A Constraint for Conditional Independence

Paulius Dilkas

5th February 2020

Goal: conditional independence of two predicates conditioned on a conjunction/disjunction of any number of predicates or a negation of a predicate.

Central question: does this clause contain this predicate (filtering out the possibility that the predicate is mentioned in the conditioning part)? We can answer this question by iterating over each position and answering two other questions:

- Does this position contain this predicate? Yes/No/Maybe.
- Is this position part of the structure we are conditioning on? Yes/No/Maybe.

For each clause, we have two position x predicate matrices. One of them says (in a yes/no/maybe fashion) whether this predicate can be in this position (that's A). The other says whether this predicate is masked in this position (yes/no/maybe).

The propagation algorithm for independence is extended with masks that are either potential or definite. Masking happens in two stages: first, we mask expressions within formulas, and then predicates. Masking algorithm uses an algorithm for perfect bipartite matching.

For any integer n, let $[n] = \{0, 1, \ldots, n-1\}$.

Definition 1 (Conditional independence)**.** When assigning edges to the graph, mask off all possible instances of the expression we're conditioning on. For example, if we're conditioning on $a \wedge b$, then $c \leftarrow a \wedge a \wedge b$ should produce no edges.

Definition 2. A predicate is *definitely masked* if all instances are definitely masked. A predicate is *potentially masked* if all instances are definitely/potentially masked.

Definition 3 (Primitives)**.** For each clause, let $\mathbf{P}$ be a $\mathcal{M}_\mathcal{N} \times |\mathcal{P} \cup \{\wedge, \vee, \neg\}|$ matrix defined as

$$P_{p,i} = \begin{cases} 1 & \text{if } \texttt{values}[i] = \{p\} \\ 0 & \text{otherwise.} \end{cases}$$

This matrix shows what symbol can (or must) be in what position.

Likewise, let $\mathbf{C}$ be a $\mathcal{M}_\mathcal{N} \times \mathcal{M}_\mathcal{N}$ matrix defined as

$$C_{i,j} = \begin{cases} 1 & \text{if } \texttt{structure}[i] = \{j\} \\ 0 & \text{otherwise.} \end{cases}$$

The (i, j)-th element shows whether node i can be a child of node j.

The adjacency matrix $\mathbf{A} \in \{0, 1/2, 1\}^{|\mathcal{P}| \times |\mathcal{P}|}$ is defined as:

$$A[h \leftarrow p] = \begin{cases} 1 & \text{if } \exists c \in [\mathcal{M}_\mathcal{C}] : \texttt{head}[c] = \{h\} \text{ and } \exists i \in [\mathcal{M}_\mathcal{N}] : A_{c;p,i} = 1 \\ 0 & \text{otherwise} \end{cases}$$

Table 1: Combining $\mathbf{M}$ and $\mathbf{P}$. The # stands for 'impossible'.

	$P=1$	$P=0$
$M=1$	0	#
$M=0$	1	0

Algorithm 1: Constructing the adjacency matrix $\mathbf{A}$

$\mathbf{A} \leftarrow \mathbf{0}_{|\mathcal{P}|,|\mathcal{P}|}$;
foreach *clause* c **do**
  Construct $\mathbf{P}$ and $\mathbf{C}$;
  $\mathbf{M} \leftarrow$ `constructMask(`$\odot$`,` $\mathcal{C}$`)`;
  Let $\mathbf{A}' \in \{0, 1/2, 1\}^{|\mathcal{P}| \times \mathcal{M}_\mathcal{N}}$ be such that $A'_{i,j} = \begin{cases} 0 & \text{if } P_{i,j} = M_{i,j} \neq 1/2 \\ 1 & \text{if } P_{i,j} = 1 \text{ and } M_{i,j} = 0; \\ 1/2 & \text{otherwise} \end{cases}$
  for $h \in$ `head`$[c]$ **do**
    for $p \in \mathcal{P}$ **do**
      for $i \in [\mathcal{M}_\mathcal{N}]$ **do**
        if $|$`head`$[c]| = 1$ **and** $A'_{p,i} = 1$ **then**
          $A_{h \leftarrow p}$`.upgradeTo(1)`;
        else if $|$`head`$[c]| = 1$ **and** $A'_{p,i} = 1/2$ **then**
          $A_{h \leftarrow p}$`.upgradeTo(`$1/2$`)`;

