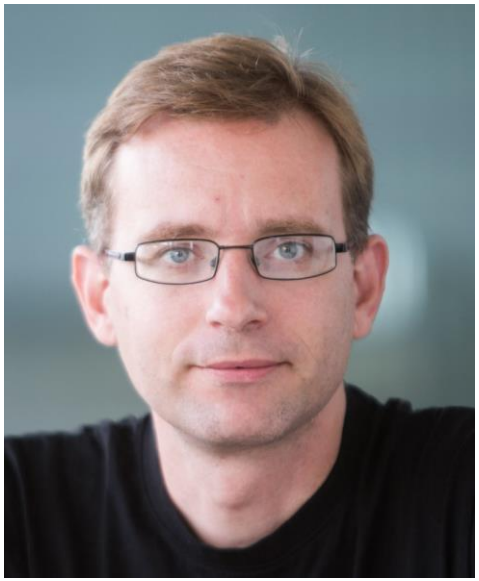
CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Decision Analytics

Lecture 1: Module Introduction

Introduction



- Dr Christian Beder
  - Lecturer at the Department of Computing
  - Email: Christian.Beder@cit.ie
  - Office Room: B180A

- Qualification & Experience:
  - Researcher at CIT's Nimbus centre 2010-2019 working on large-scale collaborative European Research Projects
  - PhD in Photogrammetry 2006
  - MSc in Computer Science 2002

What is Decision Analytics?

- Decision analytics is a collection of methodologies and tools to facilitate optimal decision making based on data
- Real-world problems are formulated by identifying variables and their mutual constraints
- Often an objective function is formulated to quantify “how good” a solution is
- An algorithmic approach is taken to find the “best” solution for the given problem

What is Decision Analytics?

- The most challenging task is typically creating a suitable model for a real-world application

- This comprises
  - Identifying relevant decision variables
  - Identifying necessary auxiliary variables

  - Identifying application specific constraints linking variables with data
  - Identifying internal constraints representing implicit assumptions and linking variables with each other

  - Identify an objective function to characterise the optimisation goal

  - Choose the most suitable tools and algorithms to solve the problem

What is Decision Analytics?

- In this module we will mainly focus on two types of algorithmic approaches
  - **Constraint Programming**
  - **Linear Programming**

- Many decision problems are so hard that there is no rule for any a-priori “better” approach
- (the best-known worst-case optimal solution is often to try out all possible combinations, which is not possible for even very small problem sizes)

Real world example: Warehouse logistics

Real world example: Optimal feeding

Real world example: Production lines

Real world example: Crop planning

Real world example: Project management

Real world example: Gate allocation

Real world example: Grid management

Real world example: Supply chain management

Learning outcomes

Module descriptor
https://courses.cit.ie/index.cfm/page/module/moduleId/13407

- LO1: Assess a wide range of real-world problems, which can be solved through the application of decision analytics.
- LO2: Model a variety of real-world problems, identifying their main characteristics as constraint satisfaction and optimisation problems.
- LO3: Apply constraint programming algorithms to solve real-world problems.
- LO4: Apply linear programming algorithms to solve real-world optimisation problems.
- LO5: Evaluate an algorithm to determine its soundness and completeness for a particular optimisation problem.

More informal learning outcomes

- Understand what type problems can be solved using the methods presented here
- Understand how to formulate these problems so that they can be solved using constraint programming
- Use the methods of constraint programming to solve these problems
- Understand what type of problems can be solved by linear programming, how to formulate and solve these
- Learn how to analyse the method for your particular problem

Course breakdown & Assessment

Module descriptor
https://courses.cit.ie/index.cfm/page/module/moduleId/13407

- 2x1h lecture / week (Mon & Thu)
  - Presentation of theory and concepts
- 1x2h lab / week split into two groups (Thu)
  - Apply concepts seen in class
  - Support on assignments

- Labs will commence next week (6/2/2020)

- 100% Continuous Assessment
  - 50% assignment due in week 8 (20/03/2020)
  - 50% assignment due in week 13 (08/05/2020)

Module content

- L1-L6: Introduction
  - Introductory examples, NP-completeness, Python, Google OR Tools, types of optimisation problems, soundness/completeness, constraint programming principles
- L7-L12: Modelling
  - More modelling examples with different representations, symmetries, constraint satisfaction vs. optimisation

Module content

- L13-L16: Constraint Propagation
  - Value propagation, domain propagation, Node consistency, Arc consistency, Domain consistency
- L17-L18: Search
  - Branching, exploration, heuristics
- L19-L23: Linear Programming
  - Modelling examples, transformation of constraints, simplex algorithm, duality, integer linear programming

Assignments

- A1: Constraint Programming (due: 20/03/2020)
- A2: Linear Programming (due: 08/05/2020)

Both assignments will comprise
- Model a real-world problem
- Implement this model and generate a solution
- Evaluate the properties of the model and the solution

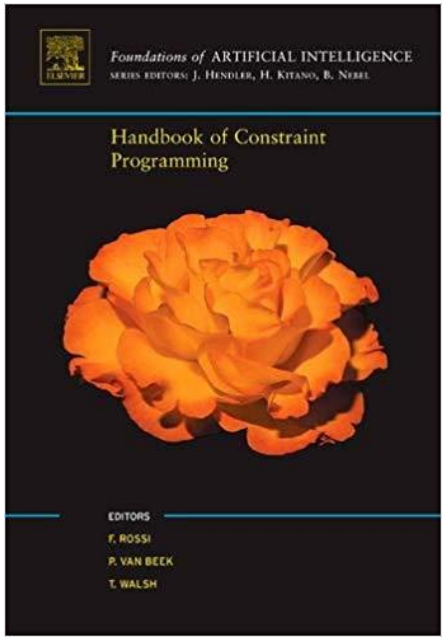
Recommended resources

- Google OR Tools:

  https://developers.google.com/optimization/

- Francesca Rossi (Editor), van Beek, Peter (Editor), Toby Walsh (Editor)
  Handbook of Constraint Programming
  Elsevier, 2006

