CS 1571 Introduction to AI
Lecture 7

Constraint satisfaction search.
Combinatorial optimization search.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Constraint satisfaction problem (CSP)

Objective:

- **Find a configuration** satisfying goal conditions

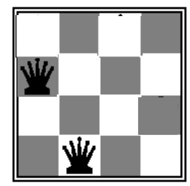
- **Constraint satisfaction problem (CSP)** is **a configuration search problem** where**:**
  - A **state** is defined by **a set of variables and their values**
  - **Goal condition** is represented by **a set constraints on possible variable values**

CSP example: N-queens

Goal: n queens placed in non-attacking positions on the board

Variables:

- Represent queens, one for each column:
  - Q_1, Q_2, Q_3, Q_4
- Values:
  - Row placement of each queen on the board
    {1, 2, 3, 4}



$Q_1 = 2, Q_2 = 4$

Constraints: $Q_i \neq Q_j$ Two queens not in the same row

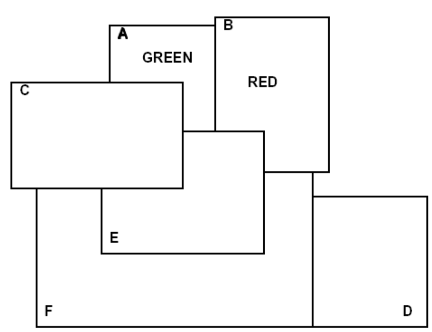
$|Q_i - Q_j| \neq |i - j|$ Two queens not on the same diagonal

Map coloring

Color a map using k different colors such that no adjacent countries have the same color

Variables:

- Represent countries
  - A, B, C, D, E
- Values:
  - K -different colors
    {Red, Blue, Green,..}



Constraints: $A \neq B, A \neq C, C \neq E$, etc

An example of a problem with **binary constraints**

Constraint satisfaction as a search problem

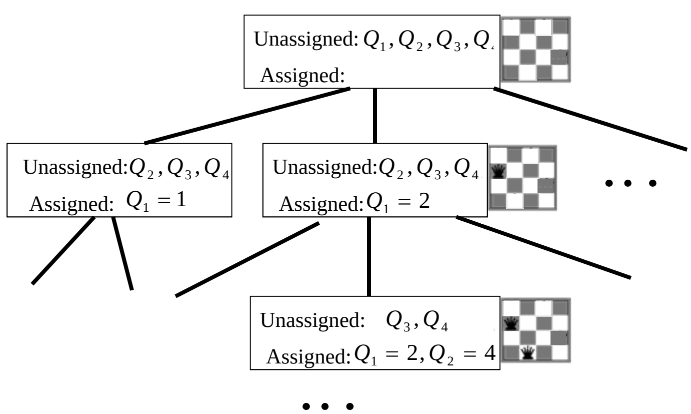
Formulation of a CSP as a search problem:

- **States.** Assignment (partial, complete) of values to variables.
- **Initial state.** No variable is assigned a value.
- **Operators.** Assign a value to one of the unassigned variables.
- **Goal condition.** All variables are assigned, no constraints are violated.

- **Constraints** can be **represented**:
  - **Explicitly** by a set of allowable values
  - **Implicitly** by a function that tests for the satisfaction of constraints

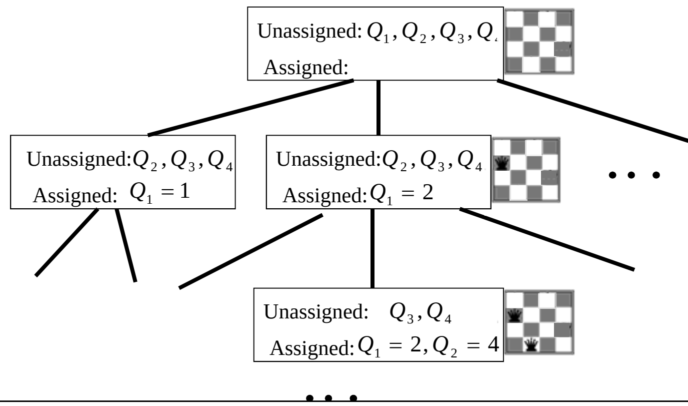
Solving a CSP through standard search

- **Maximum depth of the tree (m): ?**
- **Depth of the solution (d) : ?**
- **Branching factor (b) : ?**



