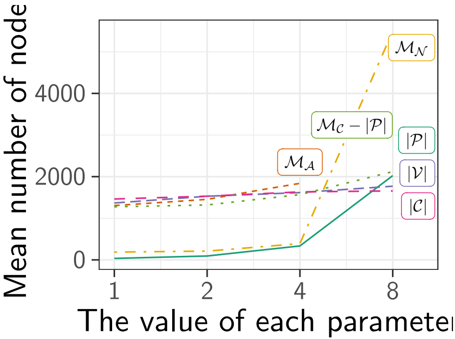
Generating Random Logic Programs Using Constraint Programming

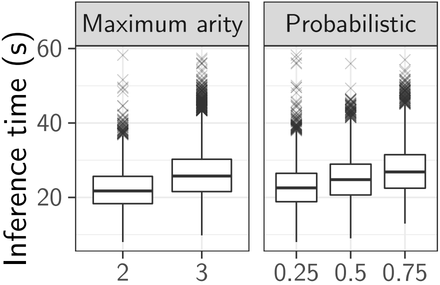
Paulius Dilkas[1] Vaishak Belle[1,2]

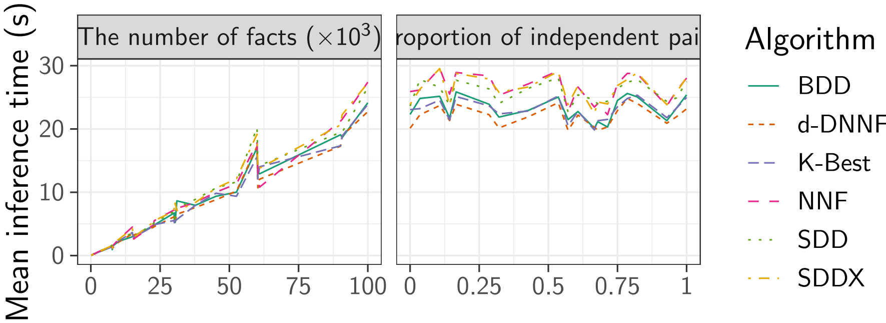
[1]University of Edinburgh, Edinburgh, UK

[2]Alan Turing Institute, London, UK

CP 2020


Mean number of node
0
2000
4000
1
2
4
8
The value of each parameter
$\mathcal{M}_{\mathcal{N}}$
$\mathcal{M}_{\mathcal{C}} - |\mathcal{P}|$
$|\mathcal{P}|$
$\mathcal{M}_{\mathcal{A}}$
$|\mathcal{V}|$
$|\mathcal{C}|$


Maximum arity
Probabilistic
Inference time (s)
60
40
20
2
3
0.25
0.5
0.75


Mean inference time (s)
The number of facts (×10^3)
roportion of independent pai
0
10
20
30
0
25
50
75
100
0
0.25
0.5
0.75
1
Algorithm
BDD
d-DNNF
K-Best
NNF
SDD
SDDX

Overview

General parameters

- maximum number of solutions
- `maxNumNodes` (in the tree representation of a clause)
- list of predicates with their variables
- maximum number of clauses
- option to forbid all cycles or just negative cycles
- list of probabilities that are randomly assigned to clauses: $\{0.1, 0.2, \dots, 0.9, 1, 1, 1, 1, 1, 1\}$

Decision variables

- `IntVar[] clauseAssignments`: a predicate or disabled
- `Clause[] clauses`

Constraints

Each predicate should get at least one constraint

- $\texttt{numDisabledClauses}$: defined by a count constraint
- $\texttt{numDistinctValues} = \begin{cases} \texttt{numPredicates} + 1 & \text{if } \texttt{numDisabledValues} > 0 \\ \texttt{numPredicates} & \text{otherwise.} \end{cases}$
  - also constrained using the $\texttt{nValues}$ constraint

Miscellaneous

- $\texttt{clauseAssignments}$ are sorted.
- If $\texttt{clauseAssignments}[i-1] = \texttt{clauseAssignments}[i]$,
  - then $\texttt{clause}[i-1] \preceq \texttt{clause}[i]$.

Clauses

A clause is defined by...

- `IntVar[] treeStructure`
  - $\texttt{treeStructure}[i] = i$: the i-th node is a root.
  - $\texttt{treeStructure}[i] = j$: the i-th node's parent is node j.
- `IntVar[] treeValues`: $\neg$, $\wedge$, $\vee$, $\top$, and any predefined predicates with variables.

Auxiliary variables

- $\texttt{numNodes}, \texttt{numTrees} \in \{1, \ldots, \texttt{maxNumNodes}\}$

Clause constraints

- $\texttt{treeStructure}$ represents $\texttt{numTrees}$ trees.
- $\texttt{treeStructure}[0] = 0$
- $\texttt{numTrees} + \texttt{numNodes} = \texttt{maxNumNodes} + 1$
- $\texttt{treeStructure}$ is sorted
- For $i = 0, \dots, \texttt{maxNumNodes} - 1$,
  - If $\texttt{numNodes} \leq i$,
  - then $\texttt{treeStructure}[i] = i$ and $\texttt{treeValues}[i] = \top$,
  - else $\texttt{treeStructure}[i] < \texttt{numNodes}$.
  - has 0 children $\iff$ $\texttt{treeValues}[i]$ is a predicate
  - has 1 child $\iff$ $\texttt{treeValues}[i] = \neg$
  - has > 1 child $\iff$ $\texttt{treeValues}[i] \in \{\wedge, \vee\}$
  - $\texttt{treeStructure}[i] \neq i \implies \texttt{treeValues}[i] \neq \top$
- If the clause should be disabled, $\texttt{numNodes} = 1$ and $\texttt{treeValues}[0] = \top$.

Adjacency matrix representation

$A[i][j] = 0 \iff \nexists k : \texttt{clauseAssignments}[k] = j$ and $i \in \texttt{clauses}[k]\texttt{.treeValues}$

New constraints

- No (negative) cycles
  - No clever propagation, just entailment checking.
- Independence. Propagation:
  - Two types of dependencies: determined and one-undetermined-edge-away-from-being-determined.
  - Look up the dependencies of both predicates. For each pair of matching dependencies:
    - If both are determined, fail.
    - If one is determined, the selected edge of the other must not exist.
- Conditional independence
  - Same propagation, but with a ‘filter’ that masks out the expression that the independence is conditioned on.