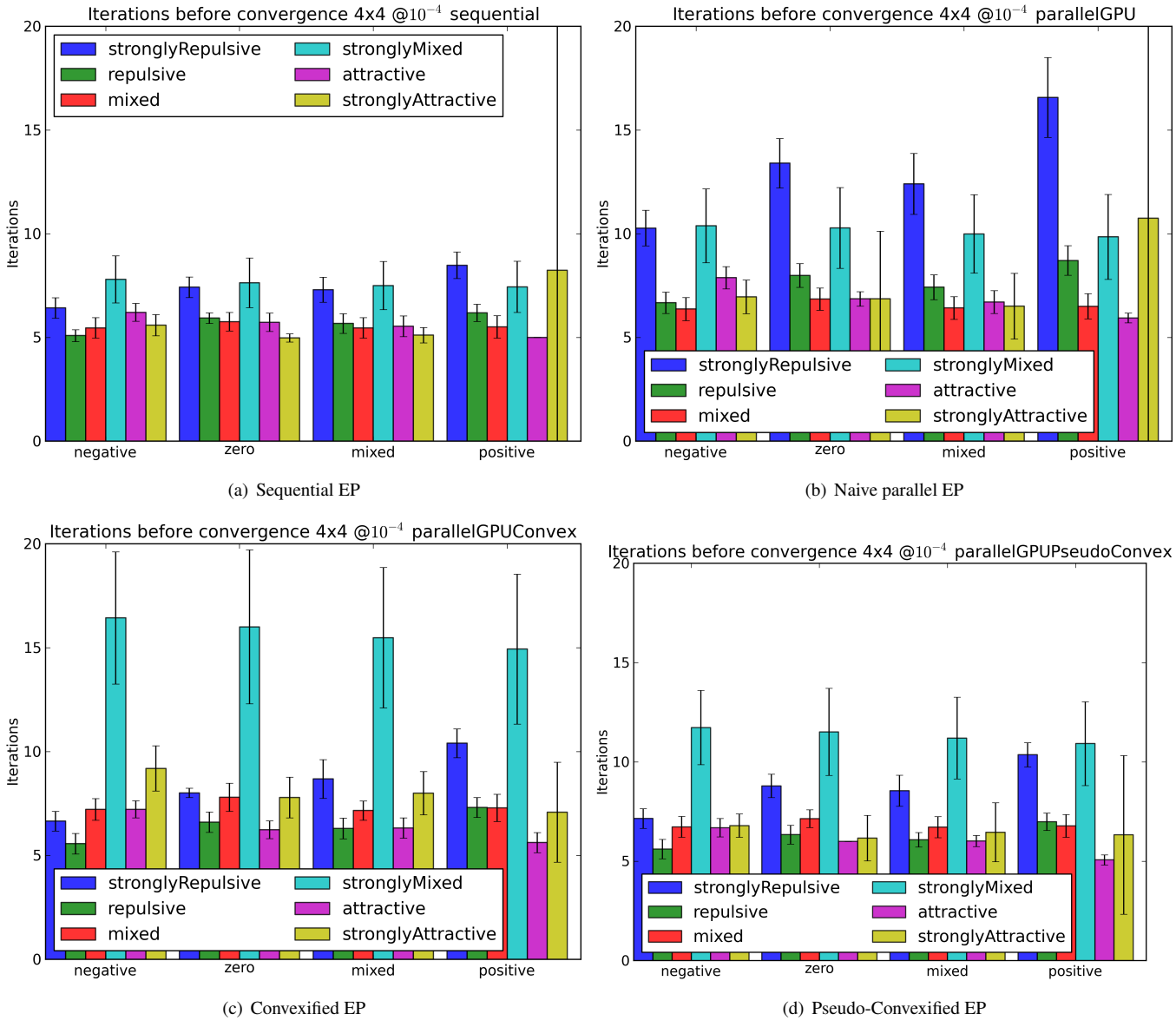
[4] T. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 362–369, San Francisco, CA, 2001. Morgan Kaufmann.

[5] T. Minka. Power EP. Technical Report MSR-TR-2004-149, Microsoft Research, Cambridge, Jan. 2004.

[6] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[7] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching. In *In AISTATS*, 2003.

[8] M. J. Wainwright and M. I. Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc., Hanover, MA, USA, 2008.