  http://www.dcs.gla.ac.uk/~pat/cpM/papers/CP_Handbook-20060315-final.pdf



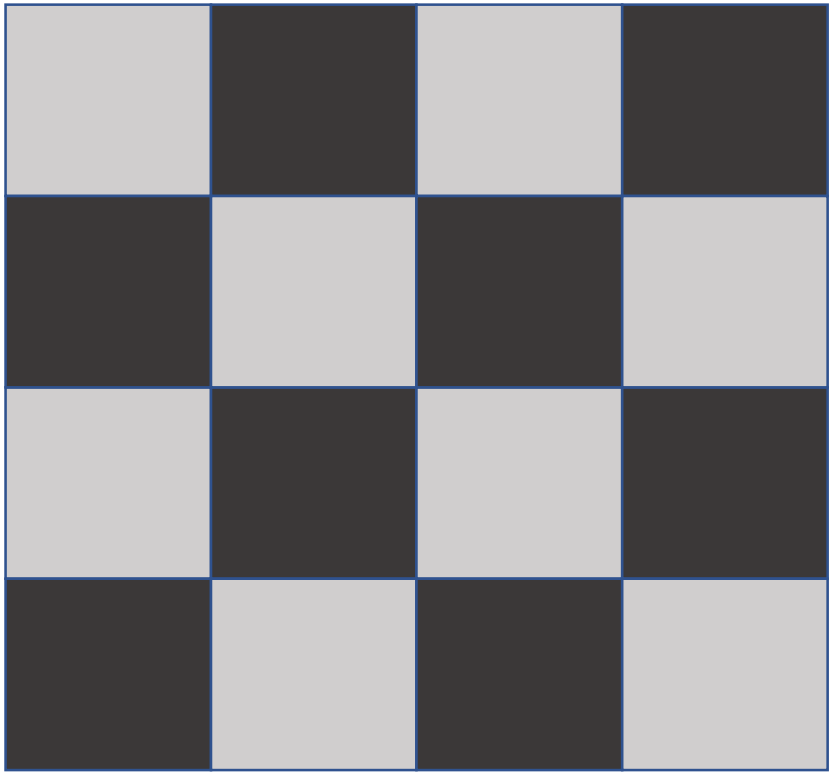
Development environment

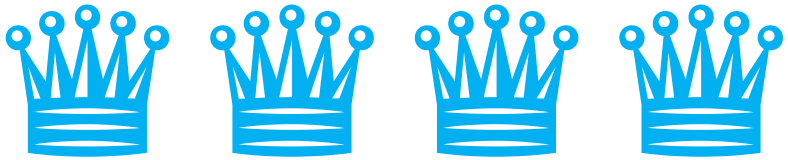
- All lab exercises and assignments will be developed in Python3
- The recommended IDE is Spyder and is part of the Anaconda package: https://www.anaconda.com/
- The Anaconda distribution already includes NumPy https://numpy.org/ and Pandas https://pandas.pydata.org/
- In addition you will be needing the Google OR Tools https://developers.google.com/optimization/

Full labs will commence from week 2.

Example: 4-Queens puzzle

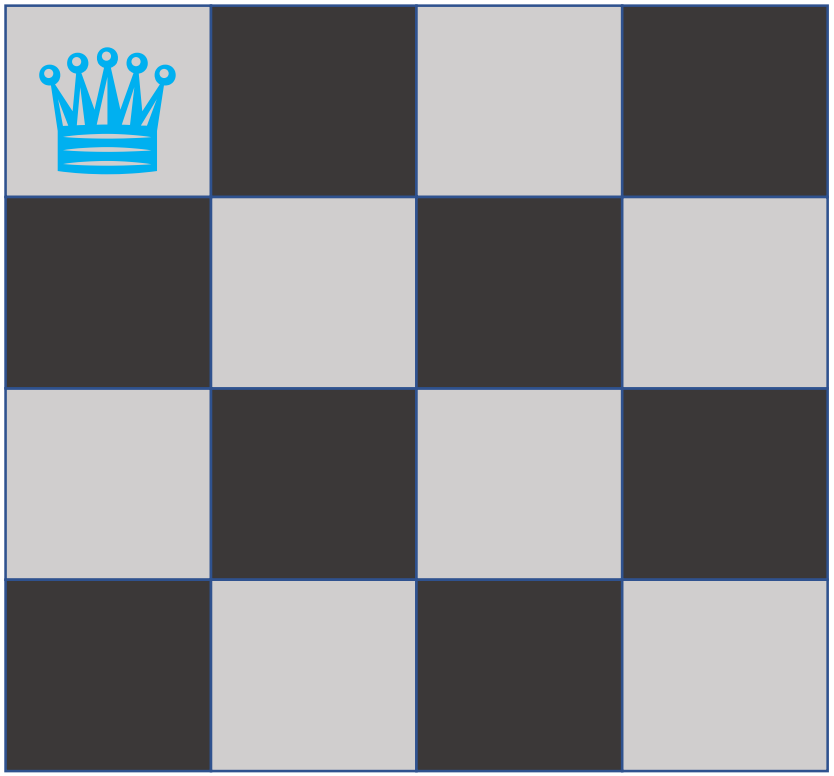
Can we place 4 queens on a chessboard so that they do not threaten each other?





Example: 4-Queens puzzle

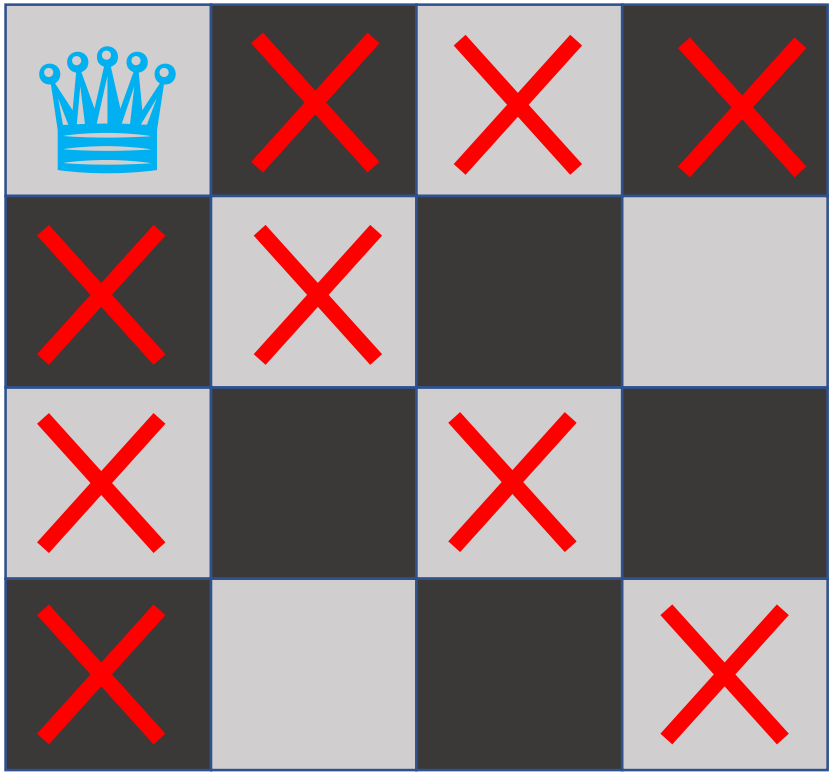
Can we place 4 queens on a chessboard so that they do not threaten each other?



