Solving Combinatorial Optimization Problems using Relaxed Linear Programming: A High Performance Computing Perspective

Chen Jin†*, Qiang Fu♯*, Huahua Wang♯, Ankit Agrawal†, William Hendrix†,
Wei-keng Liao†, Md. Mostofa Ali Patwary†, Arindam Banerjee♯, Alok Choudhary†

♯University of Minnesota, Twin Cities †Northwestern University

†{chen.jin, ankitag, whendrix, wkliao, mpatwary, choudhar}@eecs.northwestern.edu
♯{qifu, huwang, banerjee}@cs.umn.edu

ABSTRACT

Several important combinatorial optimization problems can be formulated as maximum a posteriori (MAP) inference in discrete graphical models. We adopt the recently proposed parallel MAP inference algorithm Bethe-ADMM and implement it using message passing interface (MPI) to fully utilize the computing power provided by the modern supercomputers with thousands of cores. The empirical results show that our parallel implementation scales almost linearly even with thousands of cores.

Keywords

alternating direction method of multipliers, Markov random field, maximum a posteriori inference, message passing interface

1. INTRODUCTION

Several important combinatorial optimization problems such as max-cut, vertex cover and independent set, etc., can be formulated as *maximum a posteriori* (MAP) inference problems on discrete graphical models with appropriate underlying dependency graphs and potentials [20]. In general, the combinatorial problems are NP-hard and hence the corresponding MAP inference problems. Existing approaches to solving them often consider a linear programming (LP) relaxation of the integer program. Over the past few years, several algorithms have been proposed to solve such graph-structured LPs [19, 10, 14, 13]. Such approaches can be broadly classified into two groups: primal methods which work with the original variables [17] and dual methods, which works on the dual variables [18].

One of the key limitations of many existing MAP inference algorithms is that they are inherently sequential and thus do not scale to large graphical models. However, given that in many application domains, datasets are available at much higher resolutions, we need algorithms for solving graph structured LPs which efficiently scale to problem sizes of millions or hundreds of millions of nodes. Consider the problem of detecting droughts from precipitation data of the past 100 years at a temporal resolution of a month and spatial resolution of $0.5^\circ \times 0.5^\circ$ over land. A 3-dimensional MRF (latitude × longitude × time) with neighborhood dependencies is a suitable model for such analysis since droughts have both spatial and temporal continuity. Assuming a boolean indicator variable of drought at each space-time location, the graph-structured LP relaxation of the MAP inference problem in this context has to work with approximately 7 million variables and about double that many constraints.

In this paper, we adopt the recently proposed Bethe-ADMM algorithm [8] for solving graph-structured LPs. The overall structure of the algorithm is based on two ideas: tree-based decomposition of a graph-structured LP [19] and the alternating direction method of multipliers (ADMM) [4]. The tree decomposition breaks the problem into small but overlapping parts, each involving small number of variables and constraints. The algorithm iterates between doing updates to variables in individual parts in parallel followed by suitable aggregation, all within the framework of ADMM. However, unlike standard ADMM, Bethe-ADMM is a novel inexact ADMM augmented by a Bregman divergence induced by the Bethe entropy. The unusual modification in Bethe-ADMM leads to an efficient projection of partial solutions to subsets of constraints, which results in highly efficient iterations and avoids double-loop algorithm.

To illustrate the efficiency of the Bethe-ADMM algorithm, we implement it using message passing interface (MPI), which is a natural fit for the parallel algorithm given its flexible message passing mechanism, along with its portability and wide adoption in distributed and high performance clusters. The other advantage of using MPI is that its I/O interface is optimized for a wide variety of underlying parallel file systems (PFS) and sustains high I/O bandwidth. We evaluate our algorithms on a simulation and a real precipitation dataset, which are both of large scale. The empirical results show that we manage to obtain almost linear speedup in the number of cores used.

*indicates equal contribution.

BigMine'13, August 11-14 2013, Chicago, IL, USA
Copyright is held by the owner/author(s).
Publication rights licensed to ACM.
ACM 978-1-4503-2324-6/13/08 ...$15.00.

The rest of the paper is organized as follows: We briefly review the MAP inference problem and its connections to some combinatorial problems in Section 2. We introduce the Bethe-ADMM algorithm in Section 3, and discuss its MPI implementation in detail in Section 4. We present the experimental results in Section 5 and conclude in Section 6.

2. PROBLEM DEFINITION

A pairwise Markov random field (MRF) is defined on an undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set. Each node $u \in V$ has a random variable X_u associated with it, which can take value x_u in some discrete space $\mathcal{X} = \{1, \ldots, k\}$. Concatenating all the random variables X_u, $\forall u \in V$, we obtain an n dimensional random vector $\boldsymbol{X} = \{X_u | u \in V\} \in \mathcal{X}^n$. We assume that the distribution P of $\boldsymbol{X}$ is a Markov Random Field [20], meaning that it factors according to the structure of the undirected graph G as follows: With $f_u : \mathcal{X} \mapsto \mathbb{R}$, $\forall u \in V$ and $f_{uv} : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, $\forall (u,v) \in E$ denoting nodewise and edgewise potential functions respectively, the distribution takes the form $P(\boldsymbol{x}) \propto \exp\left\{\sum_{u \in V} f_u(x_u) + \sum_{(u,v)\in E} f_{uv}(x_u, x_v)\right\}$.

