Week 3: Solving CSPs

Dr. Miqing Li

Cryptarithmetic

Cryptarithmetic is a puzzle where the digits of numbers are represented by letters. Each letter represents a unique digit. The goal is to find the digits such that a given equation is verified.

- Variables:
  - T, W, O, F, U, R
  - X1, X2, X3 (they are the carries in the tenths, hundredths and thousandths places, respectively)
- Domain:
  - T, W, O, F, U, R $\in \{0,1,2,3,4,5,6,7,8,9\}$
  - X1, X2, X3 $\in \{0, 1\}$
- Constraints:
  - *alldiff*(T, W, O, F, U, R)
  - $O + O = R + 10 \cdot X1$
  - $W + W + X1 = U + 10 \cdot X2$
  - $T + T + X2 = O + 10 \cdot X3$
  - $X3 = F$
  - $T, F \neq 0$

$$\begin{array}{r} \ T\ W\ O \\ +\ \ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

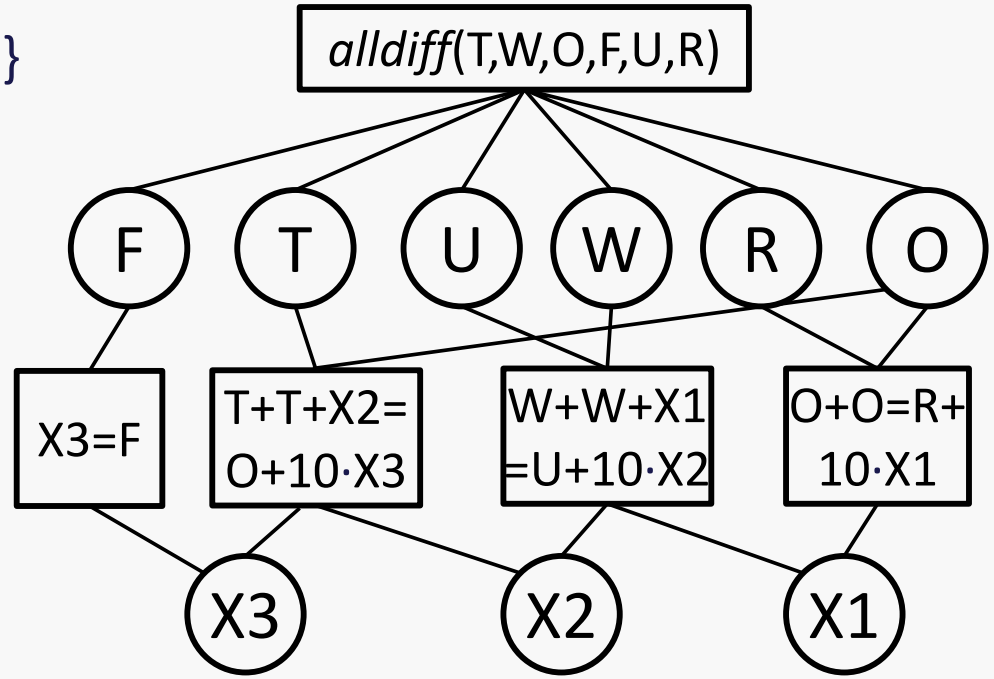
$$\begin{array}{r} \ 8\ 3\ 6 \\ +\ \ 8\ 3\ 6 \\ \hline 1\ 6\ 7\ 2 \end{array}$$

Cryptarithmetic (constraint graph)

Cryptarithmetic is a puzzle where the digits of numbers are represented by letters. Each letter represents a unique digit. The goal is to find the digits such that a given equation is verified.

```
    T W O
+   T W O
-------
  F O U R
```

- Variables:
  - T, W, O, F, U, R
  - X1, X2, X3 (they are the carries in the tenths, hundredths and thousandths places, respectively)
- Domain:
  - T, W, O, F, U, R ∈ {0,1,2,3,4,5,6,7,8,9}
  - X1, X2, X3 ∈ {0, 1}
- Constraints:
  - *alldiff*(T, W, O, F, U, R)
  - O + O = R + 10 · X1
  - W + W + X1 = U + 10 · X2
  - T + T + X2 = O + 10 · X3
  - X3 = F
  - T, F ≠ 0



Generate and Test

- The exhaustive generate-and-test algorithm is to generate all the complete assignments, then test them in turn, and return the first one that satisfies all of the constraints.

- It needs to store all d^n complete assignments, where d is the domain size and n is the number of variables.