- Let's start and see

Example: 4-Queens puzzle

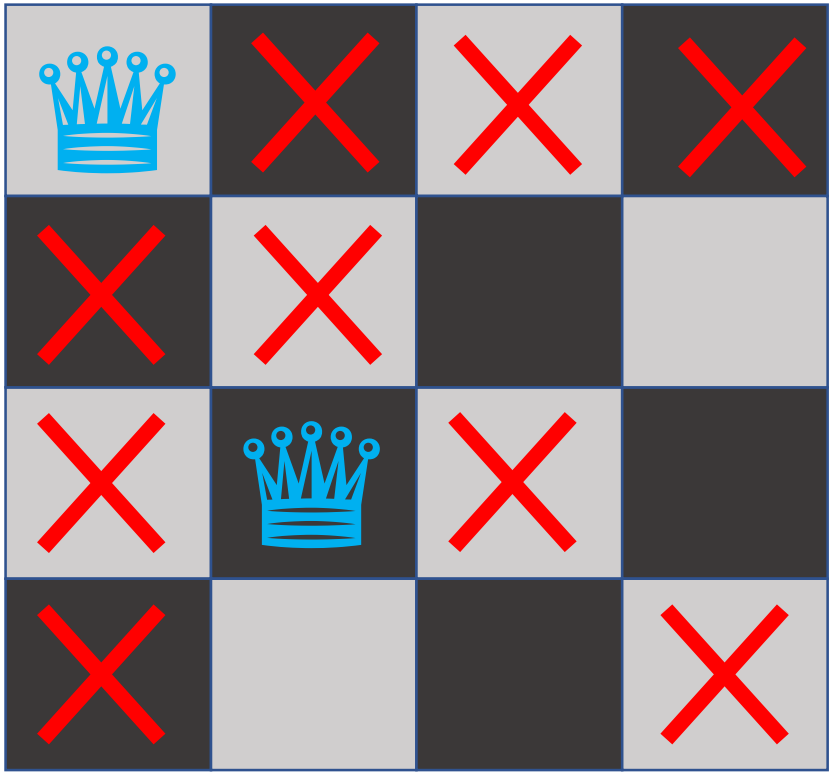
Can we place 4 queens on a chessboard so that they do not threaten each other?



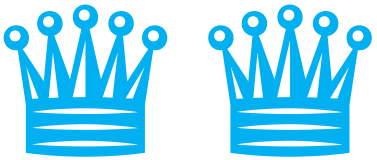
- Let's start and see
- This puts constraints on other fields

Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?

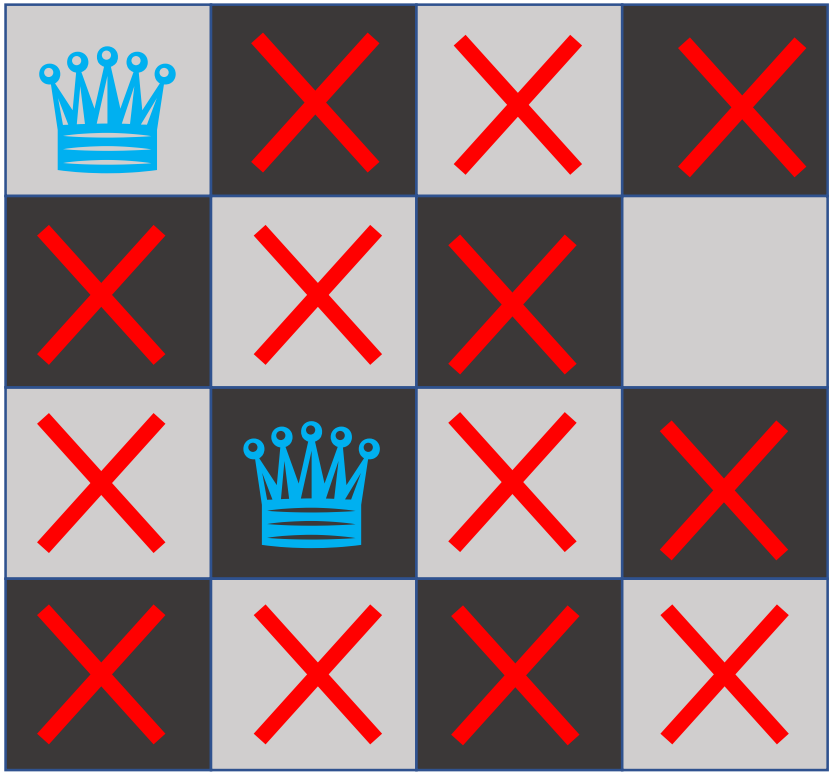


- Let's start and see
- This puts constraints on other fields
- For the second row we have to choices

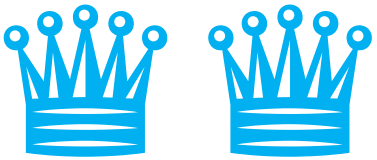


Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?

