AUTHOR
Lesley Smits[1]
SUPERVISOR
Dr. A.L.D. Latour[1]
[1]EEMCS, Delft University of Technology, The Netherlands

Smashing Hitori

An analysis of the strengths and weaknesses of constraint programming paradigm Pumpkin

What is Hitori

A n x n grid with numbers ranging from 1 to n inclusive

Uniqueness: A number appears once in every row / column.
Adjacency: No marked tiles can touch each other.
Connectivity: The remaining white tiles are all connected.

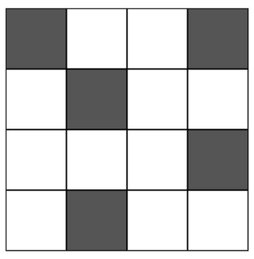
We use **Unique** puzzle instances, meaning there is only 1 valid solution for the puzzle

2	2	3
3	2	2
2	3	1

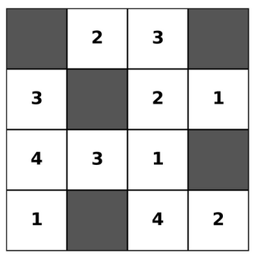
2	2	3
3	**2**	2
2	3	1

Generating Hitori instances

Solution Topology

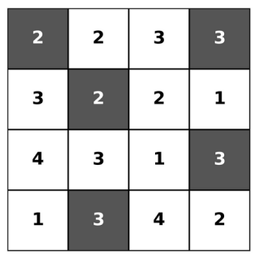


White number assignment



$v \in \{1, ..., n\} \wedge v$ is unique in row/col

Black number assignment



$v \in \{v_{white} \in$ row $\cup$ col$\}$

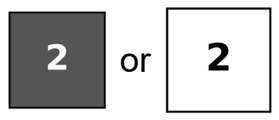
Modelling Hitori in CSP

Variables

a set of n x n $is\_black_{r,c}$

Domain

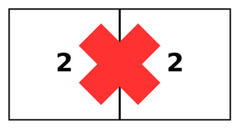
$D_{is\_black} \in$ {true, false}



Local Constraints

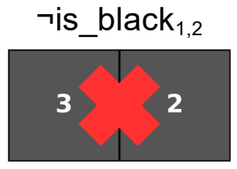
Uniqueness

$is\_black_{1,1} \vee$ is_black$_{1,2}$



Adjacency

$\neg is_black_{1,1} \vee \neg$is_black$_{1,2}$

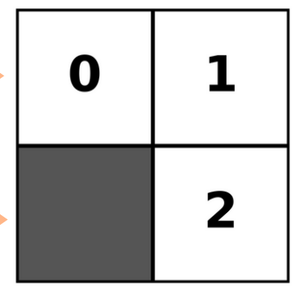


Distance Connectivity

Non-root White
Distance = smallest white neighbour distance + 1

Root
Distance = 0
Always white

Non-root Black
No distance value



Distances in Puzzle

Lazy Connectivity

Create solution

External Connectivity Check

Solution

Redundant Constraint Impact

RQ1: "*What is the impact of adding redundant constraints to our base encoding on solving time?*"

Lazy

Base Lazy: **36.9%** solved

LazyWN: **100%** solved

LazyWN + UC: **-16%** n {5, .. , 25}

LazyWN + UC: **-93%** n = 25

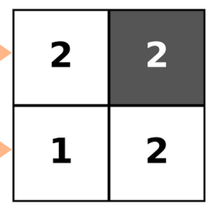
Others: Minimal or insignificant

White Neighbours (WN)

Every white tile must have ≥ 1 white neighbour.

Forces tile below white with WN

Forced white with UC



Unique Cell (UC)

Symbols unique in their row/col are set to white.

Distance

All configurations: **100%** solved

All contraints with significant impact had minimal actual impact

Setup

Dataset
40 puzzles per grid (5x5 to 25x25)

Metric
solving time

Significance
Wilcoxon Signed-Rank test ($p < 0.05$)

Puzzle Characteristics Impact

RQ2: "*What impact do the puzzle size and puzzle characteristics have on encoding size and solve time?*"

High ($x > Q_3$)
Medium ($Q_1 \leq x \leq Q_3$)
Low ($x < Q_1$)

We compare Low and High to Medium

Setup

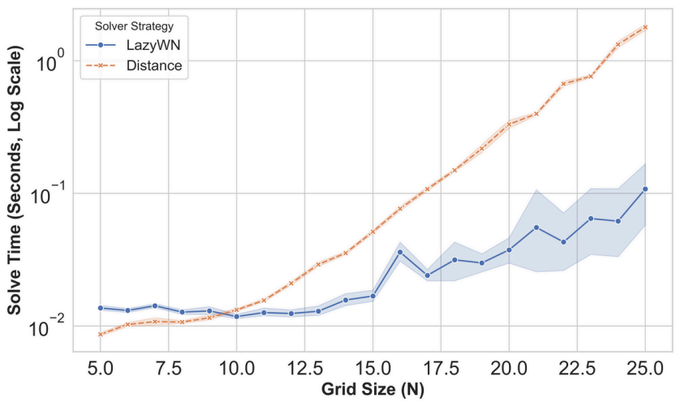
Dataset
1000 puzzles (20x20)
Metric
solving time & encoding size
Significance
Mann-Whitney U test ($p < 0.05$)