Solving a CSP through standard search

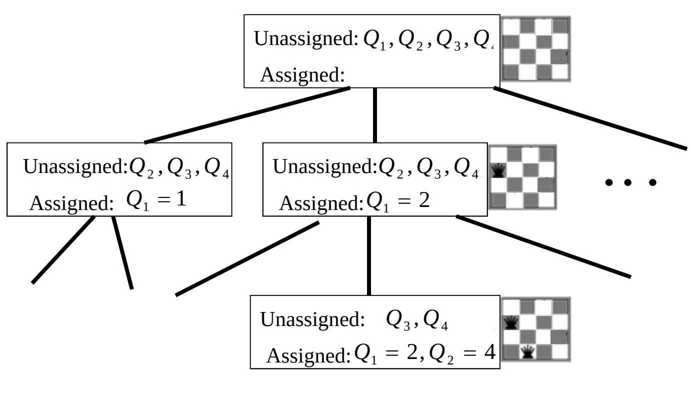
- **Maximum depth of the tree:** Number of variables in the CSP
- **Depth of the solution:** Number of variables in the CSP
- **Branching factor:** if we fix the order of variable assignments the branch factor depends on the number of their values



Solving a CSP through standard search

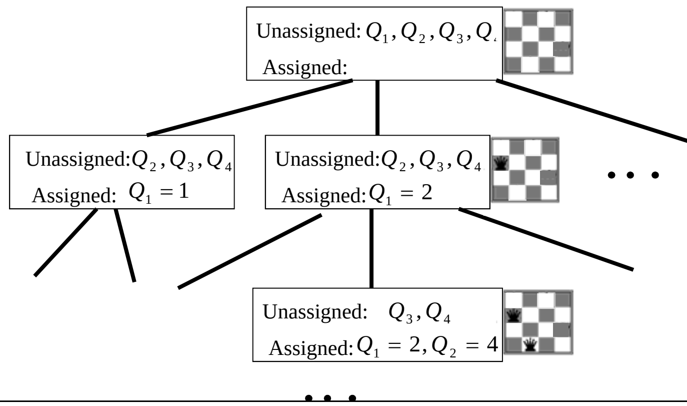
- **What search algorithm to use: ?**

  Depth of the tree = Depth of the solution=number of vars



Solving a CSP through standard search

- **What search algorithm to use:** **Depth first search !!!**
  - Since we know the depth of the solution
  - We do not have to keep large number of nodes in queues



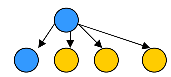
Constraint consistency

Assuring consistency of constraints:

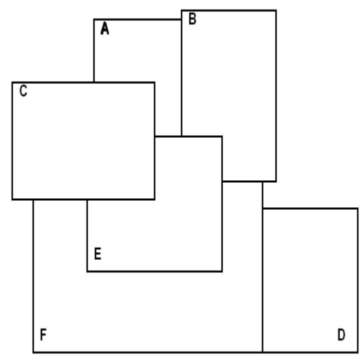
- Current **variable assignments** together **with constraints restrict remaining legal values of unassigned variables**
- The remaining **legal and illegal values of variables may be inferred** (effect of constraints propagates)
- To prevent "blind" exploration we can keep track of the remaining legal values, so we know when the constraints are violated and when to terminate the search

Constraint propagation

- Assign A=Red

A=Red


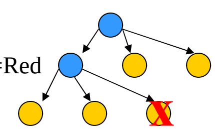
	Red	Blue	Green
A	✓		
B			
C			
D			
E			
F			



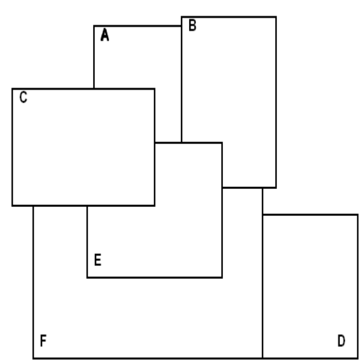
✓ - Assignments (equations) ✗ - Invalid assignments (disequations)