An important problem in the context of MRF is that of *maximum a posteriori* (MAP) inference, which is the following integer programming (IP) problem:

$$\boldsymbol{x}^* \in \operatorname*{argmax}_{\boldsymbol{x} \in \mathcal{X}^n} \left\{ \sum_{u \in V} f_u(x_u) + \sum_{(u,v)\in E} f_{uv}(x_u, x_v) \right\} . \quad (1)$$

Several important combinatorial optimization problems can be formulated as MAP inference problems, e.g., the max-cut problem, which is given a nonnegative weight $w_{uv} \leq 0$ for each edge (u,v) of an undirected graph, find a patition (U, U^c) of the vertex set such that the weight of edges across the partition is maximized:

$$\max_{U} \sum_{\{(u,v)|u \in U, v \in U^c\}} W_{uv} . \quad (2)$$

To cast (2) as a MAP inference problem, let $\boldsymbol{X} \in \{0,1\}^n$ be an binary membership vector, meaning that $\boldsymbol{X}_u = 1$ if and only if $u \in U$. Then define nodewise potentials $f_u(x_u) = 0$ for all vertices and define the pairwise potentials

$$f_{uv}(x_u, x_v) = \begin{pmatrix} 0 & w_{uv} \\ w_{uv} & 0 \end{pmatrix} . \quad (3)$$

It is easy to show (1) is equivalent to the max-cut problem.

The complexity of (1) depends critically on the structure of the underlying graph. When G is a tree structured graph, the MAP inference problem can be solved efficiently via the max-product algorithm [11]. However, for an arbitrary graph G, the MAP inference algorithm is usually computationally intractable. The intractability motivates the development of algorithms to solve the MAP inference problem approximately. In this paper, we focus on the Linear Programming (LP) relaxation method [19, 6]. The LP relaxation of MAP inference problem is defined on a set of pseudomarginals μ_u and μ_{uv}, which are non-negative, normalized and locally consistent [19, 6]:

$$\begin{aligned} \mu_u(x_u) &\geq 0 , \quad \forall u \in V , \\ \sum_{x_u \in \mathcal{X}_u} \mu_u(x_u) &= 1, \quad \forall u \in V , \\ \mu_{uv}(x_u, x_v) &\geq 0, \quad \forall (u,v) \in E , \\ \sum_{x_u \in \mathcal{X}_u} \mu_{uv}(x_u, x_v) &= \mu_v(x_v), \quad \forall (u,v) \in E . \end{aligned} \quad (4)$$

We denote the polytope defined by (4) as $L(G)$ and the LP relaxation of MAP inference problem (1) becomes solving the following LP:

$$\begin{aligned} \max_{\boldsymbol{\mu} \in L(G)} \langle \boldsymbol{\mu}, \boldsymbol{f} \rangle &= \sum_{u \in V} \sum_{x_u} \mu_u(x_u) f_u(x_u) \quad (5) \\ &+ \sum_{(uv)\in E} \sum_{x_u, x_v} \mu_{uv}(x_u, x_v) f_{uv}(x_u, x_v) , \end{aligned}$$

subject to the constraint that $\boldsymbol{\mu} \in L(G)$. If the solution $\boldsymbol{\mu}$ to (5) is an integer solution, it is guaranteed to be the optimal solution of (1). Otherwise, one can apply rounding schemes [16, 17] to round the fractional solution to an integer solution.

3. ALGORITHM

In this section, we first show how to solve (5) by the ADMM based on tree decomposition. The resulting algorithm can be a double loop algorithm since some updates do not have closed form solution. We then introduce the Bethe-ADMM algorithm where every update can be computed efficiently.

3.1 ADMM for MAP Inference

We first show how to decompose (5) into a series of subproblems. We can decompose the graph G into overlapping subgraphs and rewrite the optimization problem with consensus constraints to enforce the pseudomarginals on subgraphs (local variables) to agree with $\boldsymbol{\mu}$ (global variable). Throughout the paper, we focus on tree-structured decompositions. To be more specific, let $\mathbb{T} = \{(V_1, E_1), \ldots, (V_{|\mathbb{T}|}, E_{|\mathbb{T}|})\}$ be a collection of subgraphs of G which satisfies two criteria: (i) Each subgraph $\tau = (V_\tau, E_\tau)$ is a tree-structured graph and (ii) Each node $u \in V$ and each edge $(u,v) \in E$ is included in at least one subgraph $\tau \in \mathbb{T}$. We also introduce local variable $\boldsymbol{m}_\tau \in L(\tau)$ which is the pseudomarginal [19, 6] defined on each subgraph τ. We use $\boldsymbol{\theta}_\tau$ to denote the potentials on subgraph τ. We denote $\boldsymbol{\mu}_\tau$ as the components of global variable $\boldsymbol{\mu}$ that belong to subgraph τ. Note that since $\boldsymbol{\mu} \in L(G)$ and τ is a tree-structured subgraph of G, $\boldsymbol{\mu}_\tau$ always lies in $L(\tau)$. In the newly formulated optimization problem, we will impose consensus constraints for sharing nodes and edges. For the ease of exposition, we simply use the equality constraint $\boldsymbol{\mu}_\tau = \boldsymbol{m}_\tau$ to enforce the consensus.

The new optimization problem we formulate based on graph decomposition is then as follows:

$$\min_{\boldsymbol{m}_\tau, \boldsymbol{\mu}} \sum_{\tau=1}^{|\mathbb{T}|} \rho_\tau \langle \boldsymbol{m}_\tau, \boldsymbol{\theta}_\tau \rangle \quad (6)$$

$$\text{subject to} \quad \boldsymbol{m}_\tau - \boldsymbol{\mu}_\tau = 0, \quad \tau = 1, \ldots, |\mathbb{T}| \quad (7)$$

$$\boldsymbol{m}_\tau \in L(\tau), \quad \tau = 1, \ldots, |\mathbb{T}| \quad (8)$$

where ρ_τ is a positive constant associated with each subgraph. We use the consensus constraints (7) to make sure that the pseudomarginals agree with each other in the sharing components across all the tree-structured subgraphs. Besides the consensus constraints, we also impose feasibility constraints (8), which guarantee that, for each subgraph, the local variable $\boldsymbol{m}_\tau$ lies in $L(\tau)$. When the constraints (7) and (8) are satisfied, the global variable $\boldsymbol{\mu}$ is guaranteed to lie in $L(G)$.

