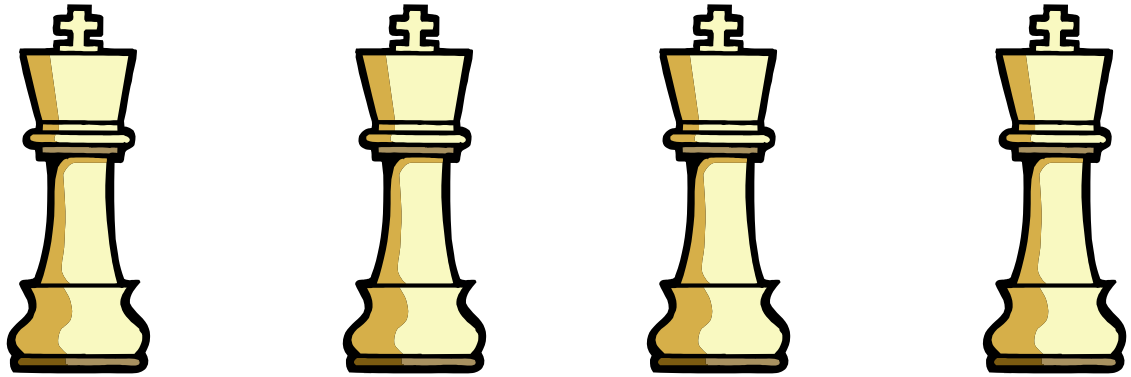
Constraint Satisfaction Problems

Fahiem Bacchus
Department of Computer Science
University of Toronto

What is it?

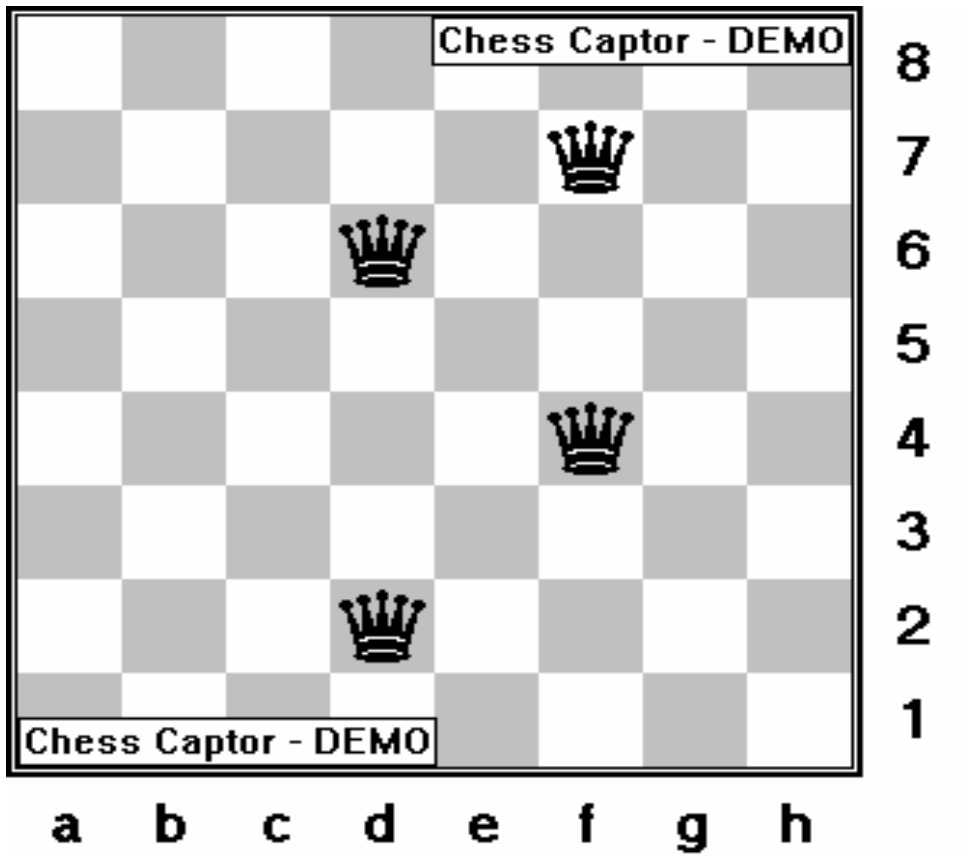
- What is a constraint satisfaction problem?
- We illustrate with a couple of examples:
  - The N-Queens puzzle.
  - Golomb Rulers.



