HPI_Quantastic (Annealing): Starbattle / Two Not Touch

Jacob Schäfer | Alexander Sohn | Richard Wohlbold

iQuHACK 2021

Why Star Battle / Two Not Touch?

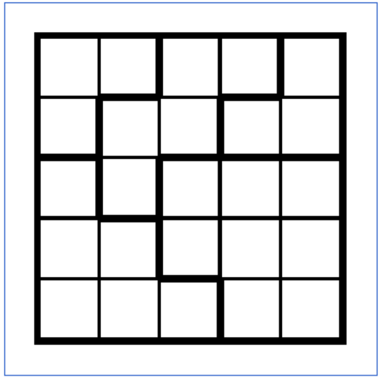
"Challenge Problems"

1. Implement the Knapsack problem with the DQM solver
2. Implement the Nurse Scheduling Problem with the DQM solver
3. Implement the game "Star Battle/Two Not Touch" (Google it)
4. Implement the puzzle "TetraVex" (Google it)
5. Find the famous Andrew Lucas paper https://arxiv.org/abs/1302.5843. Implement one of the NP-hard problems. If you find a D-Wave example for it (for example, graph partitioning), implement it using the DQM solver

Why Star Battle / Two Not Touch? (2)

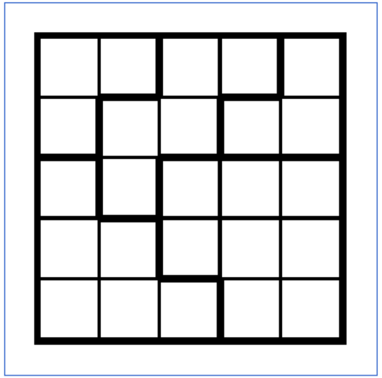
- Lots of fun to play
- Educate players on the power of Quantum Computing
    - → create interactive system

What is Starbattle / Two Not Touch?

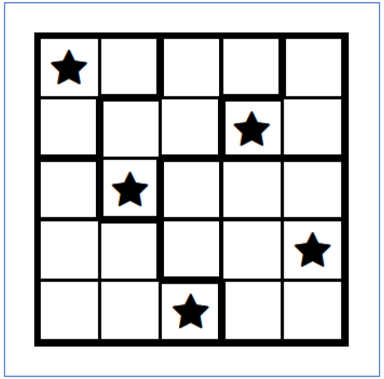


5x5/1★ Normal Star Battle Puzzle ID: 9,053,837

What is Starbattle / Two Not Touch?



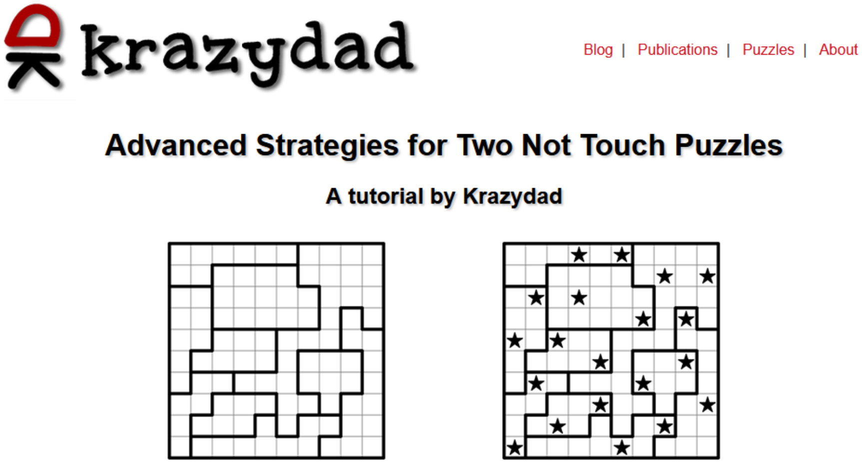
5x5/1★ Normal Star Battle Puzzle ID: 9,053,837



5x5/1★ Normal Star Battle Puzzle ID: 9,053,837

Playing the game…

- … is **hard**:
  - lots of constraints
  - lots of possiblities
  - advanced strategies limit the search space, but no one strategy



https://krazydad.com/twonottouch/adv_tutorial/

Deciding on a model

- Choice between
  - Binary Quadratic Model (BQM)
  - Discrete Quadratic Model (DQM)