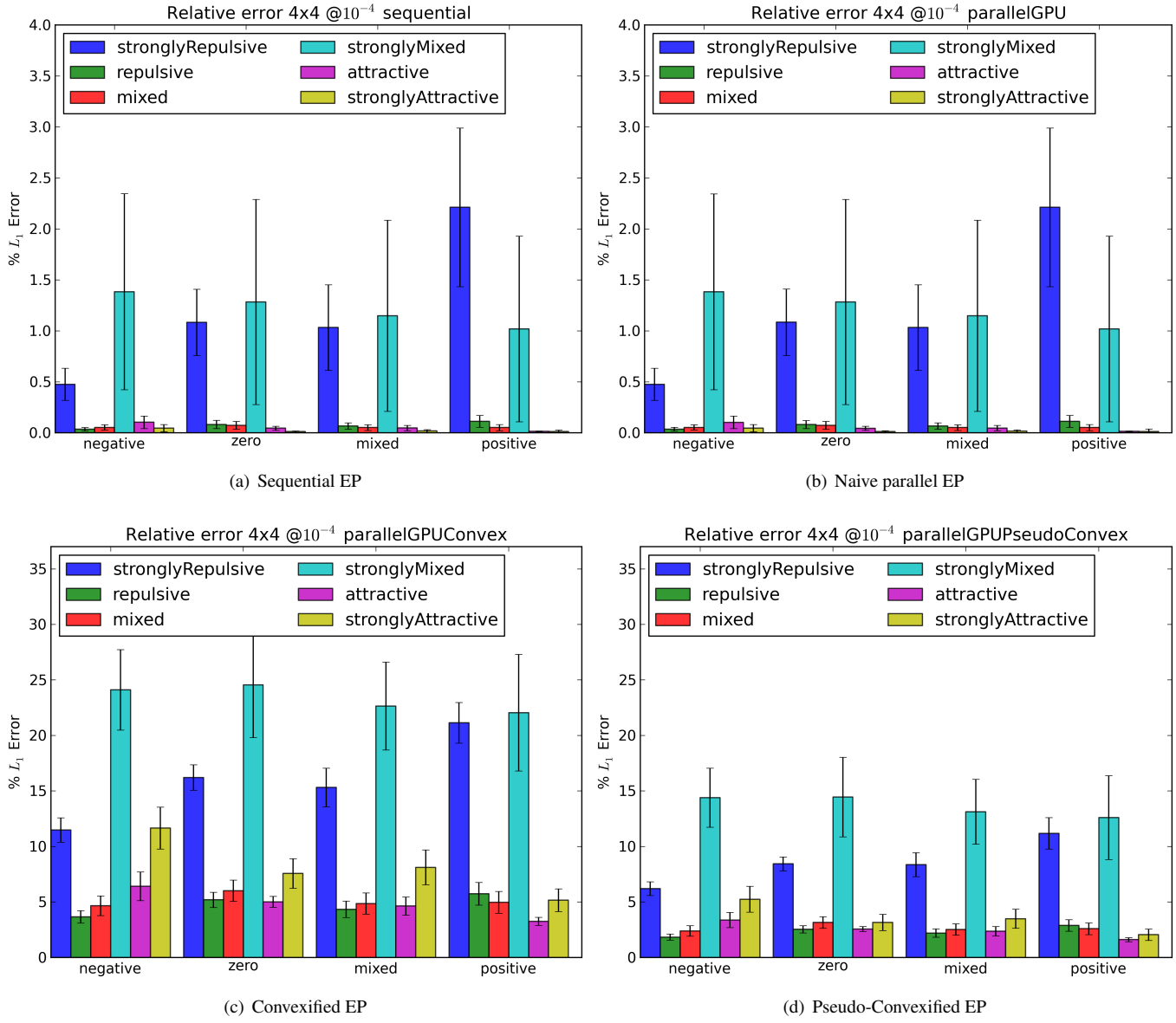
(a) Sequential EP

(b) Naive parallel EP

(c) Convexified EP

(d) Pseudo-Convexified EP

Figure 3: Number of iterations before computed marginals change less than 10^{-4} in relative L_1 norm. Results are averaged over 100 4×4 Ising instances per Ising type. Error bars are shown at 1 sigma.



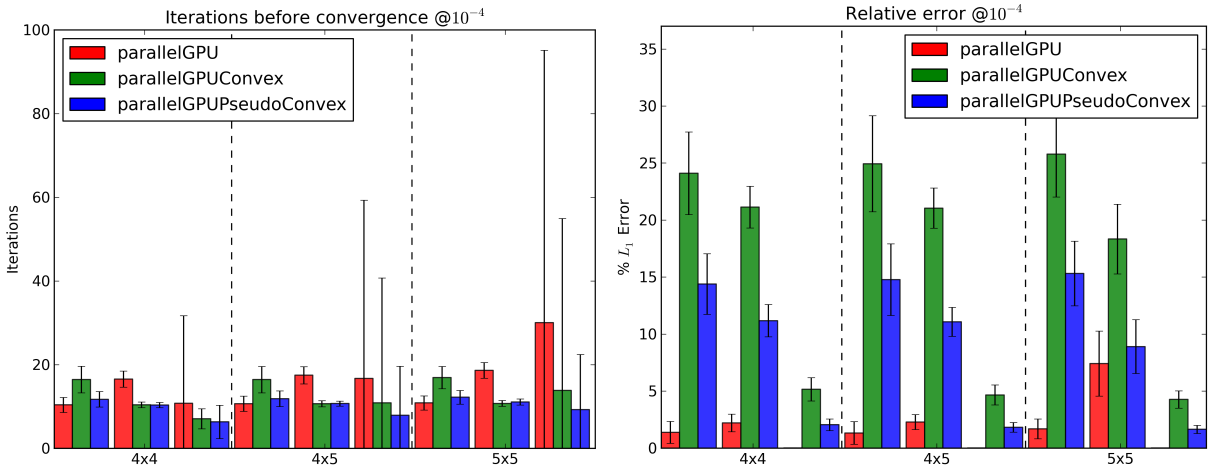
(a) Sequential EP

(b) Naive parallel EP

(c) Convexified EP

(d) Pseudo-Convexified EP

Figure 4: Relative L_1 error after 10^{-4} convergence. Results are averaged over 100 4×4 Ising instances per Ising type. Error bars are shown at 1 sigma.



(a) Convergence

(b) Accuracy

Figure 5: Convergence and accuracy measures over 4×4, 4×5 and 5×5 models. Each group of 3 bars within a given instance size respectively represent negative strongly mixed, positive strongly repulsive and positive strongly attractive instances. The convexified and pseudo-convexified EP versions do improve convergence at a significant cost in terms of accuracy with respect to the naive parallel EP.