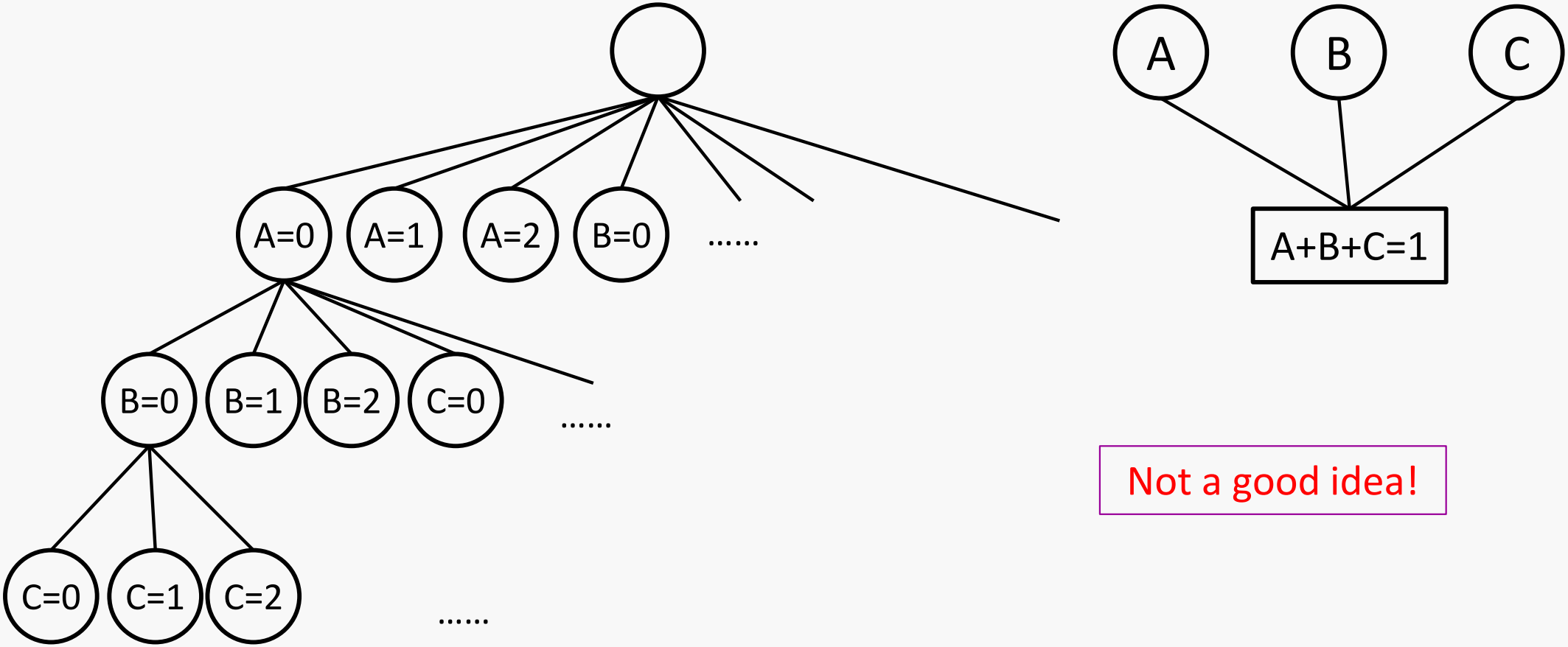
- So we must find alternative methods.

Solving CSPs by standard search formulation

- In CSPs, states defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment {}.
  - Successor function: assign a value to an unassigned variable.
  - Goal test: if the current assignment is complete and satisfies all the constraints.

Solving CSPs by BFS

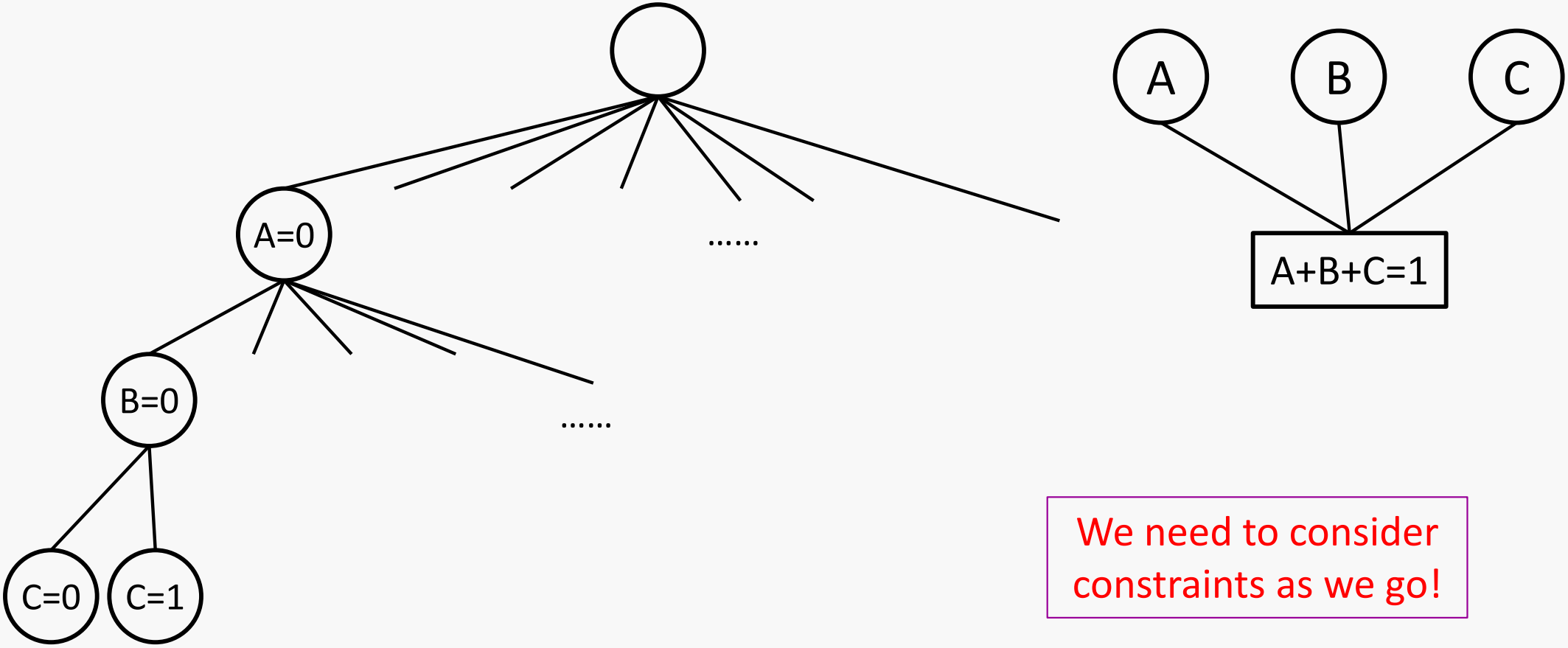
- Example: There are three variables A, B, and C, all with domain {0, 1, 2}. The constraint is A+B+C=1.
- Since the solution are always in the bottom layer, BFS needs to traverse all the nodes (partial assignments).