Deciding on a model

- Choice between
  - **Binary Quadratic Model (BQM)**
  - Discrete Quadratic Model (DQM)

- Every cell of the board is a boolean variable:
  - 0 = no star
  - 1 = star

- ➔ Quadratic Unconstrained Binary Optimization (QUBO)

Modelling the constraints (1)

- Penalize adjacent stars:

x_i	x_j	$E(x_i, x_j)$
0	0	0
0	1	0
1	0	0
1	1	1

Modelling the constraints (1)

- Penalize adjacent stars:

x_i	x_j	$E(x_i, x_j)$
0	0	0
0	1	0
1	0	0
1	1	1

- AND function
- $ax_i + bx_j + cx_i x_j + d = x_i x_j$
- c could be any other positive value → 1 is a good value

Modelling the constraints (2)

- k stars per…
  1. …row
  2. …column
  3. …block
- → exactly k of the cells to be chosen

Modelling the constraints (3)

Let R be the set of qubit indices of the selected row, column or block

- $\sum_{i \in R} x_i = k$
- $\sum_{i \in R} x_i - k = 0$
- → QUBO: $(\sum_{i \in R} x_i - k)^2$

$= (\sum_{i \in R} x_i)^2 - 2\mathrm{k}(\sum_{i \in R} x_i) + k^2$

$= \sum_{i \in R} x_i + \sum_{i,j \in R, i<j} 2x_i x_j - 2k(\sum_{i \in R} x_i) + k^2$

$= (1 - 2\mathrm{k}) \sum_{i \in R} x_i + \sum_{i,j \in R, i<j} 2x_i x_j + k^2$

Implementing the BQM (1)

```
bqm = BinaryQuadraticModel({}, {}, 0.0, BINARY)


# constraint 1: n stars per row
for y, row in enumerate(cells):
    row_coords = [(y,x) for x,_ in enumerate(row)]
    row_bqm = combinations(row_coords, num_stars)
    bqm.update(row_bqm)
```

…

Implementing the BQM (2)

- n choose k constraints:

dimod.generators.constraints.combinations

combinations(*n, k, strength=1, vartype=<Vartype.BINARY: frozenset({0, 1})>*) [source]

Generate a bqm that is minimized when k of n variables are selected.

More fully, we wish to generate a binary quadratic model which is minimized for each of the k-combinations of its variables.

The energy for the binary quadratic model is given by $(\sum_i x_i - k)^2$.

Implementing the BQM (3)

- Coupling between adjacent qubits:

dimod.BinaryQuadraticModel.add_interaction

`BinaryQuadraticModel.add_interaction(u, v, bias, vartype=None)` [source]

Add an interaction and/or quadratic bias to a binary quadratic model.

Sampling the BQM (1)

- We tried lots of samplers, but LeapHybridSampler performed the best

LeapHybridSampler

class **`LeapHybridSampler`**(*solver=None, connection_close=True, **config*) [source]

A class for using Leap’s cloud-based hybrid BQM solvers.

Leap’s quantum-classical hybrid BQM solvers are intended to solve arbitrary application problems formulated as binary quadratic models (BQM).

Jacob Schäfer,
Alexander Sohn,
Richard Wohlbold

Sampling the BQM (2)