	LazyWN Solving Time	LazyWN Encoding Size	Distance Solving Time	Distance Encoding Size
High Non-Adjacent Duplicates	-47.38%	-6.06%	N/S	<1%
Low Non-Adjacent Duplicates	N/S	+9.61%	N/S	<1%
High Black Count	N/S	N/S	+1.26%	<1%
Low Black Count	N/S	N/S	+6.39%	<1%
Low/High Adjacent Duplicates	N/S	N/S	N/S	<1%

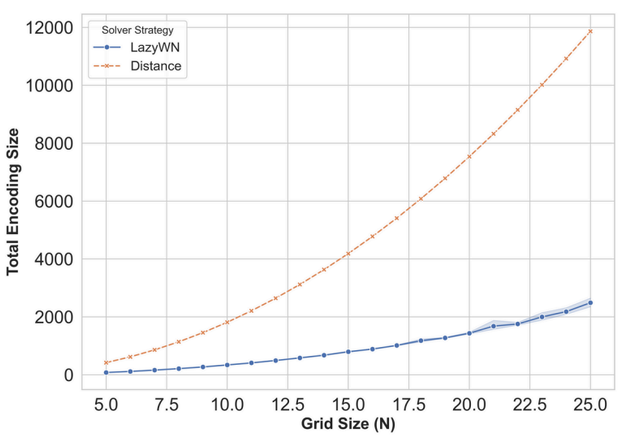
Puzzle Size Impact

RQ2: "*What impact do the puzzle size and puzzle characteristics have on encoding size and solve time?*"

Solving time
LazyWN faster for higher grid sizes but shows more variance



Encoding size
Distance is bigger and grows quicker, **80%** more then LazyWN at n = 25



Setup

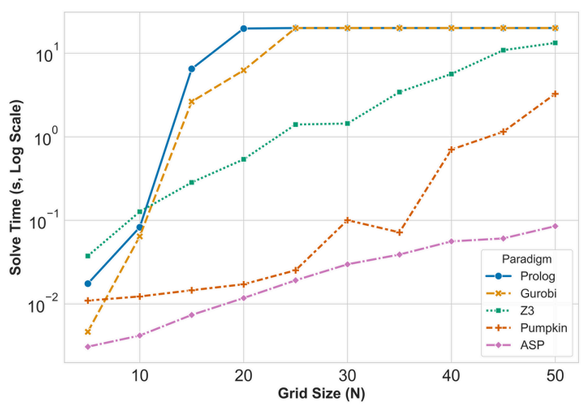
Dataset
50 puzzles per grid (5x5 to 25x25)

Metric
solving time & encoding size

Benchmarking against other paradigms

RQ3: "*How does Pumpkin compare to different paradigms at solving Hitori puzzles in solving capability and time?*"

Solving time



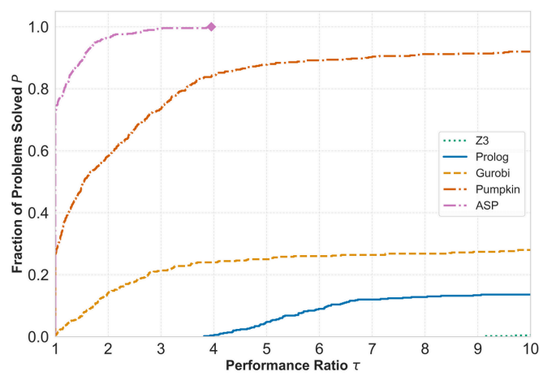
Pumpkin demonstrates superior scalability, maintaining a mean solving time of **3.3s** at n=50

Setup

Dataset
50 puzzles per grid (5x5 to 50x50, stepsize 5)

Metric
solving time & performance profile

Performance Profile
What fraction of puzzles *p* was within a performance ratio *τ* of the fastest solve time?



Pumpkin is highly competitive, achieving the fastest solve time in **30%** of instances and staying within a performance ratio of 4 for **80%** of all puzzles.

Conclusion

Our results show that, with a well-optimized connectivity approach, Pumpkin is a robust and viable option for modelling and solving Hitori.

It demonstrates strong scaling with size and efficiency on puzzles with high non-adjacent duplicate density. By integrating high-impact redundant constraints, it performs second only to ASP and surpasses Gurobi, Prolog, and Z3.