
CM0312 Artificial Intelligence

II Problem Solving by Searching

II.4 Local Search and Constraint Satisfaction

F. C. Langbein

School of Computer Science
and Informatics
Cardiff University





Overview

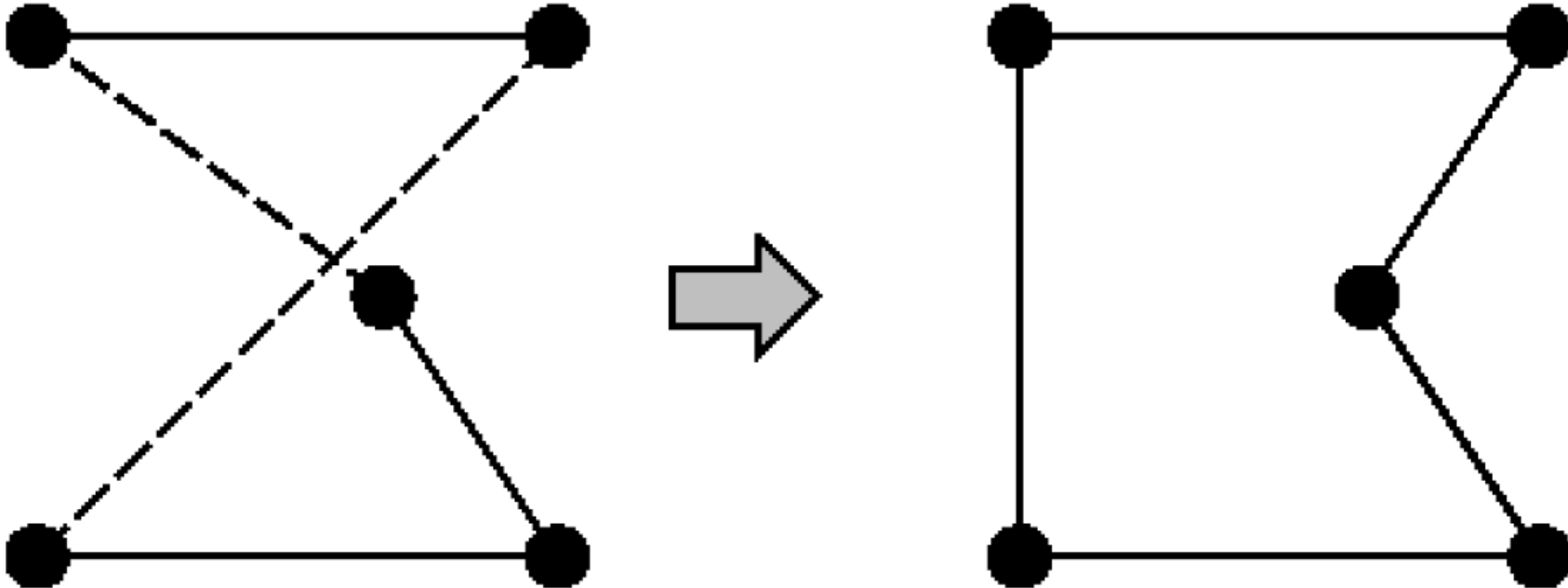
- Local search problems
 - Hill-climbing
 - Simulated annealing
 - Genetic algorithm
- Constraint satisfaction problems
 - Optimisation approach
 - Backtracking search
 - Backtracking with forward checking
 - Heuristics for constraint satisfaction problems

Local Search Problem

- Some problems **only** require to find a **goal state**
- Search space is a graph
(we do not consider continuous problems)
 - Node: a complete configuration
 - Edge: change complete configuration
- Problem: find *optimal* (or at least a good) configuration
 - Optimality is given by some evaluation function, $f(x) : X \rightarrow \mathbb{R}$, to measure quality of solution
- Not all states have to be “valid”
(e.g., some states are a solution, others are not)
- One approach: **iterative improvement** algorithms
 - Keep a single “current” state and try to improve it

