CPSC 322: Introduction to Artificial Intelligence

CSPs: Domain Splitting and Local Search

Textbook reference: [4.5, 4.7]

Instructor: Varada Kolhatkar
University of British Columbia

Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Announcements

- Assignment 2 has been released and is due on **21 Oct 11:59pm**.

- Midterm **practice questions** are available on Piazza.

- Midterm time and location
  Time: Friday, Oct 25th, from 6pm to 7pm
  Location: Woodward 2
  (Instructional Resources Centre-IRC) (WOOD) - 2

- My office hours: Fridays from 11am to noon in ICCS 185

Lecture outline

- Recap arc consistency (~5 mins) 👈🏻
- Solving CSPs
  - Domain splitting + Arc consistency (~20 mins)
  - Local Search (~40 mins)
- Summary and wrap up (~5 mins)

Solving CSPs

- Generate-and-Test
  Time complexity: $O(cd^k)$

$k \rightarrow$ **# variables**
$d \rightarrow$ **domain size**
$c \rightarrow$ **# constraints**

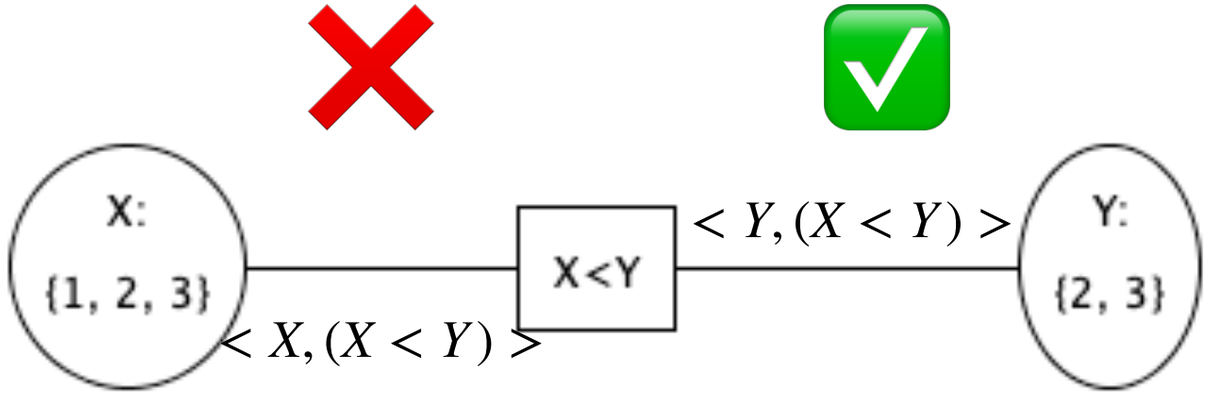
- Graph search (DFS with backtracking)
  Time complexity: $O(d^k)$

- Arc consistency
  Time complexity: $O(cd^3)$

Arc consistency

Definition: An arc $< X, r(X, Y) >$ is **arc consistent** if for each value $x \in dom(X)$, there is some value $y \in dom(Y)$, such that $r(x, y)$ is satisfied.

❌ ✅



Arc $< X, (X < Y) >$ is not consistent because no value in $dom(Y)$ that satisfies $X < Y$ for $X = 3$.

Arc $< Y, (X < Y) >$ is consistent.

Enforcing arc consistency

How can we make a constraint network arc consistent?
Arc consistency algorithm

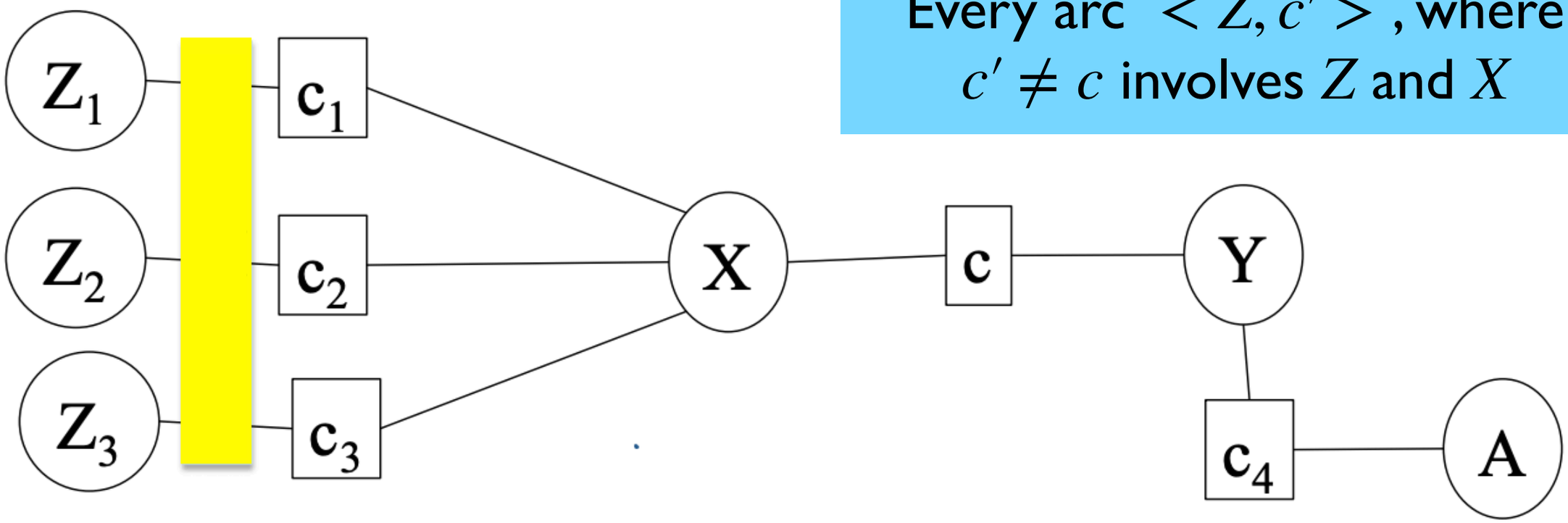
- In order to make an arc consistent we prune the domain of a variable that makes it inconsistent.

- As a consequence of that, some other arcs in the network become inconsistent.

Which arcs need to be considered?

When we reduce the domain of a variable X to make an arc $< X, c >$ arc consistent, which arcs do we need to consider?

Every arc $< Z, c' >$, where $c' \neq c$ involves Z and X



Arc consistency algorithm: Interpreting outcomes

Three possible outcomes (when all arcs are arc consistent):

Each domain has a single value.
We have a (unique) solution.
E.g., Variables = {A, B, C} with domains {1,2,3}; constraints: A < B, B <C

At least one domain is empty.
No solution! All values are ruled out for this variable.
E.g., Variables = {A, B, C}; dom{A} = {1}, dom(B) = {1,2}, dom(C) = {1,2};
constraints: A = B, B = C, A ≠ C

Some domains have more than one value
There may be a solution, multiple ones, or none. Need to solve this new CSP (usually simpler) problem: same constraints, domains have been reduced
E.g., built-in example "simple problem 2"

Domain splitting (case analysis)

- Arc consistency ends: Some domains have more than one value → may or may not have a solution
  - Apply Depth-First Search with Pruning or
  - Split the problem in a number of disjoint cases
- Solution to CSP is the **union** of solutions to these disjoint cases

Example:
CSP with $dom(X) = \{x_1, x_2, x_3, x_4\}$ becomes
CSP_1 with $dom(X) = \{x_1, x_2\}$
CSP_2 with $dom(X) = \{x_3, x_4\}$
Solution is the union of solutions of CSP_1 and CSP_2

Today: Learning outcomes

From this lecture, students are expected to be able to:

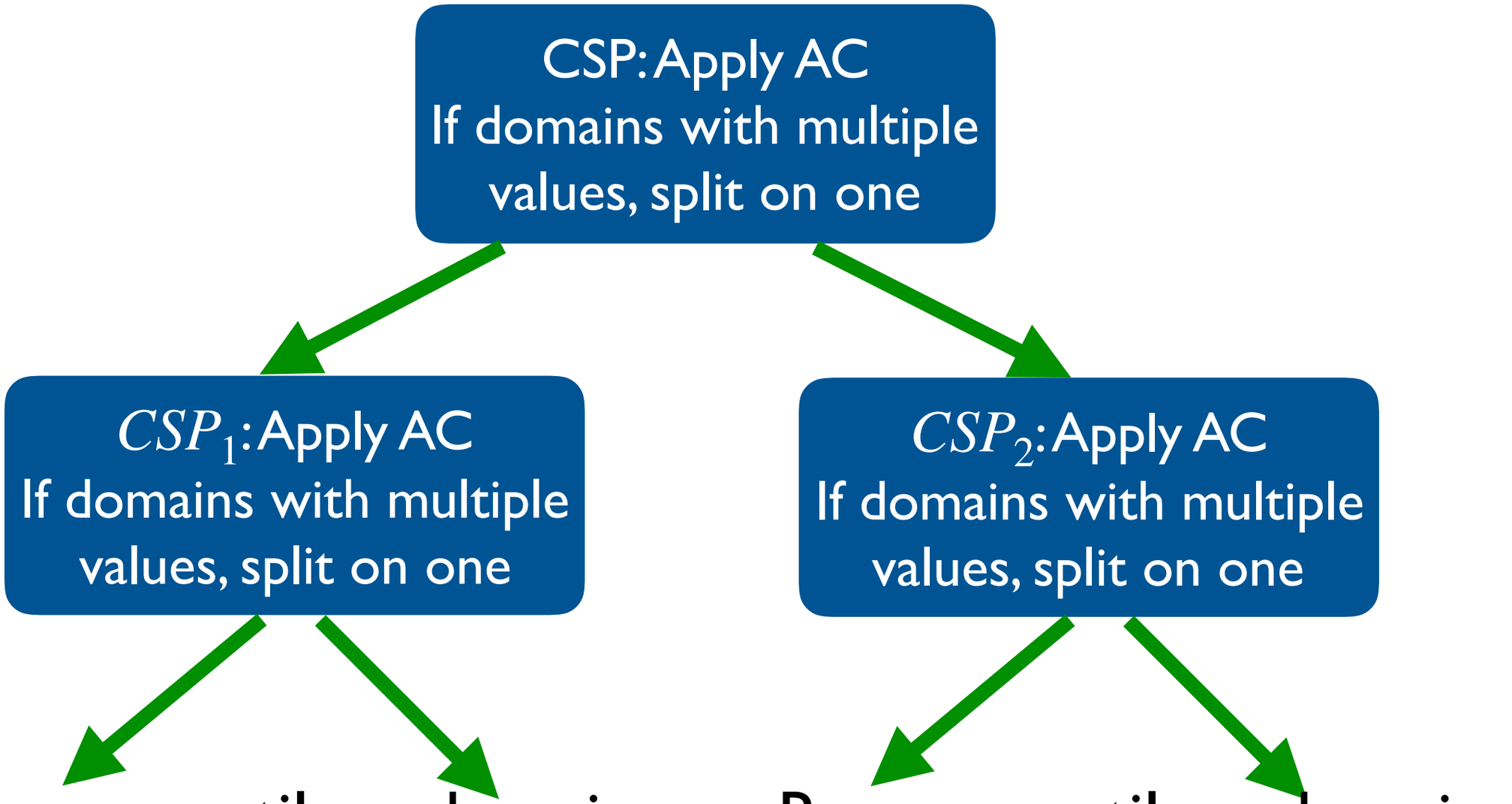
- Define/read/write/trace/debug domain splitting and its integration with arc consistency.
- Implement local search for a CSP.
  - Implement different ways to generate neighbours
  - Implement scoring functions to solve a CSP by local search through either greedy descent or hill-climbing.

Domain splitting in action

Trace it on “simple problem 1”

Searching by domain splitting

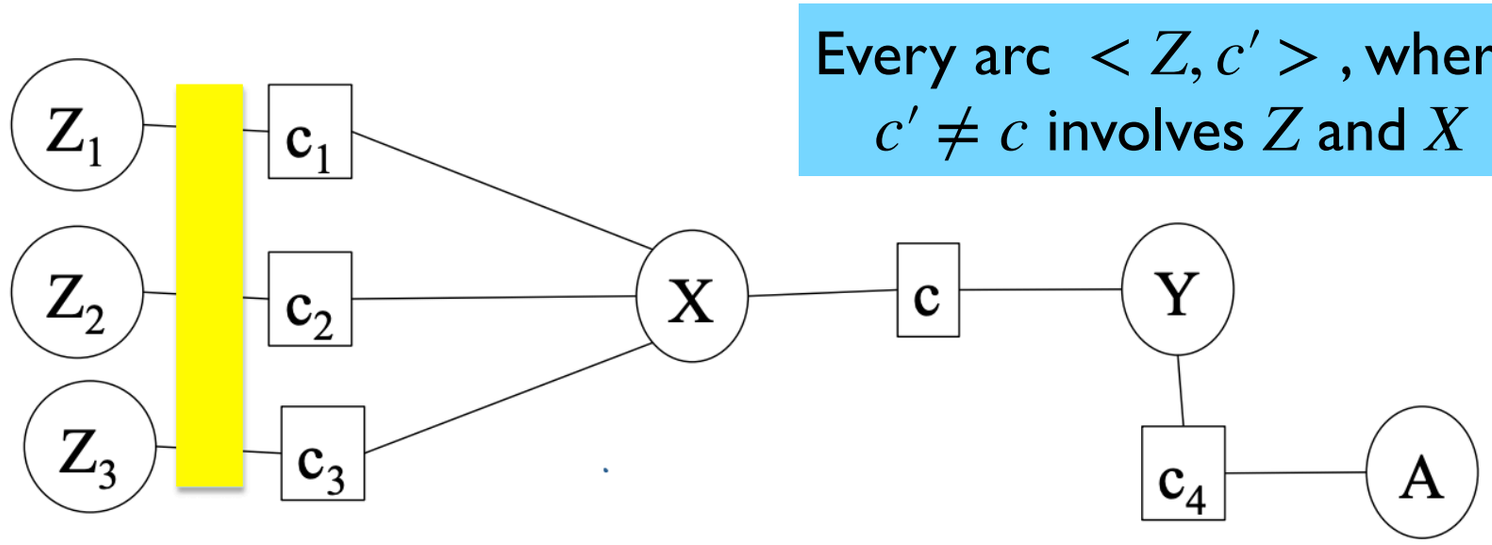
How many CSPs do we need to keep around at a time? With depth m and 2 children at each split: $O(2m)$. It's a DFS.



Domain splitting

Each smaller CSP is easier to solve

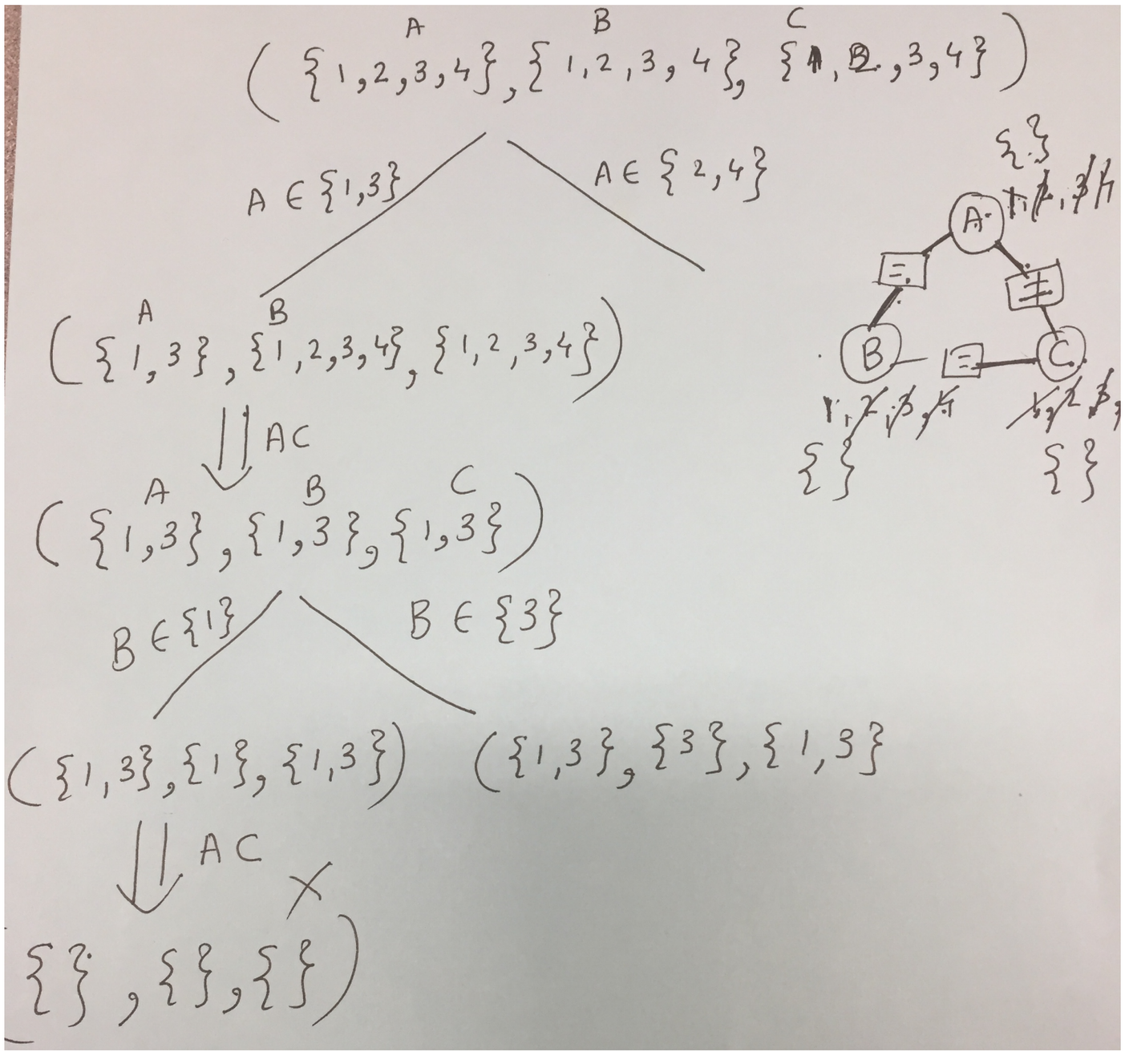
- Arc consistency might already solve it
- For each subCSP, which arcs have to be on the ToDoArcs list when we get the subCSP by splitting the domain of X?