To make sure that problem (5) and (6) are equivalent, we also need to guarantee that

$$\min_{\boldsymbol{m}_\tau} \sum_{\tau=1}^{|\mathbb{T}|} \rho_\tau \langle \boldsymbol{m}_\tau, \boldsymbol{\theta}_\tau \rangle = \max_{\boldsymbol{\mu}} \langle \boldsymbol{\mu}, \boldsymbol{f} \rangle \ , \tag{9}$$

assuming the constraints (7) and (8) are satisfied. It is easy to verify that, as long as (9) is satisfied, the choice of ρ_τ and $\boldsymbol{\theta}_\tau$ do not change the problem. Let $\mathbf{1}[.]$ be a binary indicator function and $\boldsymbol{l} = -\boldsymbol{f}$. A straightforward approach to obtain the potential $\boldsymbol{\theta}_\tau$ can be:

$$\theta_{\tau,u}(x_u) = \frac{l_u(x_u)}{\sum_{\tau'} \rho_{\tau'} \mathbf{1}[u \in V_{\tau'}]}, u \in V_\tau \ ,$$

$$\theta_{\tau,uv}(x_u, x_v) = \frac{l_{uv}(x_u, x_v)}{\sum_{\tau'} \rho_{\tau'} \mathbf{1}[(u,v) \in E_{\tau'}]}, (u,v) \in E(\tau) \ .$$

Plugging in the equality constraints, we then have the augmented Lagrangian of (6) as:

$$L(\boldsymbol{m}_\tau, \boldsymbol{\mu}_\tau, \boldsymbol{\lambda}_\tau) = \sum_{\tau=1}^{|\mathbb{T}|} \left(\rho_\tau \langle \boldsymbol{m}_\tau, \boldsymbol{\theta}_\tau \rangle + \langle \boldsymbol{\lambda}_\tau, \boldsymbol{m}_\tau - \boldsymbol{\mu}_\tau \rangle + \frac{\beta}{2} ||\boldsymbol{m}_\tau - \boldsymbol{\mu}_\tau||_2^2 \right) , \tag{10}$$

where $\boldsymbol{\lambda}_\tau$ is the dual variable and $\beta > 0$ is the penalty parameter. The following updates constitute a single iteration of the ADMM [4]:

$$\boldsymbol{m}_\tau^{t+1} = \operatorname*{argmin}_{\boldsymbol{m}_\tau \in L(\tau)} \langle \boldsymbol{m}_\tau, \rho_\tau \boldsymbol{\theta}_\tau + \boldsymbol{\lambda}_\tau^t \rangle + \frac{\beta}{2} ||\boldsymbol{m}_\tau - \boldsymbol{\mu}_\tau^t||_2^2 \ , \tag{11}$$

$$\boldsymbol{\mu}^{t+1} = \operatorname*{argmin}_{\boldsymbol{\mu}} \sum_{\tau=1}^{|\mathbb{T}|} \left(-\langle \boldsymbol{\mu}_\tau, \boldsymbol{\lambda}_\tau^t \rangle + \frac{\beta}{2} ||\boldsymbol{m}_\tau^{t+1} - \boldsymbol{\mu}_\tau||_2^2 \right) , \tag{12}$$

$$\boldsymbol{\lambda}_\tau^{t+1} = \boldsymbol{\lambda}_\tau^t + \beta(\boldsymbol{m}_\tau^{t+1} - \boldsymbol{\mu}_\tau^{t+1}) \ . \tag{13}$$

Now, the problem turns to whether the updates (11) and (12) can be solved efficiently which we analyze as follows:

Updating $\boldsymbol{\mu}$: Since we have an unconstrained optimization problem (12) and the objective function decomposes component-wisely, taking the derivatives and setting them to zero yield the solution. In particular, let S_u be the set of subgraphs which contain node u, for the node components, we have:

$$\mu_u^{t+1}(x_u) = \frac{1}{|S_u|\beta} \sum_{\tau \in S_u} \left(\beta m_{\tau,u}^{t+1}(x_u) + \lambda_{\tau,u}^t(x_u) \right) . \tag{14}$$

(14) can be further simplified by observing that $\sum_{\tau \in S_u} \lambda_{\tau,u}^t(x_u) = 0$:

$$\mu_u^{t+1}(x_u) = \frac{1}{|S_u|} \sum_{\tau=1}^{T} m_{\tau,u}^{t+1}(x_u) \ . \tag{15}$$

Similarly, let S_{uv} be the subgraphs which contain edge (u,v) and the update for the edge components is:

$$\mu_{u,v}^{t+1}(x_u, x_v) = \frac{1}{|S_{uv}|} \sum_{\tau \in S_{uv}} m_{\tau,uv}^{t+1}(x_u, x_v) \ . \tag{16}$$

Updating $\boldsymbol{m}_\tau$: We need to solve a quadratic optimization problem for each tree-structured subgraph. Unfortunately, we do not have a close-form solution for (11) in general. One possible approach, similar to the proximal algorithm, is to first obtain the solution $\tilde{\boldsymbol{m}}_\tau$ to the unconstrained problem of (11) and then project $\tilde{\boldsymbol{m}}_\tau$ to $L(\tau)$:

$$\boldsymbol{m}_\tau = \operatorname*{argmin}_{\boldsymbol{m} \in L(\tau)} ||\boldsymbol{m} - \tilde{\boldsymbol{m}}_\tau||_2^2 \ . \tag{17}$$

If we adopt the cyclic Bregman projection algorithm [5] to solve (17), the algorithm becomes a double-loop algorithm, i.e., the cyclic projection algorithm projects the solution to each individual constraint of $L(\tau)$ until convergence and the projection algorithm itself is iterative.