Travelling Salesman Problem

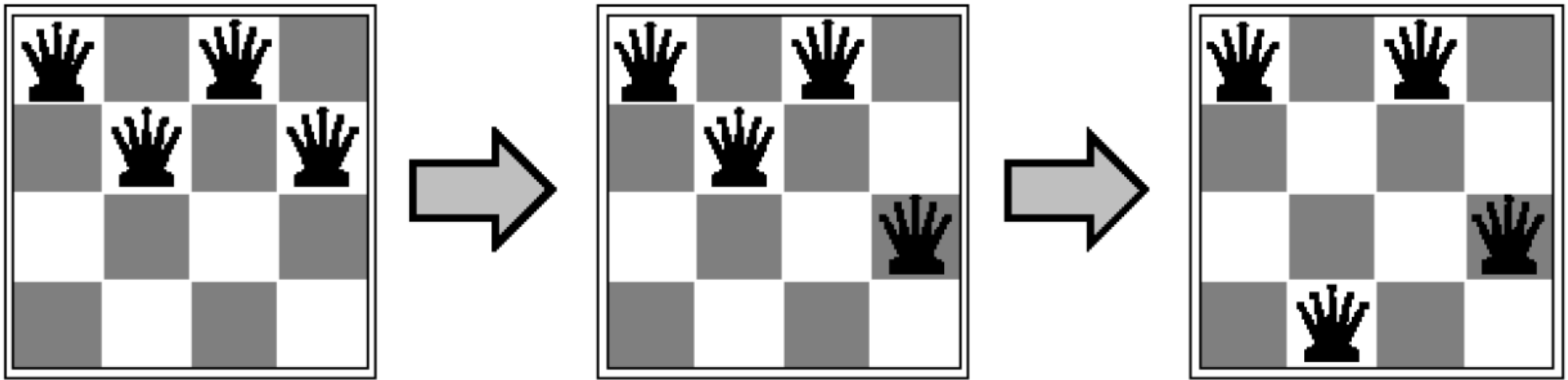
➤ Find the shortest tour that visits each city exactly once



- Each path is one state (node in the search graph)
- An edge in the search graph is a change of the path
- Not every state has to represent a valid path
- Use length of path to evaluate states

n-Queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

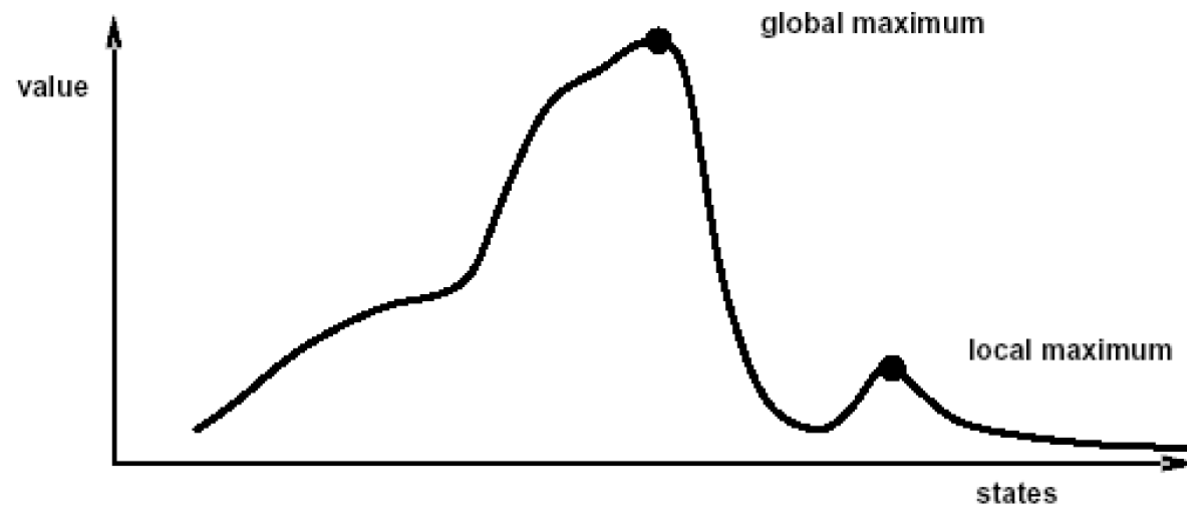


- A node is a board configuration
- An edge represents, e.g., change of position of one queen to a neighbouring field or any other unoccupied position, etc.
- Evaluate state by number of violations of constraints

Hill-Climbing

➤ Also called gradient ascent/descent

```
function state ← HILL-CLIMBING (problem)  
state ← INITIAL-STATE (problem)  
loop  
  next ← HIGHEST-VALUED-NEIGHBOUR (state)  
  if VALUE(next) < VALUE(state), return state  
  state ← next
```



➤ “Like climbing Everest in thick fog with amnesia”

Simulated Annealing

- Escape local maxima by allowing some “bad” moves,
 - but gradually decrease their size and frequency according to some agenda

```
function state ← SIMULATED-ANNEALING (problem, agenda)
state ← INITIAL-STATE (problem)
for t ← 1 to ∞
    T ← agenda[t] % map “time” to “temperature”
    % e.g. Tnew ← Told - CΔE or Tnew ← CTold, 0 < C ≤ 1.0
    if T ≤ 0, return state
    next ← RANDOM-NEIGHBOUR (state)
    ΔE ← VALUE (next) - VALUE (state)
    if ΔE > 0, state ← next
    else state ← next with probability exp(ΔE/T)
```

Genetic Algorithm

- Based on ideas taken from evolutionary biology
 - Populations, chromosomes, crossover, mutation
 - On potential solutions (states) encoded as strings

```
function GENETIC-ALGORITHM (problem)  
  population ← POTENTIAL-SOLUTIONS (problem)  
  while not CONVERGED (population)  
    population ← SELECT (population)  
    population ← CROSSOVER (population)  
    population ← MUTATE (population)  
  return BEST-INDIVIDUAL (population)
```

- Biological evolution = optimisation ???

