Eating Soup with a Fork: Solving Hitori using Integer Linear Programming

Author
Sophieke van Luenen |
s.vanluenen@student.tudelft.nl |

Supervisor
Dr. Anna L. D. Latour |

1. Introduction

In modelling-and-solving (M&S) problems, how you model a problem influences solving efficiency. We tested the **Integer Linear Programming** (ILP) solver Gurobi on the logic puzzle **Hitori**, and found that ILP is an unintuitive and slow tool for solving Hitori instances, especially when compared to other M&S paradigms.

Hitori is an NP-complete puzzle where the player is given an `n x n` grid with numbers from `1 - n`, and they have to mark tiles such that:

- **Uniqueness**: no number appears twice in any row or column,
- **Adjacency:** no marked tiles are orthogonally adjacent,
- **Connectivity:** all unmarked tiles are orthogonally connected

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

A: Unsolved Hitori instance

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

B: Solved Hitori instance

2. Integer Linear Programming

In ILP one models a problem through **integer variables** with domains

$x_1 \in [1..6]$ $x_2 \in [-1..4]$

One then defines **constraints** in the form of **linear (in)equalities**

$2x_1 \leq 5$ $x_2 \geq 2$

The computer then finds an allocation of values to the variables that satisfies the constraints, and **optimises** the **objective function**

$\text{maximise}(x_1 - x_2)$

3. Research Questions

1. **How fast can we solve Hitori puzzles using ILP?**
2. **What properties of Hitori instances influence solving speed?**

4. Modelling the connectivity constraint

The path model

Define new path variables. Create a one-directional flow through unmarked tiles from a source tile

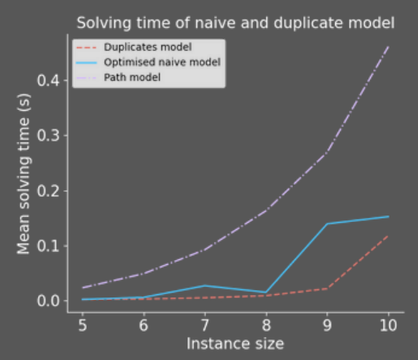
The naive model

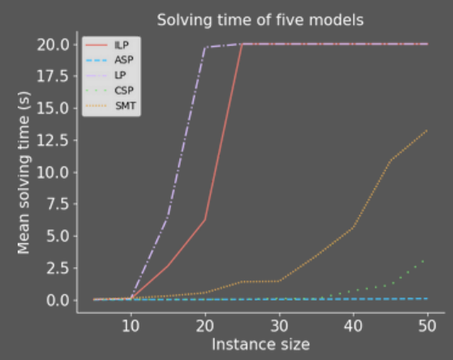
No encoding. Check constraint after optimisation, if it is not met, forbid this solution and run again

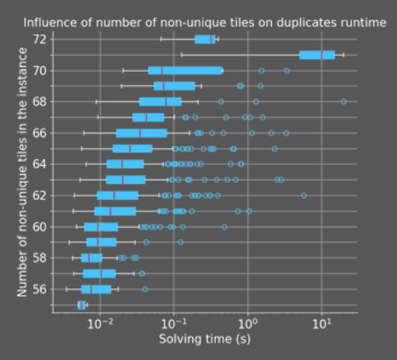
The duplicates model

Find all possible cycles of non-unique tiles and the edge of the board, and forbid them.

5. Results & Discussion







There is **no significant difference** between the naive and duplicates model. Our path model is significantly slower.

Our naive model's runtime spikes on **odd-sized instances**. We checked with a different ILP solver and did not find the same result, insinuating that this is behaviour specific to Gurobi

The naive ILP model on average solves an instance is 1.7 seconds faster than the LP (Prolog) model ($p < 0.001$)

It is, however, slower on average than the other tested models:
SMT (Z3) -9.2 seconds ($p < 0.001$)
CSP (Pumpkin) -12.4 seconds ($p < 0.001$)
ASP (Clasp) -12.9 seconds ($p < 0.001$)

The strongest determinant of the performance for our models was the **number of non-unique tiles** on the instance board.

The duplicates model was most strongly affected. More non-unique tiles → more cycles → more runtime

6. Conclusion

- Modelling Hitori in ILP is **possible but unintuitive**, because ILP made for different types of problems
- As a result the models are **slower than most** comparable models in different paradigms
- For all our models we find that **runtime grows exponentially** with instance size
- Non-unique tiles in the instance make the **naive and path models faster**, but the **duplicates model slower**
- Adding redundant constraints never significantly and meaningfully increased solving speed

Future research:

- **Extend** the comparison to different paradigms
- **Expand** the comparison to more problems

Curious? Check the paper at:
https://resolver.tudelft.nl/uuid:f4abd9b2-904e-49d4-9da5-1304254a3555

The code is available at:
https://github.com/Sophieke32/Gurobi_Hitori