Not a good idea!

Solving CSPs by DFS

- Example: There are three variables A, B, and C, all with domain {0, 1, 2}. The constraint is A+B+C=1.
- Sounds a good idea, but what if the constraint is A>B>C?

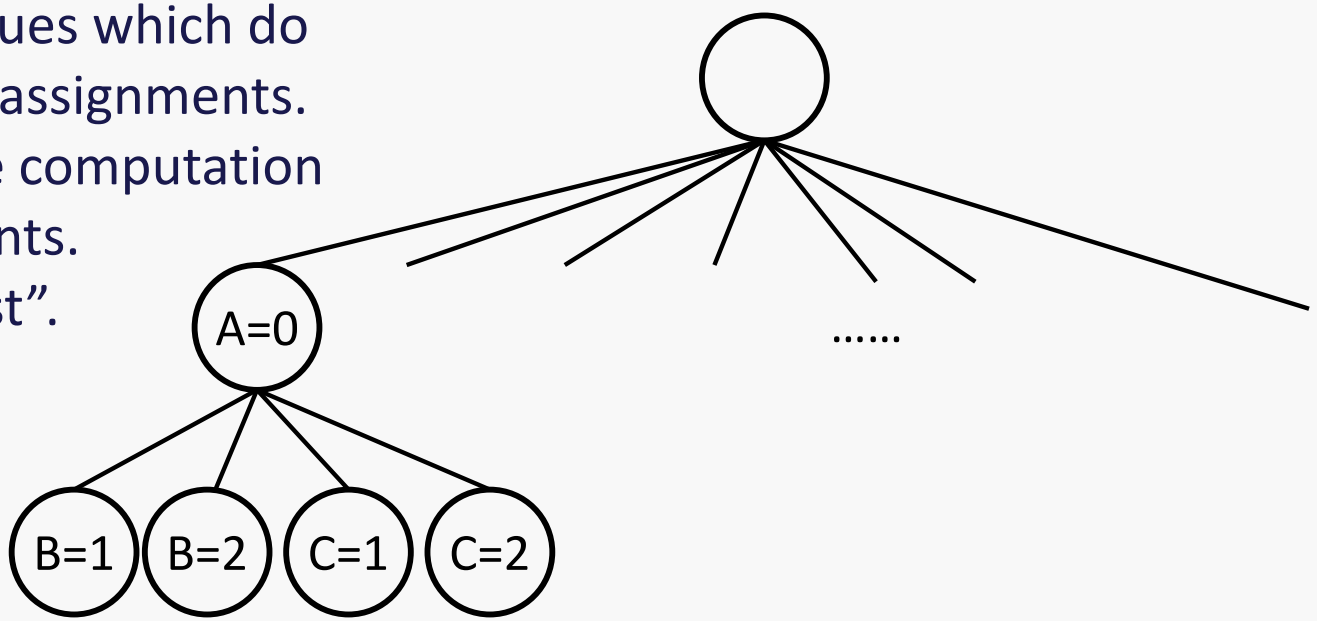


We need to consider constraints as we go!

Backtracking Search

Backtracking is a DFS method with two additional things: 1) Check constraints as you go and 2) Consider one variable at a layer.

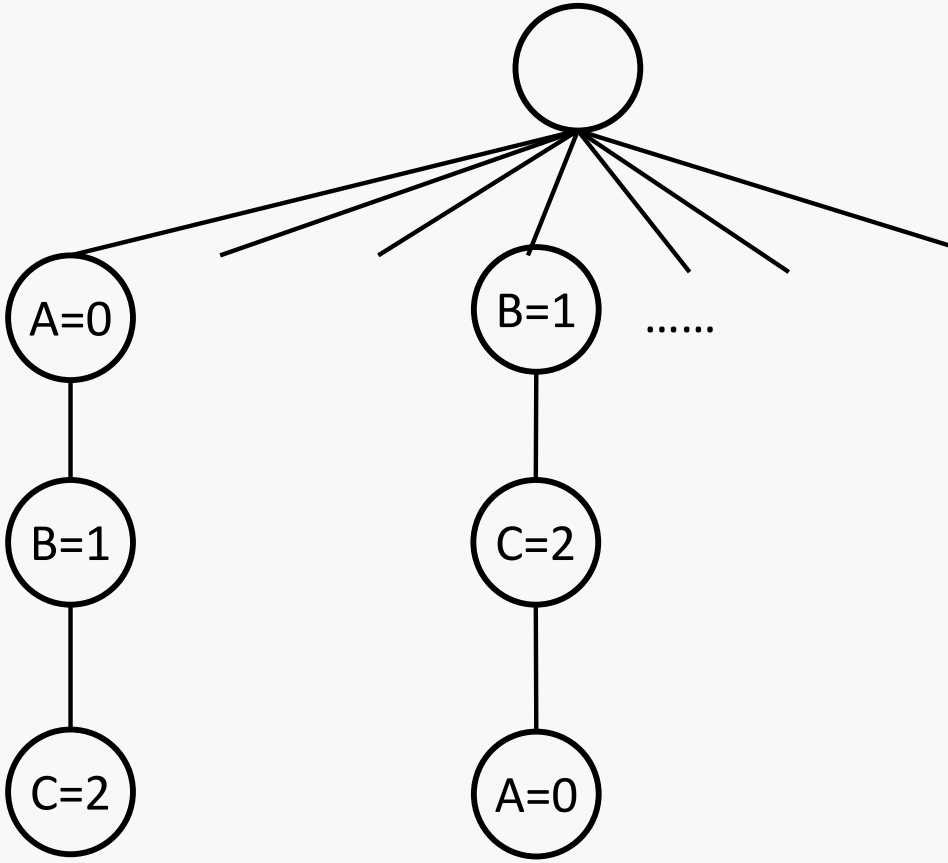
- Example: there are three variables A, B, and C, all with domain {0, 1, 2}. The constraint is A<B<C.
- Check constraints as you go
  - i.e., consider only values which do not conflict previous assignments.
  - may have to do some computation to check the constraints.
  - "incremental goal test".