Constraint satisfaction problems

- Standard search problem:
 - *state* is “black box” — any data structure supporting *goal test*, *evaluation*, *neighbour* operations
- **Constraint Satisfaction Problem** (CSP):
 - *State* is defined by **variables** v_i
 - with **values** from **domain** D_i
 - *Goal test* is set of **constraints** specifying allowable combinations of values for subsets of variables
- Essentially a **solve my problem** approach
 - where the problem is specified **formally**
 - like a generalised equation system

4-Queens as CSP

➤ Assume one queen in each column:
which row does each one go in?

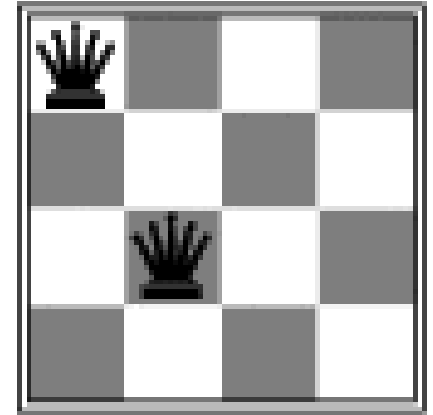
● *Variables*: q_1, q_2, q_3, q_4

● *Domains*: $D_l = \{1, 2, 3, 4\}$ for $l = 1, 2, 3, 4$

● *Constraints*:

$q_l \neq q_k$ (not in same row)

$|q_l - q_k| \neq |l - k|$ (not in same diagonal)



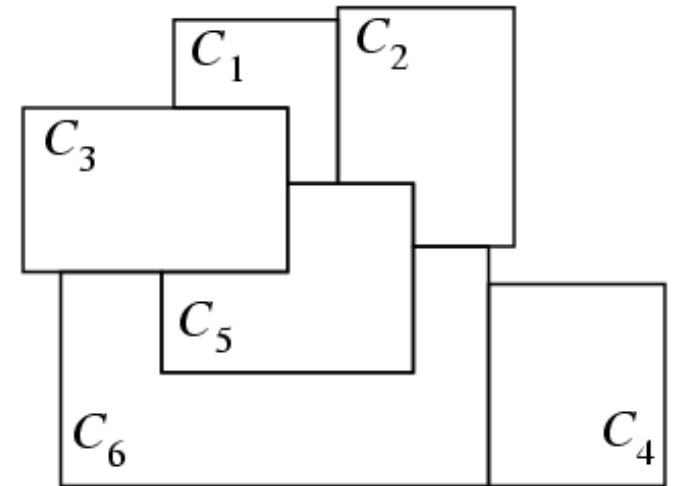
➤ We can translate each constraint into set of allowable values for its variables

● E.g. values for (q_1, q_2) are $(1, 3), (1, 4), (2, 4), (3, 1), (4, 1), (4, 2)$

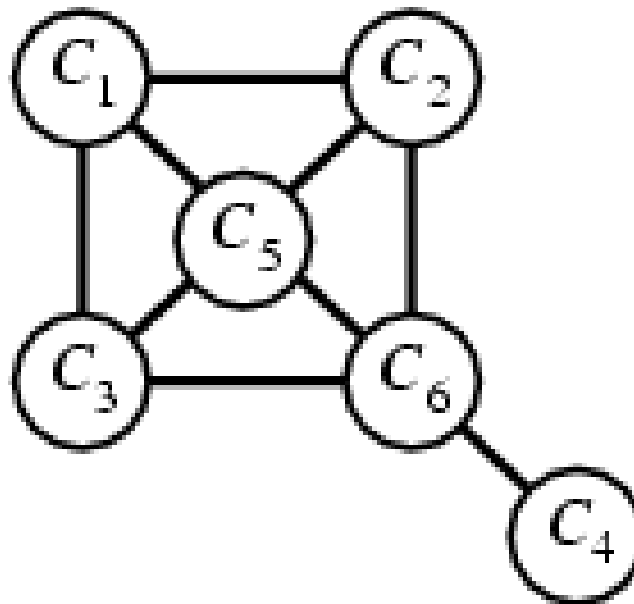
Map Colouring

➤ Colour a map such that no adjacent countries have the same colour

- *Variables*: Countries C_1
- *Domains*: {Red, Green, Blue}
- *Constraints*: $C_1 \neq C_2$, $C_1 \neq C_5$, ...