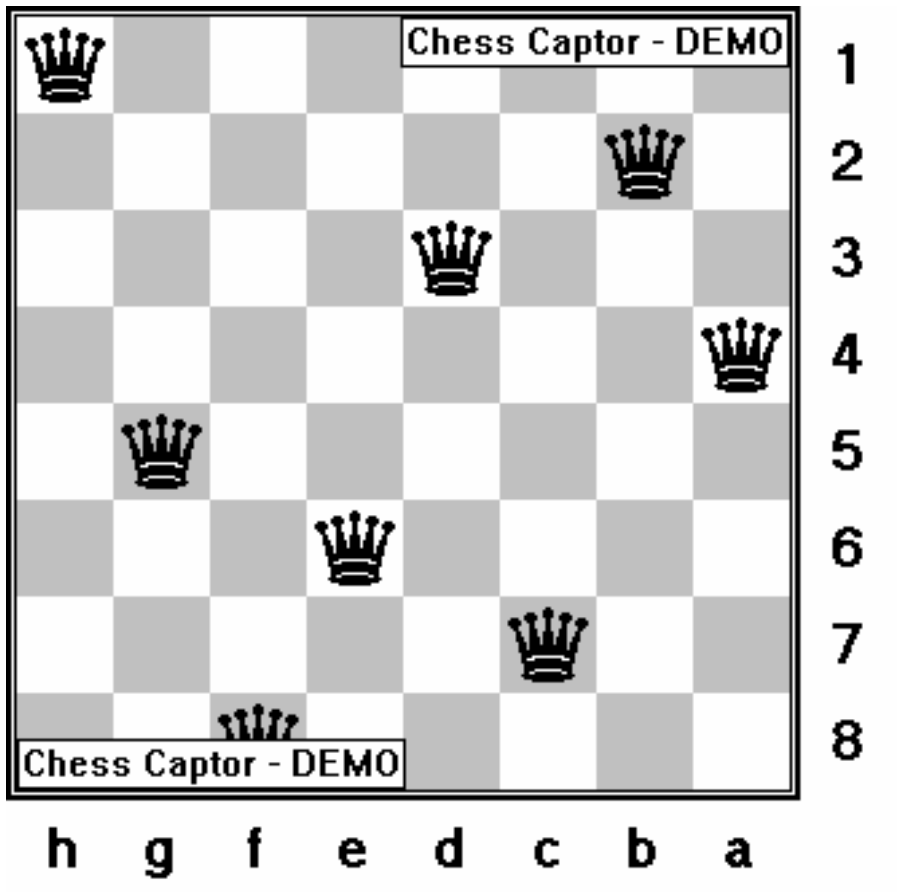
N-Queens

- Place N queens on an NxN chess board so that queen can attack any other queen.

Queen Attacks



Solution to 8-Queens



8-Queens

- There are 92 distinct solutions to the 8-Queens problem.

How do we solve N-Queens

- Humans solve this problem by experimenting with different configurations.
- They use various insights about the problem to explore only a small number of configurations before they find an answer.
- Problem is that it is unclear exactly what these insights are. Furthermore, people would find it hard to solve a 1000 Queen problem!

Generate and Test

- Computer are good at doing a large number of simple things quickly.
- So one possible solution is to systematically try every placement of queens until we find a solution.

Generate and Test…

Q			
Q			
Q			
Q			

	Q		
Q			
Q			
Q			

		Q	
Q			
Q			
Q			

			Q
Q			
Q			
Q			

Q			
	Q		
Q			
Q			

	Q		
	Q		
Q			
Q			

		Q	
	Q		
Q			
Q			

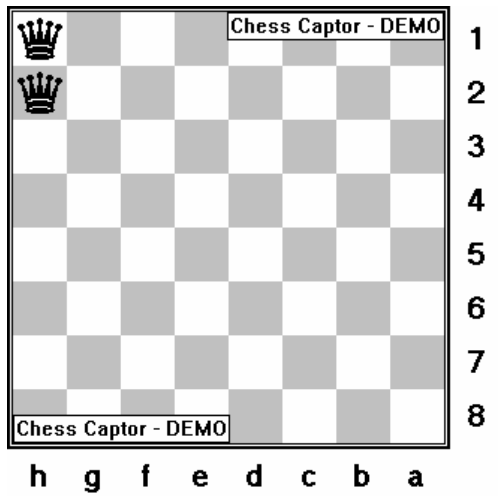
…

Problem

- For 4-Queens there are 256 different configurations.
- For 8-Queens there are 16,777,216 configurations.
- For 16-Queens there are 18,446,744,073,709,551,616 configurations.
- This would take about 12,000 years on a fast modern machine.
- In general we have N^N configurations for N-Queens.