Backtracking Search

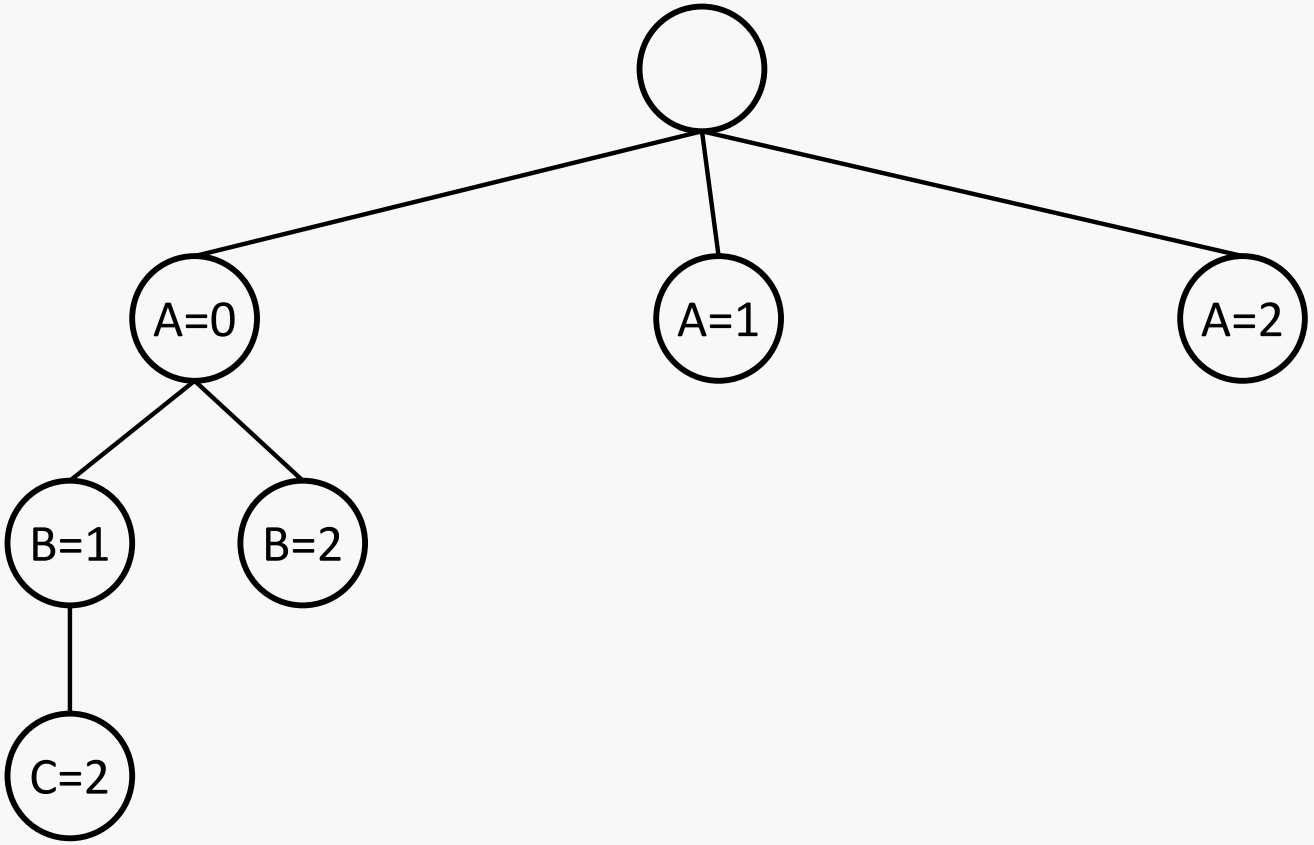
Backtracking is a DFS method with two additional things: 1) Check constraints as you go and 2) Consider one variable at a layer.

- Example: there are three variables A, B, and C, all with domain {0, 1, 2}. The constraint is A<B<C.
- Check constraints as you go
  - i.e., consider only values which do not conflict previous assignments.
  - may have to do some computation to check the constraints.
  - "incremental goal test".
- Consider one variable at a layer
  - Variable assignments are commutative, so fix ordering



Backtracking Example

- Example: there are three variables A, B, and C, all with domain {0, 1, 2}. The constraint is A<B<C.



Improving Backtracking

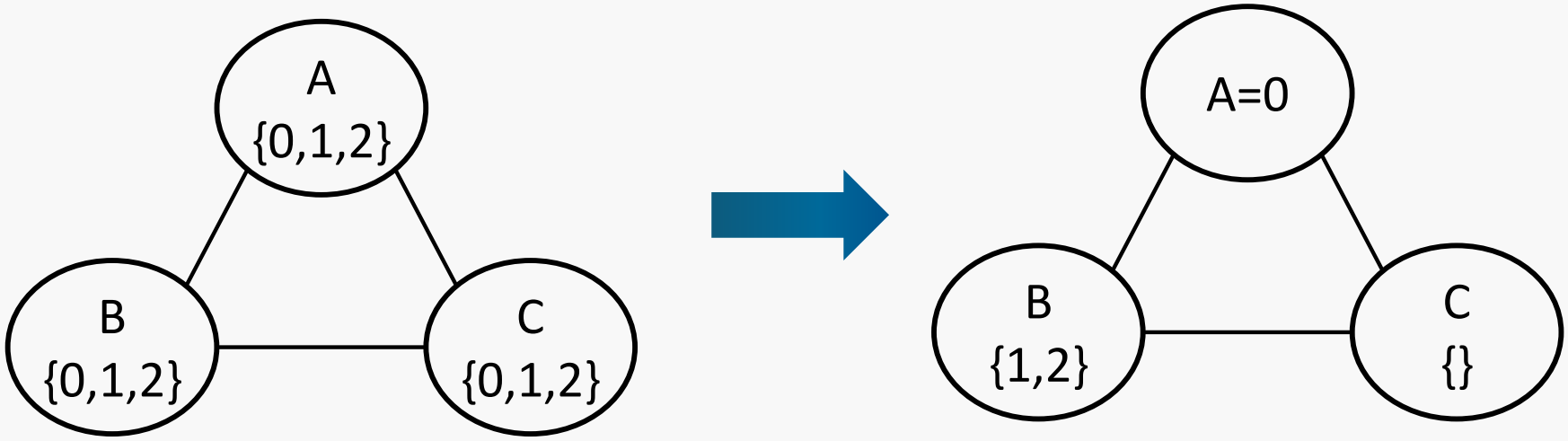
- General-purpose ideas give huge gain in speed
- Two ideas:
  - Filtering: Can we detect inevitable failure early
  - Ordering: Which variable should be assigned next