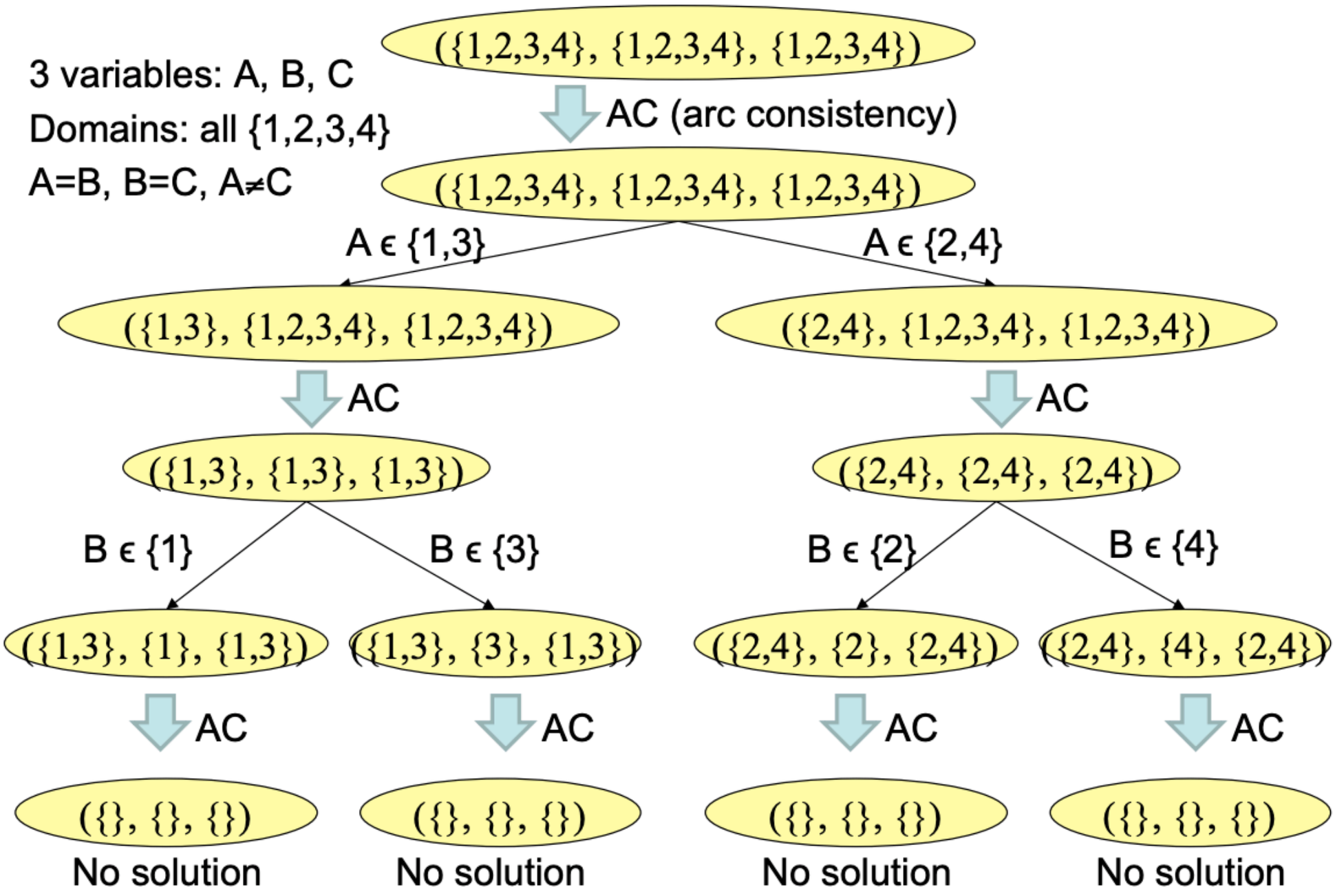
Arc consistency + domain splitting example

Toy problem

Variables: A, B, C; Domains: $\{1,2,3,4\}$; Constraints: $A = B, B = C, A \neq C$

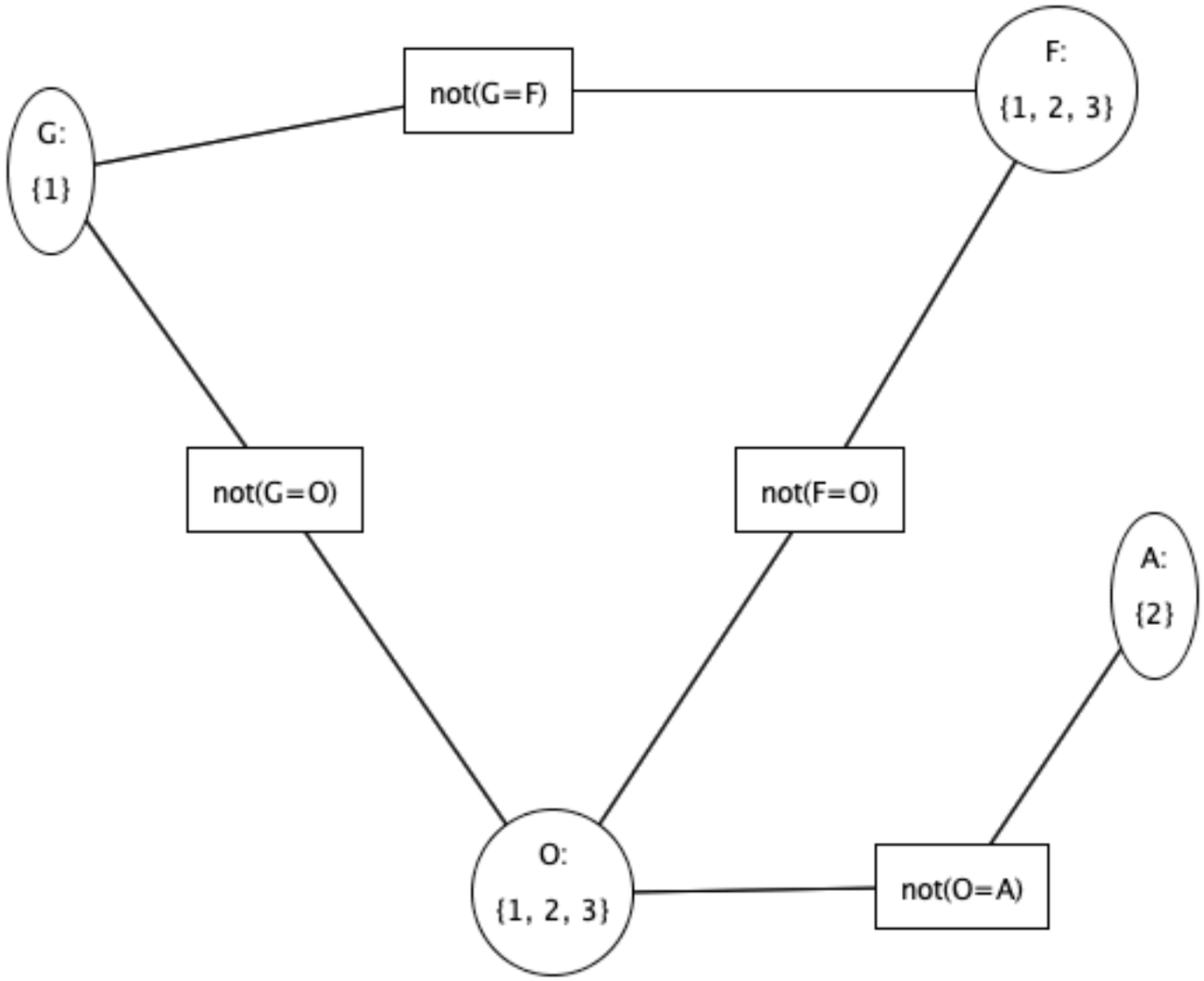


Arc consistency + domain splitting example



Take home activity: arc consistency

Trace the arc consistency algorithm on the network. Show at least 4 to 6 iterations. For each iteration, show TDA and domain values.

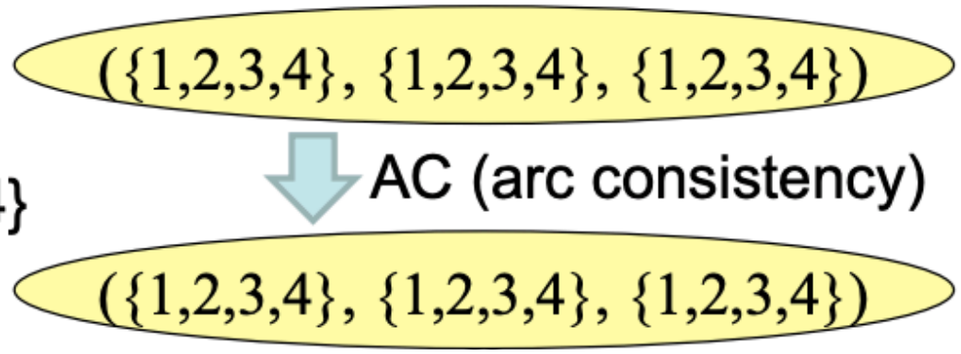


Take home activity: domain splitting

3 variables: A, B, C

Domains: all {1,2,3,4}

A=B, B=C, A=C



Third formulation of CSP as search

Arc consistency with domain splitting

- States: vector $(D(V_1), \ldots, D(V_n))$ of remaining domains, with $D(V_i) \subseteq dom(V_i) \forall V_i$

- Start state: vector of original domains $(dom(V1), \ldots, dom(Vn))$

- Successor function: reduce one of the domains + run arc consistency → new CSP

- Goal state: vector of unary domains that satisfies all constraints. That is, only one value left for each variable. The assignment of each variable to its single value is a model.