Backtracking

- One thing that we can notice is that in, e.g., the 8-Queens problem, as soon as we place some of the queens we know that an entire additional set of configurations are invalid:



Backtracking

- Backtracking is one of the main methods for solving problems like N-Queens.
- But we don’t want to create an algorithm just for solving N-Queens!
- We need to express N-Queens as an instance of a general class of problems and then design algorithms for solving this general class of problems.

CSP

- We can represent the N-queens as a constraint satisfaction problem.
- A Constraint Satisfaction Problem consists of 3 components

1. A set of **variables**.
2. A set of **values** for each of the variables.
3. A set of constraints between various collections of variables.

We must find a value for each of the variables that satisfies all of the constraints.

Constraints

- A constraint is a relation between a **local** collection of variables.
- The constraint restricts the values that these variables can simultaneously have.
- For example, **all-diff(X1,X2,X3)**. This constraint says that X1, X2, and X3 must take on different values.
  - Say that {1,2,3} is the set of values for each of these variables then:
    - X1=1, X2=2, X3=3 OK X1=1,X2=1,X3=3 NO

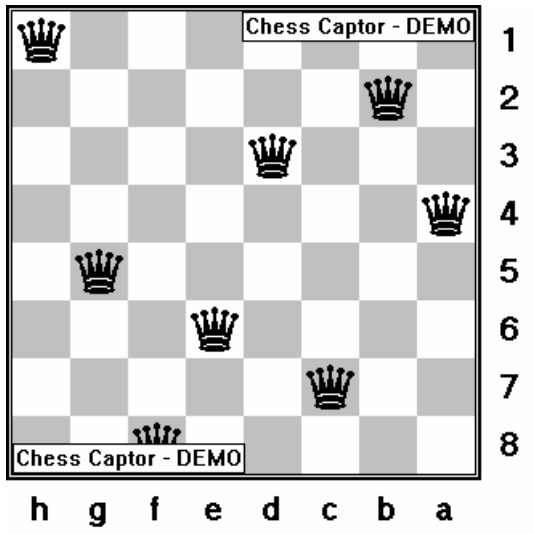
Finding A Solution

- Although each constraint is over a local collection of variables, finding a global assignment to all of the variables that satisfies **all** of the constraints is hard: NP-Complete.
- The solution techniques work by cleverly searching through the space of possible assignments of values to variables.
- If each variable has **d** values and there are **n** variables we have $\mathbf{d^n}$ possible assignments.

N-Queens as a CSP

- We need to know where to place each of the N queens. So we could have N variables each of which has as a value 1…N^2. The values represent where on the chessboard we will place the i-th variable.

N-Queens as a CSP



- Q1 = 1, Q2 = 15, Q3 = 21, Q4 = 32, Q5 = 34, Q6 = 44, Q7 = 54, Q8 = 59

N-Queens as a CSP

- This representation has

$$64^8 = 281,474,976,710,656$$

Different possible assignments in the search space.

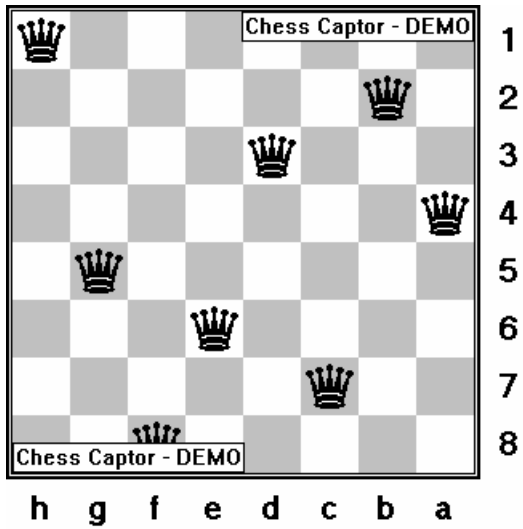
N-Queens as a CSP

- However any particular problem can be represented as a CSP in a number of different ways.
- In this case we know that we can never place two queens in the same column.
- So we can configure the problem as one where we assign one queen to each of the columns, and now we need to find out only which row each of these queens is to placed in.