3.2 Bethe-ADMM

Instead of solving (11) exactly, a common way in inexact ADMMs [21, 9] is to linearize the objective function in (11), i.e., the first order Taylor expansion at $\boldsymbol{m}_\tau^t$, and add a new quadratic penalty term such that

$$\boldsymbol{m}_\tau^{t+1} = \operatorname*{argmin}_{\boldsymbol{m}_\tau \in L(\tau)} \langle \mathbf{y}_\tau^t, \boldsymbol{m}_\tau - \boldsymbol{m}_\tau^t \rangle + \frac{\alpha}{2} ||\boldsymbol{m}_\tau - \boldsymbol{m}_\tau^t||_2^2 \ , \tag{18}$$

where α is a positive constant and

$$\mathbf{y}_\tau^t = \rho_\tau \boldsymbol{\theta}_\tau + \boldsymbol{\lambda}_\tau^t + \beta(\boldsymbol{m}_\tau^t - \boldsymbol{\mu}_\tau^t) \ . \tag{19}$$

However, as discussed in the previous section, the quadratic problem (18) is generally difficult for a tree-structured graph and thus the conventional inexact ADMM does not lead to an efficient update for $\boldsymbol{m}_\tau$. Next we show that, by taking the tree structure into account, an inexact minimization of (11) augmented with a Bregman divergence induced by Bethe entropy leads to efficient update of $\boldsymbol{m}_\tau$.

The basic idea in the new algorithm is that we replace the quadratic term in (18) with a Bregman-divergence term $d_\phi(\boldsymbol{m}_\tau || \boldsymbol{m}_\tau^t)$ such that

$$\boldsymbol{m}_\tau^{t+1} = \operatorname*{argmin}_{\boldsymbol{m}_\tau \in L(\tau)} \langle \mathbf{y}_\tau^t, \boldsymbol{m}_\tau - \boldsymbol{m}_\tau^t \rangle + \alpha d_\phi(\boldsymbol{m}_\tau || \boldsymbol{m}_\tau^t) \ , \tag{20}$$

is efficient to solve for tree τ. Expanding the Bregman divergence and removing the constants, we can rewrite (20) as

$$\boldsymbol{m}_\tau^{t+1} = \operatorname*{argmin}_{\boldsymbol{m}_\tau \in L(\tau)} \langle \mathbf{y}_\tau^t / \alpha - \nabla\phi(\boldsymbol{m}_\tau^t), \boldsymbol{m}_\tau \rangle + \phi(\boldsymbol{m}_\tau). \tag{21}$$

For a tree-structured problem, what convex function $\phi(\boldsymbol{m}_\tau)$ should we choose? Recall $\boldsymbol{m}_\tau$ defines the marginal distributions of a tree-structured distribution $p_{\boldsymbol{m}_\tau}$ over the nodes and edges:

$$\boldsymbol{m}_{\tau,u}(x_u) = \sum_{\neg x_u} p_{\boldsymbol{m}_\tau}(x_1, \ldots, x_u, \ldots, x_n), \ \forall u \in V_\tau \ ,$$

$$\boldsymbol{m}_{\tau,uv}(x_u, x_v) = \sum_{\neg x_u, \neg x_v} p_{\boldsymbol{m}_\tau}(x_1, \ldots, x_u, x_v, \ldots, x_n), \ \forall (uv) \in E_\tau \ .$$

It is well known that the sum-product algorithm [11] efficiently computes the marginal distributions for a tree structured graph. It can also be shown that the sum-product algorithm solves the following optimization problem [20] for tree τ:

$$\max_{\boldsymbol{m}_\tau \in L(\tau)} \langle \boldsymbol{m}_\tau, \boldsymbol{\eta}_\tau \rangle + H_{Bethe}(\boldsymbol{m}_\tau) \ , \tag{22}$$

where $H_{Bethe}(\boldsymbol{m}_\tau)$ is the Bethe entropy of $\boldsymbol{m}_\tau$. The Bethe entropy on tree τ is defined as:

$$H_{Bethe}(\boldsymbol{m}_\tau) = \sum_{u \in V_\tau} H_u(\boldsymbol{m}_{\tau,u}) - \sum_{(u,v) \in E_\tau} I_{uv}(\boldsymbol{m}_{\tau,uv}) \ , \quad (23)$$

where $H_u(\boldsymbol{m}_{\tau,u})$ is the entropy function on each node $u \in V_\tau$ and $I_{uv}(\boldsymbol{m}_{\tau,uv})$ is the mutual information on each edge $(u,v) \in E_\tau$.

Combing (21) and (22), we set $\boldsymbol{\eta}_\tau = \nabla\phi(\boldsymbol{m}_\tau^t) - \mathbf{y}_\tau^t/\alpha$ and choose ϕ to be the negative Bethe entropy of $\boldsymbol{m}_\tau$ so that (21) can be solved efficiently in linear time via the sum-product algorithm.

For the sake of completeness, we summarize Bethe-ADMM algorithm as follows :

$$\boldsymbol{m}_\tau^{t+1} = \underset{\boldsymbol{m}_\tau \in L(\tau)}{\text{argmin}} \ \langle \mathbf{y}_\tau^t/\alpha - \nabla\phi(\boldsymbol{m}_\tau^t), \boldsymbol{m}_\tau \rangle + \phi(\boldsymbol{m}_\tau) \ , \quad (24)$$

$$\boldsymbol{\mu}^{t+1} = \underset{\boldsymbol{\mu}}{\text{argmin}} \sum_{\tau=1}^{T} \left(-\langle \boldsymbol{\lambda}_\tau^t, \boldsymbol{\mu}_\tau \rangle + \frac{\beta}{2} ||\boldsymbol{m}_\tau^{t+1} - \boldsymbol{\mu}_\tau||_2^2 \right) , \quad (25)$$

$$\boldsymbol{\lambda}_\tau^{t+1} = \boldsymbol{\lambda}_\tau^t + \beta(\boldsymbol{m}_\tau^{t+1} - \boldsymbol{\mu}_\tau^{t+1}) \ , \quad (26)$$

where $\mathbf{y}_\tau^t$ is defined in (19) and ϕ is defined in (23). Due to the space constraint, we refer the readers to [8] for the detailed convergence analysis of the Bethe-ADMM algorithm.

It is easy to see that the update of $\boldsymbol{m}_\tau$ in (24) is independent for each tree τ, which motivates the parallel implementation of the Bethe-ADMM algorithm. We describe our implementation in detail in the next section.