Filtering

- Keep track of domains for unassigned variables and cross off bad options.
- There are different filtering methods. Forward Checking is one of them.
- Forward Checking: cross off values that violate a constraint when added to the existing assignment. That is, when assign a variable, cross off anything that is now violated on all its neighbours' domains.

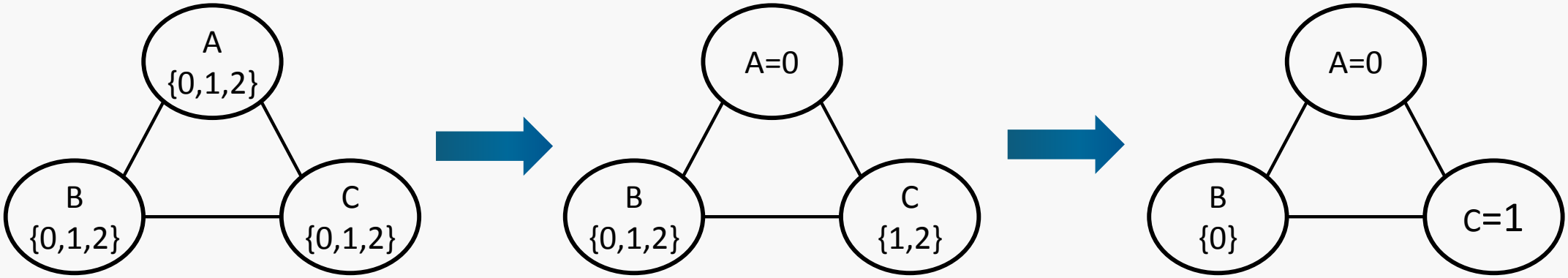
Forward Checking

- Example: There are three variables A, B, and C, all with domain {0, 1, 2}. The constraint is B>A>C.
- When A is assigned 0, domains of its neighbours B and C are reduced, so it is quick to know this assignment is not legal (as C is empty now).



