CMSC 471
Artificial Intelligence

Constraints

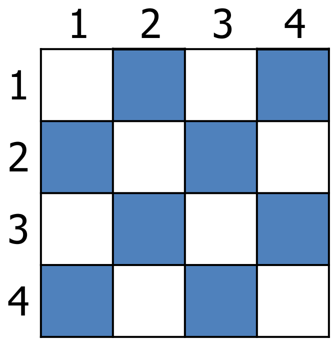
Many slides courtesy Tim Finin

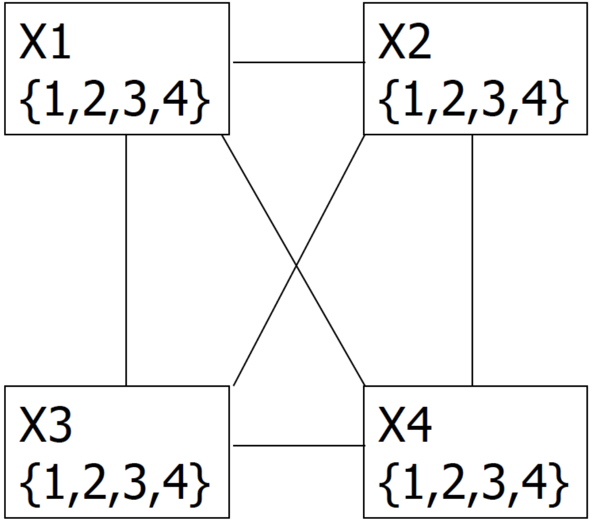
General Methods of Solving CSPs

- Generate-and-Test, aka Brute Force
- Search (backtracking)
- Consistency checking
  - Forward checking
  - Arc consistency
  - Domain splitting
  - Variable Elimination
- Localized search

Is AC3 Alone Sufficient?

Consider the four queens problem





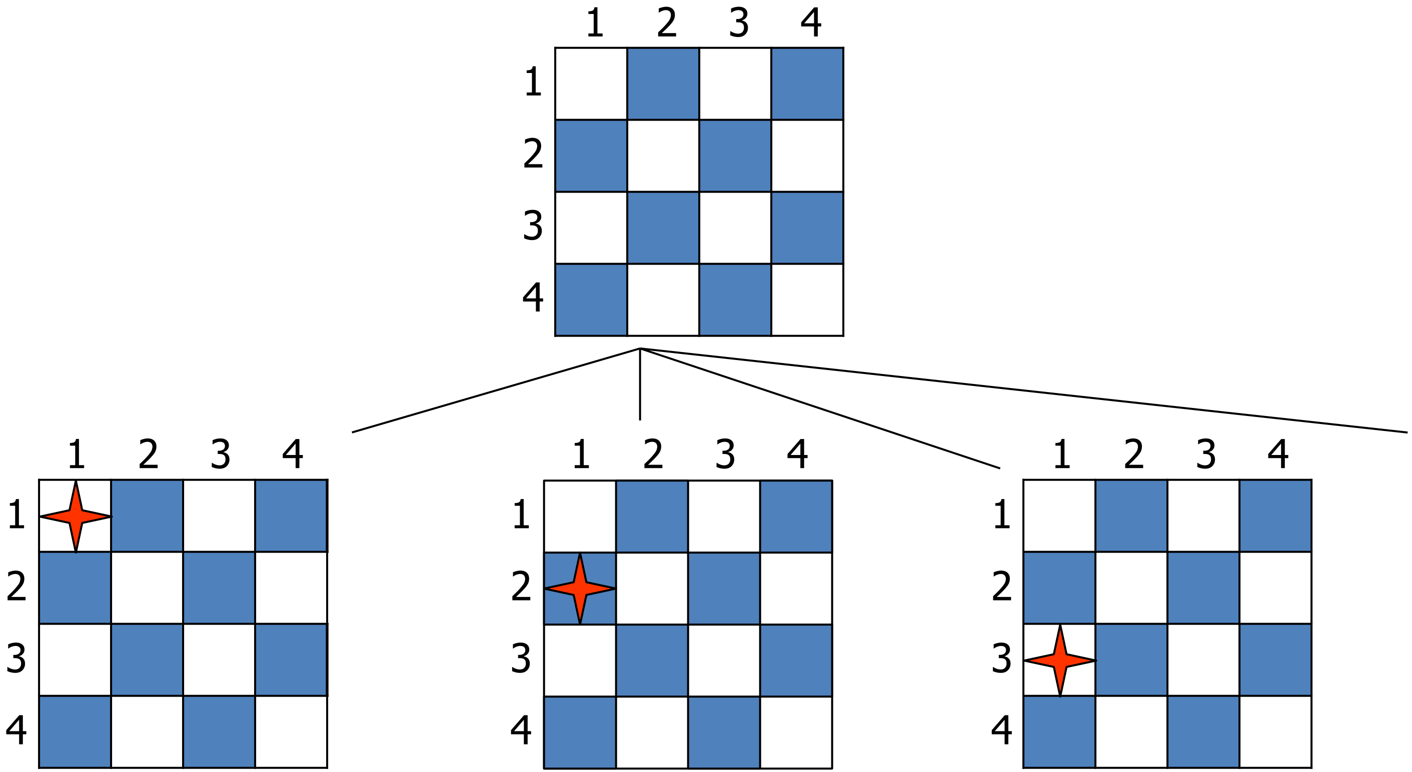
Solving a CSP still requires search

- Search:
  - can find good solutions, but must examine non-solutions along the way

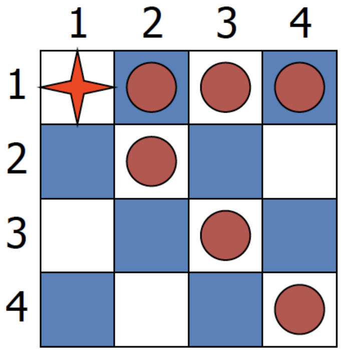
- Constraint Propagation:
  - can rule out non-solutions, but this is not the same as finding solutions

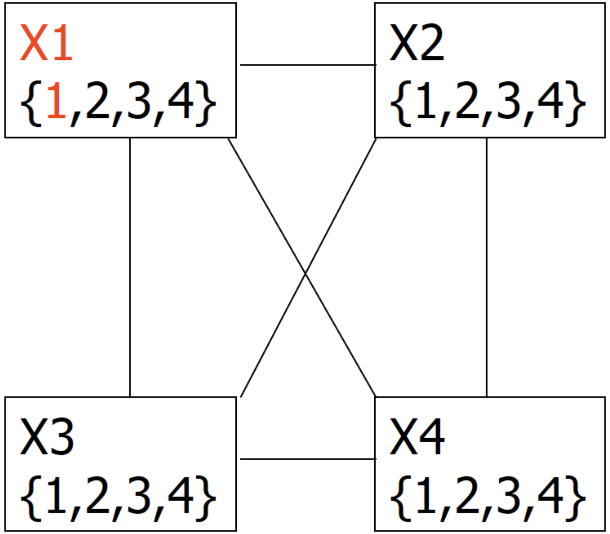
Solving a CSP still requires search

- Search:
  - can find good solutions, but must examine non-solutions along the way
- Constraint Propagation:
  - can rule out non-solutions, but this is not the same as finding solutions
- Interweave constraint propagation & search:
  - perform constraint propagation at each search step

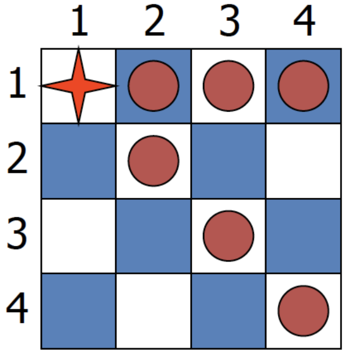

1 2 3 4
1
2
3
4
1 2 3 4
1
2
3
4
1 2 3 4
1
2
3
4
1 2 3 4
1
2
3
4

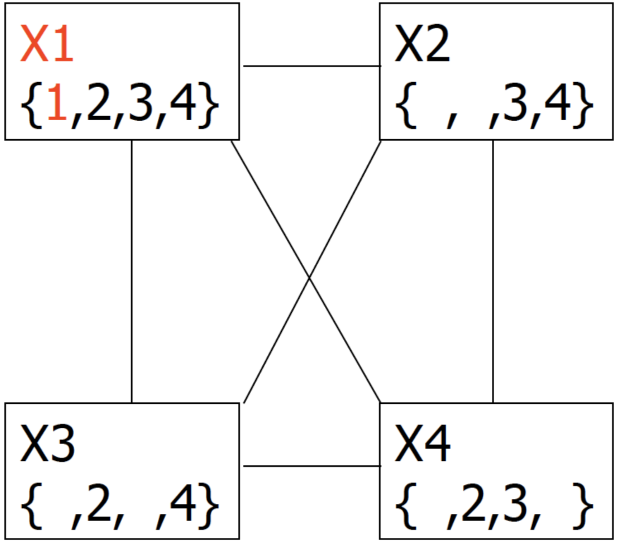
4-Queens Problem



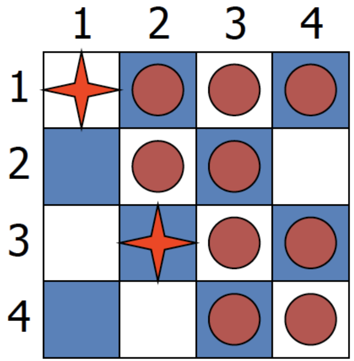


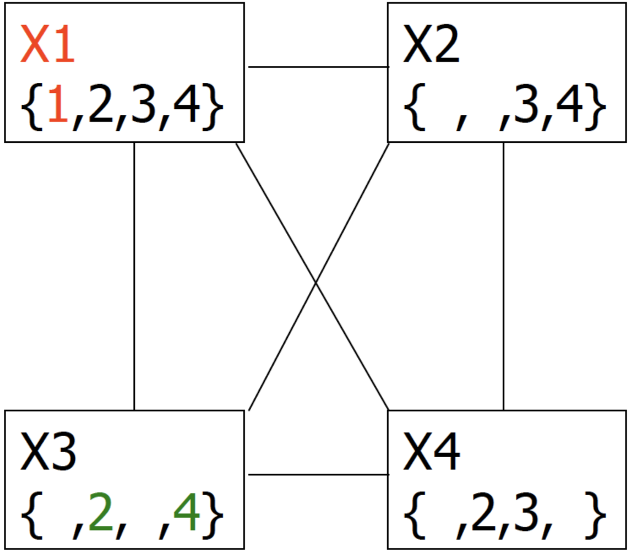
4-Queens Problem





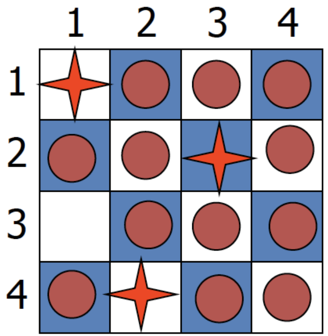
4-Queens Problem

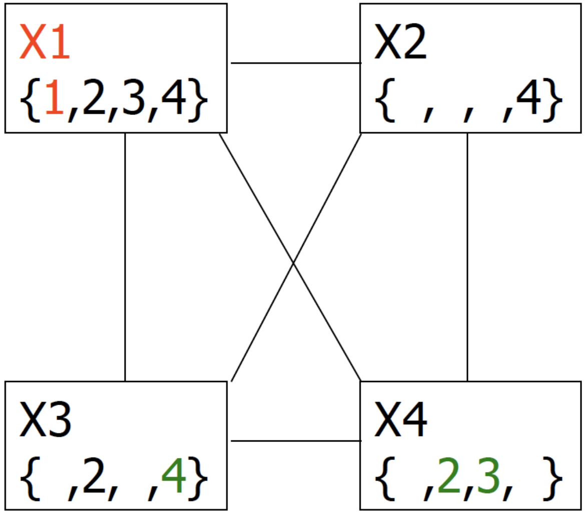




X2=3 eliminates { X3=2, X3=3, X3=4 }
⇒ inconsistent!