- Solution: that assignment

Local Search

Local search motivation: Scale

- Many CSPs (scheduling, DNA computing, etc.) are simply too big for systematic approaches
- If you have 10^5 variables with $dom(V_i) = 10^4$
- Time complexity with DFS: ?
- Time complexity with arc consistency: ?

Local search motivation: Scale

- Many CSPs (scheduling, DNA computing, etc.) are simply too big for systematic approaches

- If you have 10^5 variables with $dom(V_i) = 10^4$

- Time complexity with DFS: $(10^4)^{(10^5)}$

- Time complexity with arc consistency: $(10^{10}) \times (10^{12})$

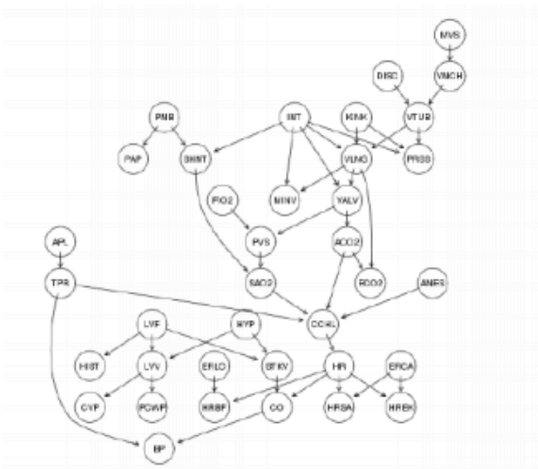
Alternative: **Local search!**
Often finds solutions quickly but
cannot prove that there is no solution.

Local search: Motivation

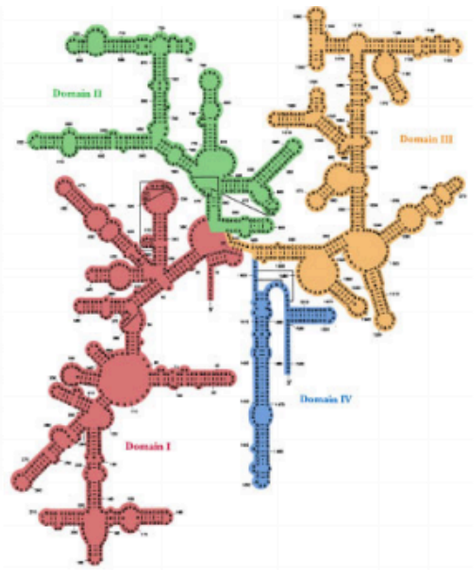
Useful method in practice!

- Best available method for many constraint satisfaction and constrained optimization problems
- Extremely general!
  - Works for problems other than CSPs
  - E.g., arc consistency only works for CSPs

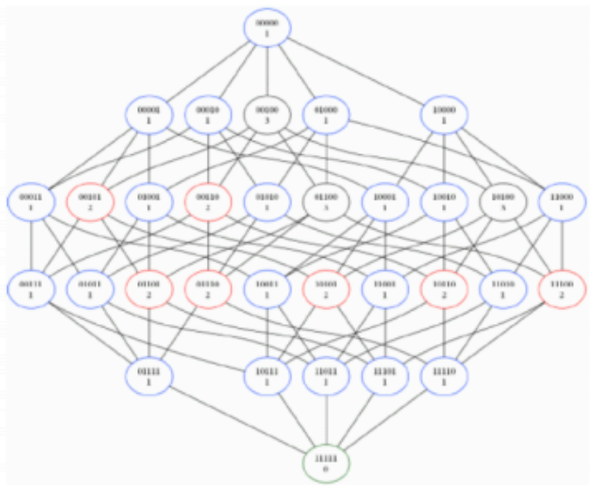
Some successful applications of local search



Probabilistic Reasoning

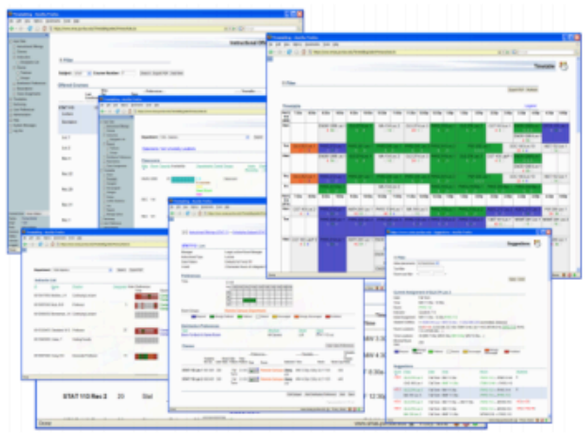


RNA structure design

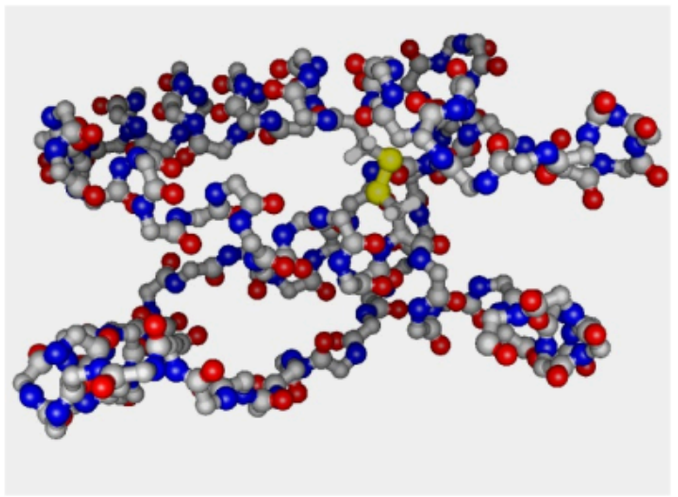


Propositional satisfiability (SAT)

Scheduling of Hubble Space Telescope:
1 week → 10 seconds



University Timetabling



Protein Folding

Local search: General idea

- Start with a **possible world**

- Repeat: move to the "best" **neighbouring state**

- If no neighbours better than current (**scoring function**), quit

Remember, for CSP a solution is a possible world, not a path

Credit: Adapted from Berkeley AI course material

Local search problem

A **local search** problem consists of a:

CSP: a set of variables, domains for these variables, and constraints on their joint values. A node in the search space will be a complete assignment to all of the variables.

Neighbour relation: an edge in the search space will exist when the neighbour relation holds between a pair of nodes.

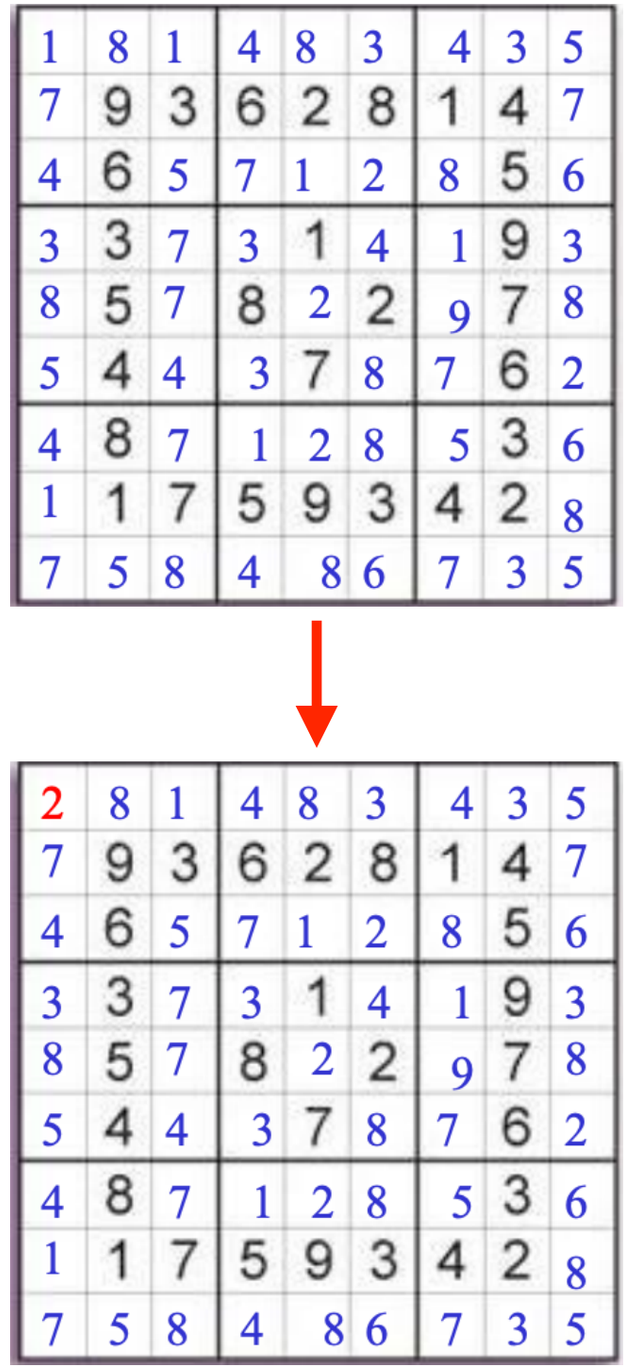
Scoring function: $h(n)$, judges cost of a node (want to minimize).
- E.g., the number of constraints violated in node n
- E.g., the cost of a state in an optimization context.