➤ *Constraint graph*



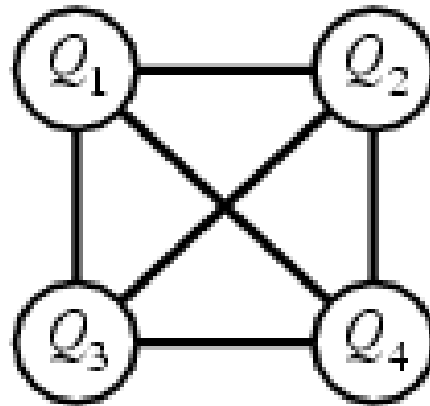
Constraint Graph

Binary CSP

- Each constraint relates to at most two variables

Constraint graph

- Nodes represent variables
- Edges represent constraints



Non-binary CSP form a hyper-graph
(edges can have more than two nodes)

- But can in principle be converted to a graph

Optimisation Approach to CSP

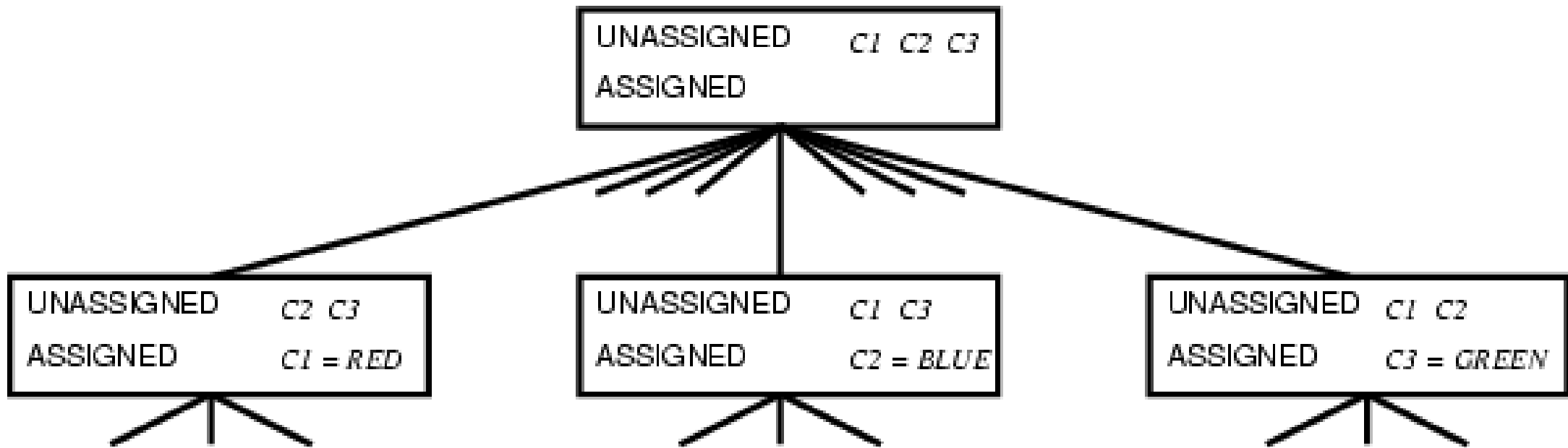
- Local search works with complete states
 - A state assigns a value to each variable
- To apply this to CSPs:
 - Allow states with unsatisfied constraints
 - Operators **reassign** variable values
(randomly select a variable involved in a constraint violation and assign another value)
- **Min-conflicts** Heuristic:
 - Choose value that violated the fewest constraints,
 - E.g. hill-climb with $h(x) =$ “total number of violated constraints”

Path Search Approach to CSP

- Try a path search approach to solve CSPs:
 - *States* are defined by values assigned so far
 - *Initial state*: all variables unassigned
 - *Operators*: assign a value to an unassigned variable
 - *Goal test*: all variables assigned, no constraints violated
- Assign values to unassigned variables, depth-first
 - Yields a path in CSP search graph
- This is the same for all CSPs!

Search Applied to Map-Colouring

➤ Search tree:



➤ Is this a good approach ???

Complexity of Dumb Approach

➤ *Search algorithm*: depth-first

● *Max. depth* of search tree: $M = n$
(number of variables)

● *Solution state depth*: $D = n$
(all variables assigned)

● *Branching factor*: $B = \sum_i |D_i|$
(at top of tree)

➡ $O(nB)$ memory and $O(B^n)$ time

➤ Note:

● **Order** of assignment is **irrelevant**
(many paths are equivalent)