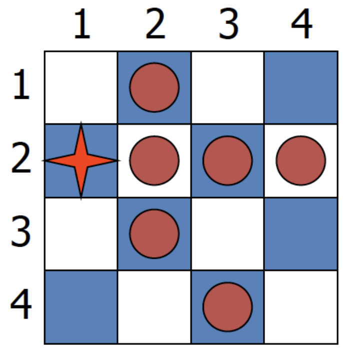
4-Queens Problem

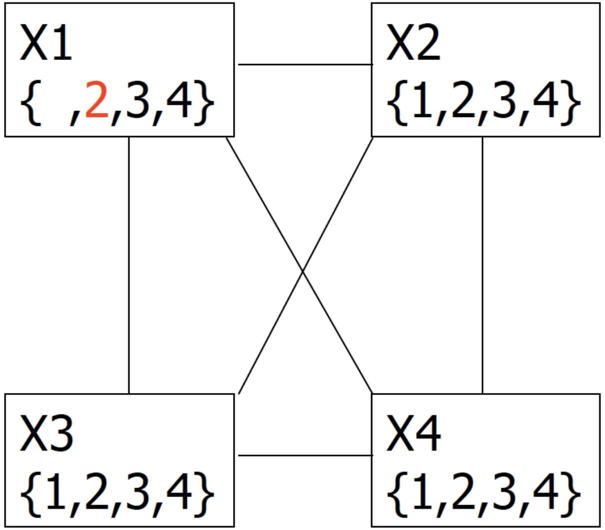




X2=4 ⇒ X3=2, which eliminates { X4=2, X4=3}
⇒ inconsistent!

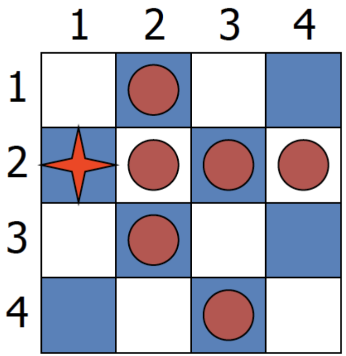
4-Queens Problem

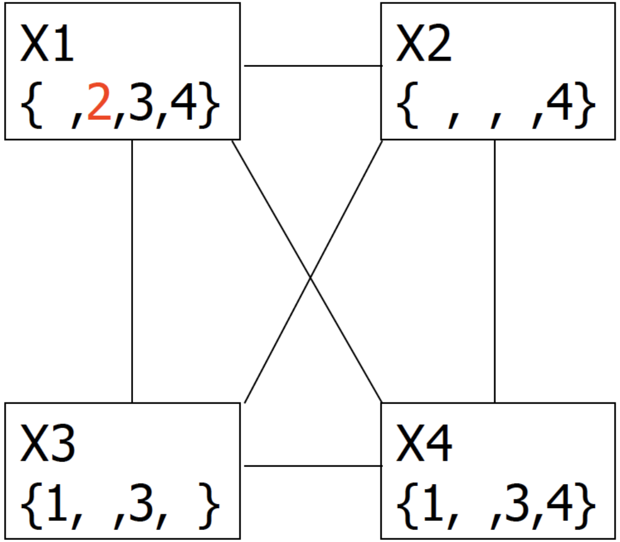




X1 can't be 1, let's try 2

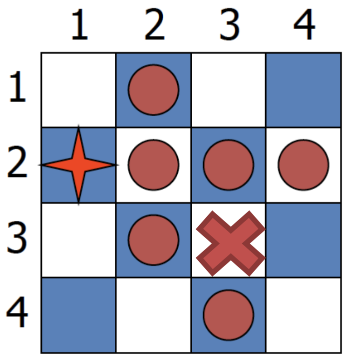
4-Queens Problem

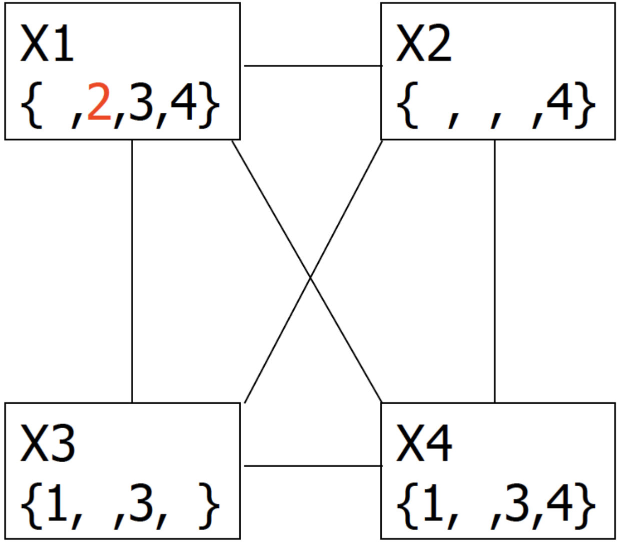




Can we eliminate any other values?

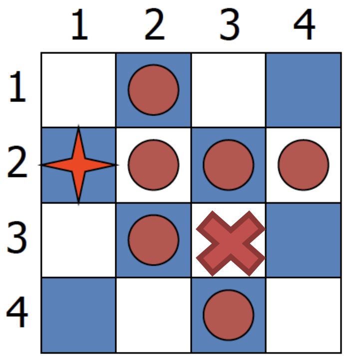
4-Queens Problem

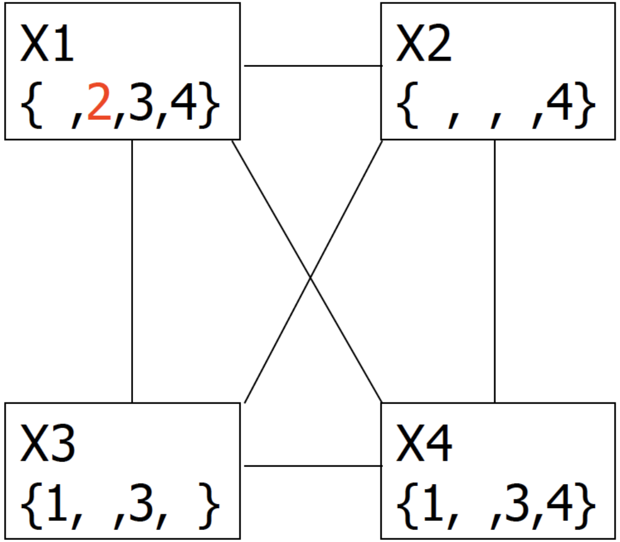




Can we eliminate any other values?

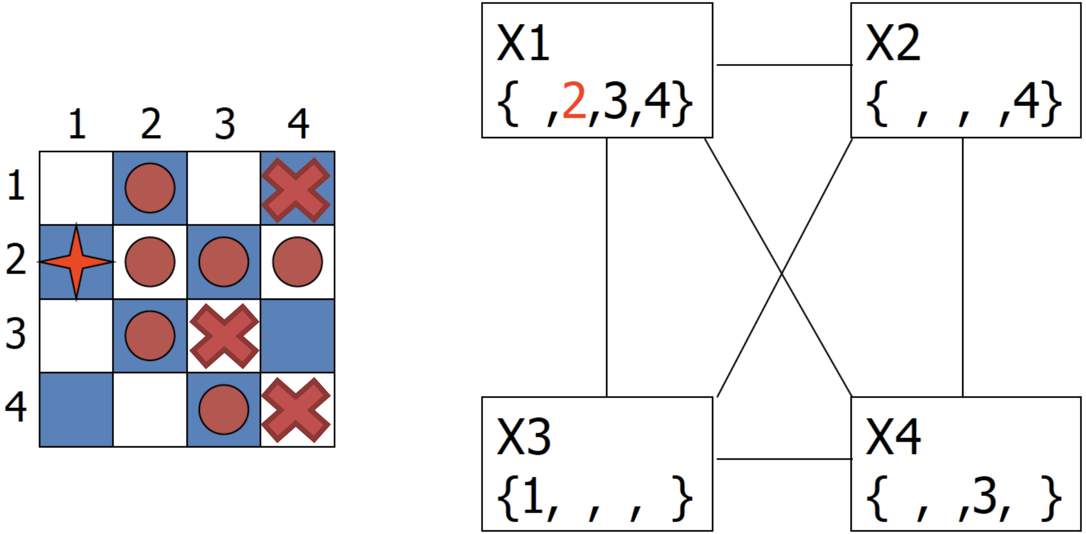
4-Queens Problem





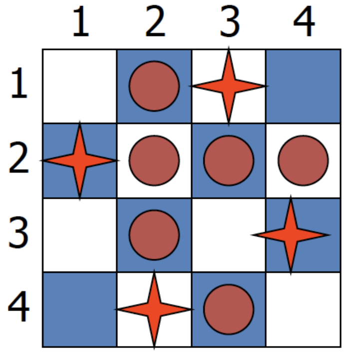
Can we eliminate any other values?

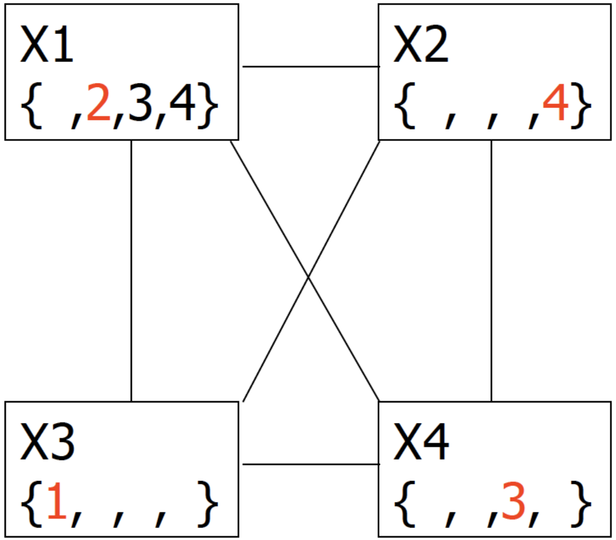
4-Queens Problem



Arc constancy eliminates x3=3 because it's not consistent with X2's remaining values

4-Queens Problem





There is only one solution with X1=2

Sudoku

- Digit placement puzzle on 9x9 grid with unique answer
- Given an initial partially filled grid, fill remaining squares with a digit between 1 and 9
- Each column, row, and nine 3×3 sub-grids must contain all nine digits

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

- Some initial configurations are easy to solve and others very difficult

Sudoku Example

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

initial problem

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

a solution

How can we set this up as a CSP?