Example: Sudoku as a local search problem

CSP: usual Sudoku CSP
Variables: One variable per cell;
domains: {1,…,9};
Constraints: each number occurs once per row, per column, and per 3x3 box.

Neighbour relation: value of a single cell differs

Scoring function: number of constraint violations

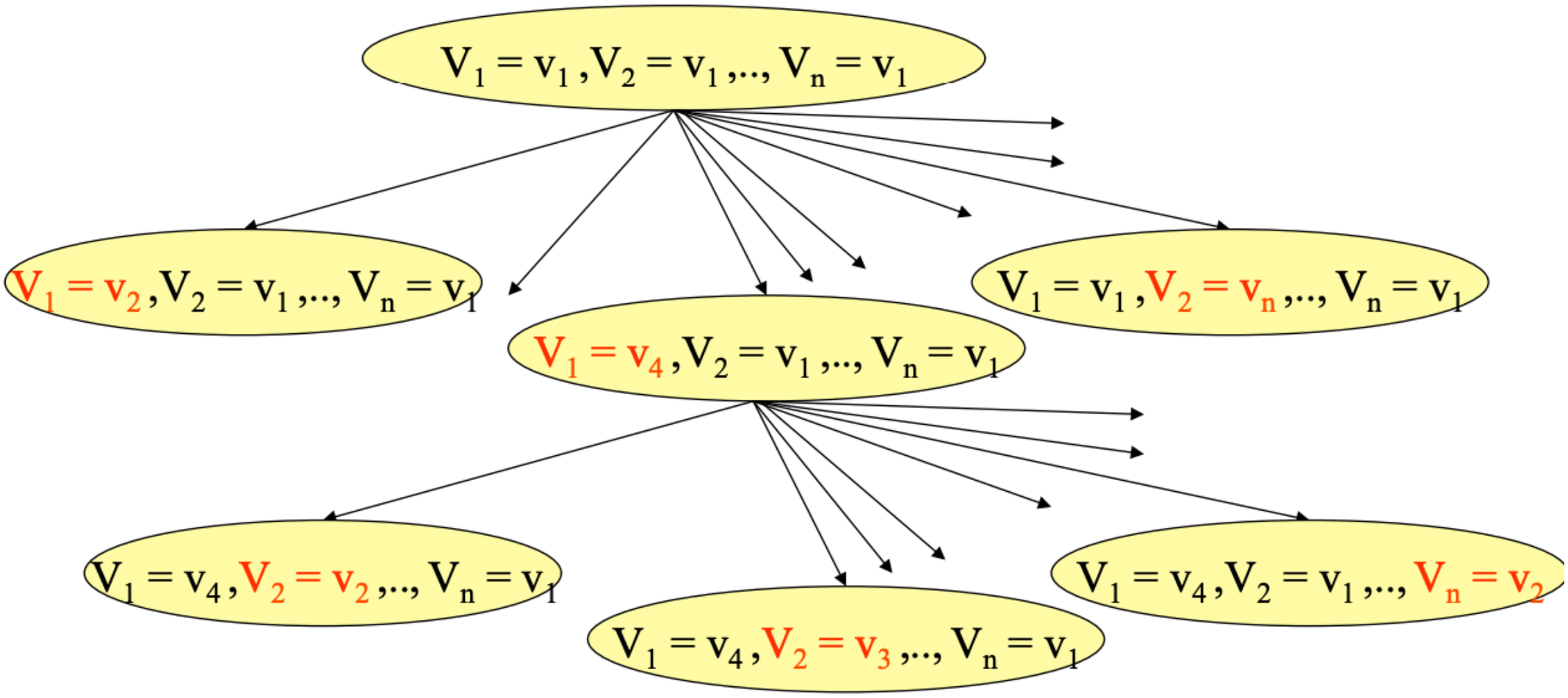


How do we determine neighbours

Neighbours of a current node are **similar variable assignments**. One way to do this is making some small incremental change to the variable assignment.

- assignments that differ in one variable's value, by (for instance) a value difference of +1
- assignments that differ (by any amount) in one variable's value
- assignments that differ in two variables' values, etc.

Search space for local search



- Only the current node is kept in memory at each step.

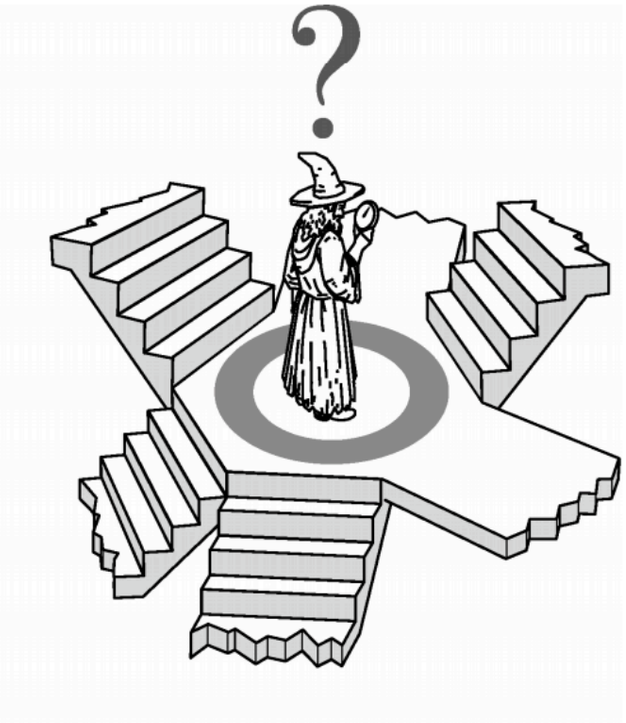
- Very **different from** the **systematic tree search** approaches we have seen so far! **Local search does NOT backtrack**!

Determining the "best" neighbour

- We select the neighbour that most improves some **scoring** or **evaluation** function.

- In CSPs a common evaluation function is the number of constraints that are violated (**conflicts)**.

- If there are several possible neighbours that most improve the evaluation function, one is chosen at random.



Selecting the "best" neighbour

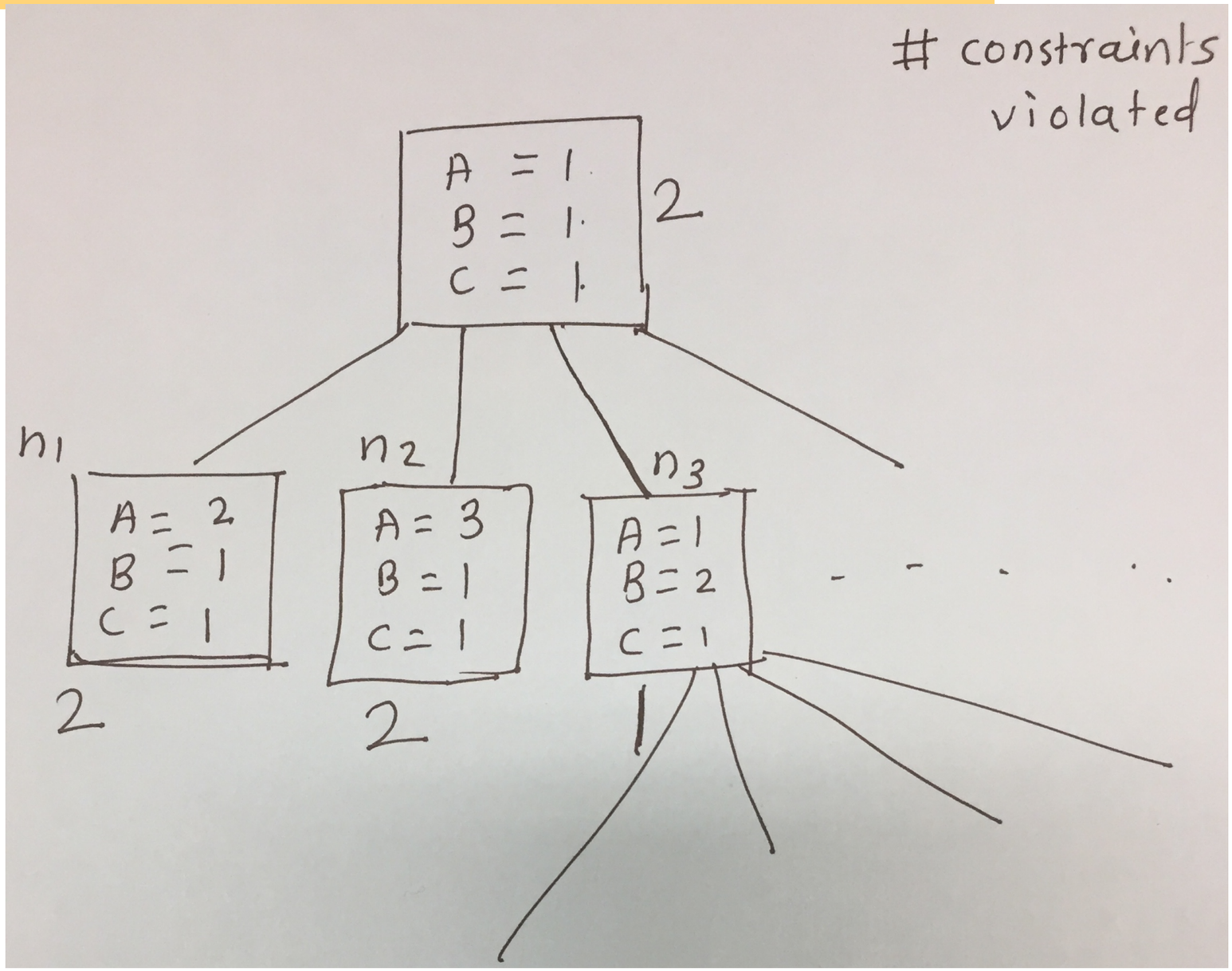
Example:
Variables: A,B,C; same domain {1,2,3};
Constraints: A<B, B<C;
Suppose we start with {A=1, B=1, C=1}

Neighbours?

