Parallel Gibbs Sampling
From Colored Fields to Thin Junction Trees

Joseph Gonzalez
Yucheng Low
Arthur Gretton
Carlos Guestrin
Suppose we wanted to know the probability that coin lands “heads”

We use the same idea for graphical model inference

Inference:
\[ P(X_1 = X_2) \approx 2/4 \]
Focus on **discrete** factorized models with **sparse** structure:
The goal is to estimate:

$$\mathbb{E} \left[ h(X_1, \ldots, X_n) \right]$$

Example: marginal estimation

$$h_i(x) = I[x == i] \Rightarrow \mathbb{E}[h_i(X_k)] = P(X_k = i)$$

If the sampler is **ergodic** the following is true*:

$$\lim_{m \to \infty} \frac{1}{m} \sum_{t=1}^{m} h(x_1^{(t)}, \ldots, x_n^{(t)}) \xrightarrow{a.s.} \mathbb{E} \left[ h(X_1, \ldots, X_n) \right]$$

*Consult your statistician about potential risks before using.
Gibbs Sampling [Geman & Geman, 1984]

- **Sequentially** for each variable in the model
  - Select variable
  - Construct conditional given adjacent assignments
  - Flip coin and update assignment to variable

Initial Assignment

...
Why Study Parallel Gibbs Sampling?

“The Gibbs sampler ... might be considered the workhorse of the MCMC world.”

–Robert and Casella

- Ergodic with geometric convergence
- Great for high-dimensional models
  - No need to tune a joint proposal
- Easy to construct algorithmically
  - WinBUGS
- Important Properties that help Parallelization:
  - Sparse structure → factorized computation
Is the Gibbs Sampler trivially parallel?
“...the MRF can be divided into collections of [variables] with each collection assigned to an independently running asynchronous processor.”

-- Stuart and Donald Geman, 1984.
The problem with **Synchronous** Gibbs

Adjacent variables **cannot** be sampled simultaneously.
How has the machine learning community solved this problem?
Two Decades later


- Same problem as the original Geman paper
  - Parallel version of the sampler is **not ergodic**.

- Unlike Geman, the recent work:
  - Recognizes the issue
  - Ignores the issue
  - Propose an “approximate” solution
Two Decades Ago

Parallel computing community studied:

Sequential Algorithm

\[
\begin{align*}
x_1^{(t+1)} &= f_1(x_3^{(t)}) \\
x_2^{(t+1)} &= f_2(x_2^{(t)}, x_3^{(t)}) \\
x_3^{(t+1)} &= f_3(x_1^{(t+1)}, x_1^{(t+1)})
\end{align*}
\]

Directed Acyclic Dependency Graph

Construct an Equivalent Parallel Algorithm

Using Graph Coloring

\[
\begin{align*}
x_1^{(t+1)} &= f_1(x_3^{(t)}) \\
x_2^{(t+1)} &= f_2(x_2^{(t)}, x_3^{(t)}) \\
x_3^{(t+1)} &= f_3(x_1^{(t+1)}, x_1^{(t+1)})
\end{align*}
\]
Chromatic Sampler

- Compute a k-coloring of the graphical model
- Sample all variables with the same color in parallel
- Sequential Consistency:
Chromatic Sampler Algorithm

For $t$ from 1 to $T$ do

\[ x^{(t)} \leftarrow x^{(t-1)} \]

For $k$ from 1 to $K$ do

\textbf{Parfor} $i$ in color $k$:

\[ x^{(t)}_i \sim P(X_i \mid x^{(t)}_{-i}) \]
Asymptotic Properties

- **Quantifiable** acceleration in mixing

  Time to update all variables once

  \[ O \left( \frac{n}{p} + k \right) \]

  - # Variables
  - # Colors
  - # Processors

- **Speedup:**

  \[ O \left( p \left( \frac{n}{n + pk} \right) \right) \]

  - Penalty Term
Proof of Ergodicity

- **Version 1 (Sequential Consistency):**
  - Chromatic Gibbs Sampler is *equivalent* to a *Sequential Scan* Gibbs Sampler.