UMBC

```python
def sudoku(initValue):
  p = Problem()
  # Define a variable for each cell: 11,12,13...21,22,23...98,99
  for i in range(1, 10) :
    p.addVariables(range(i*10+1, i*10+10), range(1, 10))
  # Each row has different values
  for i in range(1, 10) :
    p.addConstraint(AllDifferentConstraint(), range(i*10+1, i*10+10))
  # Each column has different values
  for i in range(1, 10) :
    p.addConstraint(AllDifferentConstraint(), range(10+i, 100+i, 10))
  # Each 3x3 box has different values
  p.addConstraint(AllDifferentConstraint(), [11,12,13,21,22,23,31,32,33])
  p.addConstraint(AllDifferentConstraint(), [41,42,43,51,52,53,61,62,63])
  p.addConstraint(AllDifferentConstraint(), [71,72,73,81,82,83,91,92,93])

  p.addConstraint(AllDifferentConstraint(), [14,15,16,24,25,26,34,35,36])
  p.addConstraint(AllDifferentConstraint(), [44,45,46,54,55,56,64,65,66])
  p.addConstraint(AllDifferentConstraint(), [74,75,76,84,85,86,94,95,96])

  p.addConstraint(AllDifferentConstraint(), [17,18,19,27,28,29,37,38,39])
  p.addConstraint(AllDifferentConstraint(), [47,48,49,57,58,59,67,68,69])
  p.addConstraint(AllDifferentConstraint(), [77,78,79,87,88,89,97,98,99])

  # add unary constraints for cells with initial non-zero values
  for i in range(1, 10) :
    for j in range(1, 10):
      value = initValue[i-1][j-1]
      if value:
        p.addConstraint(lambda var, val=value: var == val, (i*10+j,))
  return p.getSolution()
```

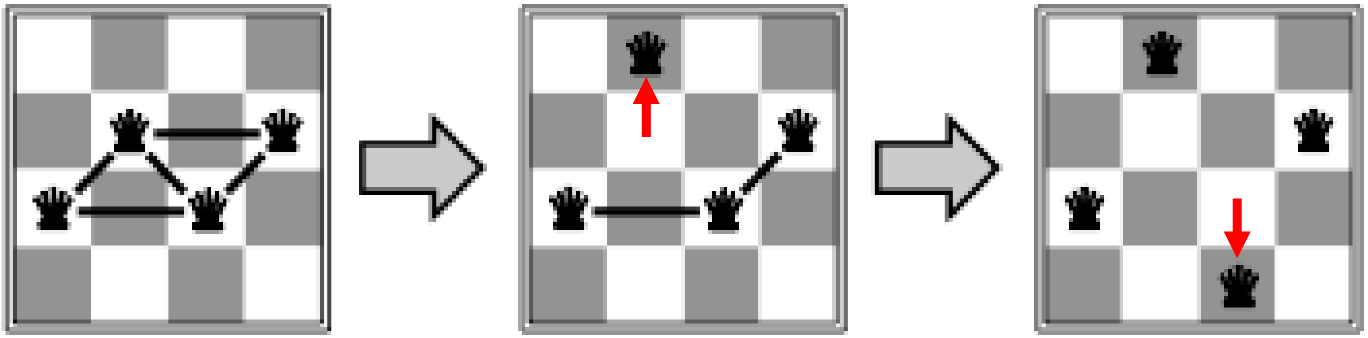
```python
# Sample problems
easy = [
 [0,9,0,7,0,0,8,6,0],
 [0,3,1,0,0,5,0,2,0],
 [8,0,6,0,0,0,0,0,0],
 [0,0,7,0,5,0,0,0,6],
 [0,0,0,3,0,7,0,0,0],
 [5,0,0,0,1,0,7,0,0],
 [0,0,0,0,0,0,1,0,9],
 [0,2,0,6,0,0,0,5,0],
 [0,5,4,0,0,8,0,7,0]]

hard = [
 [0,0,3,0,0,0,4,0,0],
 [0,0,0,0,7,0,0,0,0],
 [5,0,0,4,0,6,0,0,2],
 [0,0,4,0,0,0,8,0,0],
 [0,9,0,0,3,0,0,2,0],
 [0,0,7,0,0,0,5,0,0],
 [6,0,0,5,0,2,0,0,1],
 [0,0,0,0,9,0,0,0,0],
 [0,0,9,0,0,0,3,0,0]]

very_hard = [
 [0,0,0,0,0,0,0,0,0],
 [0,0,9,0,6,0,3,0,0],
 [0,7,0,3,0,4,0,9,0],
 [0,0,7,2,0,8,6,0,0],
 [0,4,0,0,0,0,0,7,0],
 [0,0,2,1,0,6,5,0,0],
 [0,1,0,9,0,5,0,4,0],
 [0,0,8,0,2,0,7,0,0],
 [0,0,0,0,0,0,0,0,0]]
```

Local search for constraint problems

- Basic idea:
  - generate a random “solution”
  - Use metric of “number of conflicts”
  - Modifying solution by reassigning one variable at a time to decrease metric until solution found or no modification improves it
- Has all features and problems of local search like….?

Min Conflict Example

- **States:** 4 Queens, 1 per column
- **Operators:** Move a queen in its column
- **Goal test:** No attacks
- **Evaluation metric:** Total number of attacks **(direct and indirect)**



How many conflicts does each state have?

Basic Local Search Algorithm

Assign one domain value d_i to each variable v_i

While no solution & not stuck & not timed out:

for each variable v_i where Cost(Value(v_i)) > =0:

bestCost $\leftarrow \infty$; bestList $\leftarrow$ [ ];

domain value d_i of v_i

if Cost(d_i) < bestCost

bestCost $\leftarrow$ Cost(d_i)

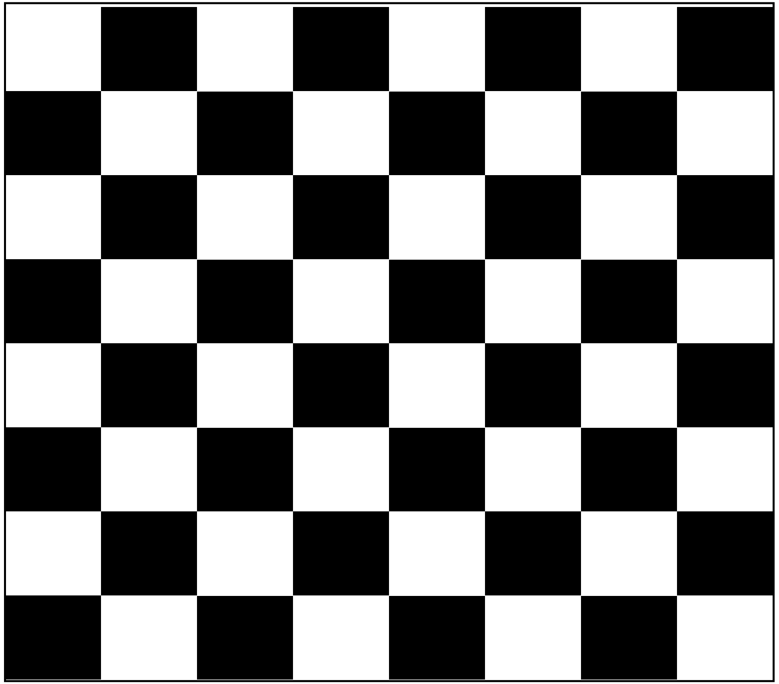
bestList $\leftarrow$ [d_i]