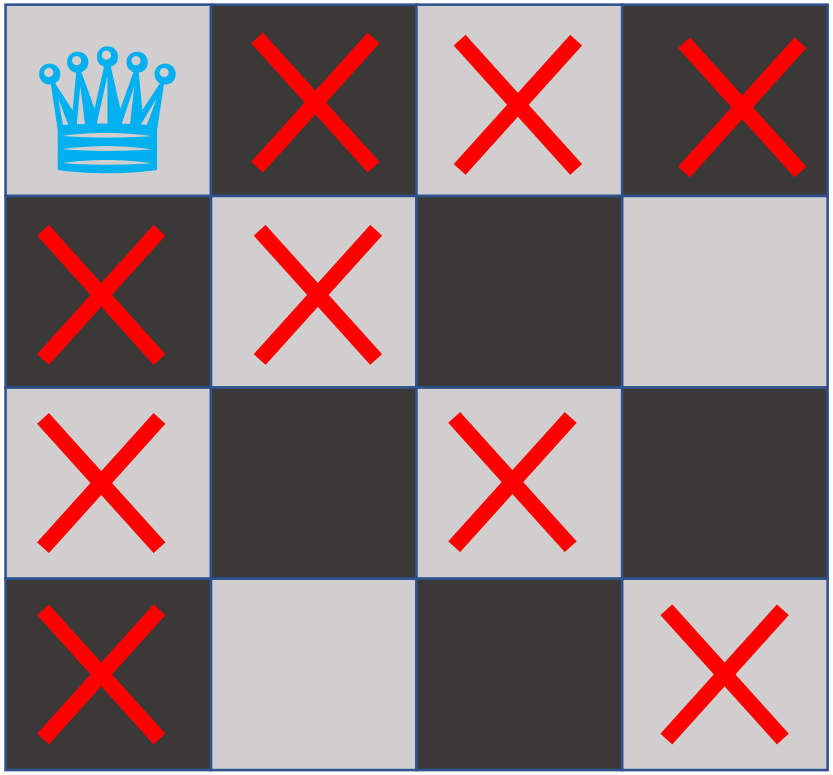


- Let's start and see
- This puts constraints on other fields
- For the second row we have to choices
- After constraint propagation, only one space is left



Example: 4-Queens puzzle

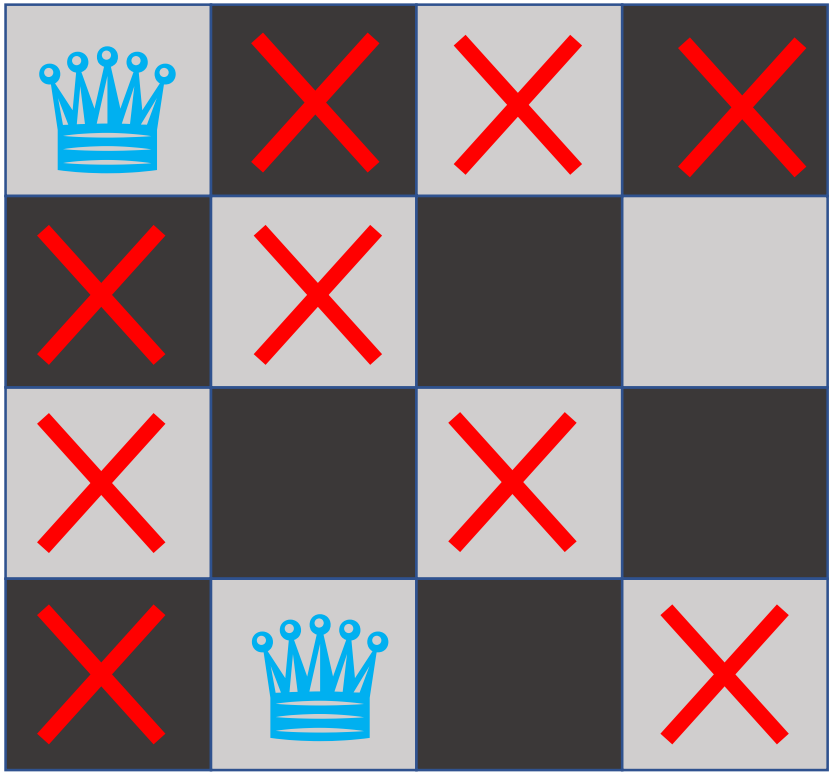
Can we place 4 queens on a chessboard so that they do not threaten each other?



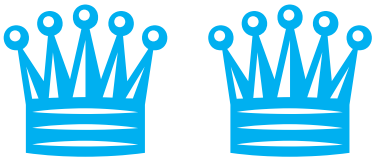
- Lets remove it again

Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?

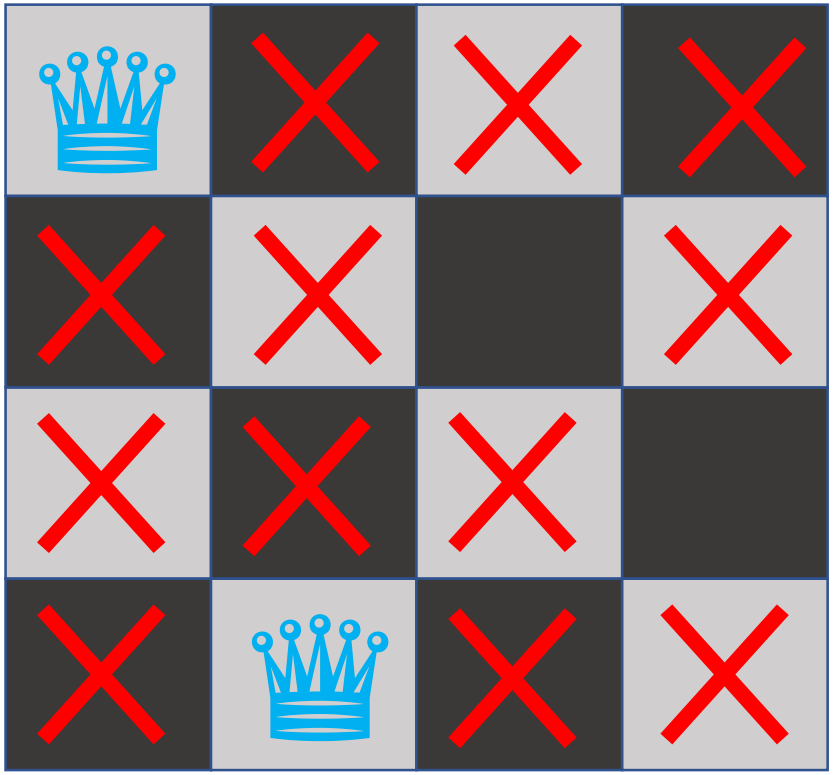


- Lets remove it again
- Try a different location

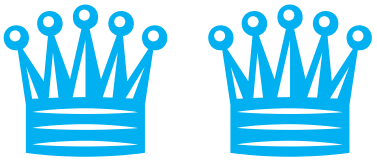


Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?