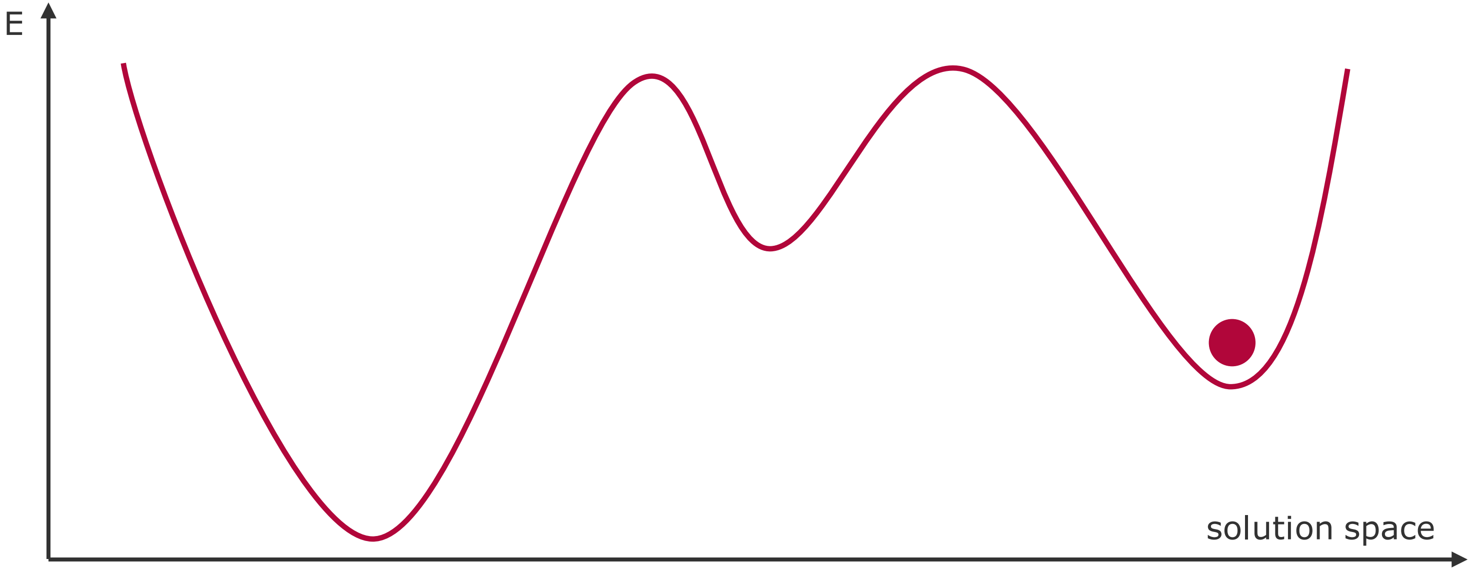
- 3 puzzle categories (KrazyDad):
  - *Easy* (8x8, 1☆)
  - *Medium* (10x10, 2☆)
  - *Hard* (14x14, 3☆)
- 100% valid samples on *Easy* & *Medium*
- Samples on *Hard* are not always valid

Sampling the BQM (2)

- 3 puzzle categories (KrazyDad):
  - *Easy* (8x8, 1☆)
  - *Medium* (10x10, 2☆)
  - *Hard* (14x14, 3☆)
- 100% valid samples on *Easy* & *Medium*
- Samples on *Hard* are not always valid
  - ~ 90% of samples are valid

Sampling the BQM (3)

- Why might the BQM be inaccurate on big problems?
  - → Lots of local minima that are far away from the optimal solution
- Energy landscape:



User Interface: game files (1)