- **Version 2 (Probabilistic Interpretation):**
  - Variables in same color are *Conditionally Independent*.
  - *Joint Sample* is *equivalent* to Parallel Independent Samples.
Special Properties of 2-Colorable Models

- Many common models have two colorings

For the [Incorrect] Synchronous Gibbs Samplers
  - Provide a method to correct the chains
  - Derive the stationary distribution
Correcting the *Synchronous* Gibbs Sampler

We can derive two **valid** chains:

```
<table>
<thead>
<tr>
<th>t=0</th>
<th>t=1</th>
<th>t=2</th>
<th>t=3</th>
<th>t=4</th>
<th>t=5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
We can derive two valid chains:

Chain 1

Chain 2

Strong Positive Correlation

Invalid Sequence

Converges to the Correct Distribution
Theoretical Contributions on 2-colorable models

Stationary distribution of **Synchronous Gibbs**:

\[
\mathbf{P}(x') = \sum_x K(x' | x) \pi(x_{\kappa_1}) \pi(x_{\kappa_2})
\]

\[
= \sum_{x_{\kappa_1}} \sum_{x_{\kappa_2}} \pi(x'_{\kappa_1} | x_{\kappa_2}) \pi(x'_{\kappa_2} | x_{\kappa_1}) \pi(x_{\kappa_1}) \pi(x_{\kappa_2})
\]

\[
= \sum_{x_{\kappa_1}} \sum_{x_{\kappa_2}} \pi(x_{\kappa_1}, x'_{\kappa_2}) \pi(x'_{\kappa_1}, x_{\kappa_2})
\]

\[
= \pi(x'_{\kappa_1}) \pi(x'_{\kappa_2})
\]
Theoretical Contributions on 2-colorable models

- Stationary distribution of **Synchronous Gibbs**

\[ \mathbf{P}_{\text{sync}} (X_1, \ldots, X_n) = \pi (X_{\kappa_1}) \pi (X_{\kappa_2}) \]

- **Corollary**: Synchronous Gibbs sampler is correct for single variable marginals.

\[ \lim_{m \to \infty} \frac{1}{m} \sum_{t=1}^{m} h(x_{i}^{(t)}) \overset{a.s.}{\longrightarrow} \mathbb{E} [h(X_i)] \]
From Colored Fields to Thin Junction Trees

Chromatic Gibbs Sampler

- Ideal for:
  - Rapid mixing models
  - Conditional structure does not admit Splash

Splash Gibbs Sampler

- Ideal for:
  - Slowly mixing models
  - Conditional structure admits Splash
  - Discrete models
Models With Strong Dependencies

- **Single variable** Gibbs updates tend to mix **slowly**:

  ![Diagram](image)

  - Single site changes move slowly with strong correlation.

- Ideally we would like to draw joint samples.
  - Blocking
Blocking Gibbs Sampler

Based on the papers:

1. Jensen et al., *Blocking Gibbs Sampling for Linkage Analysis in Large Pedigrees with Many Loops*. TR 1996
Splash Gibbs Sampler

An asynchronous Gibbs Sampler that adaptively addresses strong dependencies.
Step 1: Grow multiple Splashes in parallel:

Conditionally Independent
**Step 1:** Grow multiple Splashes in parallel:

- Conditionally Independent
- Tree-width $= 1$
Step 1: Grow multiple Splashes in parallel:

Tree-width = 2

Conditionally Independent
Step 2: Calibrate the trees in parallel
**Step 3:** Sample trees in parallel
Recall:

**Tree-width = 2**

Junction Trees
Junction Trees

- Data structure used for exact inference in loopy graphical models

Tree-width = 2
Splash Thin Junction Tree

Parallel Splash Junction Tree Algorithm

- Construct multiple conditionally independent thin (bounded treewidth) junction trees **Splashes**
  - Sequential junction tree extension

- Calibrate the each thin junction tree in parallel
  - Parallel belief propagation

- Exact backward sampling
  - Parallel exact sampling
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

A B
Splash generation

Frontier extension algorithm:

- Markov Random Field
- Corresponding Junction tree

A B
C B C
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

A B D

B C D
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

A B D
B C D
A D E
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

A B D
B C D
A D E
A E F
Splash generation

- Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

- A B D
  - B C D
- A D E
  - A E F
  - A G
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

A B D
B C D
A D E
A E F
A G
B G H
Splash generation

- Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

- A B D
- B C D
- A B D E
- A B E F
- A B G
- B G H
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree
Splash generation

Frontier extension algorithm:

Markov Random Field

Corresponding Junction tree

A B D  B C D

A D E  A E F

D I  A G
Splash generation

Challenge:
- Efficiently reject vertices that violate treewidth constraint
- Efficiently extend the junction tree
- Choosing the next vertex

Solution Splash Junction Trees:
- Variable elimination with reverse visit ordering
  - I,G,F,E,D,C,B,A
- Add new clique and update RIP
  - If a clique is created which exceeds treewidth terminate extension