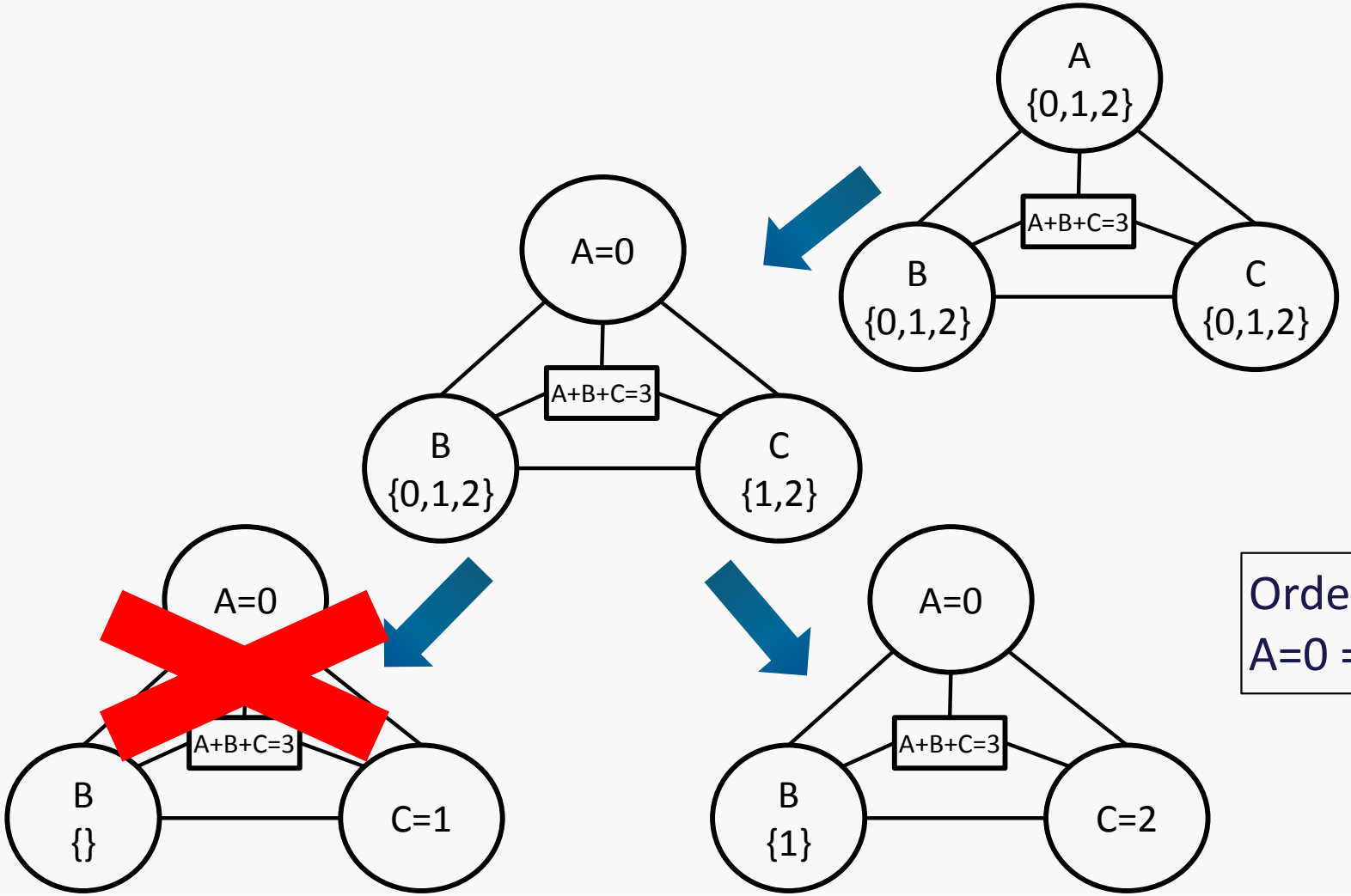
Ordering

- Consider minimum remaining values, i.e., choose the variable with the fewest legal values left in its domain.
- Example: There are three variables A, B, and C, all with domain {0, 1, 2}. The constraint is $A \leq B < C$.
- Once A is assigned 0, after forward checking C will be assigned since its domain is smaller than B's domain.
- Also called "most constrained variable" or "fail-fast" ordering



Example: Backtracking + Forward Checking + Ordering

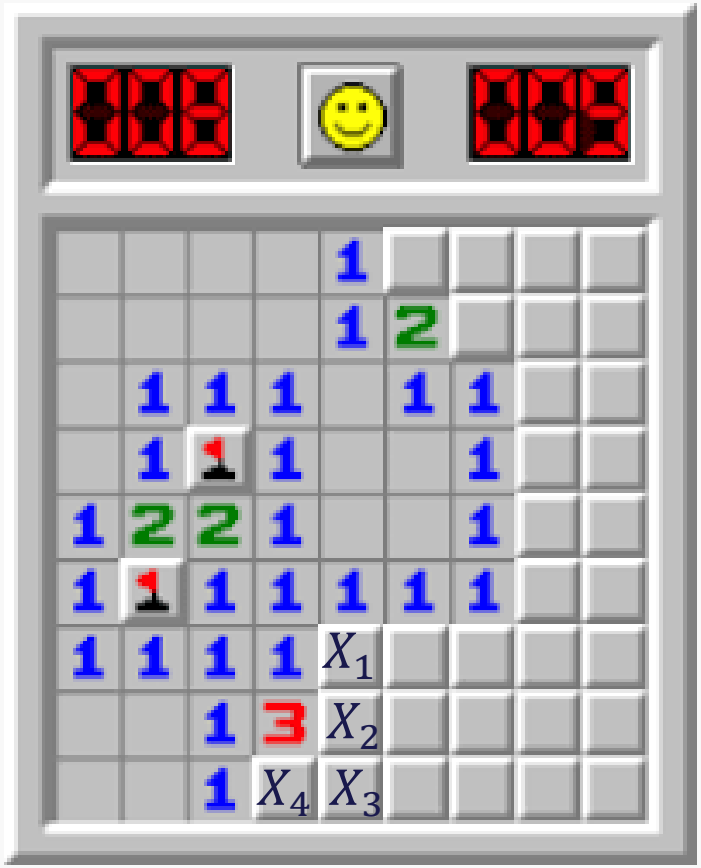
- Example: There are three variables A, B, C, all with domain {0, 1, 2}. The constraints: $A \leq B < C$ and $A + B + C = 3$. Tie is broken alphabetically/numerically.



Order of states to be visited:
A=0 => C=1 => C=2 => B=1

Minesweeper - example

- Variables:
  - X_1, X_2, X_3, X_4
- Domain:
  - $D = \{0, 1\}$, where 0 denotes not a mine and 1 denotes a mine
- Constraints:
  - $X_1 = 1$
  - $X_1 + X_2 = 1$
  - $X_1 + X_2 + X_3 + X_4 = 3$
  - $X_4 = 1$
  - …



Based on the forward checking and ordering, the order of variables to be visited is (tie is broken numerically):

$$X_1 \rightarrow X_2 \rightarrow X_4 \rightarrow X_3$$

Tree Search vs Local Search

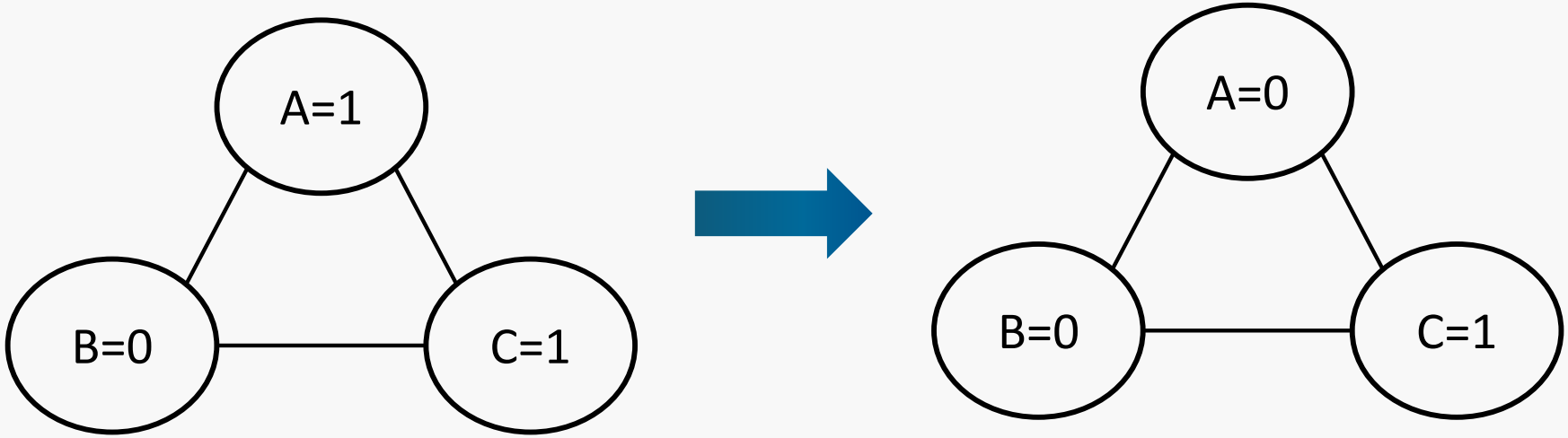
- Tree Search methods: systematically search the space of assignments.
  - Start with an empty assignment.
  - Assign a value to an unassigned variable and deal with constraints on the way until a solution is found.