N-Queens as a CSP

- So we can have N variables: Q1, …, QN.
- The set of values for each of these variables will be {1, 2, …, N}.

N-Queens as a CSP



- Q1 = 1, Q2 = 5, Q3 = 8, Q4 = 6, Q5 = 3, Q6 = 7, Q7 = 2, Q8 = 4

N-Queens as a CSP

- This representation has

  $8^8 = 16{,}777{,}216$

  Different possible assignments in the search space.
- Still too large to examine all of them, but a big improvement.

The Constraints

- The constraints are the key component in expressing a problem as a CSP.
- The constraints are determined by how the variables and the set of values are chosen.
- Each constraint consists of
  1. A set of variables it is over.
  2. A specification of the sets of assignments to those variables that satisfy the constraint.

Constraints

- The idea is that we break the problem up into a set of distinct conditions each of which have to be satisfied for the problem to be solved.
- In N-Queens:
  - No queen can attack any other queen.
    - Given any two queens Qi and Qj they cannot attack each other.

Constraints

- Now we translate each of these individual conditions into a separate constraint.
  - Qi cannot attack Qj ($i \neq j$)
    - Qi is a queen to be placed in column i, Qj is a queen to be placed in column j.
    - The value of Qi and Qj are the rows the queens are to be placed in.
- Note the translation is dependent on the representation we chose.

Constraints

- Queens can attack each other
  1. Vertically, if they are in the same column---this is impossible as Qi and Qj are placed in different columns.
  2. Horizontally, if they are in the same row---we need the constraint Qi ≠ Qj.
  3. Along a diagonal---they cannot be the same number of columns apart as they are rows apart: we need the constraint
     |i-j| ≠ |Qi-Qj| (|.| is absolute value)

Representing the Constraints

1. Between every pair of variables (Qi,Qj) (i ≠ j), we have a constraint Cij.
2. For each Cij, an assignment of values to the variables Qi = A and Qj = B, satisfies this constraint if and only if
    1. A ≠ B
    2. |A-B| ≠ |i-j|

Solutions

- A solution to the N-Queens problem will be any assignment of values to the variables Q1,…,QN that satisfies all of the constraints.
- Constraints can be over any collection of variables. In N-Queens we only need binary constraints---constraints over pairs of variables.

Another Problem---Golomb Rulers

- This problem has various practical applications, e.g., sensor placement in radio astronomy and in x-ray crystallography.
- We have a ruler of length L units. We can place marks along this ruler at any unit interval.
- For example, if L = 7, we can place a mark at any of the positions 0,1, 2, 3, 4, 5, 6, 7. But we cannot place a mark at position 1.5.

Golomb Rulers

- We want to place M marks (m1,m2,…mM) on the ruler, such that all of the M(M-1)/2 differences mi-mj are distinct.
- The objective is to find the minimal length ruler such that the M marks will all have distinct differences.

Golomb Rulers

- For example, for 5 marks, the optimal (shortest) ruler has length 11, and the marks are placed at
  - 0 1 4 9 11
    - e.g., 11-4 ≠ 9-1
- An optimal ruler for 23 marks has length 372. The optimal ruler for 24 marks is not known.

Expressing Golomb Rulers as CSPs

- We can represent this problem as a CSP. However, the CSP will only tell us whether or not a ruler of a fixed length L for M marks exists.
- To find the optimal length we must successively decrease L until we find that the CSP has no solution.

Golomb Rulers

- Variables m1,…mM one variable for each mark.
- Each variable has the domain of values {0,1,…,L-1}. If we assign, e.g., m1 = 4, this means that we place mark 1 at position 4 along the ruler.
- We have a constraint between every collection of 4 variables, {mi, mj, mk, mn} such that they form two distinct pairs: (mi,mj) ≠ (mk,mn).