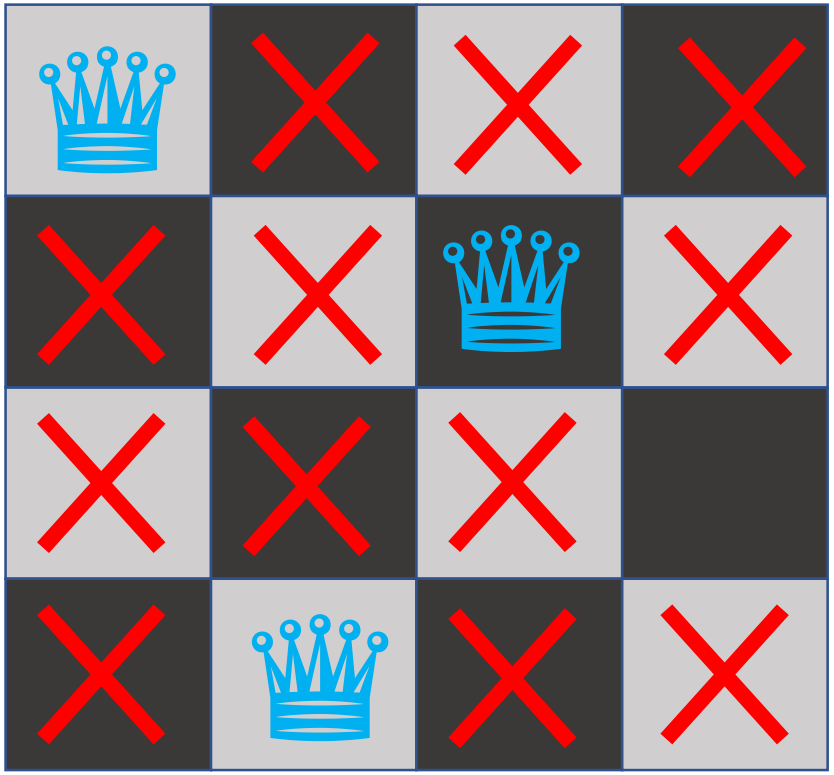


- Lets remove it again
- Try a different location
- Propagate the constraints



Example: 4-Queens puzzle

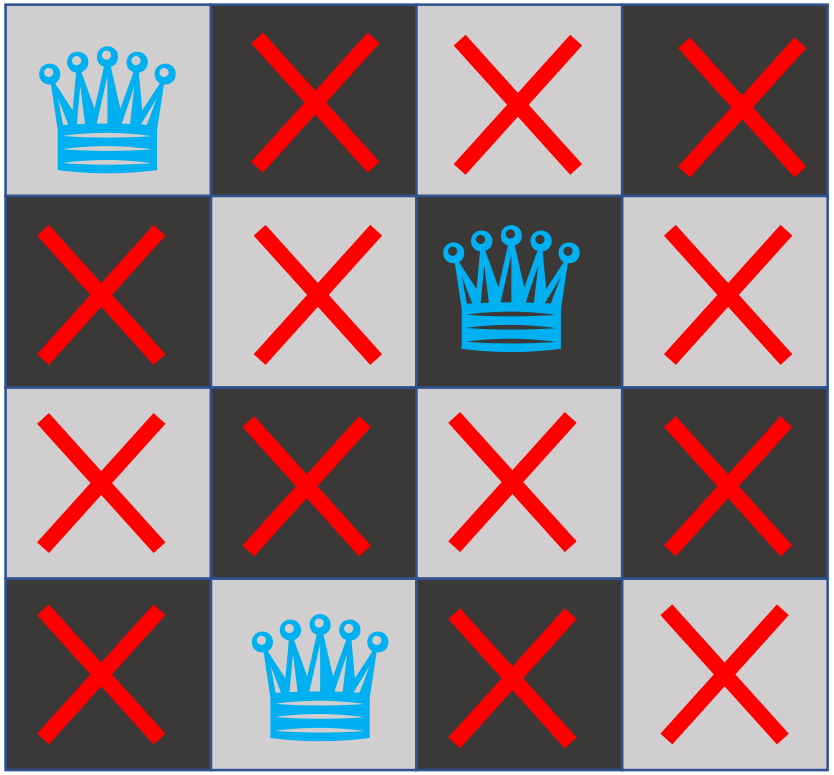
Can we place 4 queens on a chessboard so that they do not threaten each other?



- Lets remove it again
- Try a different location
- Propagate the constraints
- Place the third queen

Example: 4-Queens puzzle

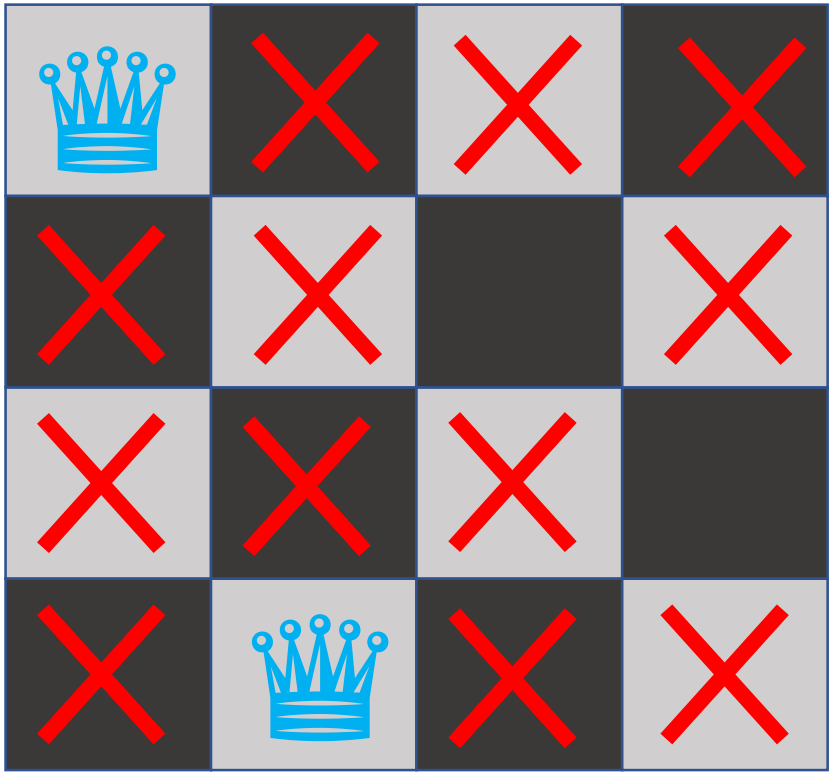
Can we place 4 queens on a chessboard so that they do not threaten each other?



