Evaluating Logic Programming in PROLOG for solving Hitori puzzles

A comparison of competing modeling-and-solving paradigms

Author: Tom D. Friederich, Supervisor: Dr. A.L.D. Latour, EEMCS, Delft University of Technology, The Netherlands

1. Hitori puzzles

Hitori is a NP-complete [1] puzzle consisting of a N x N grid of numbers. The goal of the puzzle is to mark tiles as black until the puzzle satisfies all of these constraints:

- Uniqueness: each unmarked number must be unique in its row and column.
- Non-adjacency: marked tiles may never be orthogonally adjacent.
- Connectivity: the set of tiles that are not marked must always remain orthogonally connected.

2	4	4	3
1	4	1	2
2	2	1	4
4	3	2	2

2	■	4	3
1	4	■	2
■	2	1	4
4	3	2	■

A small 4x4 Hitori puzzle (left), with its solution (right)

2. Research Objectives

Communities for modeling for competing modeling-and-solving paradigms rarely interact [2]. We evaluate PROLOG and compare it to other paradigms.

Paradigm evaluation criteria

1. Hitori solving performance: what size puzzles can solved in a reasonable timeframe?
2. How naturally and easily can the problem be expressed in this paradigm?
3. Ecosystem and documentation quality.

References

[1] Robert A. Hearn and Erik D. Demaine. Games, puzzles and computation. A K Peters, 2009

[2] Nina Narodytska Peter J. Stuckey Katalin Fazekas, Matti J¨arvisalo. Interactions in constraint optimization, Dagstuhl seminar motivation, 2025.

More information and code:

github.com/tom0334/PrologHitori

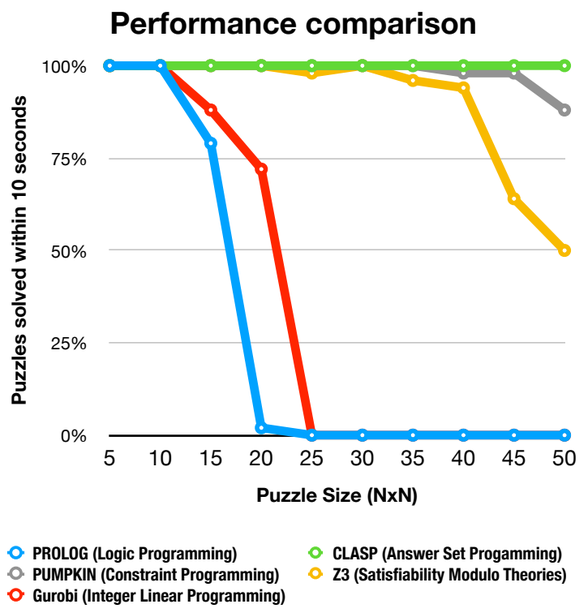
github.com/sappho3/Thesis-Hitori-shared

Our PROLOG model performed worse at solving Hitori puzzles

than Z3, Gurobi, Pumpkin, and Clasp.

It is limited by PROLOG's depth first backtracking search that lacks advancements from SAT solvers such as backjumping and conflict-driven clause learning.

Incorporating these techniques would be difficult, because existing PROLOG code relies on the depth first search order, but may be possible for predicates marked 'explorable in non DFS order' by the programmer.



4. Results

- Our PROLOG model can solve puzzles up to 15x15 quickly. Others can solve 50x50 easily. CLASP was the fastest.
- Our model is reasonably elegant, but its more complicated, less declarative and longer than the CLASP model.
- PROLOG is wel established and has a wealth of knowledge about it available, but development and support seems to be moving on to other languages.
- PROLOG can still be useful for less computationally difficult problems: its querying style is unique.

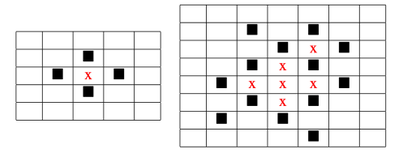
3. Solving approach

Our model uses an iterative solving approach to build up a solution step by step, where rows and columns are solved top to bottom and left to right, letting PROLOG backtrack when needed.

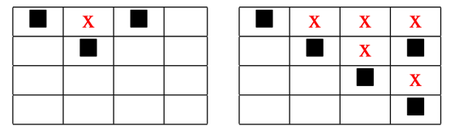
Cutoff detection for connectivity constraint

We check if **marked** tiles cut off any *unmarked* tiles from the rest. This happens when:

1. A path of **marked** tiles forms a cycle around an *unmarked* tile:



2. A Path of **marked** tiles cuts the board in two:



We check for these using an edge based depth first search from newly **marked** tiles in our iterative search process.

5. Further research

Would it be possible to mark predicates to allow the search tree for them to be explored in a non DFS way?

Extensions of PROLOG that add concepts from Constraint Programming exists. How does this compare to regular logic programming, and to other paradigms?

What redundant constraints can be added to improve performance?