Constraint propagation

- Assign A=Red

A=Red
E=


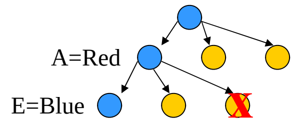
	Red	Blue	Green
A	✓		
B	✗		
C	✗		
D			
E	✗		
F			



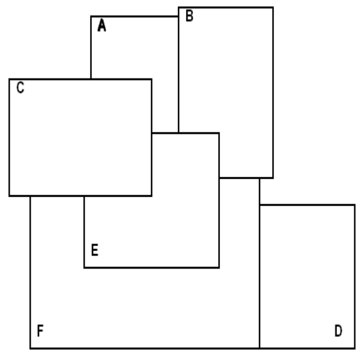
✓ - Assignments (equations) ✗ - Invalid assignments (disequations)

Constraint propagation

- Assign E=Blue

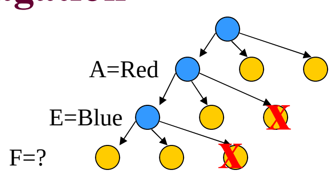


	Red	Blue	Green
A	✓		
B	✗		
C	✗		
D			
E	✗	✓	
F			

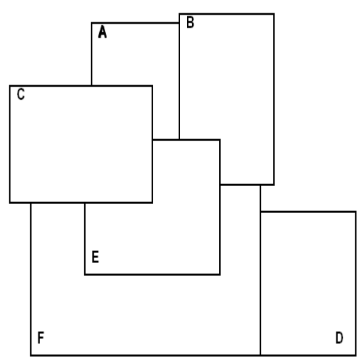


Constraint propagation

- Assign E=Blue



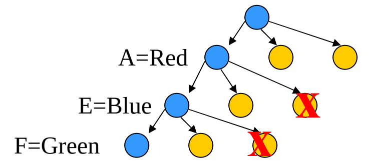
	Red	Blue	Green
A	✓	✗	
B	✗	✗	
C	✗	✗	
D			
E	✗	✓	
F		✗	

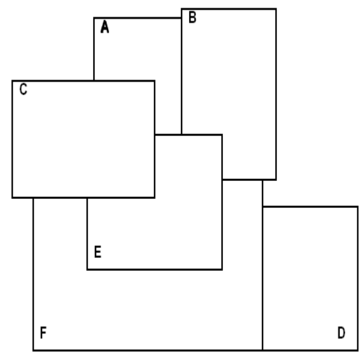


Constraint propagation

- Assign F=Green

	Red	Blue	Green
A	✓	✗	
B	✗	✗	
C	✗	✗	
D			
E	✗	✓	
F		✗	✓





Constraint propagation

Three known techniques for propagating the effects of past assignments and constraints:

- **Node propagation**
- **Arc consistency**
- **Forward checking**

- **Difference:**
  - Completeness of inferences
  - Time complexity of inferences.

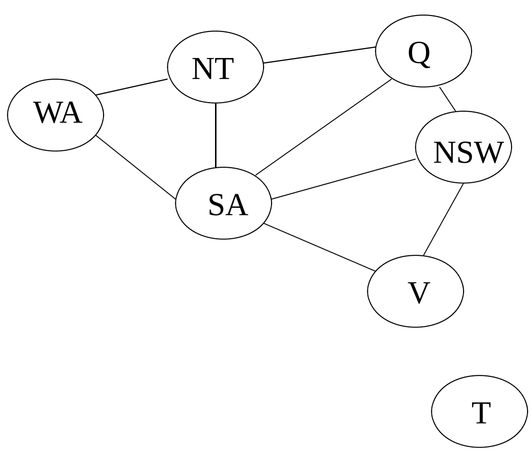
Constraint propagation

1. **Node consistency. Infers:**
   - **equations (valid assignments) or disequations (invalid assignments)** for an individual variable by applying **a unary constraint**
2. **Arc consistency. Infers:**
   - **disequations from** the set of **equations and disequations** defining the partial assignment, and **a constraint**
   - **equations through the exhaustion of alternatives**
3. **Forward checking. Infers:**
   - **disequations from** a set of **equations** defining the partial assignment, and a constraint
   - **Equations through the exhaustion of alternatives**

   Restricted forward checking:
   - uses only active constraints (active constraint – only one variable unassigned in the constraint)

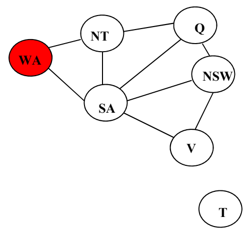
Example

Map coloring of Australia territories



Example: node consistency

Map coloring

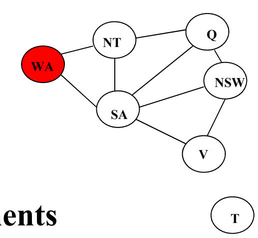


Assume a constraint:
WA≠ Green

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
inferred	?	?	?	?	?	?	?