4. PARALLEL IMPLEMENTATION

In this section, we explain the key components of our MPI implementation in detail. Our goal is to run the Bethe-ADMM algorithm on modern high performance computers with thousands of cores and it requires us to adopt the best parallelization practice. To achieve this goal, we carefully design our MPI implementation so that the underlying parallel computing architecture can be fully utilized.

Since the update of $\boldsymbol{m}_\tau$ in (24) for each tree is independent, the Bethe-ADMM algorithm is inherently parallel. In the parallel Bethe-ADMM algorithm, each process only maintains the information of a subset of trees in $\mathbb{T}$ and $\boldsymbol{m}_\tau$ is updated simultaneously. According to (25), the update of variable $\boldsymbol{\mu}$ involves averaging over $\boldsymbol{m}_\tau$ from the relevant trees. If these trees belong to different processes, the value of $\boldsymbol{m}_\tau$ needs to be exchanged among the processes so that $\boldsymbol{\mu}$ can be computed correctly. Because of the communication occurred among the processes, the message passing framework is a good fit for our parallel implementation. Hence, we implement the Bethe-ADMM algorithm using MPI. We also make the following implementation assumptions: (i) The MRF dependency graph is a regular grid shaped graph, e.g., two dimensional four nearest neighbor grid. (ii) Each tree structured subgraph is simply an edge of G. (iii)The input to the MAP inference algorithm is some data file, which has the potential and graph structure information.

An efficient parallel implementation is more challenging than an efficient sequential implementation. To fully utilize the computing power provided by the underlying parallel architecture, we need to address the following issues:

- How to design an efficient I/O scheme to load the data files, i.e., node potentials, edge potentials and graph structure?
- How to decompose the graph so that the work load on each process is balanced?
- How to efficiently figure out, for each process, what 'messages' it needs to exchange with other processes?

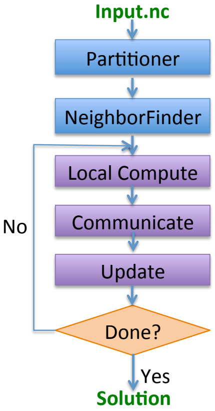


Figure 1: Bethe-ADMM parallel implementation.

We illustrate in Fig 1 the key components of our MPI implementation. We take advantage of the PFS so that processes can access the data file (input.nc) in parallel. We also design a simple heuristic to partition the graph to achieve load balancing. Making use of the graph structure information, we deploy a decentralized algorithm to figure out, for each MPI process, the information it needs to exchange with other processes. After each process reads the data file in parallel to fetch the relevant nodewise and edgewise potentials, it computes the local variables $\boldsymbol{m}_\tau$, communicate with other processes and update the global variable $\boldsymbol{\mu}$.

4.1 Parallel File Loading

The data file used as input to the MAP inference algorithm contains the nodewise and edgewise potentials and the graph structure information. We represent the graph as a set of edges with two node ids. (Figure 2(a) shows an example on a simple grid graph.) A naive way to load the data file is to have a master process read the entire data file and send to other slave processes the information they need. This approach is clearly not efficient because a slave process remains idle when other slave processes receive data from the master process. Our approach is to take advantage of the PFS, which stripes a file across multiple storage devices and enables parallel access to the data file.

To be more specific, we adopt the Pnetcdf [1] file format for parallel data file loading. The Pnetcdf is suitable for our implementation because the potential data and graph structure information can be easily stored as Pnetcdf multidimensional arrays. A Pnetcdf file also provides a rich suite of APIs that allow users to define metadata which describe datasets in details, such as the number of nodes and edges of a given graph, the type of graphs and the dimensions of the datasets. Moreover, it integrates tightly with MPI-IO and the underlying PFS so that our algorithm can achieve high degree of parallelism in terms of I/O operations.

4.2 Graph Partitioning

To take advantage of the parallel architecture, the work load should be split evenly among the processes and the partition should also minimize the intercommunication among the processes. This problem is usually NP hard and most practical solutions are based on heuristics. For example, in Pregel [12] and Giraph [2], the solution is to use node-centric partition, where assignment of a node to a partition depends solely on the node id. The simplest implementation is to calculate the hash value of each node id and modulus by N, where N is the number of partitions. However this simple heuristic comes with a cost that neighboring nodes are likely to be distributed on different processes and thus incur high communication overhead.

In our implementation, we adopt edge-centric partition, where we evenly divide the edges among all the processes. (Figure 2(b) shows the partition on a 2 × 3 grid.) Since the underlying dependency graph is a regular shaped grid graph, edge partition is empirically a good choice, as shown by the experimental results in Section 5.

4.3 Inter-process communication

After the graph decomposition step, each process reads from the input Pnetcdf file, retrieve the nodewise and edgewise potentials and compute $\boldsymbol{m}_\tau$. To compute $\boldsymbol{\mu}$, a simple solution is to have a master process collect the value of $\boldsymbol{m}_\tau$ from the slave processes and compute $\boldsymbol{\mu}$ according to (25). After $\boldsymbol{\mu}$ is updated, the master process has to send $\boldsymbol{\mu}$ back to each slave process so that $\boldsymbol{m}_\tau$ can be computed in the next iteration. This approach is clearly not efficient and we adopt a fully distributed algorithm: each process maintains a copy of the relevant elements of $\boldsymbol{\mu}$, receive $\boldsymbol{m}_\tau$ from other processes and update $\boldsymbol{\mu}$ according to (25).