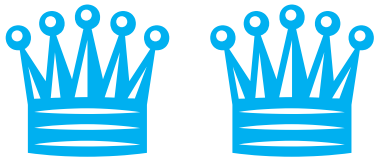
- Lets remove it again
- Try a different location
- Propagate the constraints
- Place the third queen
- After constraint propagation no space left, so lets remove it again

Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?

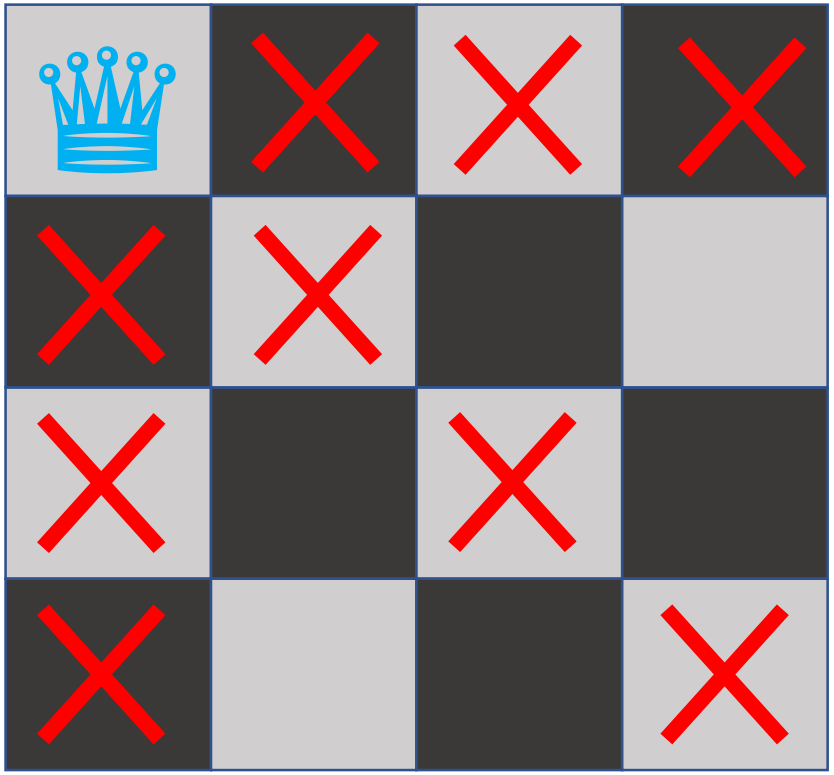


- This path did not lead anywhere, so lets backtrack one further



Example: 4-Queens puzzle

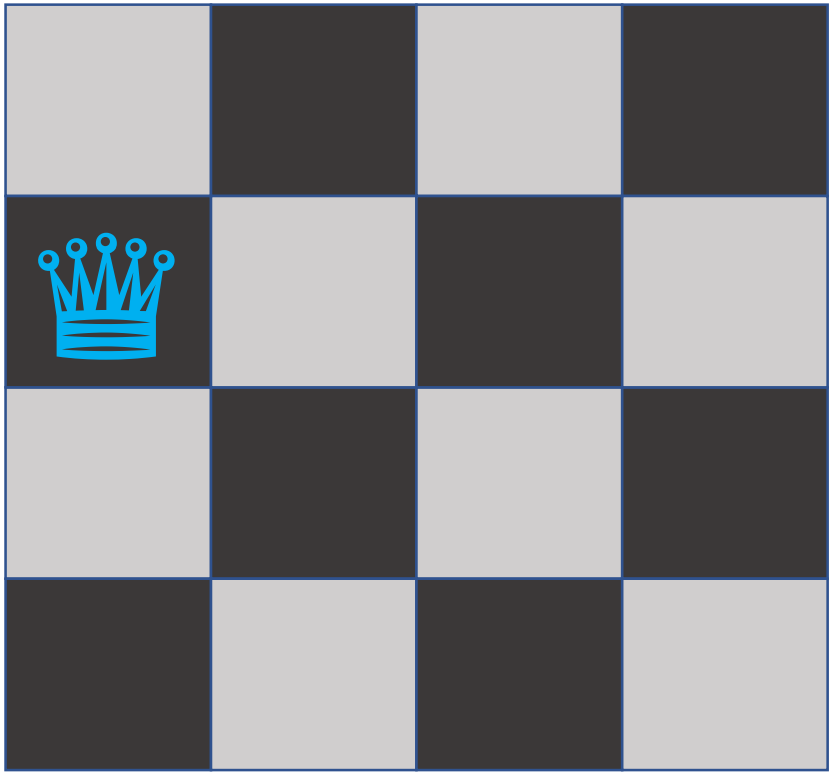
Can we place 4 queens on a chessboard so that they do not threaten each other?



- This path did not lead anywhere, so lets backtrack one further
- We are back to square one, none of these paths led anywhere

Example: 4-Queens puzzle

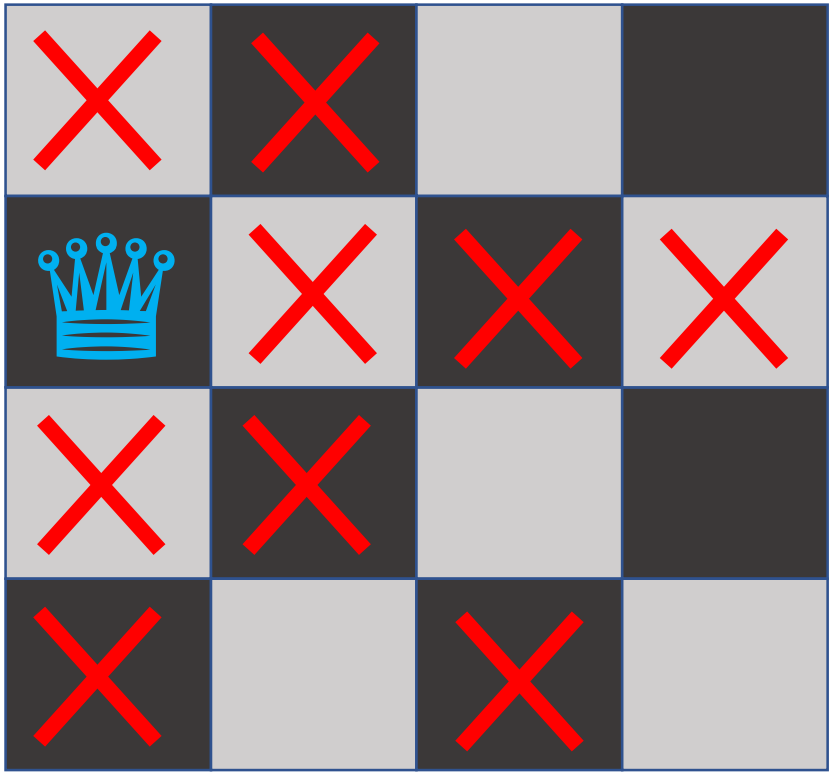
Can we place 4 queens on a chessboard so that they do not threaten each other?