else if Cost(d_i) = bestCost

bestList $\leftarrow$ bestList $\cup$ d_i

Take a randomly selected move from bestList

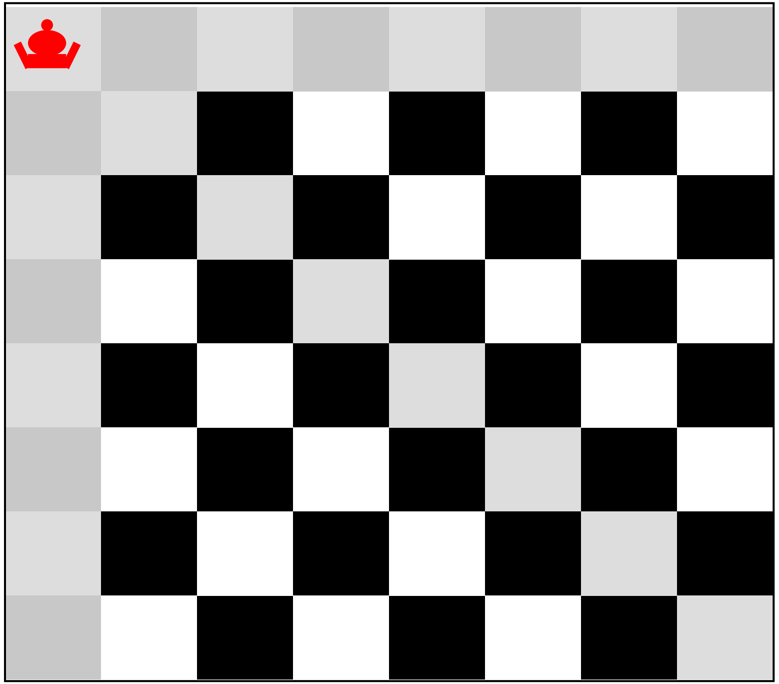
Eight Queens using Backtracking



Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

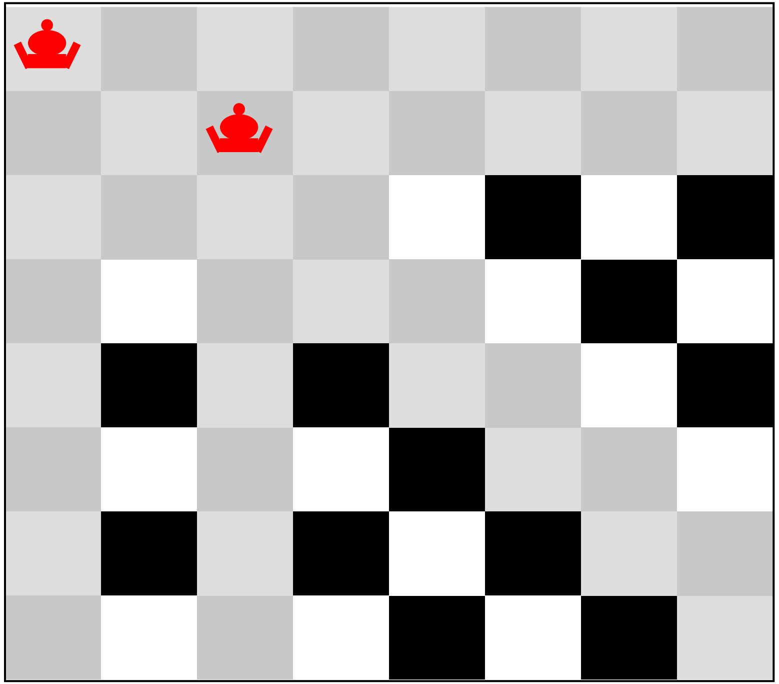
Try Queen 1



Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

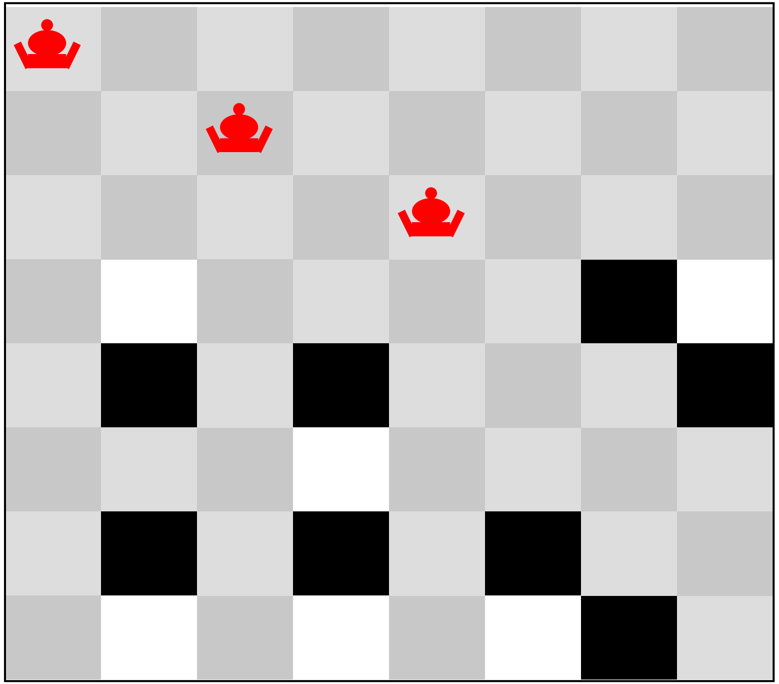
Try Queen 2



Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

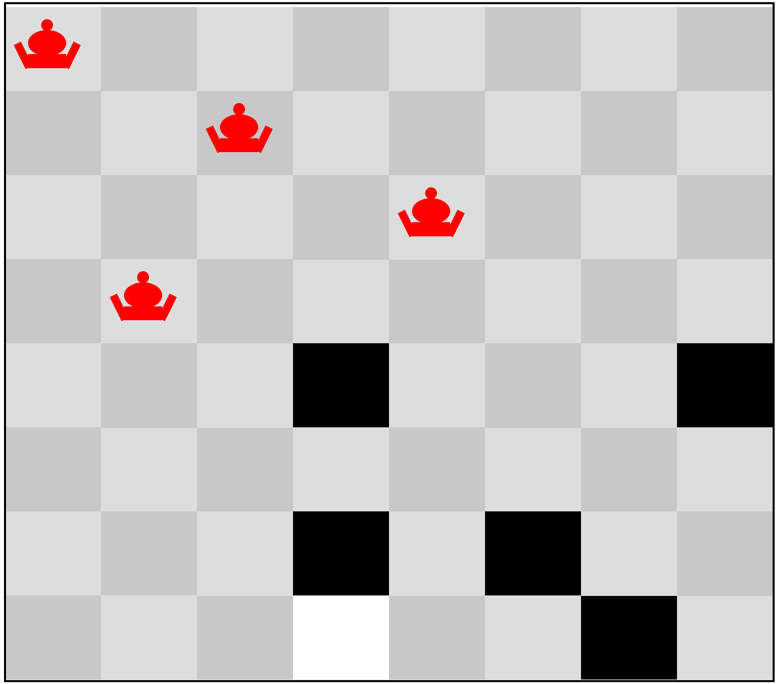
Try Queen 3



Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Try Queen 4

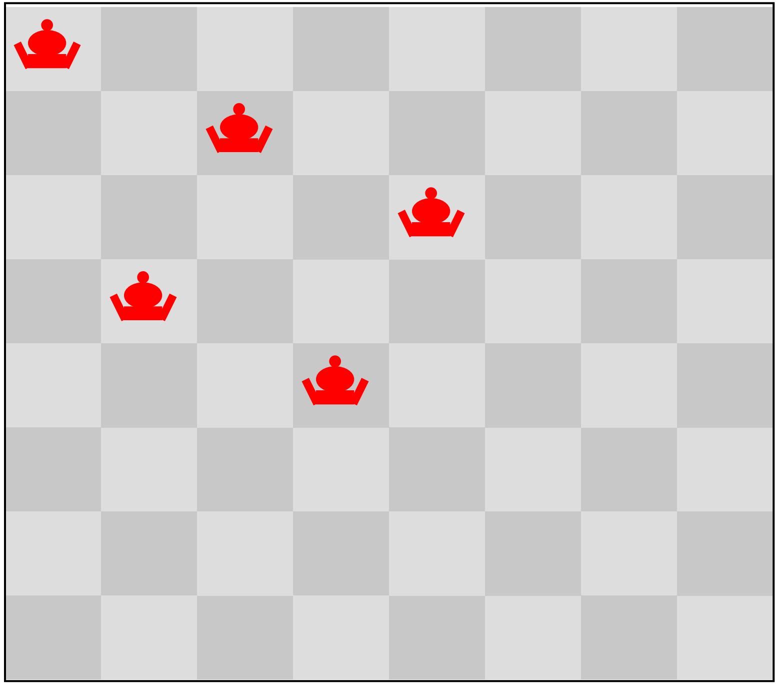


Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Try Queen 5

Stuck!

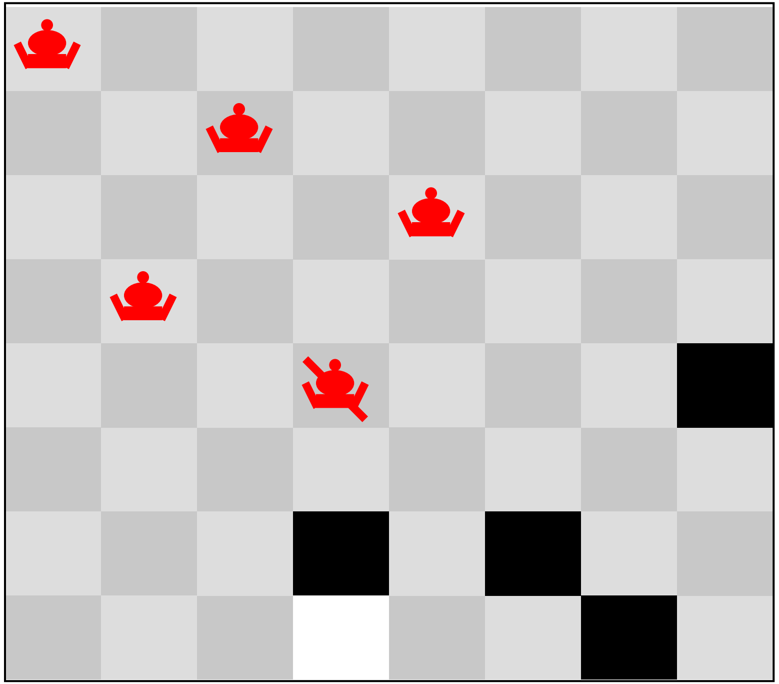


Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Undo move

for Queen 5



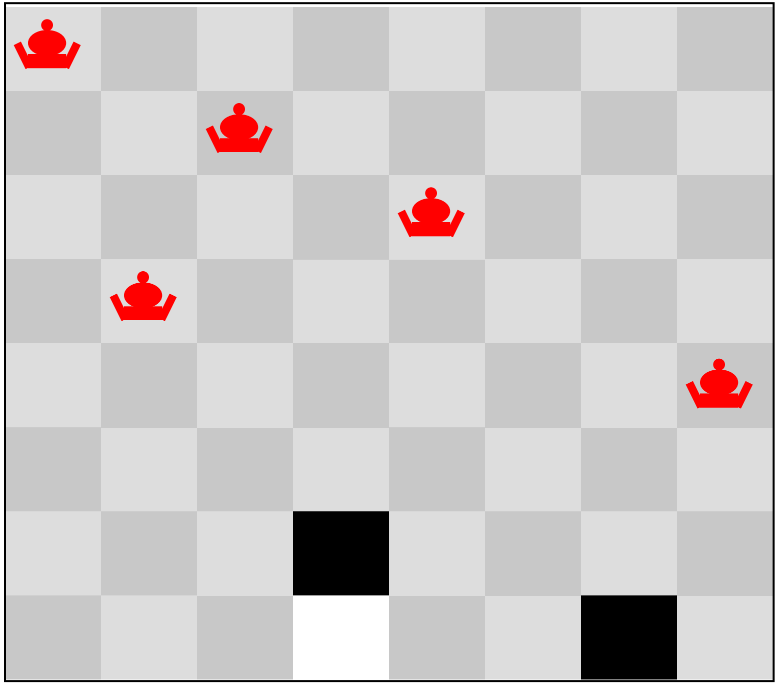
Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Try next value

for Queen 5

Still Stuck



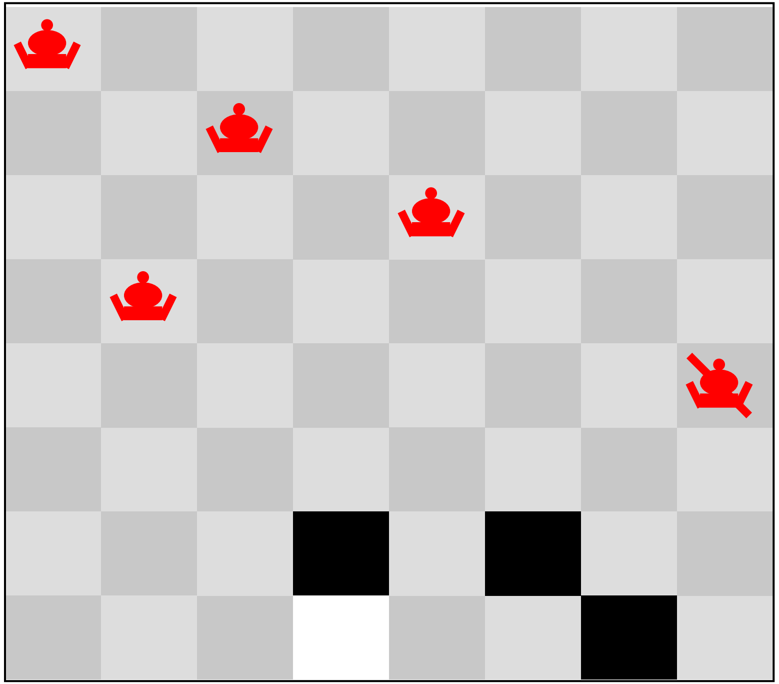
Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Undo move