Algorithm 1 NeighborFinder

```
procedure NeighborFinder
    idList = getNodeId()
    pairCount = idList.size()
    MPI_Allgather(
            $pairCount, 1, MPI\_INT,$
            $countArr, 1, MPI\_INT, comm)$
    Copy idList to sendBuf
    Construct $displacementArr$ from $countArr$
    MPI_Allgatherv(
            $sendBuf, 2 * pairCount, MPI\_INT,$
            $recvBuf, 2 * countArr, displacementArr,$
            $MPI\_INT, comm)$
    Compute neighbor processes by comparing idLists
    Count partial degree of sharing nodes
    Exchange partial degree with neighbor processes
    Compute full degree of sharing nodes
end procedure
```

To apply the above distributed algorithm, each process needs to figure out the neighbor processes with which it exchanges the value of $\boldsymbol{m}_\tau$. This can be done by comparing the node ids of each process and a pair of processes need to communicate with each other if they have sharing nodes. To be more specific, we compactly represent the node list of a process as a list of pairs $\{v_i, l_i\}$, where l_i is the length of continuous ids starting from v_i. (Figure 2(c) illustrates the compact representation of node lists on two processes.) Each process then gathers $\{(v_i, l_i)\}$ from all other processes, compare the lists with its own node list and decide what processes it communicates with. Beside deciding the neighbor processes, each process also needs to figure out the degrees of the sharing nodes. The degree (count) information will be used when the averaging operation is performed according to (25). As a result, the neighbor process also exchanges the local partial degree of the sharing nodes and compute the full degree accordingly. Algorithm 1 summaries the above procedure.

Algorithm 2 shows the details on the communication occurred among the processes: we reduce our communication cost by exchanging the partial sum of $\boldsymbol{m}_\tau$ rather than individual $\boldsymbol{m}_\tau$. We use asynchronous MPI APIs, which allows messages to be send or received asynchronously while not blocking the following operations.

Algorithm 2 Exchange $\boldsymbol{m}_\tau$ among neighbor processes

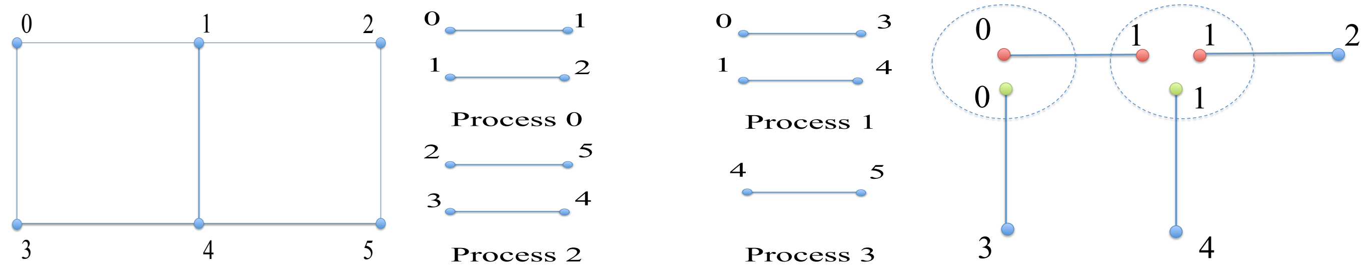
```
procedure ExchMsg
    for Each node $u$ do
        $partial\_sum[u] = 0$
    end for
    for Each edge $\tau$ $(u, v)$ do
        $partial\_sum[u]$ += $\boldsymbol{m}_\tau[u]$
        $partial\_sum[v]$ += $\boldsymbol{m}_\tau[v]$
    end for
    $idx = 0$
    MPI_Request request[neighbors.size() * 2]
    for $i$ in neighbors do
        $sharing\_node = getSharingNode(i)$
        copy partial_sum[sharing_node] to sendBuf
        MPI_ISend($sendBuf, k * sharing\_node.size(),$
                $MPI\_FLOAT, i, rank,$
                $comm, \&request[idx++])$
        MPI_IRecv($recvBuf, k * sharing\_node.size(),$
                $MPI\_FLOAT, i, i,$
                $comm, \&requests[idx++])$
    end for
end procedure
```

5. EXPERIMENTAL RESULTS

In this section, we present experimental results on a simulation dataset and a precipitation dataset. Our experiments are conducted on Hopper [3], the Cray XE6 parallel machine at the National Energy Research Scientific Computing Center. Hopper is a 6384 compute node cluster where each compute node consists of two twelve-core AMD MagnyCours processors with a theoretical peak performance of 8.4 GFlop/sec per core. 6000 compute nodes have 32 GB DD3 memory each and the rest have 64 GB memory each. Hopper runs “Cray Linux Environment” (CLE) operating system which is restricted low-overhead and optimized for high performance computing. The PFS is Lustre with 156 I/O servers (OSTs). The measured peak write performance on Hopper is 35 GB per second. To maximize the possible read bandwidth, we stripe our input file across 128 stripes and the file stripe size is set to 1 MB.

5.1 Simulation Dataset

We show experimental results on a simulation dataset. The underlying graph is a 2 dimensional 1,000 × 10,000 grid with $k = 3$ and the potentials are random numbers in [0, 1]. The resulting MRF has 10 million nodes and approximately



(a) A grid with 6 nodes and 7 edges. (b) Edge-centric partition with 4 processes. (c) Process 0 and 1 share node 0 and 1.

Figure 2: 2(a): We label the nodes row by row and represent the graph structure as a set of edges: (0, 1), (1, 2), (0, 3), (1, 4), (2, 5), (3, 4), (4, 5). 2(b): We use 4 MPI processes and apply edge-centric partition. 2(c): The node list of process 0 can be represented as: {{0, 3}} and the node list of process 1 can be represented as: {{0, 2}, {3, 2}}. The processes share node 1 and 0. The degree of node 1 is 3. The process 0 has partial degree of 2 (red nodes) and the process 1 has degree of 1 (green node). The degree of node 0 is 2 and both processes have local degree of 0.

20 million edges. We apply the edge-centric partitioning and run the Bethe-ADMM algorithm for 100 iterations.

Figure 3(a) shows the run time performance using 8 to 1024 MPI processes. The algorithm runs about half an hour on 8 process, and dramatically reduces to 16 seconds on 1024 processes. The input file size is close to 1GB and data loading only takes 1.2 seconds. We attribute the speedup to our adoption of PNetCDF as well as stripping the input file across 128 OSTs.