● New assignments **do not correct a violated constraint**

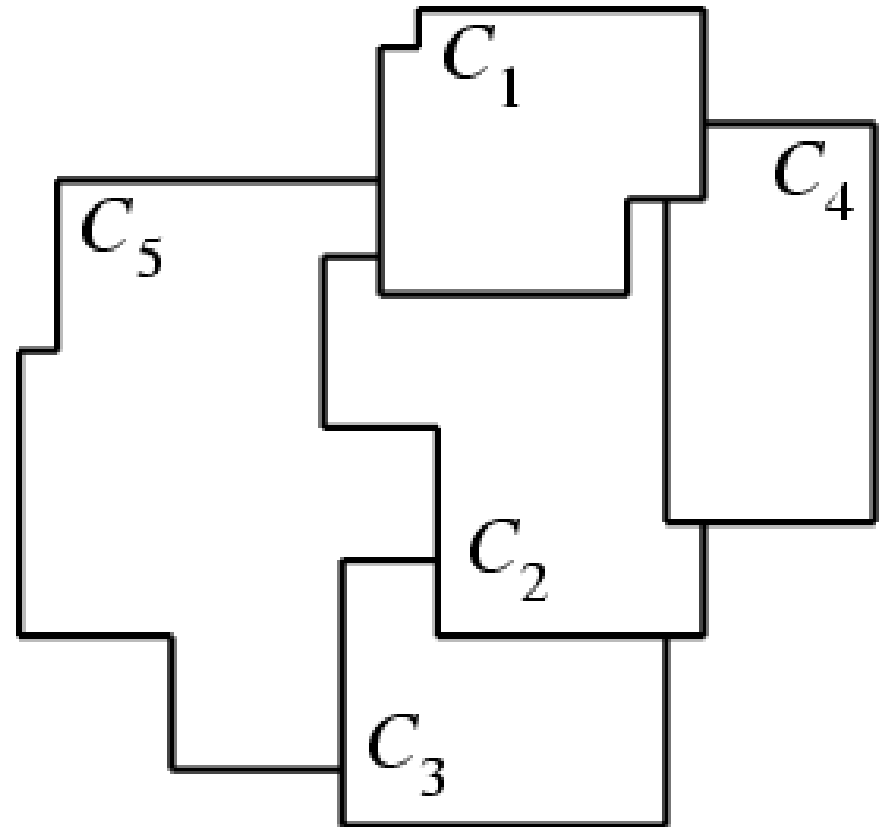
Backtracking Search

- Use depth-first, but
 - **Fix order** of assignment $\Rightarrow B = |D_i|$
(can be done in SUCCESSOR function)
 - **Check** for constraint violations
- Constraint violation check
 - Modify SUCCESSORS to **assign only values that are allowed**, given the values already assigned
 - or check **constraints are satisfied before expanding state**
- **Backtracking** search is basic uninformed search algorithm for CSPs
 - Can solve n-queens for $n \approx 15$

Forward Checking

- Expand backtracking with **forward checking**
 - Keep track of remaining legal values for unassigned variables
 - Terminate search when any variable has no legal values

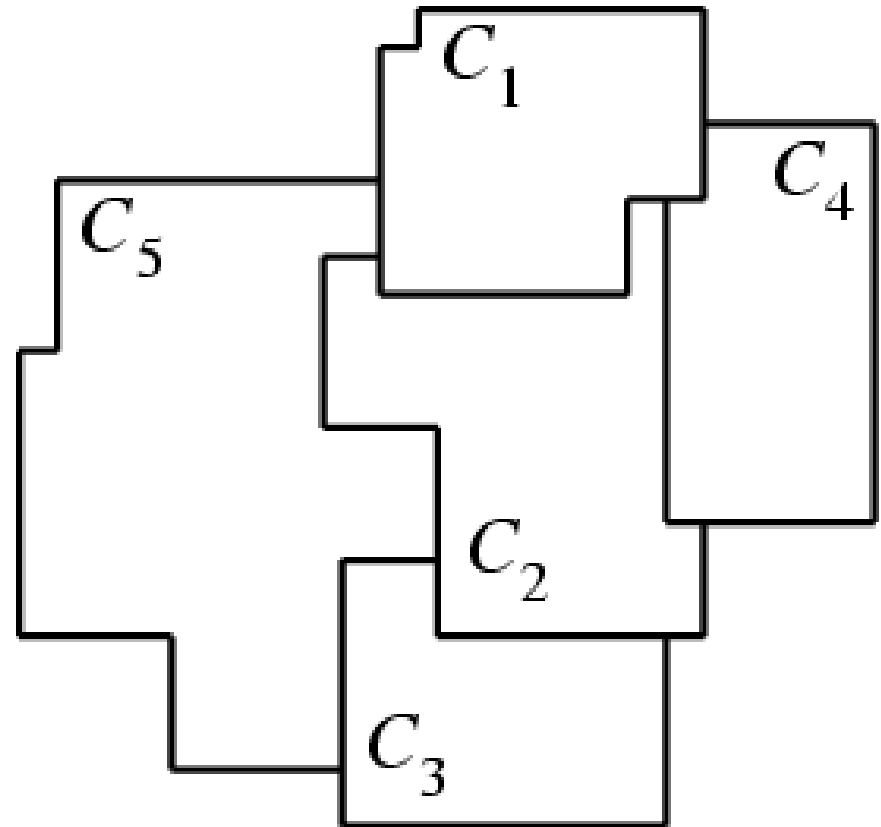
	Red	Blue	Green
C_1	✓		
C_2	×		
C_3			
C_4	×		
C_5	×		