for Queen 5

no move left



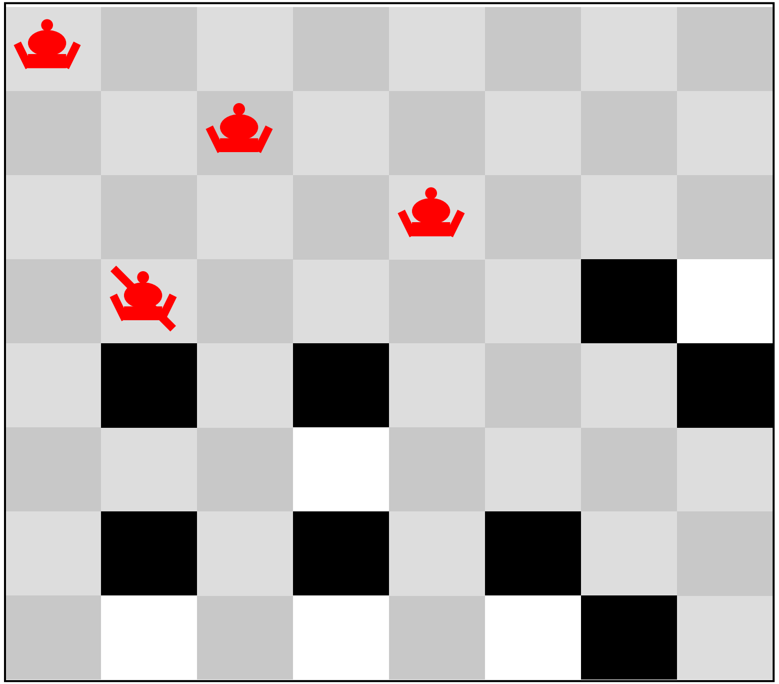
Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Backtrack and

undo last move

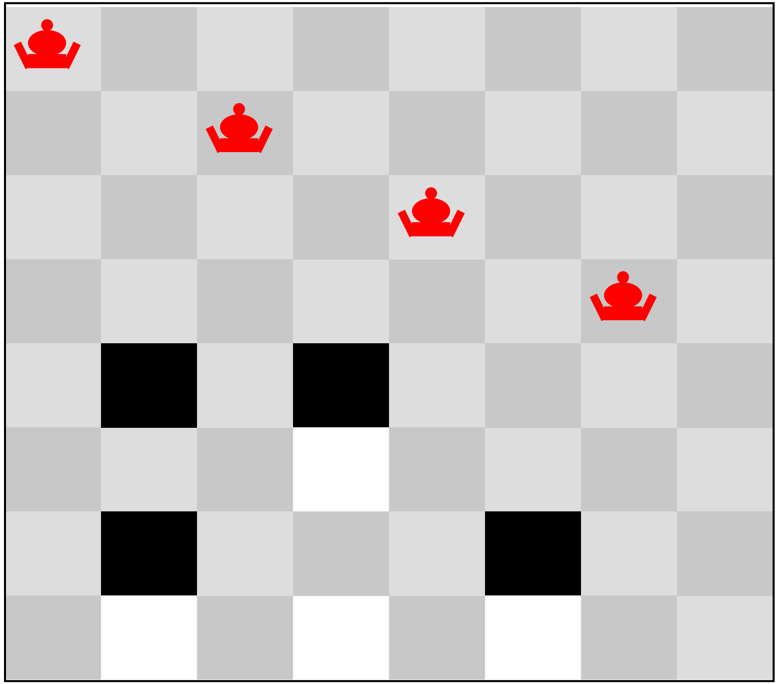
for Queen 4



Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

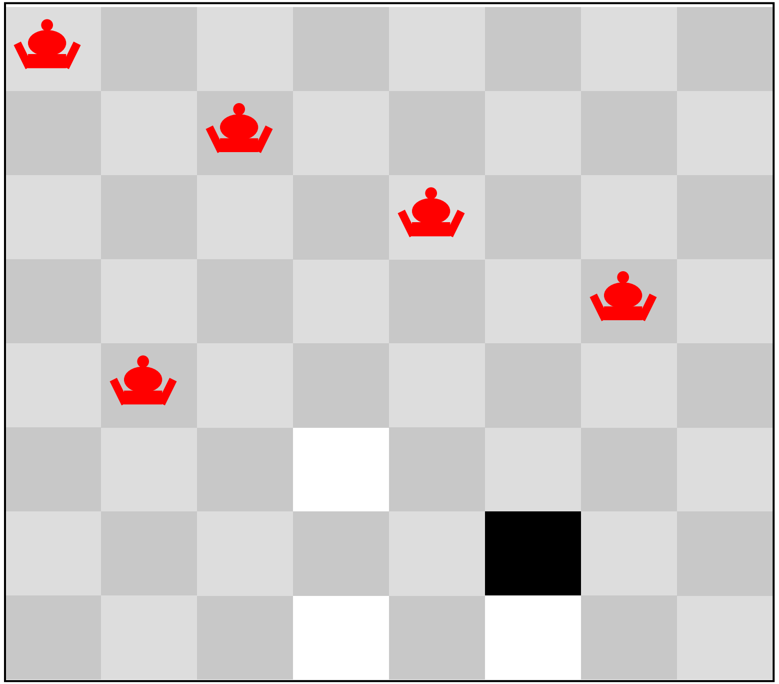
Try next value for Queen 4



Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

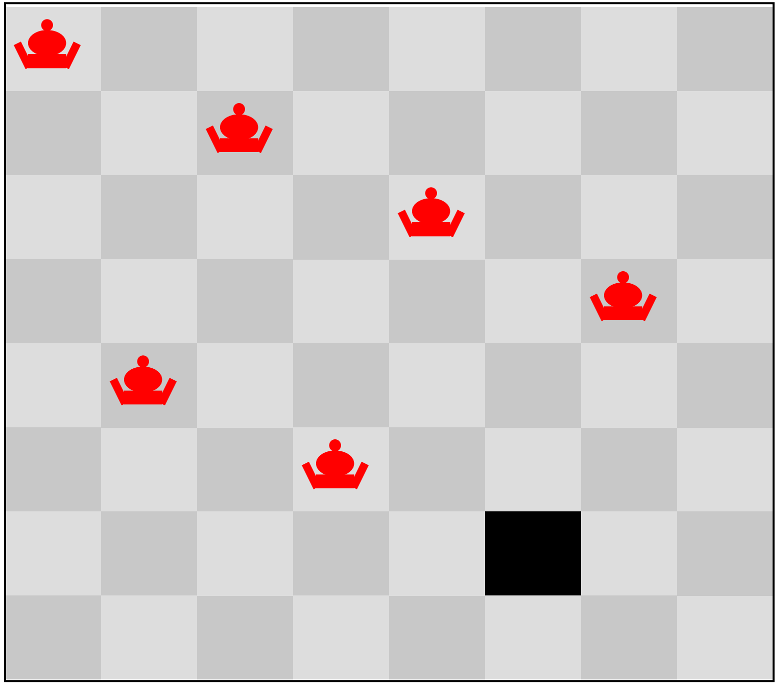
Try Queen 5



Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Try Queen 6

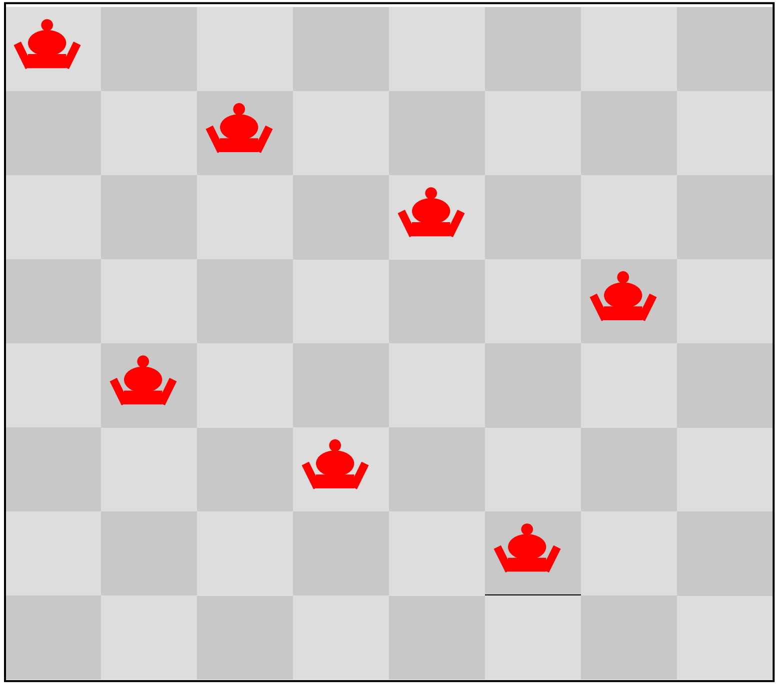


Note: in this example we put one queen in each row, not column

Eight Queens using Backtracking

Try Queen 7

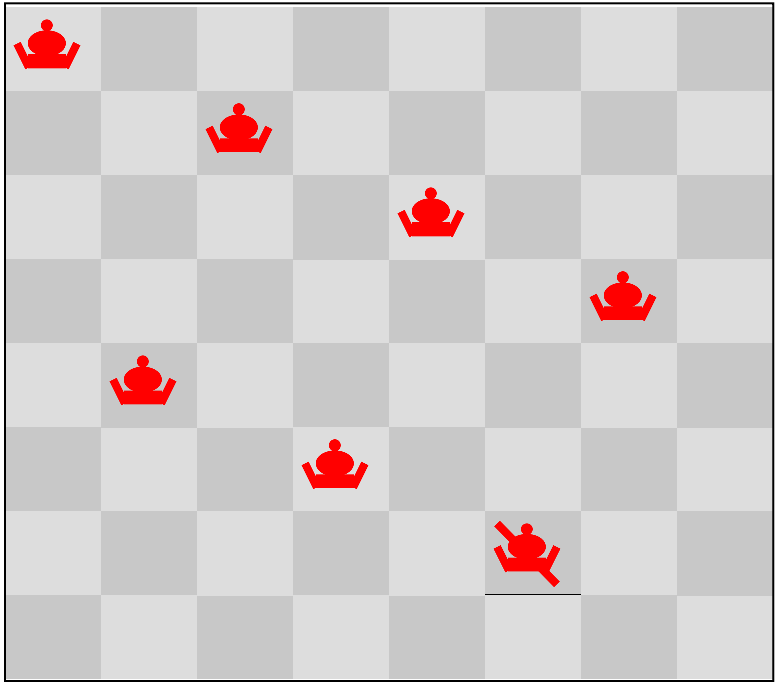
Stuck Again



Note: in this example we put one queen in each row, not column

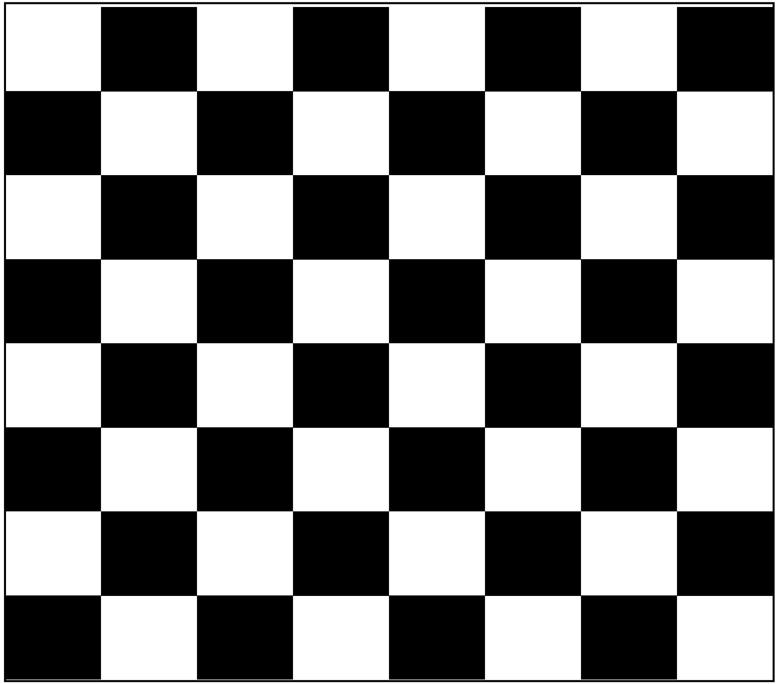
Eight Queens using Backtracking

Undo move
for Queen 7
and so on...