- But what if the space is too big and even infinite, so in any reasonable time, systematic search may fail to consider enough of the space to give any meaningful results.

- Local Search methods: not systematically search the space but design to find solutions quickly on average
  - Start with an (arbitrary) complete assignment, so constraints can be violated.
  - Try to improve the assignment iteratively.

Local Search for CSPs

- Example: There are three variables A, B, C, all with domain {0, 1, 2}. The constraints: $A \leq B < C$.



- A typical local search algorithm (i.e. hill climbing for CSPs):

Randomly generate a complete assignment

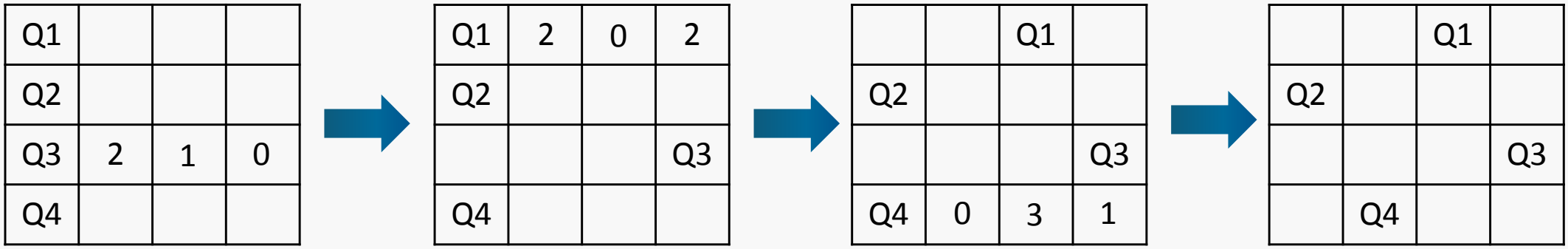
While stop criterion not met

Step 1: Variable selection - randomly select a constraint-violated variable.

Step 2: Value selection (min-conflict heuristic) – choose a value that violates the fewest constraints.

Example: N-Queens

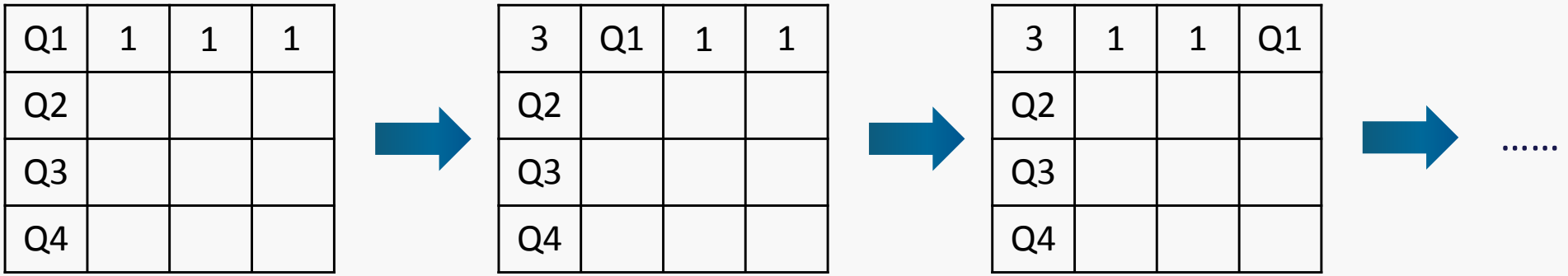
Problem: N-queens puzzle is the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other.



- Formulation – Variables: Q1, Q2, Q3, Q4; Domains: {1,2,3,4}; Constraints: ∀ i, j, not-threatening(Qi and Qj).
- Randomly generate a complete assignment;
- Assume we first consider Q3, then Q3 going to column 4 has fewest constraints violated;
- Assume then we consider Q1, then Q1 going to column 3 has fewest constraints violated;
- Assume next we consider Q4, then Q4 going to column 2 has fewest constraints violated;
- Stop execution and return the solution.

Can Local Search always guarantee finding a solution

- Local search may get stuck in somewhere based on the problem's landscape and search strategy



Strategy like "fixing the queen on the top before fixing the others" can lead to a never-ending search

- But it is effective in practice: can solve the million-queens problem in an average of 50 steps!