Forward Checking

- Expand backtracking with **forward checking**
 - Keep track of remaining legal values for unassigned variables
 - Terminate search when any variable has no legal values

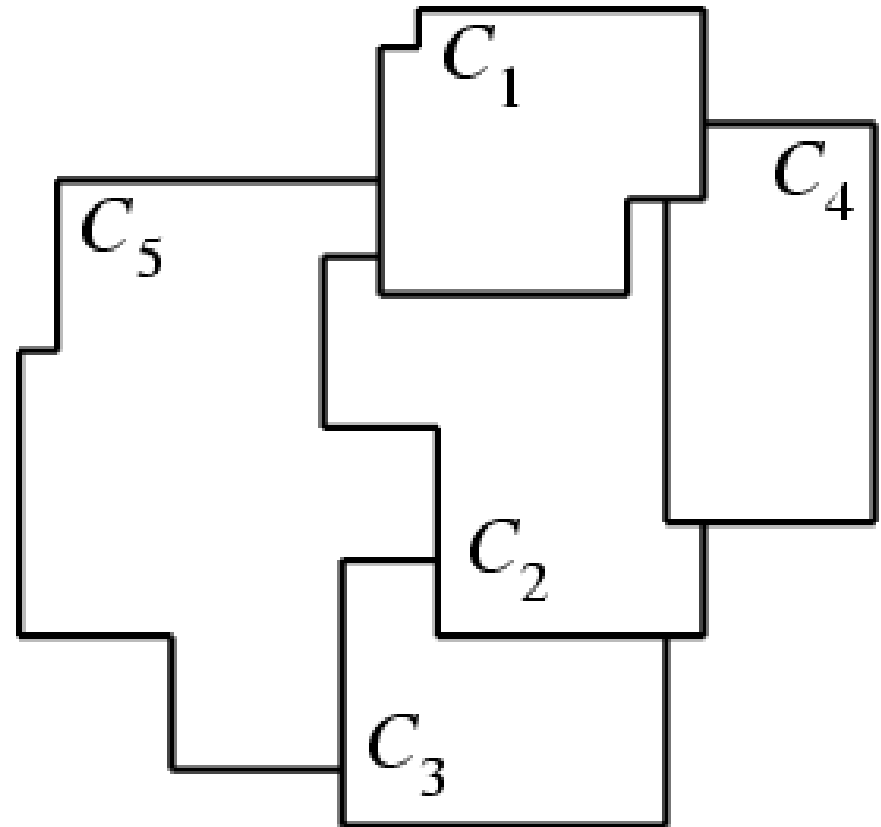
	Red	Blue	Green
C_1	✓		
C_2	×	✓	
C_3		×	
C_4	×	×	
C_5	×	×	



Forward Checking

- Expand backtracking with **forward checking**
 - Keep track of remaining legal values for unassigned variables
 - Terminate search when any variable has no legal values

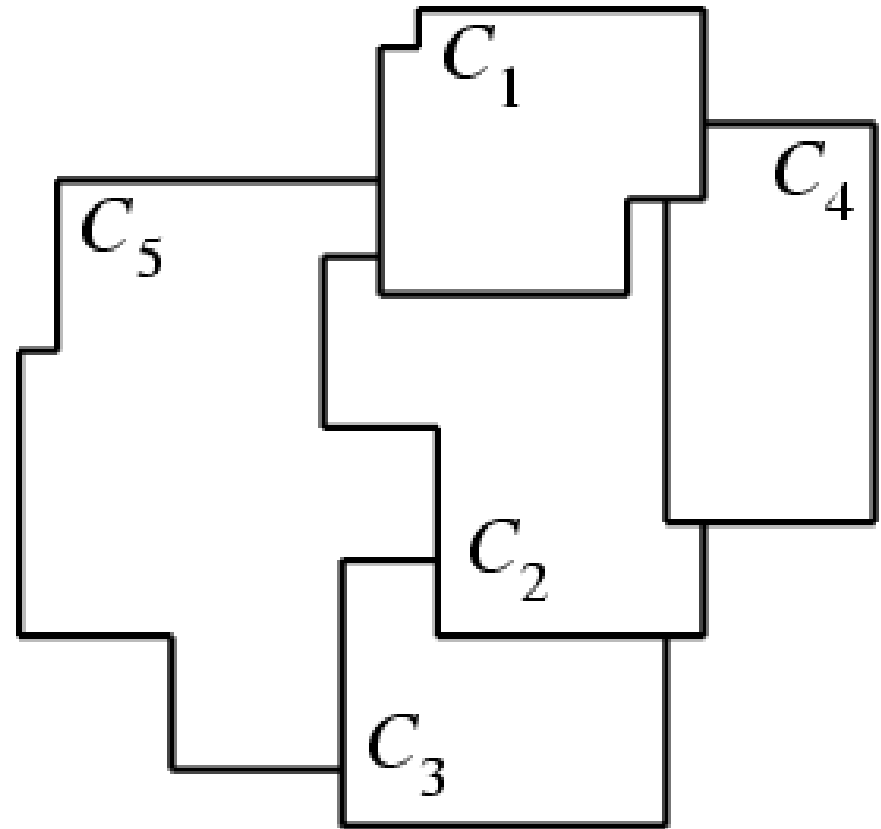
	Red	Blue	Green
C_1	✓		
C_2	×	✓	
C_3		×	✓
C_4	×	×	
C_5	×	×	×