- Adaptive prioritize boundary
Incremental Junction Trees

First 3 Rounds:

Junction Tree: 4
Elim. Order: {4}
Incremental Junction Trees

- Result of third round:
  \{2,5,4\}

- Fourth round:
  \{1,2,5,4\}
Incremental Junction Trees

- Results from 4\textsuperscript{th} round:
  \{1,2,5,4\}

- 5\textsuperscript{th} Round:
  \{6,1,2,5,4\}
Incremental Junction Trees

- Results from $5^{th}$ round:
  \{6,1,2,5,4\}

- $6^{th}$ Round:
  \{3,6,1,2,5,4\}
Incremental Junction Trees

- Finishing 6\textsuperscript{th} round:

\{3,6,1,2,5,4\}
Algorithm Block [Skip]

**Input:** The original junction tree \((C, E) = J_S\).

**Input:** The variable \(X_i\) to add to \(J_S\).

**Output:** \(J_{S+i}\)

**Define:** \(C_u\) as the clique created by eliminating \(u \in S\).

**Define:** \(V[C] \in S\) as the variable eliminated when creating \(C\).

**Define:** \(t[v]\) as the time \(v \in S\) was added to \(S\).

**Define:** \(P[v] \in N_v \cap S\) as the next neighbor of \(v\) to be eliminated.

1. \(C_i \leftarrow (N_i \cap S) \cup \{i\}\)
2. \(P[i] \leftarrow \arg \max_{v \in C_i \setminus \{i\}} t[v]\)
   
   // ---------------- Repair RIP ----------------
3. \(R \leftarrow C_i \setminus \{i\}\) // RIP Set
4. \(v \leftarrow P[i]\)
5. while \(|R| > 0\) do
6.   \(C_v \leftarrow C_v \cup R\) // Add variables to parent
7.   \(w \leftarrow \arg \max_{w \in C_v \setminus \{v\}} t[w]\) // Find new parent
8.   if \(w = P[v]\) then
9.     \(R \leftarrow (R \setminus C_i) \setminus \{i\}\)
10. else
11.   \(R \leftarrow (R \cup C_i) \setminus \{i\}\)
12.   \(P[v] \leftarrow w\) // New parent
13. \(v \leftarrow P[v]\) // Move upwards
Challenge:
- Efficiently reject vertices that violate treewidth constraint
- Efficiently extend the junction tree
- Choosing the next vertex

Solution: Splash Junction Trees:
- Variable elimination with reverse visit ordering
  - I,G,F,E,D,C,B,A
- Add new clique and update RIP
  - If a clique is created which exceeds treewidth terminate extension

Adaptive prioritize boundary
Adaptive Vertex Priorities

- Assign priorities to boundary vertices:

\[ s[X_v] = \left\| \log \frac{\sum_x \pi(X_S, X_v = x \mid X_-S = x^{(t)}_{-S})}{\pi(X_S \mid X_v = x^{(t)}_v, X_-S = x^{(t)}_{-S})} \right\|_1 \]

- Can be computed using only factors that depend on \( X_v \)
- Based on current sample
- Captures difference between marginalizing out the variable (in Splash) fixing its assignment (out of Splash)
- Exponential in treewidth

- Could consider other metrics ...
Adaptively Prioritized Splashes

- Provably converges to the correct distribution
  - Requires vanishing adaptation
  - Identify a bug in the Levine & Casella seminal work in adaptive random scan
Experiments

- Implemented using GraphLab
  - Treewidth = 1:
    - Parallel tree construction, calibration, and sampling
    - No incremental junction trees needed
  - Treewidth > 1:
    - Sequential tree construction (use multiple Splashes)
    - Parallel calibration and sampling
    - Requires incremental junction trees
  - Relies heavily on:
    - Edge consistency model to prove ergodicity
    - FIFO/ Prioritized scheduling to construct Splashes
- Evaluated on 32 core Nehalem Server
Rapidly Mixing Model

- Grid MRF with weak attractive potentials
  - 40K Variables
  - 80K Factors

- The Chromatic sampler slightly outperforms the Splash Sampler
Slowly Mixing Model

- Markov logic network with strong dependencies
  - 10K Variables
  - 28K Factors

- The Splash sampler outperforms the Chromatic sampler on models with **strong** dependencies
Conclusion

- **Chromatic Gibbs** sampler for models with *weak* dependencies
  - Converges to the correct distribution
  - Quantifiable improvement in mixing

- **Theoretical analysis** of the Synchronous Gibbs sampler on *2-colorable models*
  - Proved marginal convergence on 2-colorable models

- **Splash Gibbs** sampler for models with *strong* dependencies
  - Adaptive asynchronous tree construction
  - Experimental evaluation demonstrates an improvement in mixing
Future Work

- Extend Splash algorithm to models with continuous variables
  - Requires continuous junction trees (Kernel BP)
- Consider “freezing” the junction tree set
  - Reduce the cost of tree generation?
- Develop better adaptation heuristics
  - Eliminate the need for vanishing adaptation?
- Challenges of Gibbs sampling in high-coloring models
  - Collapsed LDA
- High dimensional pseudorandom numbers
  - Not currently addressed in the MCMC literature