How does this propagate back to the variables? Each 'maybe' edge should hold a set of (variable, value) pairs that can be eliminated if the propagation algorithm decides that it cannot exist. Problem: propagating through $\mathbf{A}'$ would mean propagating through $\mathbf{M}$ and $\mathbf{P}$. If they're both uncertain, then we can't do anything. If $\mathbf{P}$ is uncertain, we can remove the predicate, but if $\mathbf{M}$ is uncertain, there might be multiple ways to complete the mask.

Essentially, we have three layers, i.e., we build two layers of abstraction on top of the CP model. We make a decision on the top layer, and propagate the changes to the bottom layers. A weakness is that some high-level problems can have multiple solutions in the bottom layers, in which case we can't commit to any particular solution and so can't do anything.

Algorithm 2: Construct the mask matrix $\mathbf{M}$

Data: the expression we are conditioning on: $\bigodot_{\mathsf{p} \in \mathcal{C}} \mathsf{p}$

Function `constructMask(`$\odot$`,` $\mathcal{C}$`):`

- $\mathbf{M} = \mathbf{0}_{|\mathcal{P}|, \mathcal{M}_\mathcal{N}}$;
- **for** $r \in [\mathcal{M}_\mathcal{N}]$ *such that* $P_{r,\odot} \neq 0$ **do** `// for each possible root`
  - $E \leftarrow \{\{p, i\} \mid p \in \mathcal{P}, i \in [\mathcal{M}_\mathcal{N}], C_{i,r} \neq 0, P_{i,p} \neq 0\}$;
  - $\mathbf{M}' \leftarrow$ `markIndices(`$(\mathcal{P}, [\mathcal{M}_\mathcal{N}], E)$`,` $1/2$`);` `// potential positions`
  - $\mathbf{M} \leftarrow \max\{\mathbf{M}, \mathbf{M}'\}$;
  - **if** $P_{r,\odot} = 1$ **and** $\mathbf{M}' \neq \mathbf{0}$ **then**
    - $E \leftarrow \{\{p, i\} \mid p \in \mathcal{P}, i \in [\mathcal{M}_\mathcal{N}], C_{i,r} = 1, P_{i,p} = 1, M'_{p,i} \neq 0\}$;
    - $\mathbf{M} \leftarrow \max\{\mathbf{M},$ `markIndices(`$(\mathcal{P}, [\mathcal{M}_\mathcal{N}], E)$`,` 1`)};` `// determined positions`
- **return** $\mathbf{M}$;

Algorithm 3: Predicate-perfect bipartite matching

Data: a bipartite graph $G = (\mathcal{P}, \mathcal{I}, E)$ (predicates and indices) and the value v to use for the matrix (1 or $1/2$)

Result: a $|\mathcal{P}| \times |\mathcal{I}|$ mask matrix $\mathbf{M}$

Function `markIndices(`G`,` v`):`

- $\mathbf{M} \leftarrow \mathbf{0}_{|\mathcal{P}|, \mathcal{M}_\mathcal{N}}$;
- **if** $|\mathcal{I}| < |\mathcal{P}|$ **then return** $\mathbf{M}$;
- **while** *true* **do**
  - Let $(p, i) \in E$ be such that $M_{p,i} = 0$;
  - **if** *there is none* **then** break;
  - Let G' be G without vertices p, i and all the incident edges;
  - Use alternating paths to find a $\mathcal{P}$-perfect matching P in G';
  - **if** *found* **then**
    - **foreach** *edge* $(p', i') \in P$ **do** $M_{p',i'} = v$;
  - **else**
    - remove e from G;
    - **if** *this leaves an isolated vertex* **then** break;
- **return** $\mathbf{M}$;