Forward Checking

- Expand backtracking with **forward checking**
 - Keep track of remaining legal values for unassigned variables
 - Terminate search when any variable has no legal values

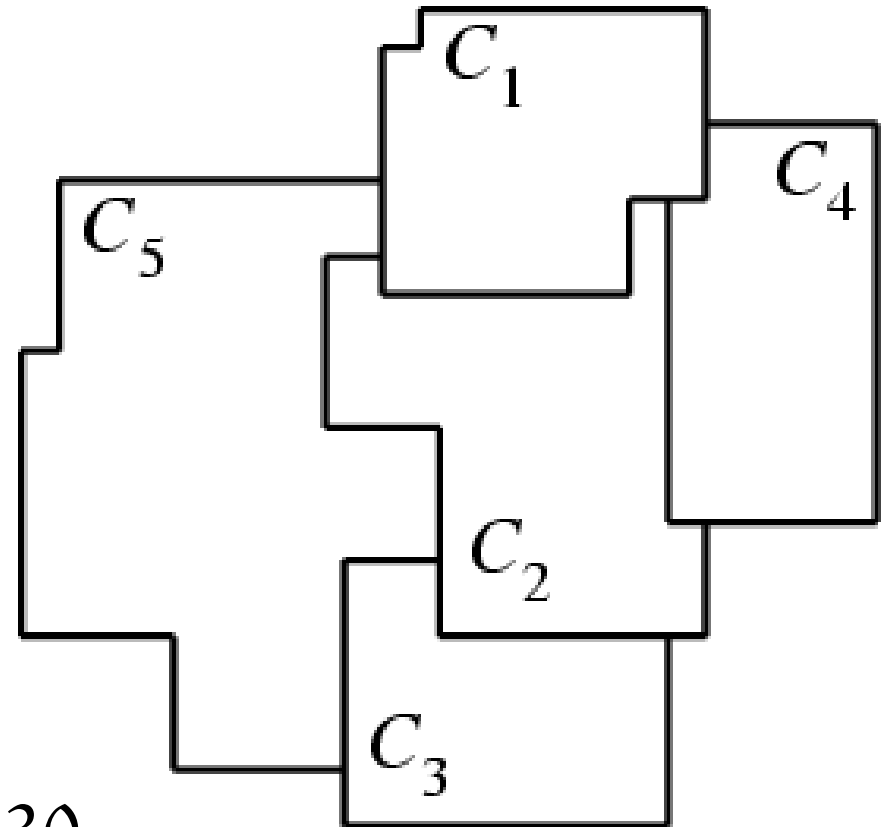
	Red	Blue	Green
C_1	✓		
C_2	×	✓	
C_3	✓	×	
C_4	×	×	
C_5	×	×	



Forward Checking

- Expand backtracking with **forward checking**
 - Keep track of remaining legal values for unassigned variables
 - Terminate search when any variable has no legal values

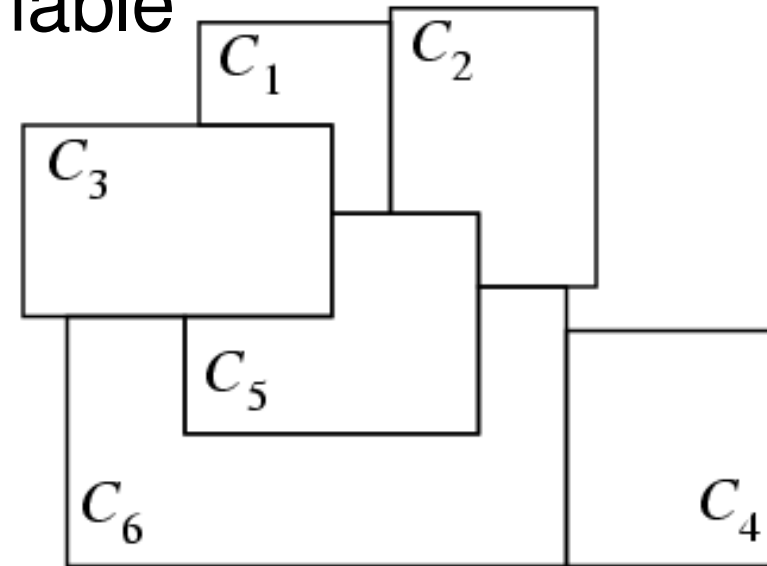
	Red	Blue	Green
C_1	✓		
C_2	×	✓	
C_3	✓	×	
C_4	×	×	✓
C_5	×	×	(✓)