Golomb Rulers

- The constraint between (mi,mj,mk,mn) is that
  - $|mi-mj| \neq |mk-mn|$

Solving CSPs

- As we saw before we can improve over simply enumerating and testing all possible assignments by recognizing that a subset of the variables can already make a solution impossible.
- By expressing the problem as a CSP we have a systematic way of achieving this extra efficiency.

Generic Backtracking

- Generic Backtracking is the simplest and oldest algorithm for solving CSP problems.
- The idea is to search in a tree of variable assignments, as we move down the tree we assign a value to a new variable.
- Once we have assigned all of the variables that participate in a constraint, we check that constraint.
- At any point if a constraint is violated we backtrack up the tree.

BT

```
BT(int level)
   if(all variables assigned)
        PRINT value of each varible;
        exit(1);
   V := PickUnAssignedVariable();
   Assigned[V] := TRUE;
   for d := each member of Domain(V)
        Value[V] := d;
        OK := TRUE;
        for each constraint C such that V is a variable of C
                                  and all other variables of C
                                  are assigned.
                if C is NOT satisfied by the current assignments
                        OK := FALSE;
        if(OK)
                BT(level+1);
   return;
```

BT

- `PickUnAssignedVariable`---simply returns one of the unassigned variables. The choice of which variable to assign next can be critical.
- Example: 4 Queens.

BT Performance

Finding a single solution.

```
26 Time =       26.84 sec
27 Time =       32.3 sec.
28 Time =      234.8 sec.
29 Time =      125.5 sec.
```

Forward Checking

- The idea of searching in a tree of variable assignments is very powerful. However generic backtracking is not a very good algorithm.
- (Note that although BT is much faster than simple enumeration all algorithms for solving CSPs take time that can grow exponentially with the size of the problem.)

Forward Checking

- Forward Checking is based on the idea of looking ahead in the tree to see if we have already made assigning a value to one of the unassigned variable impossible.
- It is based on the idea of pruning the domains of the unassigned variables.

FC

```
Initially CurrDomain[V] = Domain[V] for all variables V;


FC(int level)
   if(all variables assigned)
        PRINT value of each varible;
        exit(1);
   V := PickUnAssignedVariable();
   Assigned[V] := TRUE;
   for d := each member of CurrDomain(V)
        Value[V] := d;
        for each constraint C such that
                1. V is a variable of C
                2. C has an unassigned variable V'.
                3. All other variables of C are assigned.
           DWO := FCCheck(C,V,V',level);
        if(DWO == FALSE)
           FC(level+1);
        Restore(level);
   return;
```

FC

```
FCCheck(Constraint C, Variable V, Variable V', int level)
   //We have just assigned a value to V.
   //We need to check every value the CurrDomain(V')
   //to see if it is compatiable.

   for d := each member of CurrDomain(V')
       Value[V'] := d;
       if C is not satisfied by the current set of assignments
              CurrDomain(V') := CurrDomain(V') - {d};
              remember that d was pruned from the domain of V'
                     at level
   if CurrDomain(V') is empty
       return TRUE;
   return FALSE;
```

FC

```
Restore(int level)
   for all d,V such that d was pruned from the domain of V
       at level
       CurrDomain(V) := CurrDomain(V) U {d};
```

FC

- Example 4 Queens.

FC Performance

Finding a single solution.

```
26 Time = 0.58 sec.
27 Time = 0.68 sec.
28 Time = 4.78 sec.
29 Time = 2.39 sec.
30 Time = 89.69 sec.
31 Time = 22.01 sec.
32 Time = 143.2 sec.
33 Time = 240.9 sec.
```

Variable Ordering

- Remember I said that the variable choosen by `PickUnAssignedVariable` is critical to performance.
- If we always chose as the next variable the variable with smallest `CurrDomain` we get a tremendous improvement in performance.

FC Variable Ordering Performance

Finding a single solution.

```
  33 Time  = 0.00 sec.
  100 Time = 0.02 sec.
  101 Time = 4.79 sec.
  102 Time = 0.01 sec.
```

Then 104 Queens is hard to solve.

Work in CSPs

- Algorithms that use randomization have been able to solve 6,000,000 Queens (but randomization does not always work).
- In practice it is found that the way we model a problem as a CSP makes a tremendous difference.
- Optimization---finding the best solution is also an area where a great deal of work is being done.