Example: node consistency

Map coloring



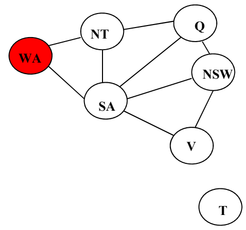
Assume a constraint:
WA≠ Green

Infer: invalid assignments from WA≠ Green constraint

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
inferred	**R B**	R G B	R G B	R G B	R G B	R G B	R G B

Example: forward checking

Map coloring

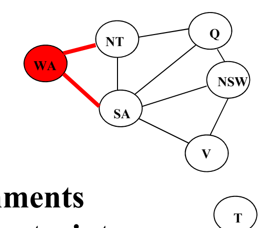


Set: WA=Red

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	?	?	?	?	?	?

Example: forward checking

Map coloring

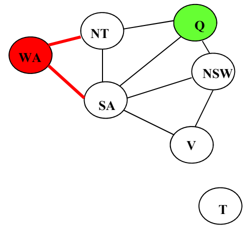


Infer: invalid assignments from WA=Red + constraints

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B

Example: forward checking

Map coloring

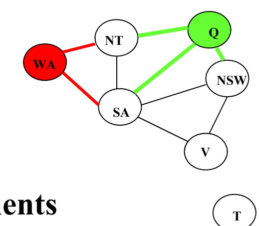


Set: Q=Green

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	?	**G**	?	?	?	?

Example: forward checking

Map coloring



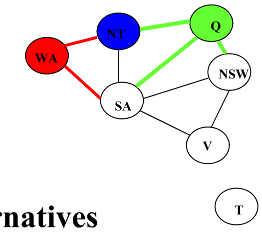
Infer: **invalid assignments from Q=Green + constraints**

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B

CS 1571 Intro to AI M. Hauskrecht

Example: forward checking

Map coloring



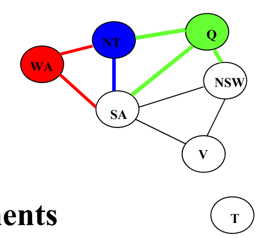
Infer: NT=B

Exhaustions of alternatives

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B
Infer NT	**R**	**B**	**G**	?	?	?	?

Example: forward checking

Map coloring

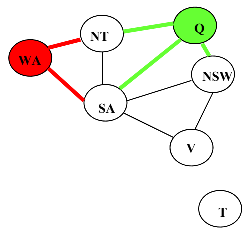


Infer: invalid assignments from NT=B

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B
Infer NT	**R**	**B**	**G**	R B	R G B	!	R G B