Figure 3(b) illustrates the average time it takes per process to compute $\boldsymbol{m}_\tau$, update $\boldsymbol{\mu}$ and communicate with neighbor processes respectively and the error bars show the minimum and maximum time spent on these three steps across all the processes. Since we evenly distribute the edges to the processes, the time spent on computing $\boldsymbol{m}_\tau$ has little fluctuation among the processes. The time to update $\boldsymbol{\mu}$, however, also depends on the number of neighbor processes and the number of shared nodes, hence the fluctuation between the min and max time among all the processes becomes more obvious as the number of processes increases. The plot shows the communication cost incurred by the edge-centric partition is negligible. The main reason that the communication cost is so small is because when we partition the grid, we sweep row edges and column edges from top to bottom, which essentially behaves as row partitioning where each process has at most 2 neighbors and only the boundary data are exchanged.

Figure 3(c) shows that the Bethe-ADMM algorithm implementation achieves almost linear speedup while the speedup of the entire implementation (I/O phase + Bethe-ADMM optimization) starts to deviate from the ideal case after 256 processes. This is because as the number of MPI processes increases, each process has less work load and optimization part becomes less dominating compared with the I/O part.

5.2 CRU Precipitation Dataset

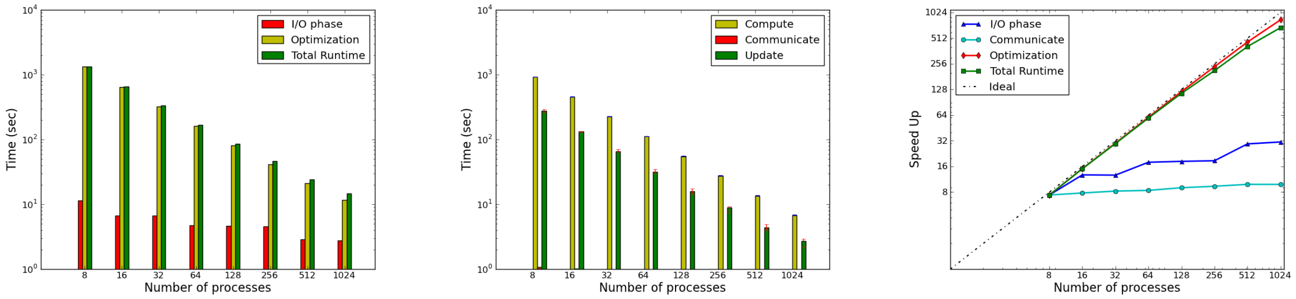
The dataset used in this section is the Climate Research Unit (CRU) precipitation dataset [15], which has monthly precipitation from the years 1901-2006. The dataset is of high gridded spatial resolution (360 × 720, i.e., 0.5 degree latitude × 0.5 degree longitude) and only includes the precipitation over land.

Our goal is to detect major droughts of the last century based on precipitation. We formulate the drought detection problem as the one of estimating the most likely configuration of a binary hidden MRF. In the underlying graph, each node represents a location and it can be in two possible states: dry and normal. We use a four nearest neighbor grid ($m = 360, n = 720$) to model the global dependency and replicate it 106 times. The resulting graph is similar to the ones used in the previous section and the structure respects the CRU dataset, i.e, it only has the nodes that correspond to the locations with precipitation record. Overall, the three dimensional grid has 7,146,520 nodes and 20,777,480 edges.

We design the potential functions carefully from the CRU datasets to enforce label consistency, i.e., neighboring nodes should take same values. We refer the readers to [7] regarding the details on designing potential functions. We obtain the integer solution after rounding the node pseudo-marginals and we can detect droughts based on it. Figure 5 shows the detected droughts in the 1960s.

We run the Bethe-ADMM algorithm on the CRU dataset for 500 iterations with edge-centric partitioning. The input PNetcdf file is around 530 MB. The runtime performance, as shown in Figure 4(a) exhibits the nice decreasing trend as it does on the simulation data. The algorithm takes less than 2 minutes to complete with 1024 MPI processes which would run more than two hours with 8 processes. The amount of time saved by our implementation is tremendous.

Figure 4(b) illustrates the average time per process to compute $\boldsymbol{m}_\tau$, communicate with neighbors and update $\boldsymbol{\mu}$ respectively. The error bars mark the minimum and maximum time spent on these three steps across all the processes. The communication cost on the CRU dataset is no longer negligible anymore. This is because the underlying 3 dimensional grid has missing nodes (CRU only has precipitation over land) and when we apply edge-centric partitioning, each process may have more than two neighbors. Hence as the number of processes increases, the number of neighbors for each process is more dynamic and the communication pattern becomes more complicated. Figure 4(c) plots the almost linear speedup on the CRU dataset. It also shows the trend that our implementation is scalable beyond 1024 processes. This is understandable because I/O time is only 2% of the total execution time, even at 1024 MPI processes.

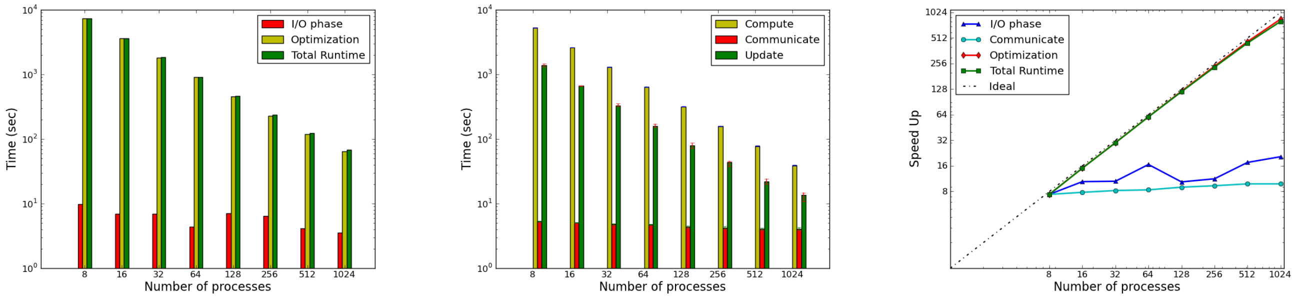


(a) Time spent on the I/O phase and the Bethe-ADMM optimization. The I/O cost is low.