- Puzzles are stored in text files, repository contains examples

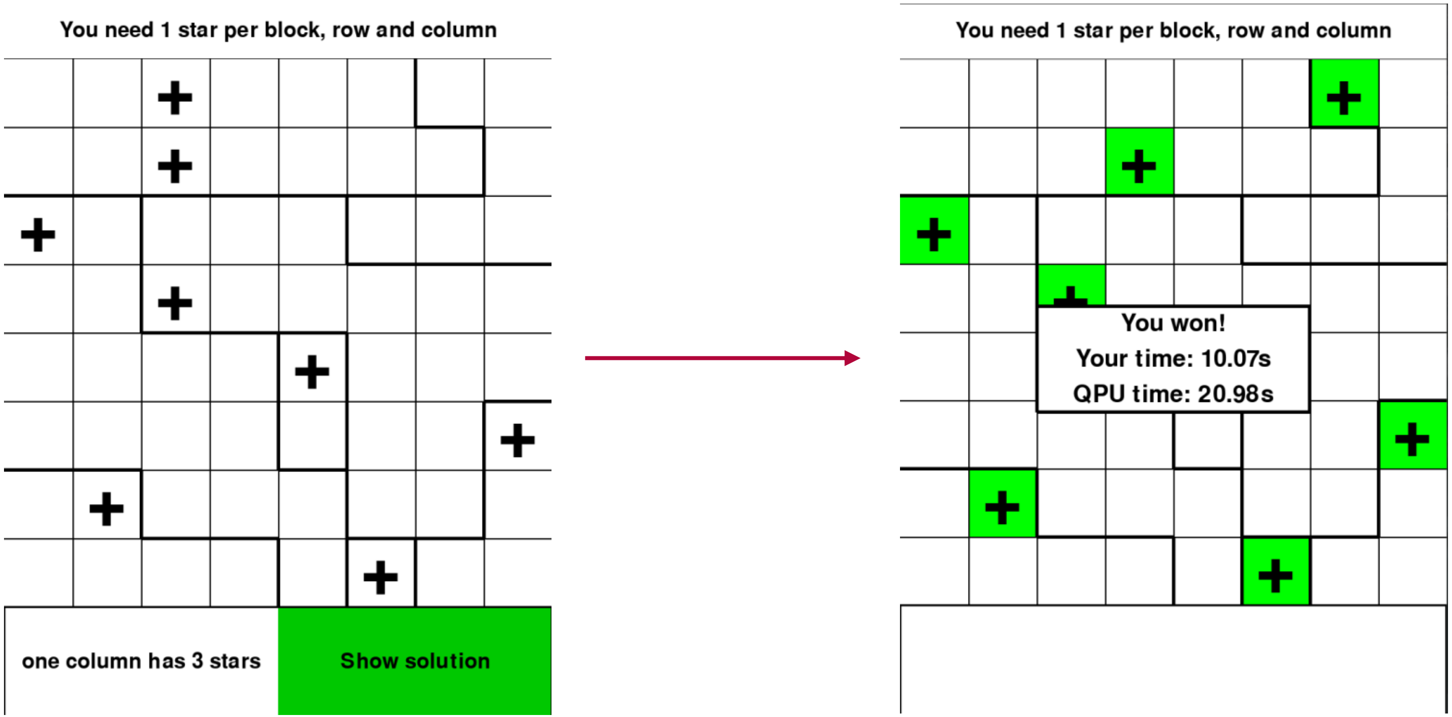
```
1
0 0 0 1 2 2 2 2
0 0 0 1 2 2 2 2
0 0 0 1 1 2 2 2
1 1 1 1 1 2 3 2
4 1 1 1 3 2 3 2
4 4 4 1 3 3 3 5
4 4 4 1 3 6 6 6
4 7 7 3 3 3 6 6
```

User Interface: solve (2)

```
$ python main.py solve games/hard/hard-1.txt
puzzle (3☆):
/ / / / $ $ $ ? ? ? ? ? ? ?
/ / / / $ $ $ ? ? ? ? ? ? ?
/ / / / | $ $ $ ? ? % % ( (
/ | | | | § + + + + % ( ( (
/ | | | | § § § + % % % ( (
/ / / | | | | § + % % % ( (
/ / / / | § § § + + % % % (
_ _ _ / / § & & + + % % % (
_ _ ~ § § § & # # + ^ ^ ( (
_ ~ ~ ~ ~ § & # + + ^ ^ ^ ^
~ ~ ~ ~ ~ & & # + ^ ^ ^ ^ ^
~ ~ ~ ~ ~ = & # # ^ ^ ^ ^ ^
~ = = ~ = = & # # ^ ^ ^ ^ ^
~ = = = = = = = # # # ^ ^ ^
```

```
found valid solution:
- - - - - * - - * - * - - -
- * - * - - - - - - - - - *
- - - - - * - * - - - * - -
- * - * - - - - - * - - - -
- - - - - * - - - - - * - *
- * - * - - - * - - - - - -
- - - - - - - - - * - * - *
* - * - - - - * - - - - - -
- - - - * - - - - * - - * -
* - * - - - * - - - - - - -
- - - - * - - - - - * - * -
* - - - - - * - * - - - - -
- - * - * - - - - - - - * -
- - - - - - * - * - * - - -
```

User Interface: play (3)



Live Demo

Hasso Plattner Institut
Digital Engineering • Universität Potsdam

Thank you for your attention!
Questions?

Jacob Schäfer | Alexander Sohn | Richard Wohlbold
iQuHACK 2021