Example: arc consistency

Map coloring

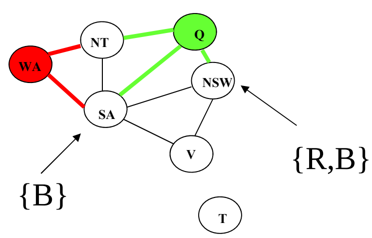


Set: WA=Red
Set: Q=Green

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B

Example: arc consistency

Map coloring

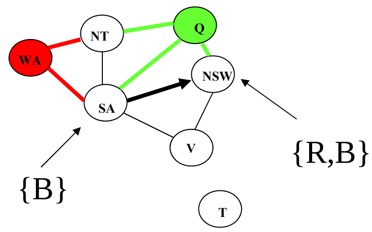


Infer: invalid assignments from valid and invalid assignments

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B

Example: arc consistency

Map coloring



SA=B
NSW=R

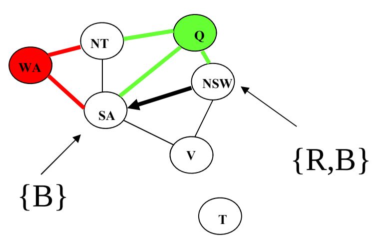
Consistent
assignment

Infer: **invalid assignments from valid and invalid assignments**

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B

Example: arc consistency

Map coloring



NSW=R
SA=B

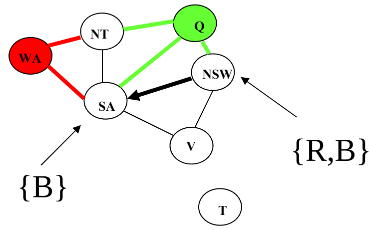
Consistent assignment

Infer: **invalid assignments from valid and invalid assignments**

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B

Example: arc consistency

Map coloring



NSW=B
SA=!

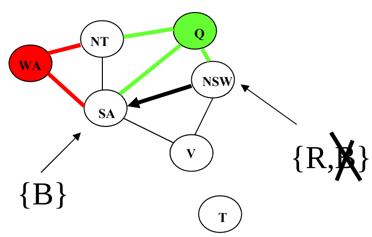
Inconsistent assignment

Infer: **invalid assignments from valid and invalid assignments**

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B

Example: arc consistency

Map coloring



NSW=B
SA=!

Inconsistent assignment

Infer: **invalid assignments from valid and invalid assignments**

vars	WA	NT	Q	NSW	V	SA	T
domain	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Red	**R**	G B	R G B	R G B	R G B	G B	R G B
Q=Green	**R**	B	**G**	R B	R G B	B	R G B
				R			

Heuristics for CSPs

CSP searches the space in the depth-first manner.

But we still can choose:

- **Which variable to assign next?**
- **Which value to choose first?**

Heuristics

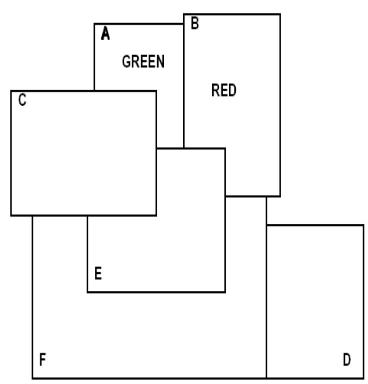
- **Most constrained variable**
  - Which variable is likely to become a bottleneck?
- **Least constraining value**
  - Which value gives us more flexibility later?

Heuristics for CSP

Example: **map coloring**

Heuristics

- **Most constrained variable**
  - ?
- **Least constraining value**
  - ?



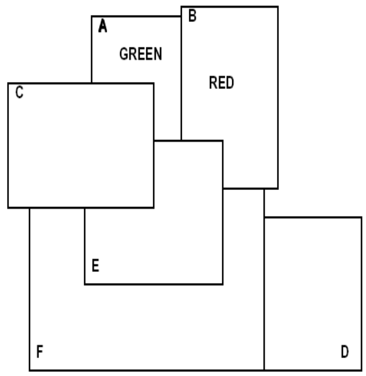
Heuristics for CSP

Examples: **map coloring**

Heuristics

- **Most constrained variable**
  - Country E is the most constrained one (cannot use Red, Green)

- **Least constraining value**
  - ?



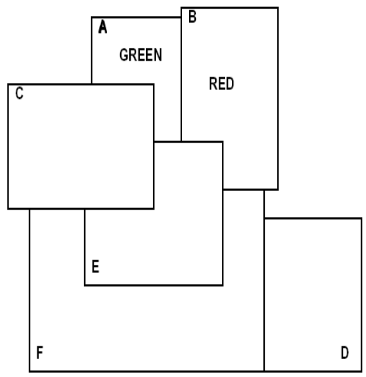
Heuristics for CSP

Examples: **map coloring**

Heuristics

- **Most constrained variable**
  - Country E is the most constrained one (cannot use Red, Green)

- **Least constraining value**
  - Assume we have chosen variable C
  - What color is the least constraining color?