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search



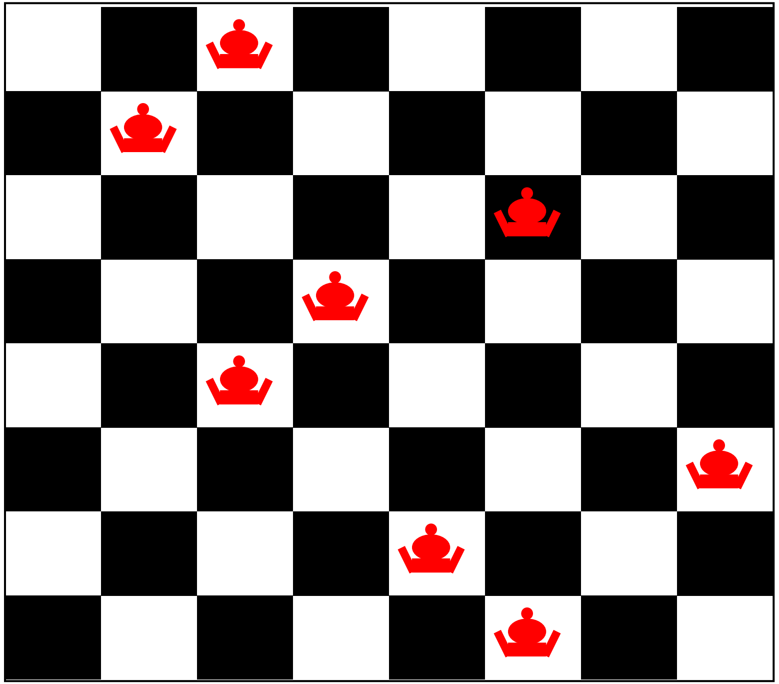
Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

Place 8 Queens

randomly on

the board

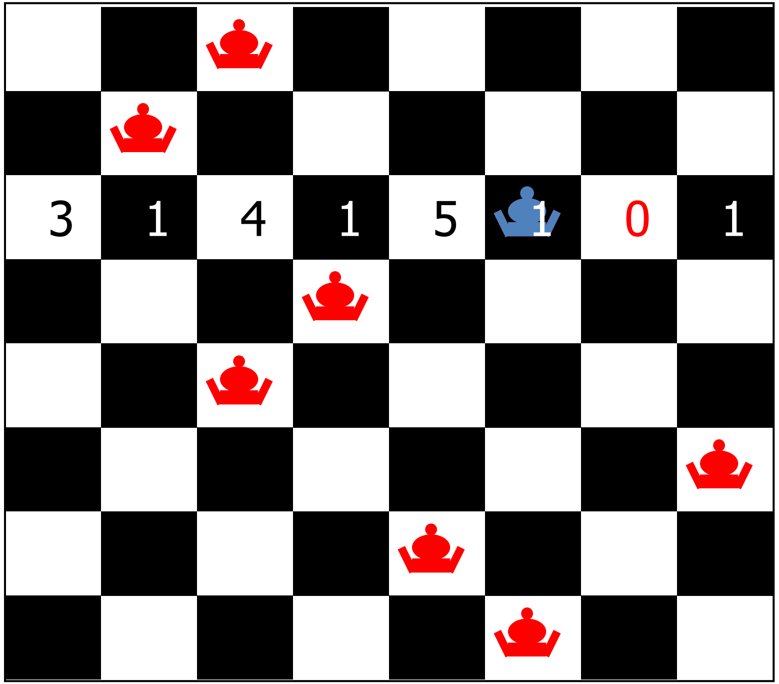


Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

Pick a Queen:

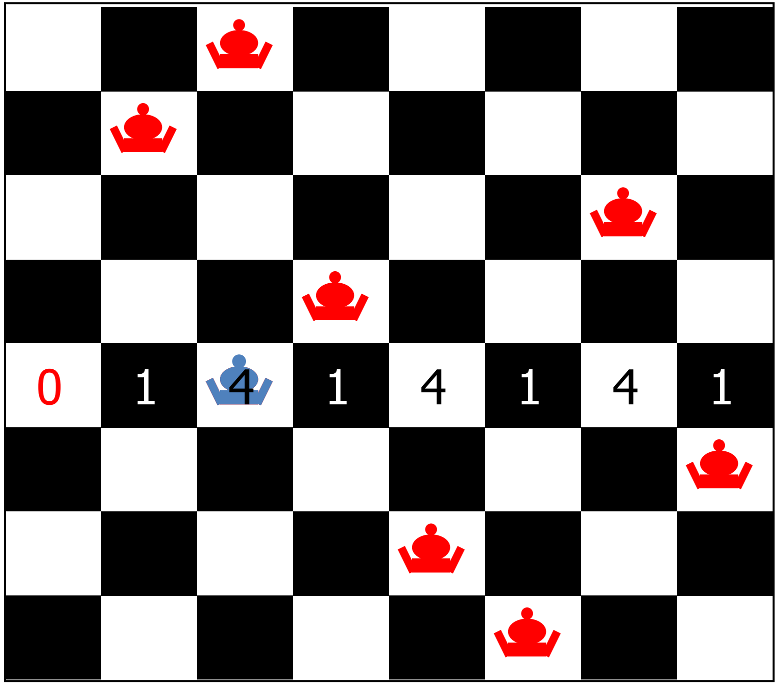
Calculate cost of each move



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

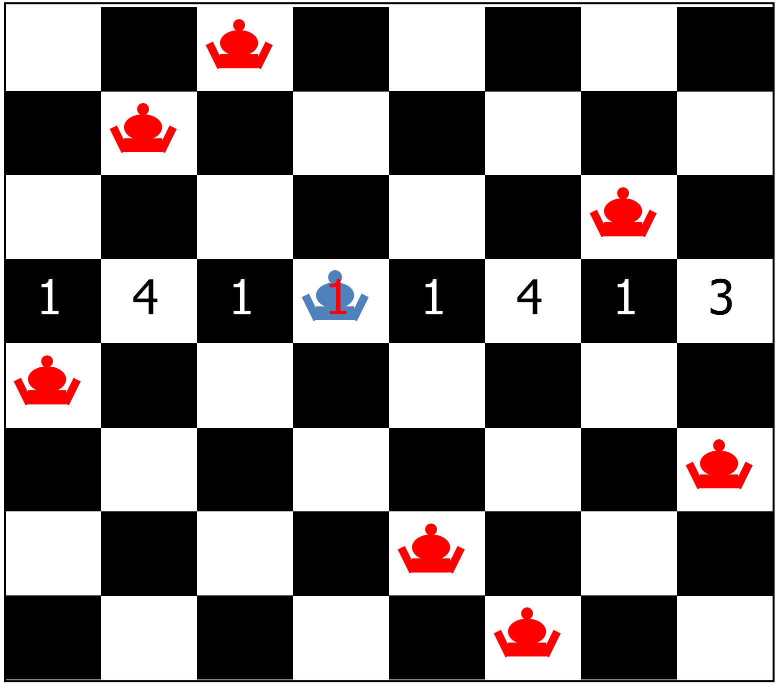
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

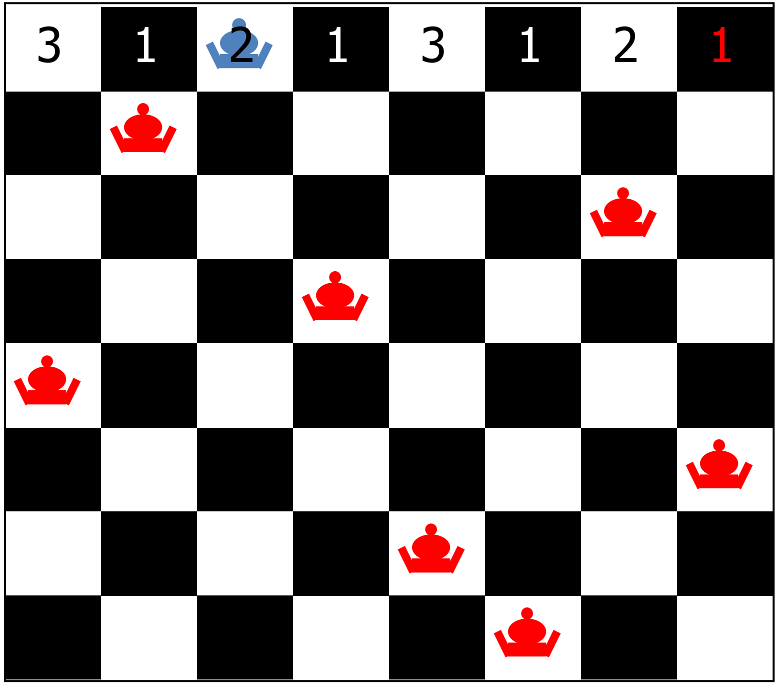
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

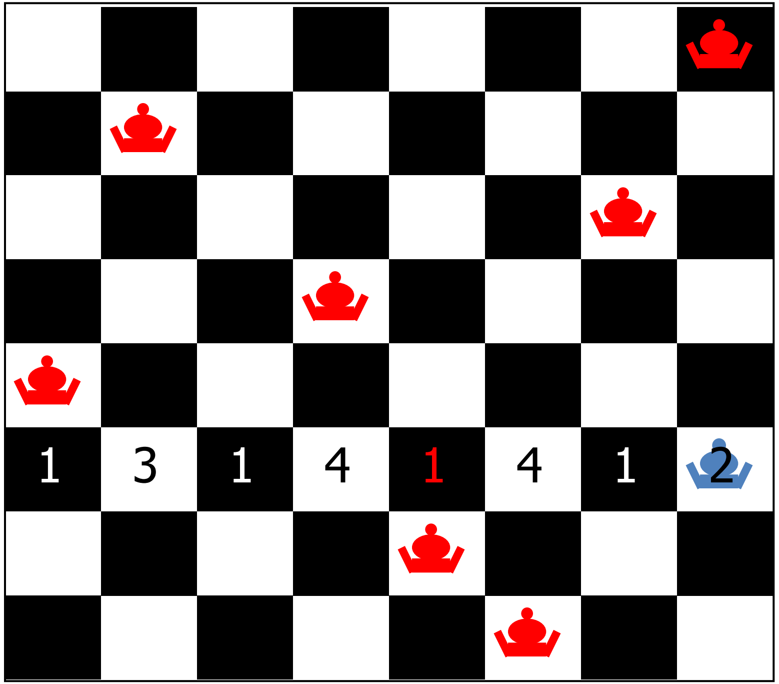
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

