Supplementary Material: Block-Value Symmetries in Probabilistic Graphical Models

Gagan Madan, Ankit Anand, Mausam and Parag Singla
Indian Institute of Technology Delhi
gagan.madan1@gmail.com, {ankit.anand, mausam, parags}@cse.iitd.ac.in

Algorithmic and Implementation Details for Finding Block Partitions

This section provides algorithmic and implementation details for the heuristic used to find the candidate set of block partitions (Section 5). There are three broad steps for obtaining a good candidate set and each one of them is described below in turn.

Algorithm 1: Procedure Get_Useful_Blocks takes a parameter r and computes potentially useful blocks with maximum block-size r. It iterates over each of the features in turn and selects all possible subsets of size $\leq r$ of variables which are part of that feature (lines 2-7). This automatically eliminates all r (or less) sized blocks which are composed of variables that never appear together in any feature in the graphical model.

Algorithm 1 Get_Useful_Blocks($\mathcal{G}, r$)

```
1: useful_blocks ← {}
2: for f ∈ features(G) do
3:     for all b ⊂ Var(f) and Size(b)≤ r do
4:         useful_blocks ← useful_blocks ∪ b
5:     end for
6: end for
7: return useful_blocks
```

Algorithm 2: For the useful blocks obtained above, our heuristic constructs a weight signature for each of the block-value pairs. Procedure Get_Weight_Sign computes a weight signature for all the features consistent with the input BV pair(B, b). We define the Feature Blanket of a variable $X_j \in B$ as the set of features in which X_j appears. In line 1, we construct feature blanket of a block B by taking union of the feature blankets of all the variables appearing in the block. Line 2 initializes the signature as an empty multi-set. We construct weight signature by iterating over features present in feature blanket of this block. For each feature f_j, we check whether the given BV pair (B, b) is consistent with f_j, i.e., whether the feature is satisfied by the block-value pair. The weight of f_j is inserted in the signature if the consistency requirement is met (line 5). The complete weight-signature so obtained after iterating over all the features in the blanket is returned as the weight-signature for the BV pair (B, b).

Algorithm 2 Get_Weight_Sign($\mathcal{G}, B, b$)

```
1: feature_blanket(B) ← ⋃_{X_j ∈ B} FeatureBlanket(X_j)
2: signature ← {}
3: for f ∈ feature_blanket do
4:     if (B, b) is consistent with f then
5:         Insert weight(f) in signature
6:     end if
7: end for
8: return signature
```

Algorithm 3: This makes use of the two procedures described above and outlines the complete process for generating multiple block partitions. It takes as input a Graphical Model $\mathcal{G}$ and maximum block-size r. After obtaining useful blocks, a weight signature dictionary is constructed with key as weight-signature and value as a list of blocks. For each block $B_i \in useful\_blocks$, we iterate over all value assignments of that block ($\mathcal{V}(B_i)$) to form all possible BV pairs (lines 3,4). For each BV pair, Procedure Get_Weight_Sign computes the weight-signature S_j for that BV pair (line 5). If S_j has already been seen in dictionary, the current block is appended to the list of blocks corresponding to the signature (lines 6,7). Else, a new weight-signature along with the list of singleton block is added to the dictionary (lines 8-10).

Once the weight-signature dictionary is built, we generate useful candidate lists by picking blocks using the weight signature dictionary (loop at line 14). Line 15 initializes an empty candidate list. Blocks are added to the

candidate list in iterative fashion until all the variables are included (line 16). A two step sampling procedure is used. The first step samples a weight-signature with a probability proportional to the size of its corresponding list of blocks (line 17). The second step samples a block uniformly from the list of blocks sampled in the first step. The sampled block is added to the current candidate list if it does not overlap with pre-existing blocks (lines 19-21) otherwise a new block is sampled as above. Once all variables are added, the candidate list is complete and the process is run again till $max\_candidate\_list$ number of lists are generated.

Algorithm 3 Generate_Block_Partitions($\mathcal{G}, r$)

1: useful_blocks $\leftarrow$ Get_Useful_Blocks($\mathcal{G}, r$)
2: $Weight\_Sign\_Dict \leftarrow \{\}$
3: **for all** $B_i \in useful\_blocks$ **do**
4: **for all** $b_i \in \mathcal{V}(B_i)$ **do**
5: $S_i \leftarrow$ Get_Weight_Sign($\mathcal{G}, B_i, b_i$)
6: **if** $S_i \in Weight\_Sign\_Dict$ **then**
7: Append B_i to $Weight\_Sign\_Dict[S_i]$
8: **else**
9: Insert $[S_i, [B_i]]$ to $Weight\_Sign\_Dict$
10: **end if**
11: **end for**
12: **end for**
13: $Candidate\_List \leftarrow$ [ ]
14: **for** $i \leftarrow 1 \rightarrow$ max_candidate_lists **do**
15: $CL \leftarrow \{\ \}$
16: **while** All variables not included in CL **do**
17: Sample S_j with probability $\propto |Weight\_Sign\_Dict[S_j]|$
18: Sample a block b uniformly from $Weight\_Sign\_Dict[S_j]$
19: **if** Variables(b) $\cap\ CL = \phi$ **then**
20: $CL \leftarrow CL \cup b$
21: **end if**
22: **end while**
23: $Candidate\_List \leftarrow Candidate\_List \cup CL$
24: **end for**
25: **return** $Candidate\_List$