- Try the first queen in the second location

Example: 4-Queens puzzle

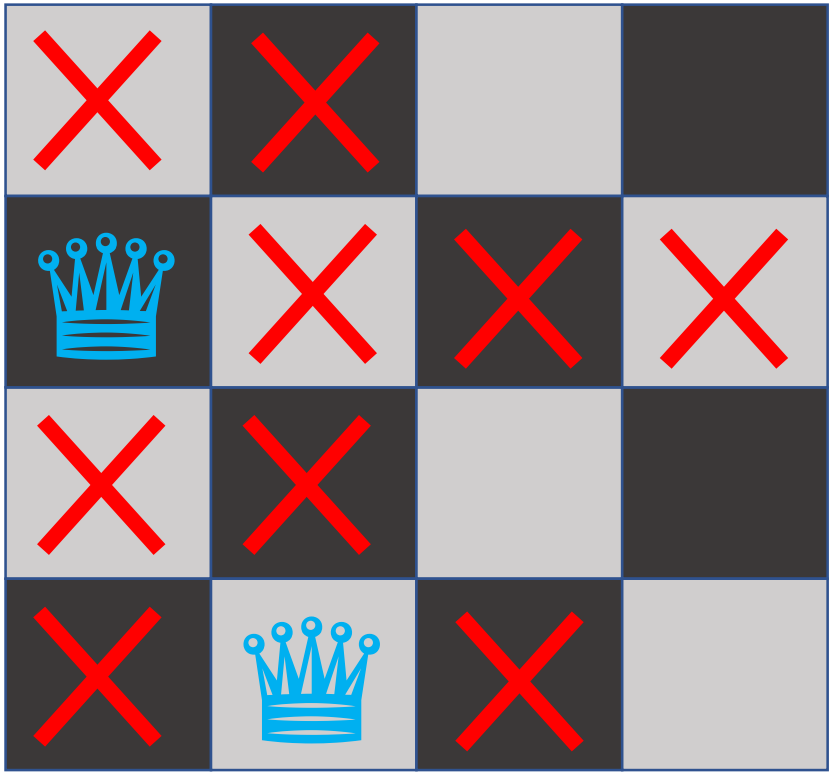
Can we place 4 queens on a chessboard so that they do not threaten each other?



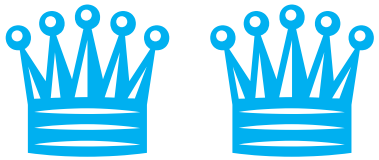
- Try the first queen in the second location
- Propagate the constraints

Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?

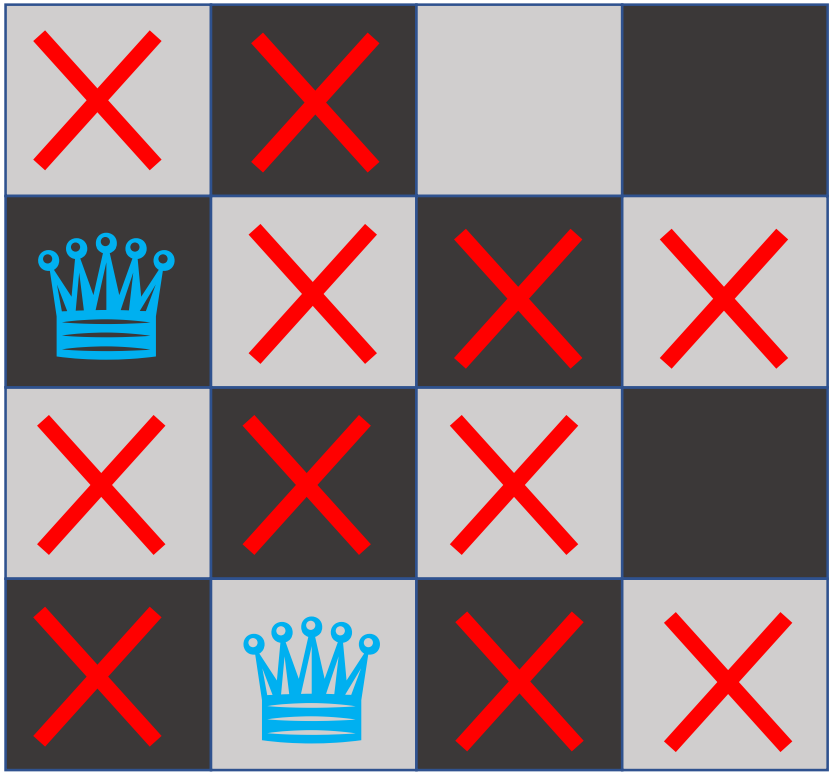


- Try the first queen in the second location
- Propagate the constraints
- Place next the queen

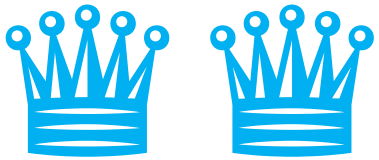


Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?