- Can solve n-queens up to $n \approx 30$

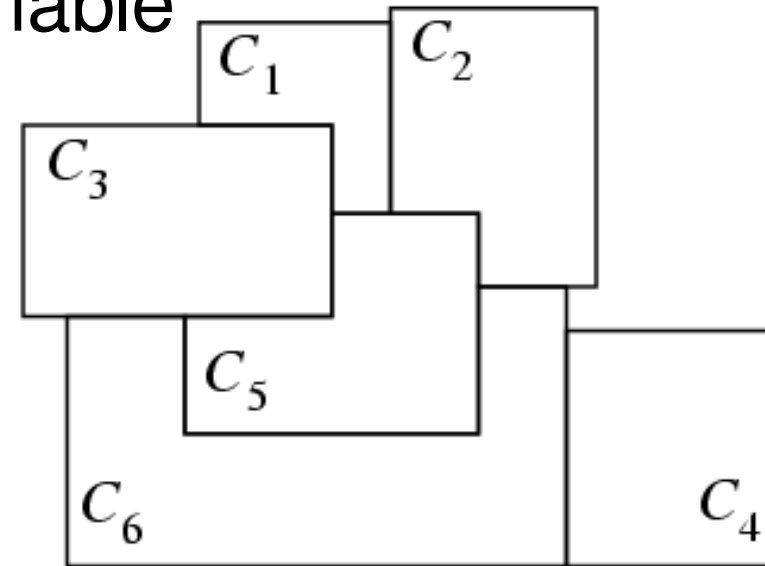
Heuristics for CSPs

- More intelligent decisions on
 - which value to choose for each variable
 - which variable to assign next
- Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, $C_3 = ?$



Heuristics for CSPs

- More intelligent decisions on
 - which value to choose for each variable
 - which variable to assign next
- Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, $C_3 = ?$
 - $C_3 = \text{Green}$ ($C_5 \neq \text{Green}$)
 - ➔ Choose **least-constraining value**
- Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, what next?



Heuristics for CSPs

- More intelligent decisions on
 - which value to choose for each variable
 - which variable to assign next

➤ Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, $C_3 = ?$

- $C_3 = \text{Green}$ ($C_5 \neq \text{Green}$)

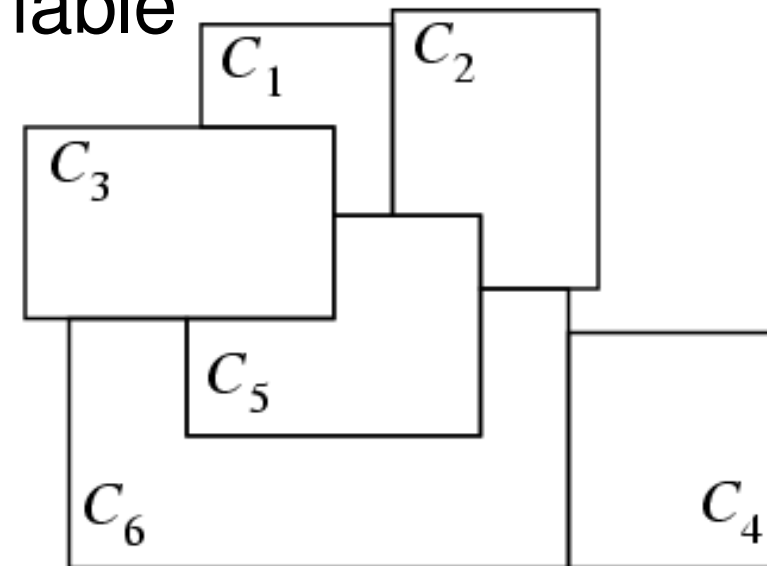
➔ Choose **least-constraining value**

➤ Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, what next?

- C_5

➔ Choose **most-constrained variable**

➤ Can solve n-queens for $n \approx 1000$





Real-World CSPs

- Assignment problems
 - E.g., who teaches what class?
- Timetabling problems
 - E.g. which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Geometric constraints
- Many real-world problems involve real-valued variables
(infinite, continuous domains; we cannot list all allowed combinations)