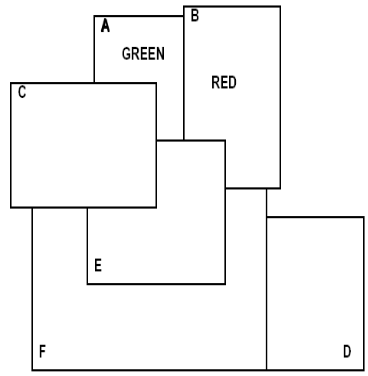


Heuristics for CSP

Examples: **map coloring**

Heuristics

- **Most constrained variable**
  - Country E is the most constrained one (cannot use Red, Green)
- **Least constraining value**
  - Assume we have chosen variable C
  - Red is the least constraining valid color for the future



Finding optimal configurations

Search for the optimal configuration

Constrain satisfaction problem:

Objective: find a configuration that satisfies all constraints

Optimal configuration problem:

Objective: find the best configuration

The quality of a configuration: is defined by some quality measure that reflects our **preference towards each configuration** (or state)

Our goal: optimize the configuration according to the quality measure also referred to as **objective function**

Search for the optimal configuration

Optimal configuration search:

- Configurations are described in terms of variables and their values
- Each configuration has a quality measure
- Goal: find the configuration with the best value

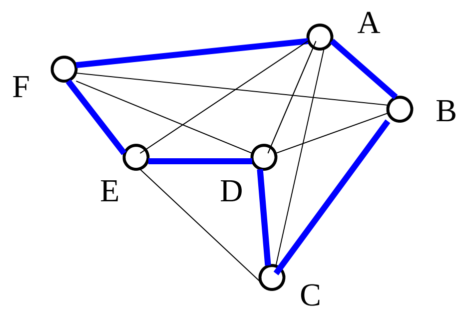
If the space of configurations we search among is

- **Discrete or finite**
  - **then it is a combinatorial optimization problem**
- **Continuous**
  - **then it is a parametric optimization problem**

Example: Traveling salesman problem

Problem:

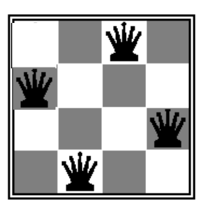
- A graph with distances
- A tour – a path that visits every city once and returns to the start e.g. ABCDEF



- **Goal:** find the shortest tour

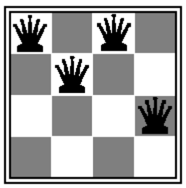
Example: N queens

- A CSP problem
- Is it possible to formulate the problem as an optimal configuration search problem ?

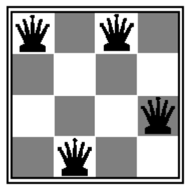


Example: N queens

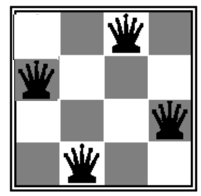
- A CSP problem
- Is it possible to formulate the problem as an optimal configuration search problem ? **Yes.**
- **The quality of a configuration in a CSP** can be measured by the number of violated constraints
- **Solving:** minimize the number of constraint violations



\# of violations =3



\# of violations =1



\# of violations =0

Iterative optimization methods

- Searching systematically for the best configuration with the **DFS** may not be the best solution
- Worst case running time:
  - Exponential in the number of variables
- Solutions to **large 'optimal' configuration** problems are often found using iterative optimization methods

- **Methods:**
  - **Hill climbing**
  - **Simulated Annealing**
  - **Genetic algorithms**

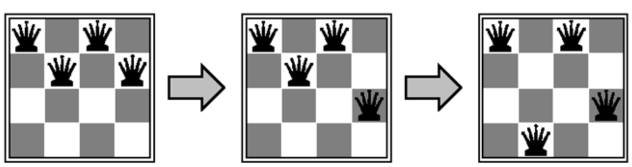
Iterative optimization methods

Properties:

- **Search the space of "complete" configurations**
- **Take advantage of local moves**
  - Operators make "local" changes to "complete" configurations
- **Keep track of just one state (the current state)**
  - no memory of past states
  - **!!! No search tree is necessary !!!**

Example: N-queens

- **"Local" operators for generating the next state:**
  - Select a variable (a queen)
  - Reallocate its position



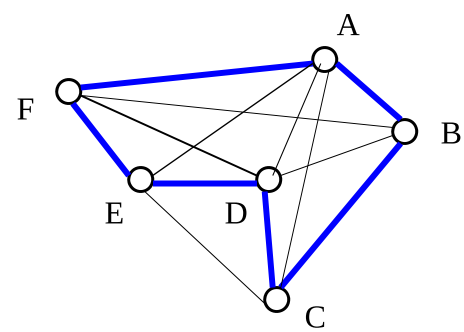
Example: Traveling salesman problem

“Local” operator for generating the next state:

