Variant Generation for Augmented Gibbs Samplers

Sachith Seneviratne
sachith.Seneviratne@unimelb.edu.au
University of Melbourne, Australia

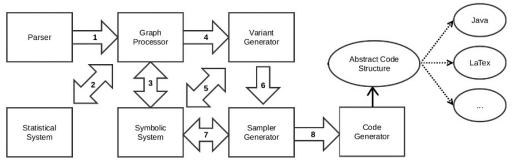
Wray Buntine
wray.buntine@monash.edu.au
Monash University, Australia

Introduction

Our research explores the possibility of developing automated collapsing and augmentation with Gibbs sampling, that could be used within a larger PPL. We introduce "variants" because different samplers can be generated once you add a variety of choices in transforming your model. Due to uncertainty in performance, these variants need to be tested to evaluate their comparable worth.

- **Collapsing** (or marginalization) is the process of integrating over some parameters of the model and may be associated with a reduction of the data to sufficient statistics of the marginal model.
- **Augmentation** is the inverse operation of collapsing. It involves adding variables or parameters to the model to make it more tractable.
- A variant can be characterised by a graph G and a likelihood L. Each variant is derived from the original model through the application of one or more statistical operations, such as collapsing and augmentation.

Methodology



Overall System Architecture

Input – Model Specification

```
model {
  for (k in 1:K) {
    phi[k,1:V] ~ ddirich(b);
  }
  for (m in 1:M) {
    theta[m,1:K] ~ ddirich(a);
    for (n in 1:N) {
      z[m,n] ~ dcat(theta[m]);
      w[m,n] ~ dcat(phi[z[m,n]]);
    }
  }
}
```

Output – Generated Code

```
for (int d = 0; d < D; d++) {
    int I = w[d].length;
    for (int i = 0; i < I; i++) {
        c0_1[d]--;
        c0[d][z[d][i]]--;
        c1_1[z[d][i]]--;
        c1[z[d][i]][w[d][i]]--;
        double[] p = new double[K];
        for (int k = 0; k < K; k++) {
            p[k] = (Math.pow(c0_1[d] + a_1[d], -1))
            * (Math.pow(c1_1[k] + b_1[k], -1))
            * (c0[d][k] + a[d][k])
            * (c1[k][w[d][i]] + b[k][w[d][i]]);
        }
        for (int k = 1; k < K; k++) {
            p[k] += p[k - 1];
        }
        int k;
        double val = Math.random() * p[K - 1];
        for (k = 0; k < K; k++) {
            if (p[k] > val) {
                break;
            }
        }
        z[d][i] = k;
        c0_1[d]++;
        c0[d][z[d][i]]++;
        c1_1[z[d][i]]++;
        c1[z[d][i]][w[d][i]]++;
    }
}
```

Experiments

1) Verify generated samplers are equivalent to existing model implementations by empirical analysis of results
2) Generate Parallelized code for the samplers using loop parallelism and Hogwild sampling and evaluate performance.

Result Similarity

By running the Hungarian algorithm on a distance metric between generated matrices, it is possible to compare the similarity of the generated results.

Parallelization

Analysis is performed on the generated abstract code structure to evaluate possible parallelization options.

Results

Model	Dataset	Full dataset			Half dataset		
		S	P	speedup	S	P	speedup
LDA (Theta collapsed)	Obesity	465	116	4.00	330	108	3.05
LDA (Theta collapsed)	WS	486	137	3.55	294	100	2.94
LDA (Theta collapsed)	Reuters	3138	667	4.70	1854	461	4.02
MetaLDA HogWild	Obesity	1699	298	5.70	1380	239	5.77
MetaLDA HogWild	WS	1560	273	5.71	1315	199	6.61
MetaLDA HogWild	Reuters	21363	3627	5.88	13586	2298	5.91
MIGA HogWild	Obesity	4912	739	6.63	2600	376	6.91
MIGA HogWild	WS	1803	372	4.84	670	117	5.73
MIGA HogWild	Reuters	20653	4318	4.78	10574	2210	4.78

Table: Generated results for Parallelization experiment

We present a demonstration system that allows collapsing and augmentation to be applied to exponential family probability models in order to generate different variants of the model, and also supports two simple methods of parallelism.

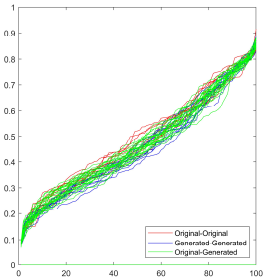


Figure: Result similarity experiment over 5 repeats