Best neighbours?

Selecting the "best" neighbour

Example: Variables: A,B,C; same domain {1,2,3}; Constraints: A<B, B<C;
Suppose we start with {A=1, B=1, C=1}



Iterative best improvement

Which strategy would make sense to select the "best" neighbour in local search?

A. Maximal number of constraint violations

B. Similar number of constraint violations as current state

C. Minimal number of constraint violations ✅

Determining the "best" neighbour

- **Iterative best improvement**: Select the neighbour that optimizes some scoring/evaluation function $h(n)$.

  - **Greedy descent**: Evaluate $h(n)$ for each neighbour, pick the neighbour n with **minimal** $h(n)$

- **Hill climbing**: Evaluate $h(n)$ for each neighbour, pick the neighbour n with **maximum** $h(n)$

- Note that Minimizing $h(n)$ is identical to maximizing $-h(n)$

General local search algorithm

```
Procedure Local-Search(V,dom,C)
        Inputs
                V: a set of variables
                dom: a function such that dom(X) is the domain of variable X
                C: set of constraints to be satisfied
        Output
                complete assignment that satisfies the constraints
        Local
                A[V] an array of values indexed by V

1:      repeat
2:              for each variable X do
3:                      A[X] ←a random value in dom(X);
4:
5:              while (stopping criterion not met & A is not a satisfying assignment):
6:                      select a variable Y and a value V ∈dom(Y)
7:                      set A[Y] ←V
8:              if (A is a satisfying assignment) then
9:                      return A
10:
11:     until termination
```

Random initialization

Local search step

General local search algorithm

```
Procedure Local-Search(V,dom,C)
        Inputs
                V: a set of variables
                dom: a function such that dom(X) is the domain of variable X
                C: set of constraints to be satisfied
        Output
                complete assignment that satisfies the constraints
        Local
                A[V] an array of values indexed by V

1:      repeat
2:              for each variable X do
3:                      A[X] ←a random value in dom(X);
4:
5:              while (stopping criterion not met & A is not a satisfying assignment):
6:                      select a variable Y and a value V ∈dom(Y)
7:                      set A[Y] ←V
8:              if (A is a satisfying assignment) then
9:                      return A
10:
11:     until termination
```

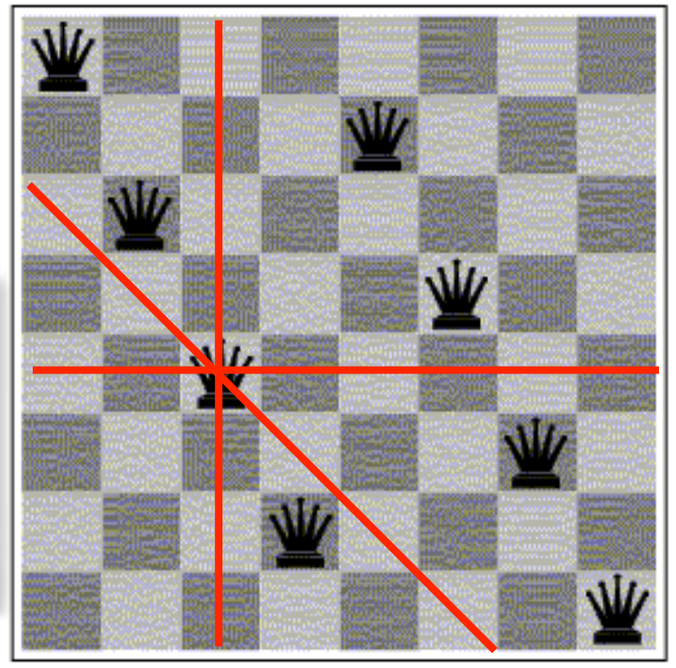
Random initialization

Based on local information. E.g., for each neighbour evaluate how many constraints are unsatisfied. Greedy descent: select Y and V to minimize #unsatisfied constraints at each step

Example: N-Queen problem

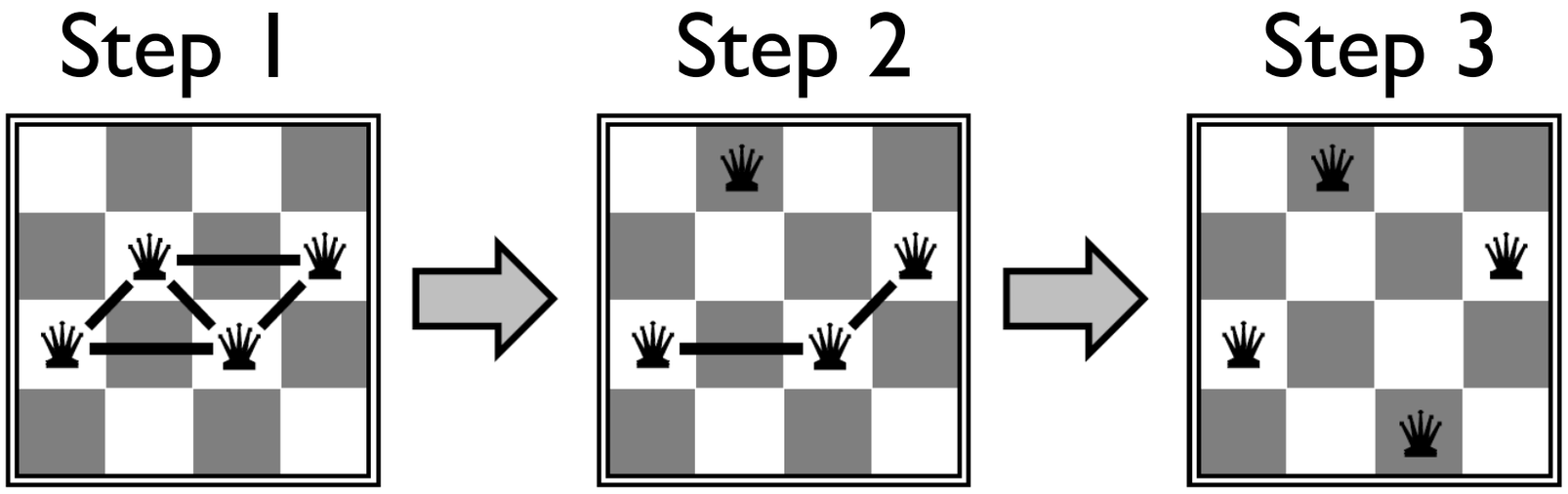
Put N queens on an $N \times N$ board with no two queens on the same row, column, or diagonal (i.e., attacking each other)

Positions a queen can attack



Example: 4-Queen problem

CSP: 4-Queen CSP
One variable per column;
Domains {1,2,3,4}: row where the queen in the i^{th} column sits;
Constraints: no two queens in the same row, column or diagonal
Neighbour relation: value of a single column differs
Scoring function: number of attacks



8-Queen problem: How many neighbours?

CSP: 8-Queen CSP; One variable per column;
Domains {1,…,8}: row where the queen in the i^{th} column sits;
Constraints: no two queens in the same row, column or diagonal
Neighbour relation: value of a single column differs
Scoring function: number of attacks

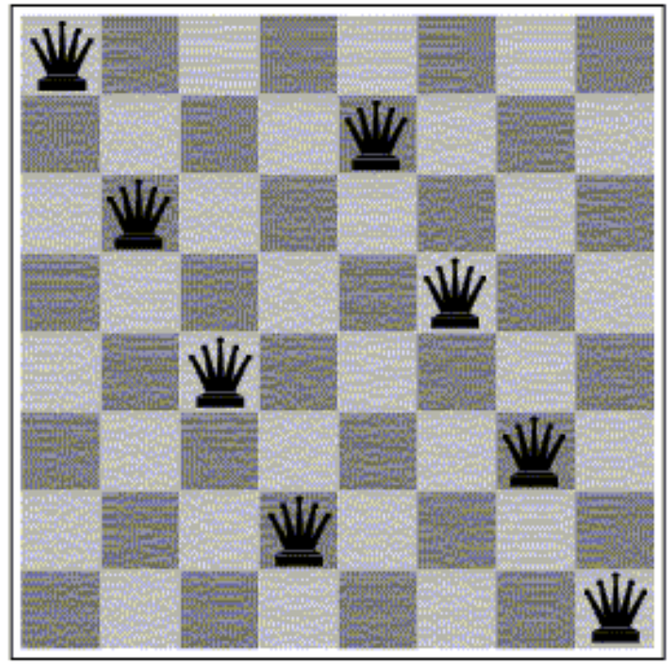
How many neighbours at a given state?