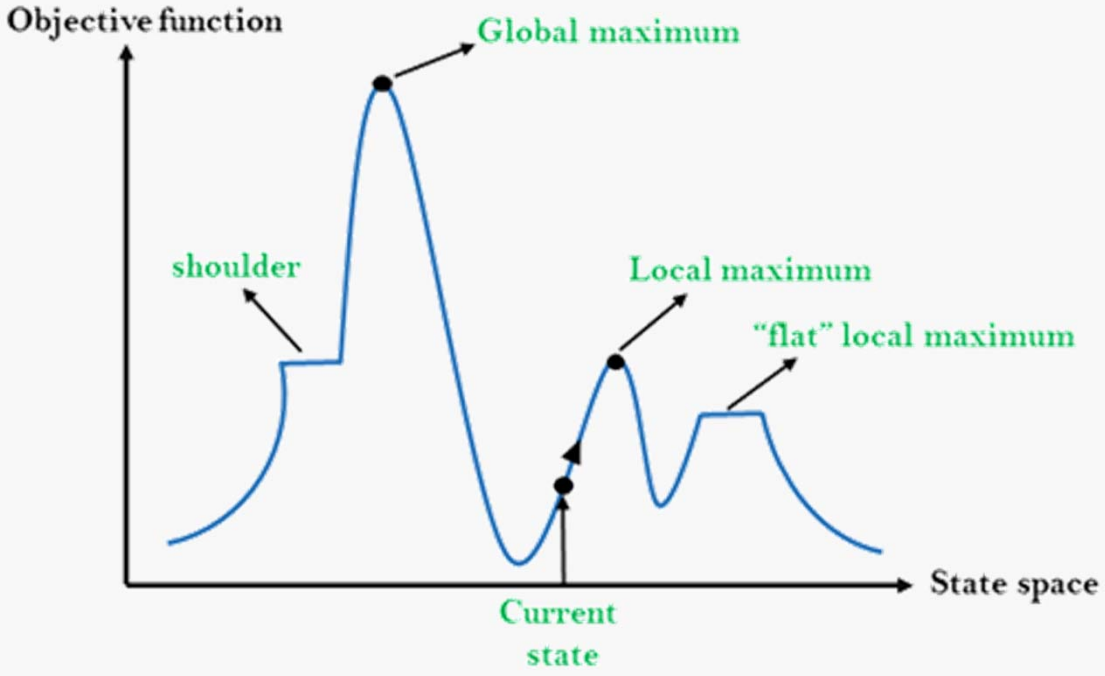
Summary: CSPs

- CSPs are a special class of search problems
  - States are partial assignment
  - Goal test defined by constraints
- Basic strategy: Backtracking
- Speed-ups
  - Filtering: forward checking
  - Ordering
- Local search: not systematically search the space, start with a (bad) complete assignment and improve it iteratively
  - Not only apply to CSPs, but to various optimisation problems

Optimisation

- Optimisation problem: search problem with preferences, i.e. objective function(s).
- They consist of variables, domains, objective function(s).
  - Objectives:
    - Single-objective optimisation problems, e.g., Travelling Salesman Problem (TSP): minimising the cost of the travelling.
    - Multi-objective optimisation problems, e.g., TSP with an additional objective: minimising the time of the travelling.
  - Constraints:
    - Unconstrained optimisation problems.
    - Constrained optimisation problems.
- Tree search methods may not work for optimisation problems, e.g. in some continuous search space.
- Local search methods can be effective for optimisation problems.

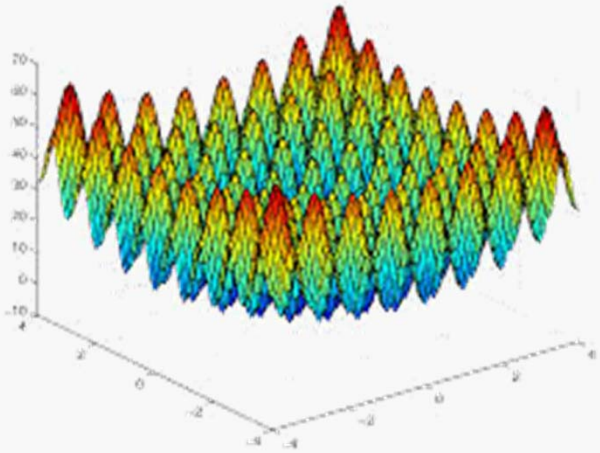
Local Search for Optimisation

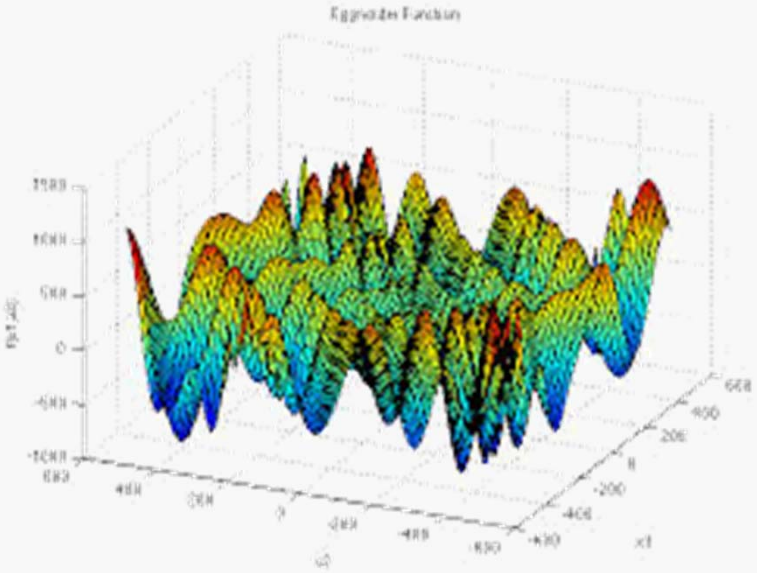


- Generally fast and memory efficient.
- Can deal with problems where the search state is difficult to represent/formulate.
- Can be used in an online setting when the problem changes, e.g., in airline scheduling problem.

Local search methods

- Hill climbing, e.g. gradient descent.
- Simulated annealing, tabu search (keep a small list of recently visited solutions and forbid the algorithm to return to those solutions).
- Population-based local search: evolutionary computation (e.g., genetic algorithms).





Population-based search can be helpful in jumping out of local optima.