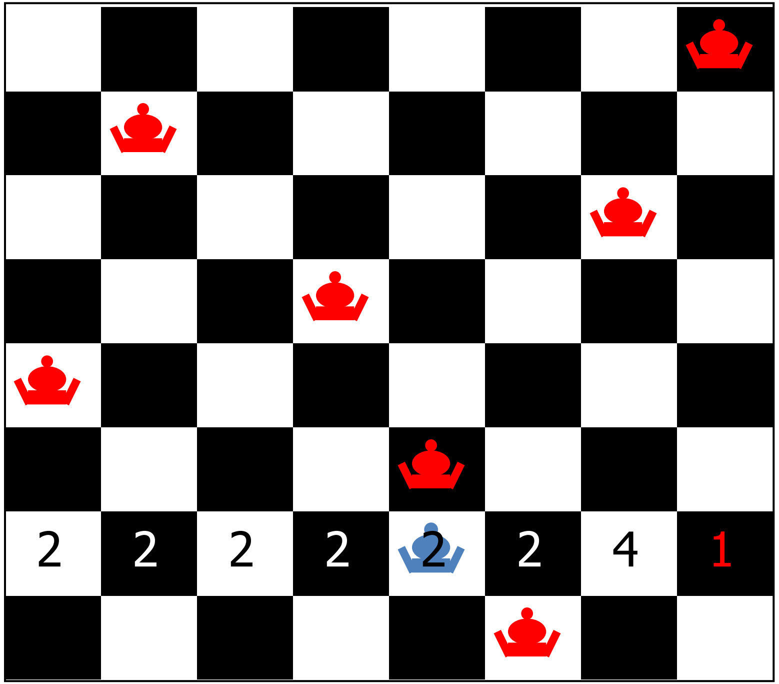
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

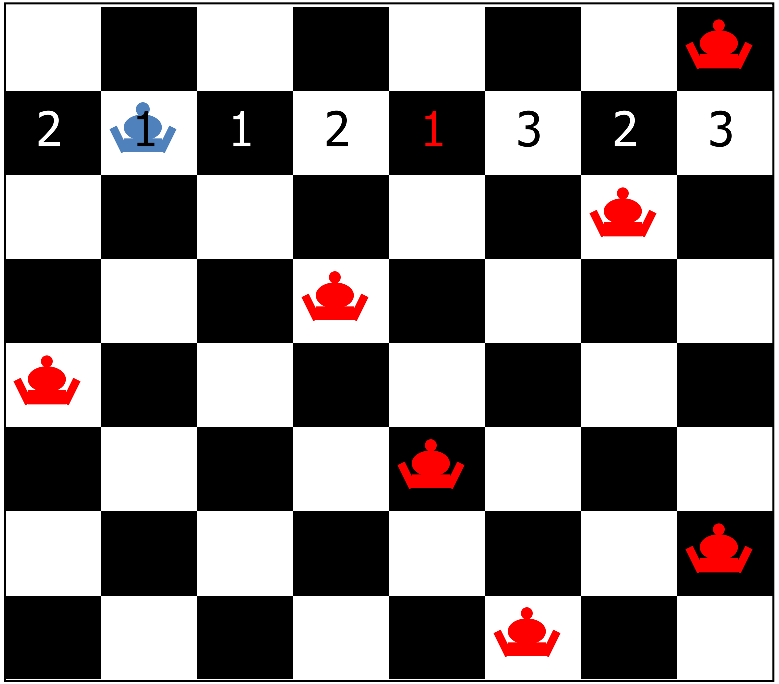
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

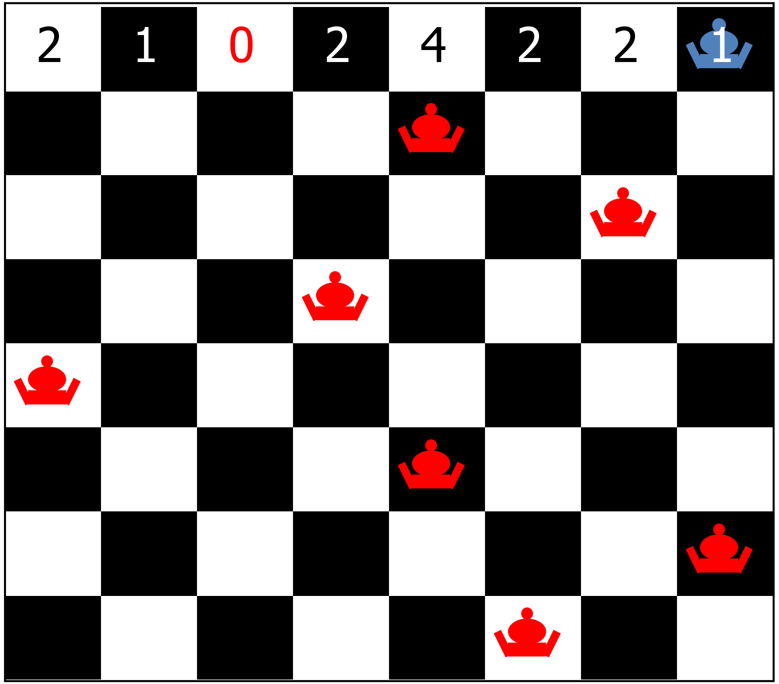
[Take least cost move then try another Queen]



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

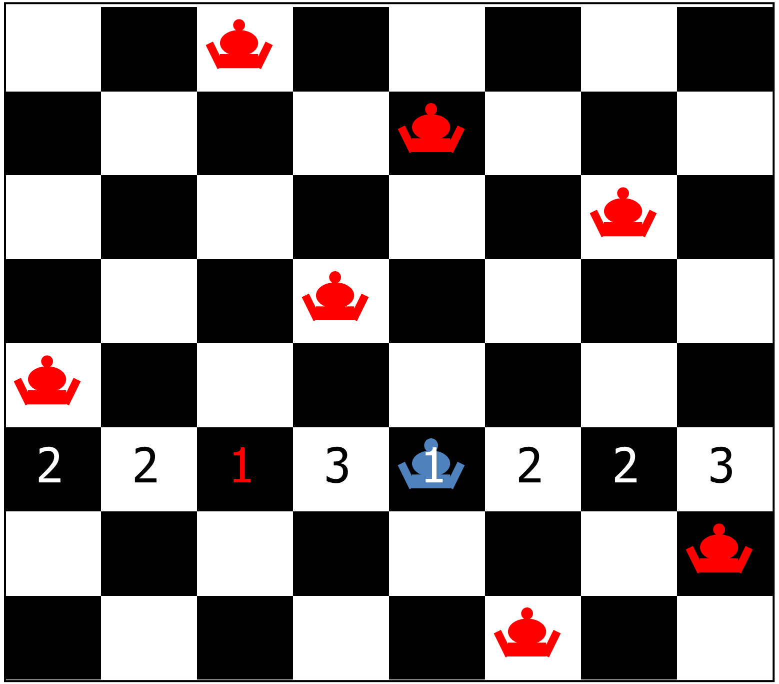
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

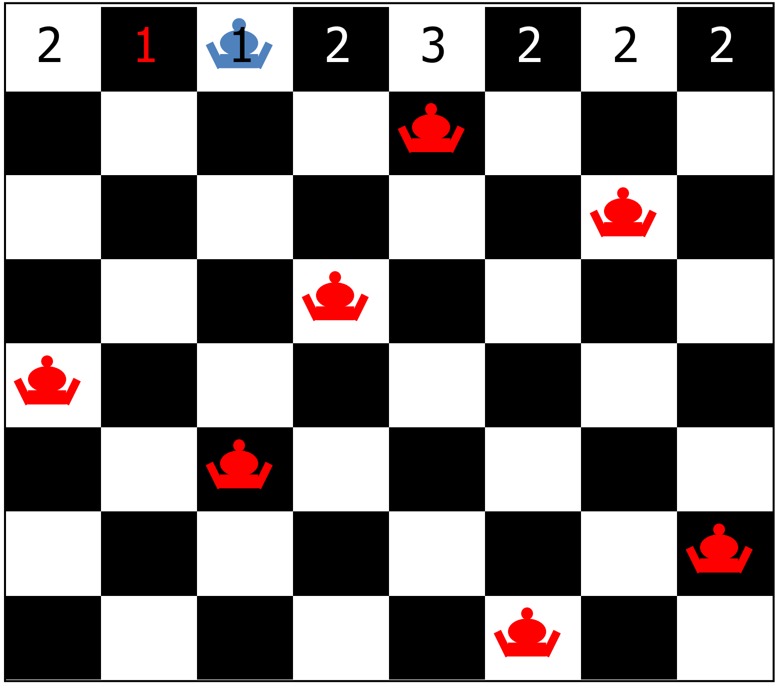
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

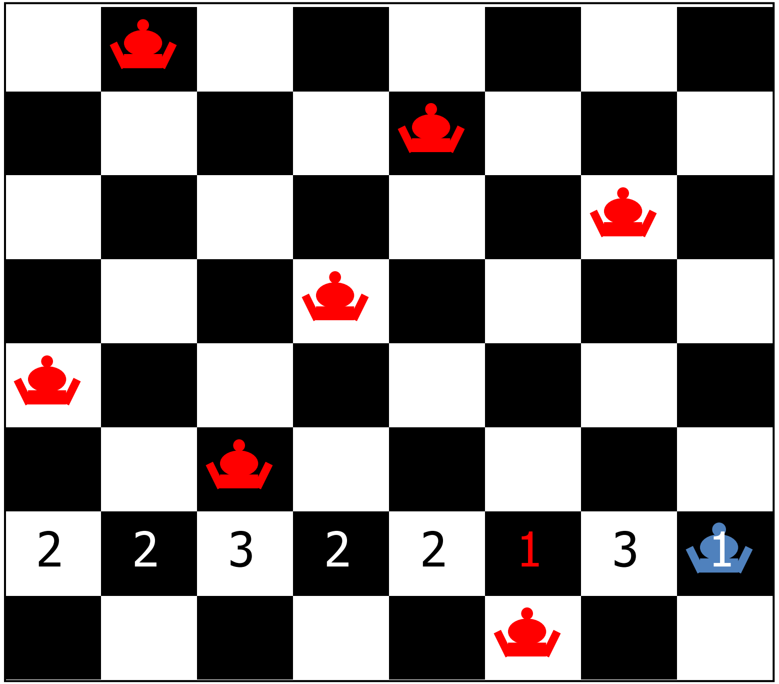
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