- divide the existing tour into two parts,
- reconnect the two parts in the opposite order

Example:

ABCDEF



Example: Traveling salesman problem

“Local” operator for generating the next state:

- divide the existing tour into two parts,
- reconnect the two parts in the opposite order

Example:

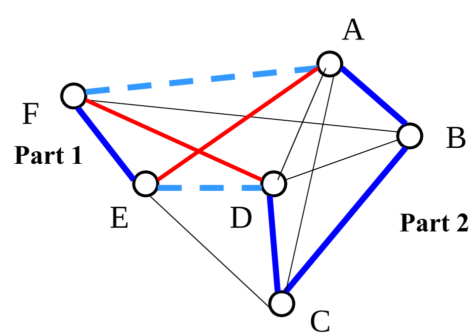
ABCDEF

↓

ABCD | EF |

↓

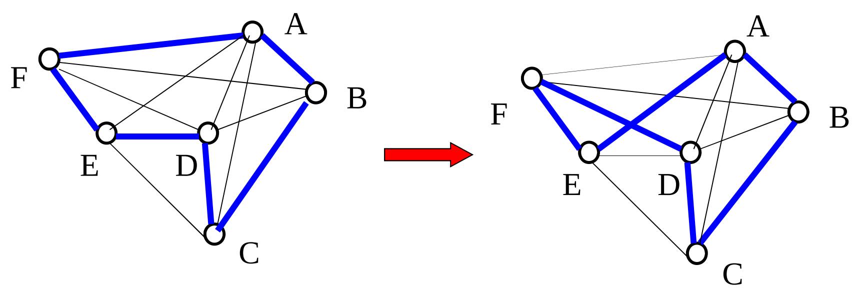
ABCDFE



Example: Traveling salesman problem

"Local" operator:

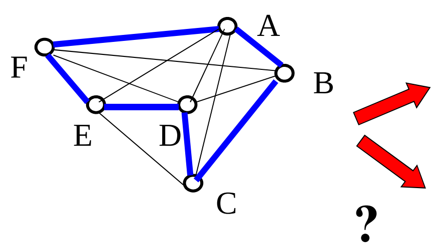
– generates the next configuration (state)

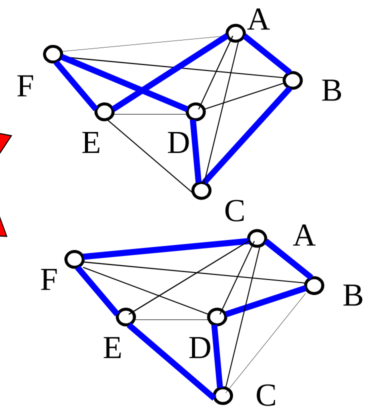


Searching the configuration space

Search algorithms

- keep only one configuration (the current configuration)





Problem:

- How to decide about which operator to apply?

Search algorithms

Two strategies to choose the configuration (state) to be visited next:

- **Hill climbing**
- **Simulated annealing**

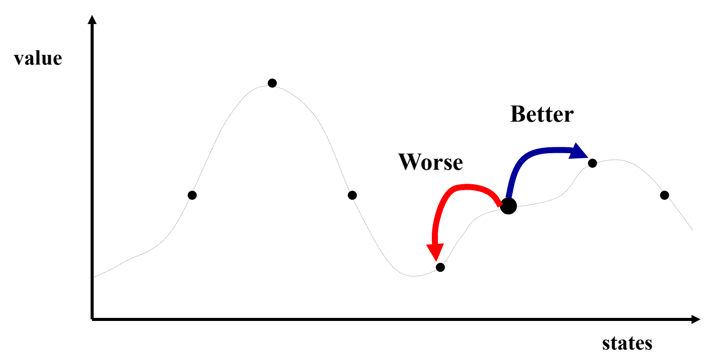
- Later: Extensions to multiple current states:
  - **Genetic algorithms**

- **Note:** Maximization is inverse of the minimization

$$\min \ f(X) \Leftrightarrow \max \left[- f(X) \right]$$

Hill climbing

- Look around at states in the local neighborhood and choose the one with the best value
- Assume: we want to maximize the