A. 100

C. 56 ✅

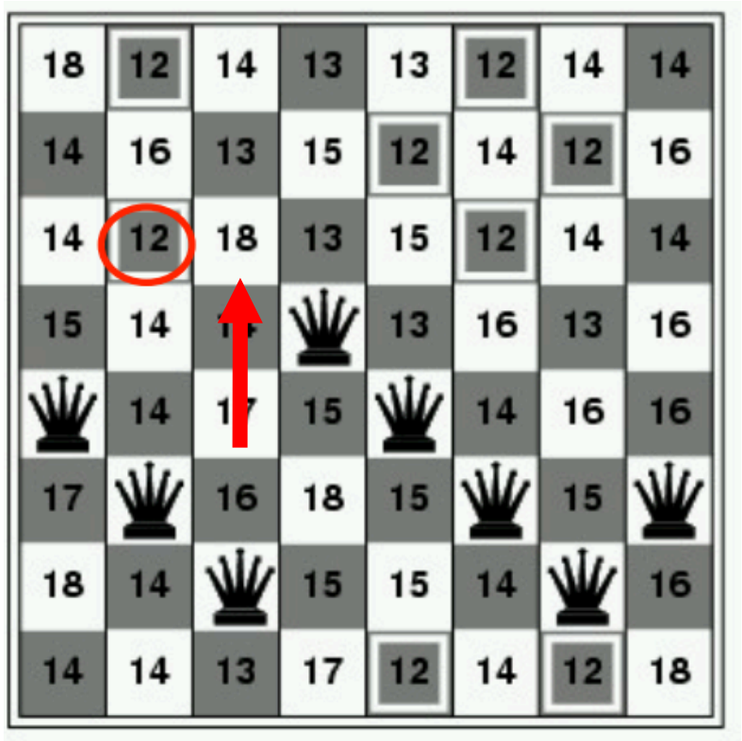
B. 8

D. 7

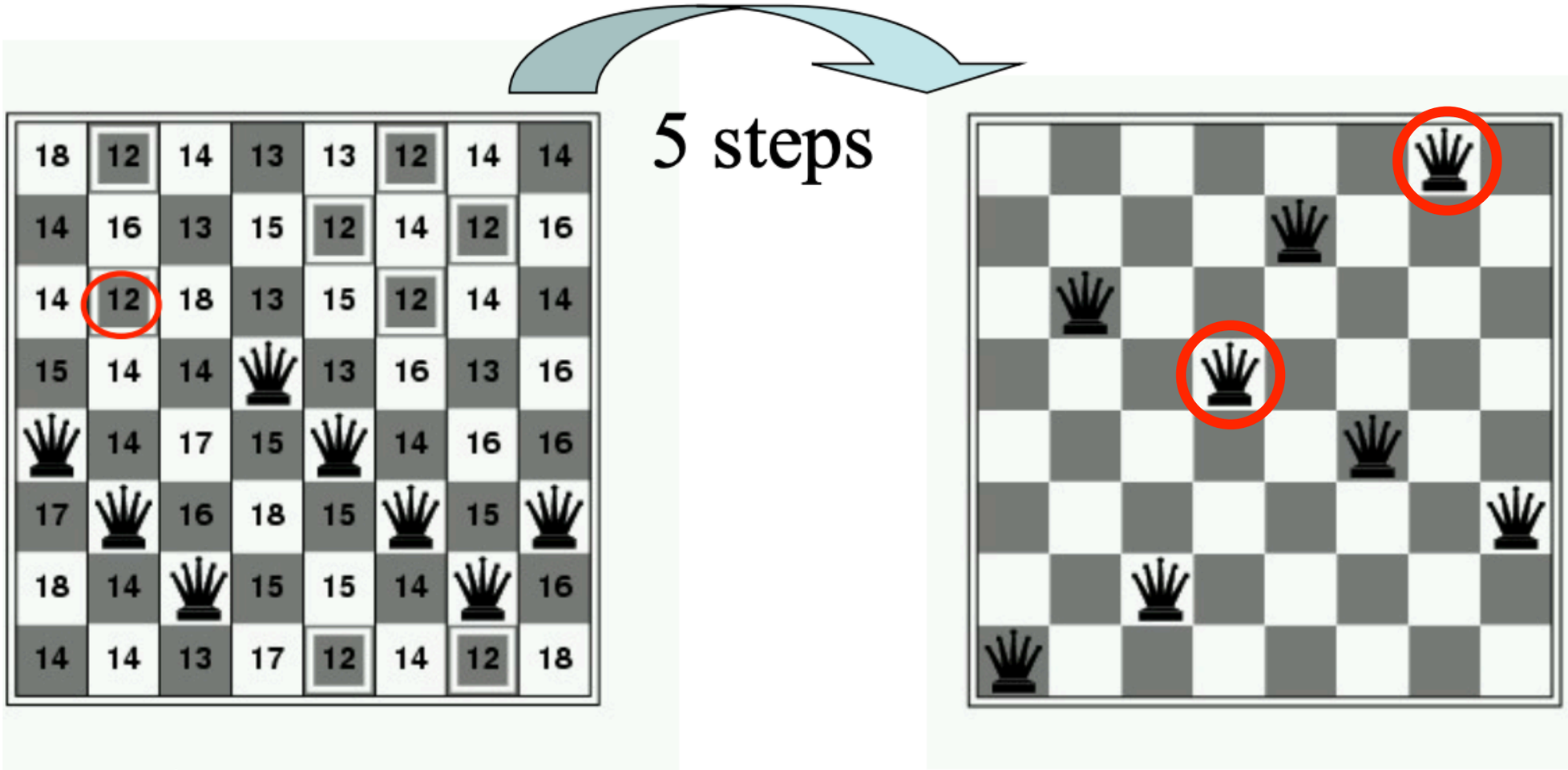


Example: Local search for N-Queen problem

For each column, assign randomly each queen to a row
(a number between 1 and N)
Repeat
For each column & each number: Evaluate how many constraint violations changing the assignment would yield.
Choose the column and number that leads to the fewest violated constraints; change the assignment
Until solved



Example: 8-Queen problem



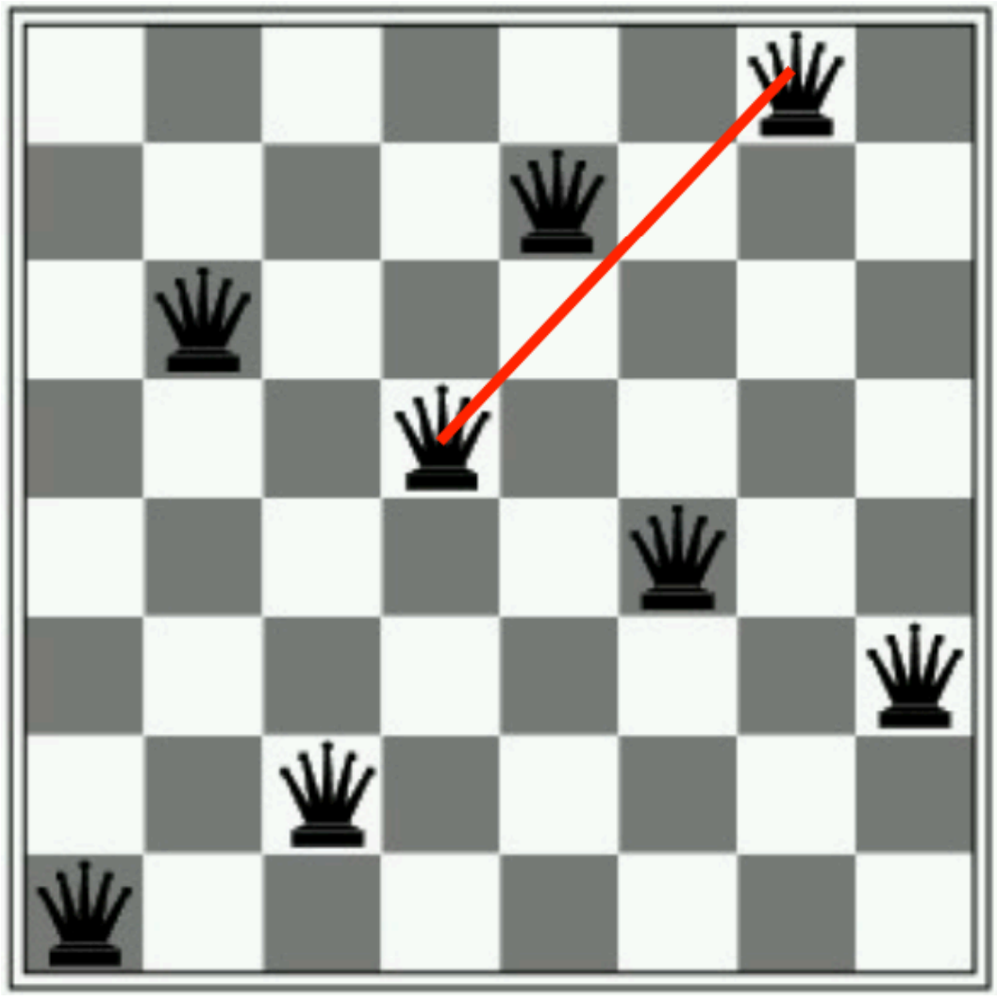
h = 17

h = 1

Each cell lists h (i.e. #constraints unsatisfied) if you move the queen from that column into the cell