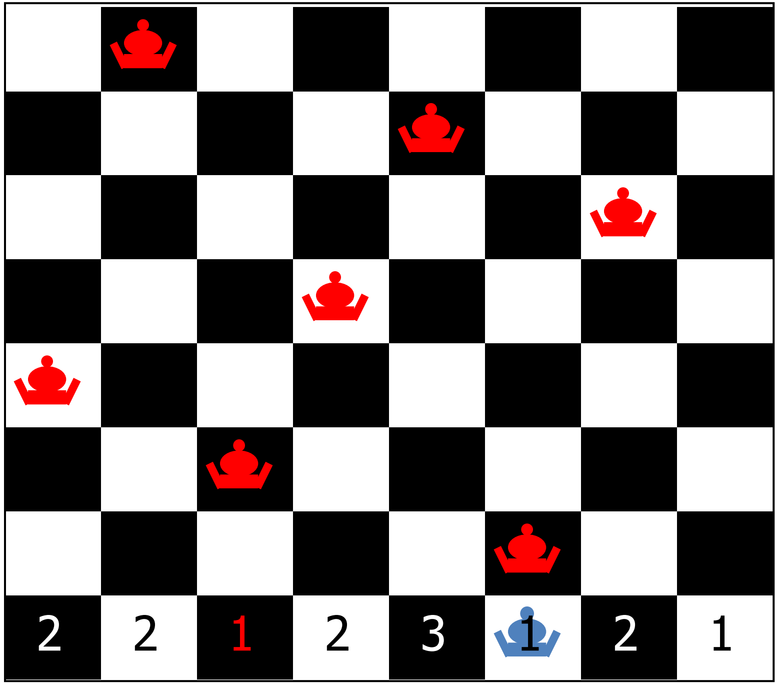
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

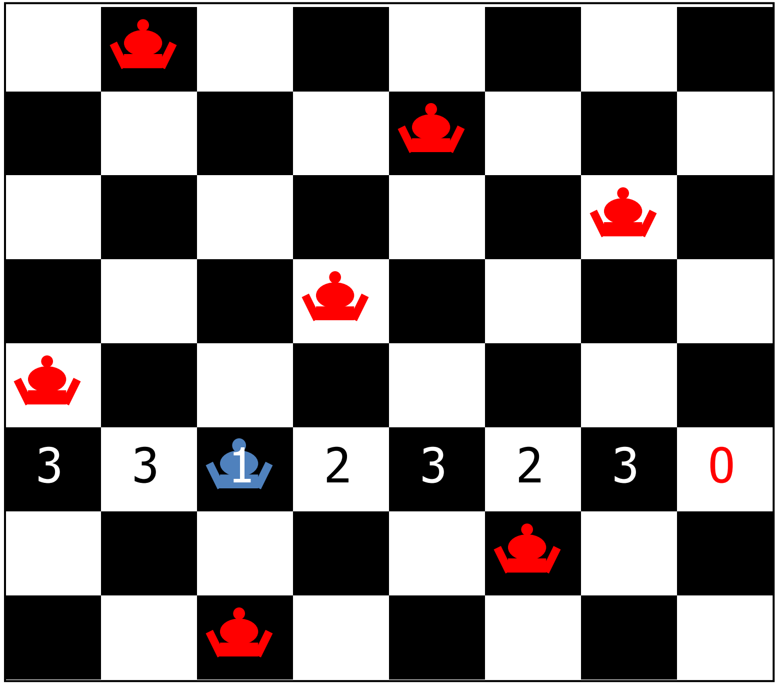
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

Eight Queens using Local Search

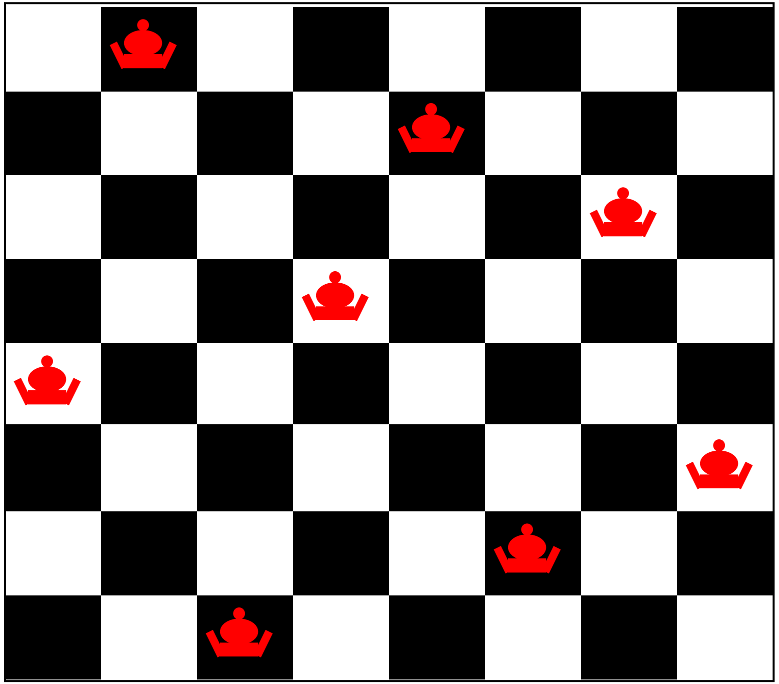
Take least cost move then try another Queen



Note: in this example we put one queen in each row, not column

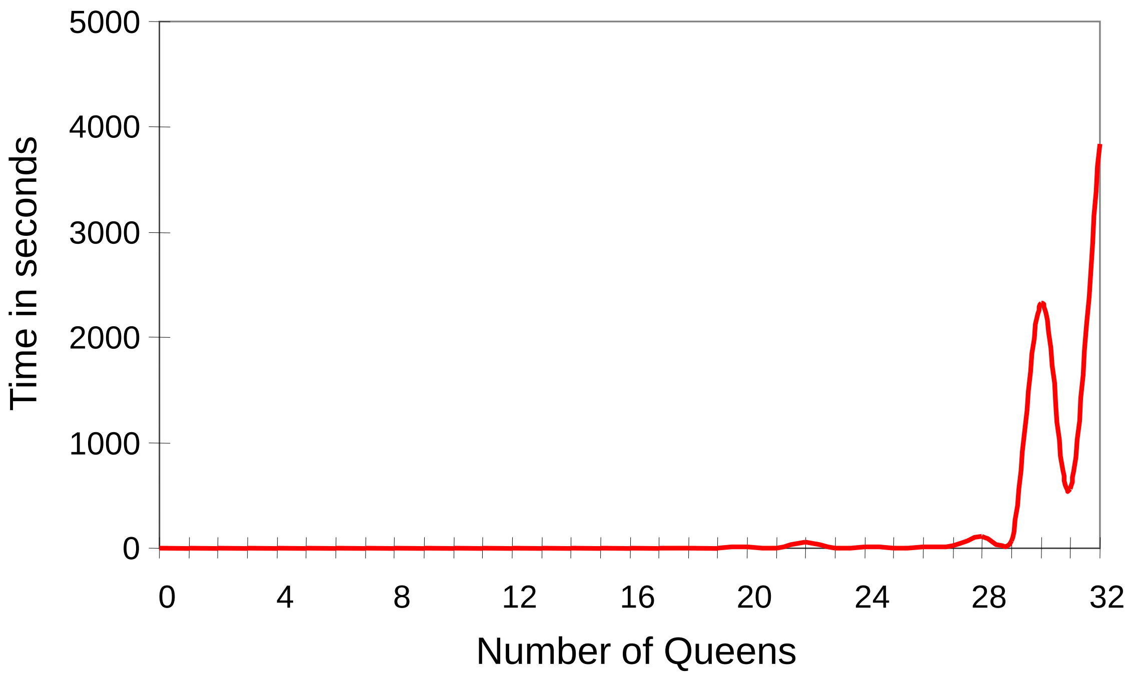
Eight Queens using Local Search

Answer Found

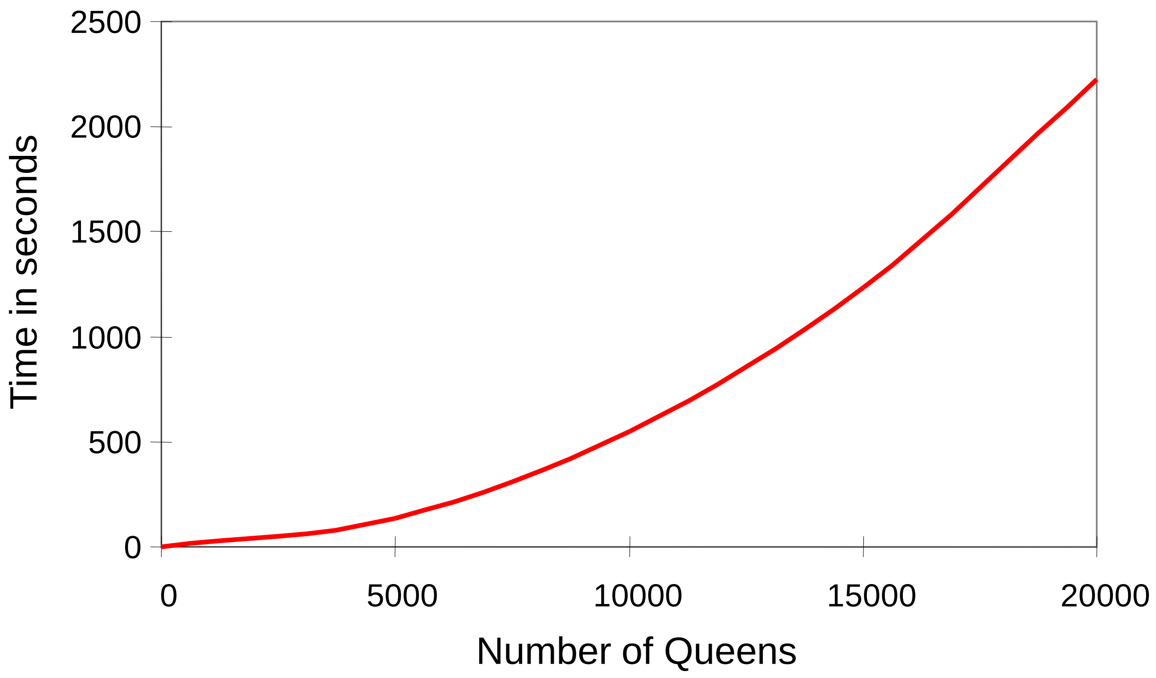


Note: in this example we put one queen in each row, not column

Backtracking Performance



Local Search Performance



Min Conflict Performance

- Performance depends on quality and informativeness of initial assignment; inversely related to distance to solution
- Min Conflict often has astounding performance
- Can solve arbitrary size (i.e., millions) N-Queens problems in constant time
- Appears to hold for arbitrary CSPs with the caveat…

Challenges for constraint reasoning

- What if not all constraints can be satisfied?
  - Hard vs. soft constraints vs. preferences
  - Degree of constraint satisfaction
  - Cost of violating constraints
- What if constraints are of different forms?
  - Symbolic constraints
  - Logical constraints
  - Numerical constraints [constraint solving]
  - Temporal constraints
  - Mixed constraints

Summary

- Many problems can be effectively modeled as constraints solving problems
- The approach is very good at reducing the amount of search needed
- Arc consistency is simple yet powerful
- Constraints are also useful for local search
- There’s a lot of complexity in many real-world problems that require additional ideas and tools