Hill climbing

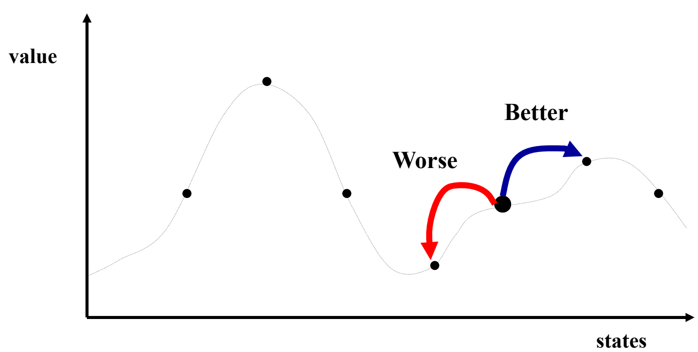
- Always choose the next best successor state
- Stop when no improvement possible

```
function HILL-CLIMBING(problem) returns a solution state
   inputs: problem, a problem
   static: current, a node
           next, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      next ← a highest-valued successor of current
      if VALUE[next] < VALUE[current] then return current
      current ← next
   end
```


Hill climbing

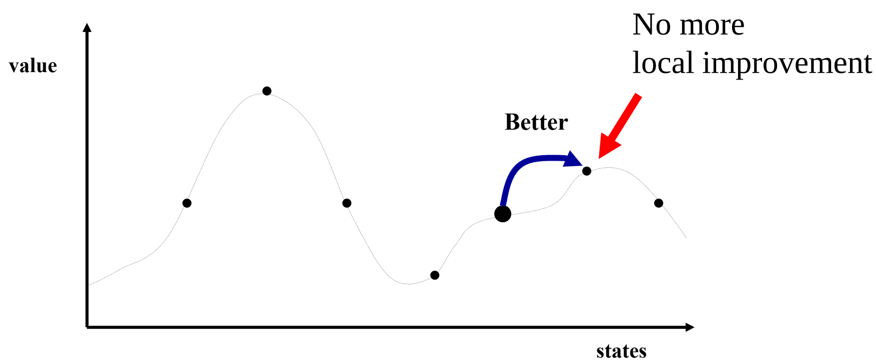
- Look around at states in the local neighborhood and choose the one with the best value



- **What can go wrong?**

Hill climbing

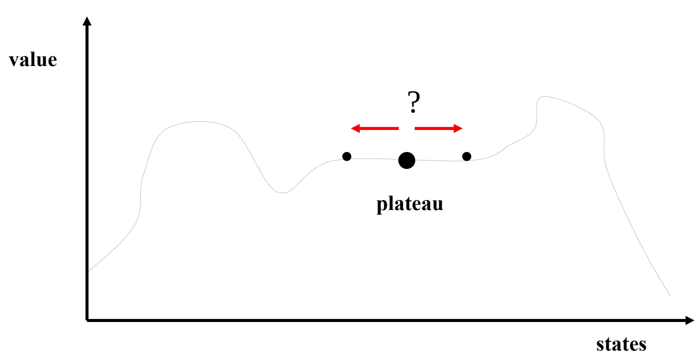
- Hill climbing can get trapped in the local optimum



- **What can go wrong?**

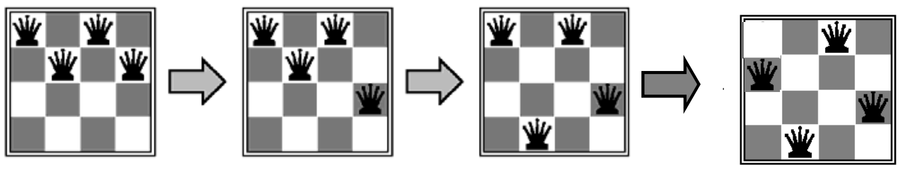
Hill climbing

- Hill climbing can get clueless on plateaus



Hill climbing and n-queens

- The quality of a configuration is given by the number of constraints violated
- **Then: Hill climbing** reduces the number of constraints
- **Min-conflict strategy (heuristic):**
  - Choose randomly a variable with conflicts
  - Choose its value such that it violates the fewest constraints



Success !! But not always!!! The local optima problem!!!