(b) Time spent on the three steps of the Bethe-ADMM optimization. The communication overhead can be negligible.

(c) Almost linear speedup in the number of MPI processes

Figure 3: Results on the simulation dataset with 10 million nodes and 20 million edges using 8-1024 MPI processes. The I/O and communication cost is relatively low. Overall, the MPI implementation achieves almost linear speedup in the number of processes.



(a) Time spent on the I/O phase and Bethe-ADMM optimization. The I/O cost is low.

(b) Time spent on the three steps of the Bethe-ADMM optimization. The communication overhead is low.

(c) Almost linear speedup in the number of MPI processes

Figure 4: Results on the CRU dataset with 7,146,520 nodes and 20,777,480 edges using 8-1024 MPI processes. The I/O and communication cost is relatively low. Overall, the MPI implementation achieves almost linear speedup in the number of processes.

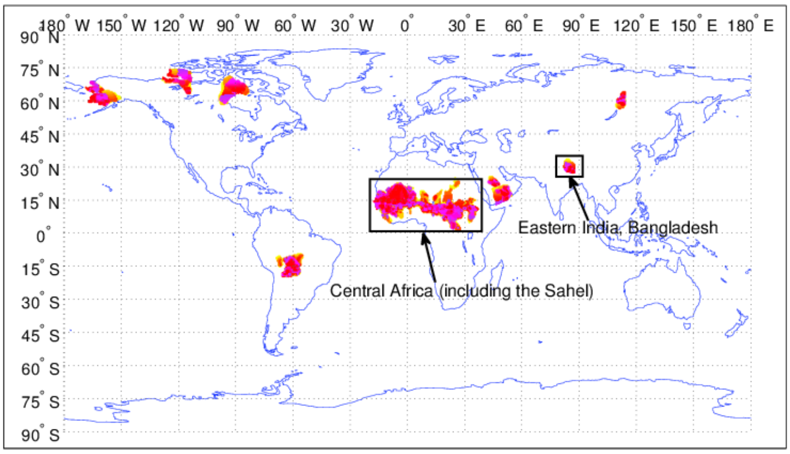

Eastern India, Bangladesh
Central Africa (including the Sahel)

Figure 5: Major droughts starting within the period 1961-1970, which include the three decade long Sahel drought and the drought in eastern India in the 1960s.

6. CONCLUSIONS

We adopt the recently proposed Bethe-ADMM algorithm for large scale MRFs. The algorithm is based on the 'tree decomposition' idea from the MAP inference literature and the alternating direction method from the optimization literature. The algorithm solves the tree structured subproblems efficiently via the sum product algorithm and is inherently parallel. We implement the algorithm using MPI and the experimental results show that our implementation scales almost linearly with the number of MPI processes for grid-structured graphs.

Acknowledgements

The research was supported in part by NSF grants IIS-1029711, IIS-0916750, IIS-0812183, NSF CAREER award IIS-0953274, CCF-0833131, CNS-0830927, IIS-0905205, CCF-0938000, CCF-1029166, OCI-1144061 and NASA grant NNX-12AQ39A and DOE awards DE-FG02-08ER25848, DE-SC0001283, DE-SC0005309, DESC0005340, DESC0007456 and AFOSR award FA9550-12-1-0458. The authors would like to thank Stefan Liess and Peter K. Snyder for the helpful discussion on the CRU experiment. The authors are grateful for the technical support from the University of Minnesota Supercomputing Institute (MSI). This research used Hopper Cray XE6 computer of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

7. REFERENCES

[1] http://cucis.ece.northwestern.edu/projects/PnetCDF.

[2] http://giraph.apache.org.

[3] http://www.nersc.gov/users/computational-systems/hopper.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[5] Y. Censor and S. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications.* Oxford University Press, 1998.

[6] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. A linear programming formulation and approximation algorithms for the metric labeling problem. *SIAM Journal on Discrete Mathematics*, 18(3):608–625, Mar. 2005.

[7] Q. Fu, A. Banerjee, S. Liess, and P. K. Snyder. Drought detection of the last century: An MRF-based approach. In *Proceedings of the SIAM International Conference on Data Mining*, 2012.

[8] Q. Fu, H. Wang, and A. Banerjee. Bethe-ADMM for tree based parallel MAP inference. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2013.

[9] S. P. Kasiviswanathan, P. Melville, A. Banerjee, and V. Sindhwani. Emerging topic detection using dictionary learning. In *Proceedings of the Twentieth ACM international conference on Information and knowledge management*, 2011.

[10] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, oct. 2006.

[11] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[12] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM International Conference on Management of Data*, 2010.

[13] A. F. Martins, P. M. Aguiar, M. A. Figueiredo, N. A. Smith, and E. P. Xing. An augmented Lagrangian approach to constrained MAP inference. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning*, 2011.

[14] O. Meshi and A. Globerson. An alternating direction method for dual MAP LP relaxation. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2011.

[15] T. D. Mitchell, T. R. Carter, P. D. Jones, M. Hulme, and M. New. *A comprehensive set of high-resolution grids of monthly climate for Europe and the globe: the observed record (1901-2000) and 16 scenarios (2001-2100).* Tyndall Centre for Climate Change Research, 2004.

[16] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[17] P. Ravikumar, A. Agarwal, and M. J. Wainwright. Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. *Journal of Machine Learning Research*, 11:1043–1080, 2010.

[18] D. Sontag, A. Globerson, and T. Jaakkola. Introduction to dual decomposition for inference. In S. Sra, S. Nowozin, and S. J. Wright, editors, *Optimization for Machine Learning.* MIT Press, 2011.

[19] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. MAP estimation via agreement on (hyper)trees: Message-passing and linear-programming approaches. *IEEE Transactions of Information Theory*, 51(11):3697–3717, 2005.

[20] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[21] J. Yang and Y. Zhang. Alternating direction algorithms for l1-problems in compressive sensing. *SIAM Journal on Scientific Computing*, 33(1):250–278, 2011.