Local minima

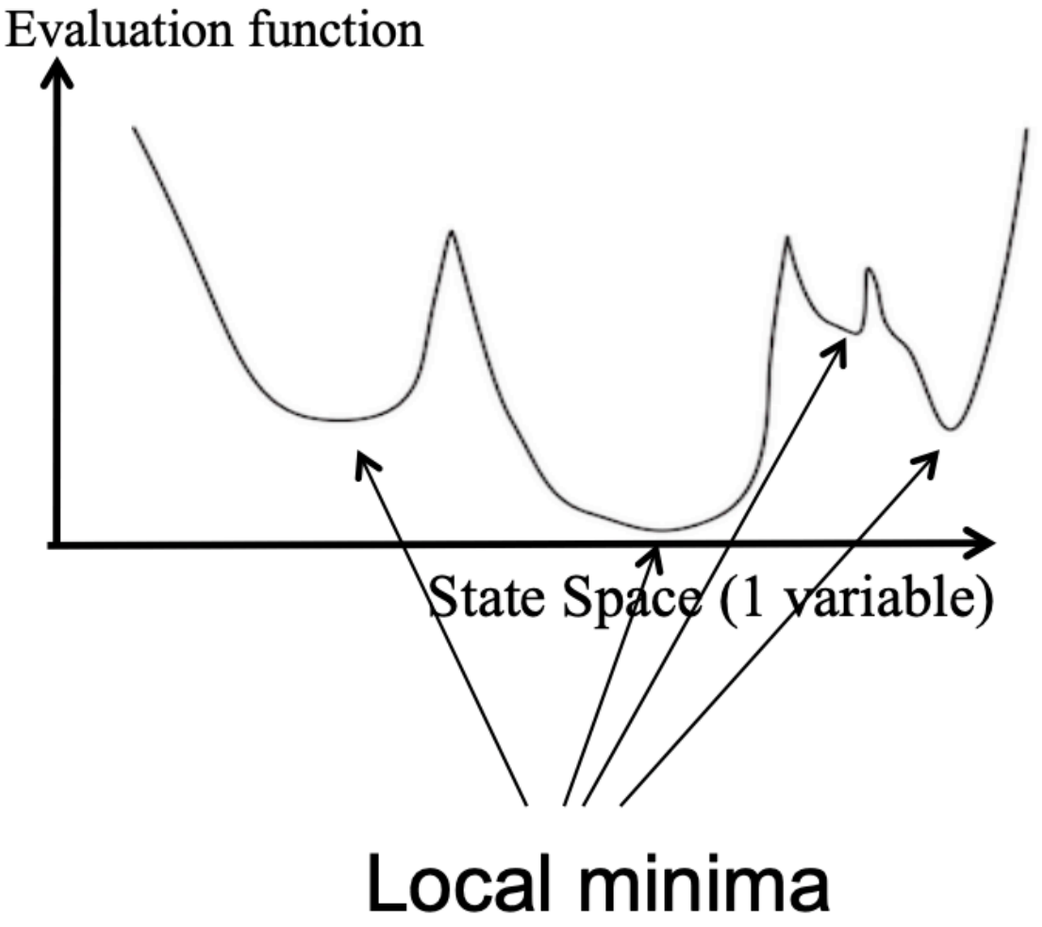
- Which move should we pick in this situation?
- Current cost: h=1
- No single move can improve on this
- In fact, every single move only makes things worse (h ≥ 2)
- Locally optimal solution. Since we are minimizing: local minimum



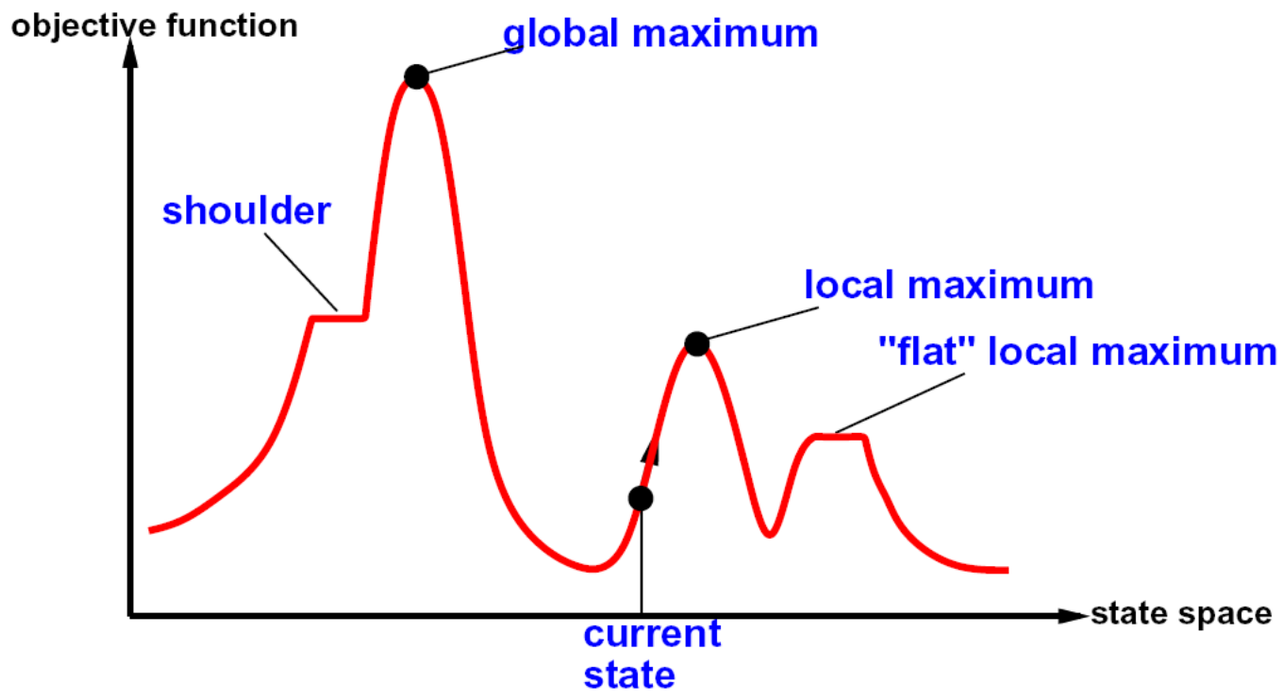
Local minima

Most research in local search concerns effective **mechanisms for escaping from local minima.**

Want to quickly explore many local minima: global minimum is a local minimum, too.



A one-dimensional state-space landscape

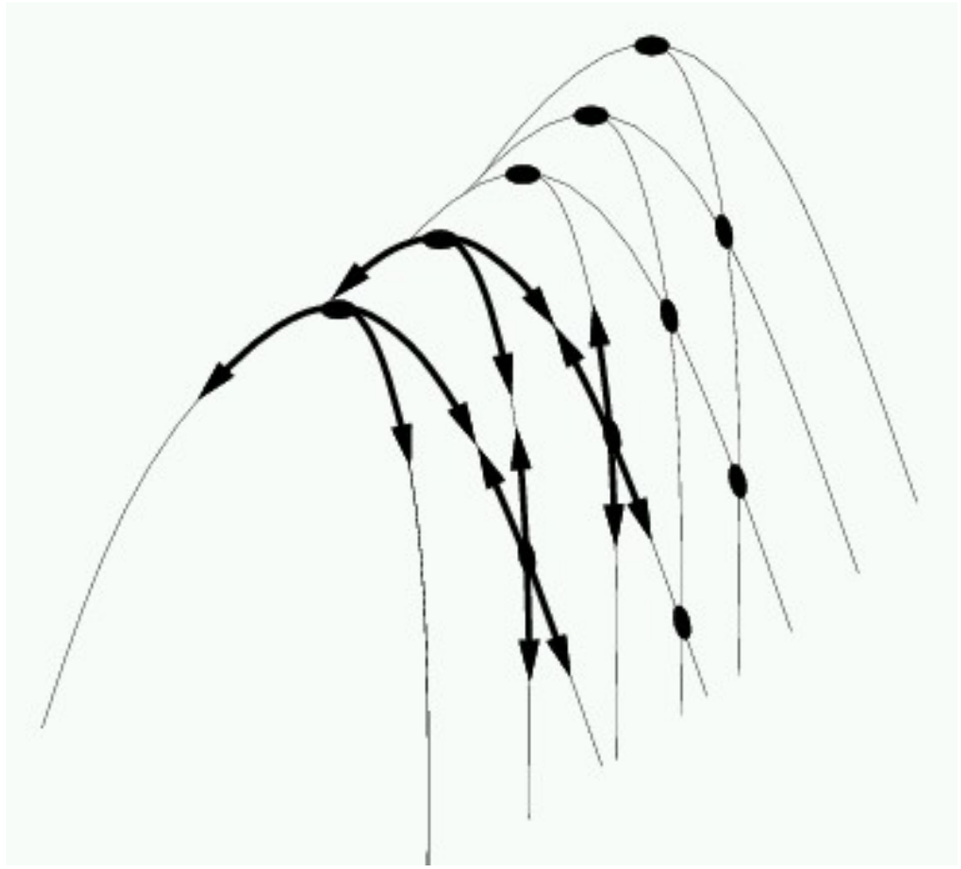


- Local maxima
- Plateaus and shoulders

Even more problems in higher dimensions

- Ridges: sequence of local maxima not directly connected to each other

- From each local maximum you can only go downhill.



Different neighbourhoods

- Local minima are defined with respect to a neighbourhood.

- Neighbourhood: states resulting from some small incremental change to current variable assignment

- Using larger neighbourhoods means fewer local minima and better chances to get to the optimal solution but time required to determining improving search steps is higher

Local search: Summary

- Often finds a solution **quickly**
- But cannot prove that there is no solution
- Useful method in practice
- Best available method for many constraint satisfaction and constrained optimization problems
- Extremely general and works for problems other than CSPs. (In contrast, arc consistency only works for CSPs.)

Coming up

Readings for next class

- 4.8 Population-Based Methods