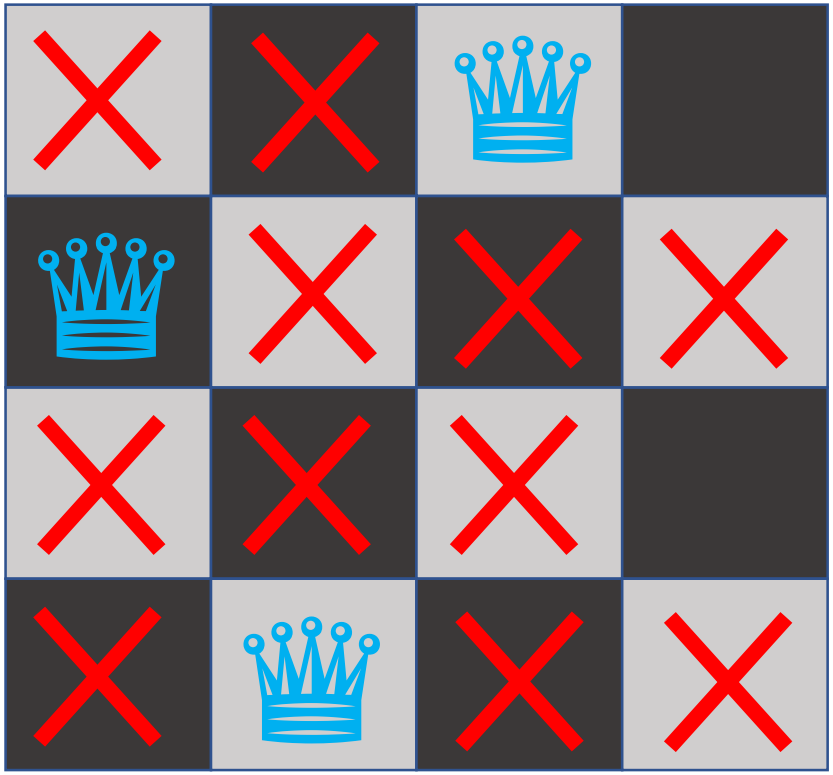


- Try the first queen in the second location
- Propagate the constraints
- Place next the queen
- Propagate the constraints



Example: 4-Queens puzzle

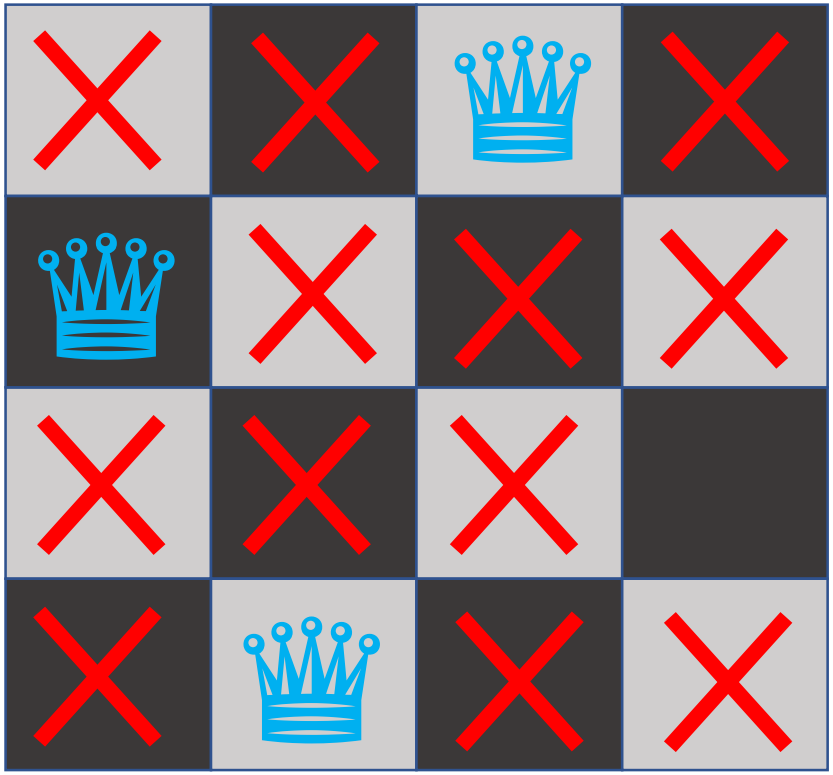
Can we place 4 queens on a chessboard so that they do not threaten each other?



- Try the first queen in the second location
- Propagate the constraints
- Place next the queen
- Propagate the constraints
- Place next queen

Example: 4-Queens puzzle

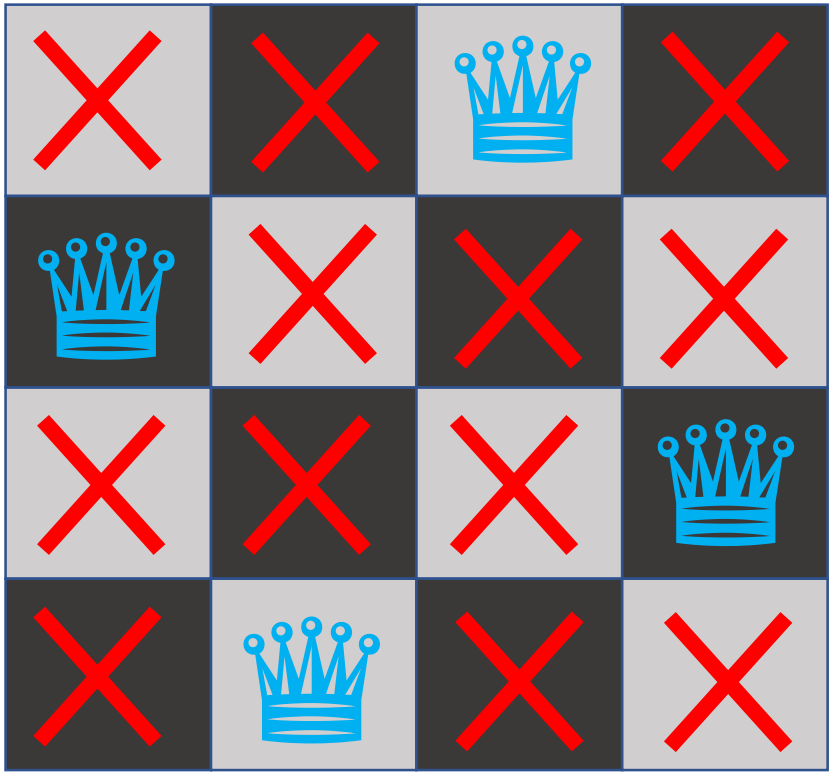
Can we place 4 queens on a chessboard so that they do not threaten each other?



- Try the first queen in the second location
- Propagate the constraints
- Place next the queen
- Propagate the constraints
- Place next queen
- Propagate the constraints

Example: 4-Queens puzzle

Can we place 4 queens on a chessboard so that they do not threaten each other?



- Try the first queen in the second location
- Propagate the constraints
- Place next the queen
- Propagate the constraints
- Place next queen
- Propagate the constraints
- Place the final queen

We have a solution

Constraint Programming

Constraint programming is an approach to solving optimisation and satisfiability problems comprising of three steps

- **Modelling**: Find an adequate formal representation of the problem, already considering reducing the size of the search space

- **Propagation**: Create constraints that can propagate and reduce the search space as much as possible early on

- **Search**: Implement a strategy that finds a good solution early on in the search

Thank you for your attention!