CONSTRAINT SATISFACTION PROBLEMS – PART VI

Chapter 6

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search
- Constraint propagation
- Local search for CSPs
- Structure of the problem

Local search

- Assume an **assignment is inconsistent**
- **Next assignment** can be constructed in such a way that **constraint violation** is **smaller**
  - Only **"small" changes** (local steps) of the assignment are allowed
  - Next assignment should be "**better**" than previous
    - better = **more constraints are satisfied**
- Assignments are **not necessarily** generated **systematically**
  - **we lose completeness, but** we (hopefully) get better **efficiency**

Local search terminology

- **Search space:** set of **all complete** variable **assignments**

- **Set of solutions:**
  - subset of the search space
  - all complete assignments **satisfying all the constraints**

- **Neighborhood relation:** indicating **what assignments** can be reached **by a search step** given the current assignment during the search procedure

- **Evaluation function:** mapping **each assignment** to a **real number** representing "how far the assignment is from being a solution"

Local search terminology

- **Initialization function:** returns an **initial position** given a possibility distribution over the assignments

- **Step function:**
  - Given an assignment, its neighborhood, and the evaluation function
  - returns **the new assignment** to be **explored** by the search

- **Set of memory states** (optional): holding information about the state of the search mechanism

- **Termination criterion:** **stopping the search** when satisfied

Local search for CSPs

- **Neighborhood of an assignment:** all **assignments differing** on **one value** of **one variable**

- **Evaluation function:** mapping **each assignment** to the **number of constraints it violates**

- **Initialization function:** returns an **initial assignment** chosen randomly

- **Termination criterion:**
  - if a solution is found or
  - if a given number of search steps is exceeded

- The **different algorithms** are characterized by **the step function** and **use of memory**

Local search for CSPs

- The point of LS: **eliminating violated constraints**

- **Heuristic** for choosing **a new value** for a variable:
  value that results in the **minimum number of conflicts** with other variables

Min-Conflicts

- **Conflict set** of an **assignment:**

  set of variables involved in some constraint that **assignment is violating**

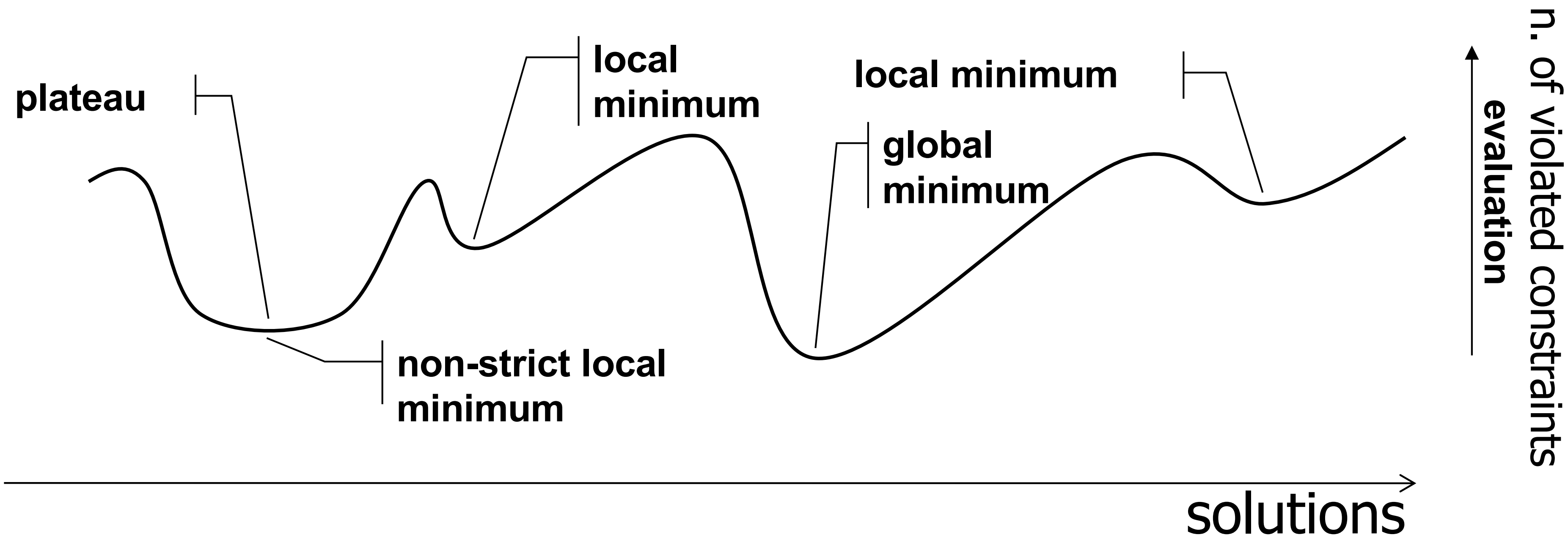
- **Min-conflict LS procedure**
  - Starts at a randomly generated assignment
  - At each step of the search
    - **Selects a variable** from the **current conflict set**
    - **Selects a value** for that variable that **minimizes the number of violated constraints**
  - **If multiple choices** choose one **randomly**
    - *neighbourhood* = different values for the selected variable
    - neighbourhood size = (d-1)

Local minima

The evaluation function can have:

- **local** minimum - a state that is **not minimal** and there is **no state with better** evaluation **in its neighbourhood**
- **strict local** minimum - a state that is **not minimal** and there are **only states** with **worse evaluation in its neighbourhood**
- **global** minimum - the state with the **best evaluation**
- **plateau** - a set of **neighbouring states** with the **same evaluation**

Graphically…



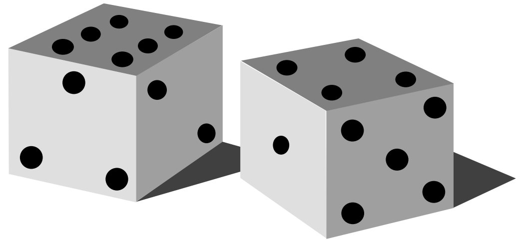
Escaping local minima

- A local search procedure **may get stuck** in a local minima
- Techniques **for preventing stagnation**
  - restart
  - allowing **non improving steps** → random walk
  - changing the **neighborhood** → tabu search

Restart

- Re-initialize the search after MaxSteps (non-strictly improving) steps
- New assignment chosen randomly
- Can be combined both with hill-climbing and Min-conflicts
- It is effective if MaxSteps is chosen correctly and often it depends on the instance

Random walk



- Add some “noise” to the algorithm
- *Random walk*
  - a new assignment from the neighbourhood is selected randomly (e.g., the value is chosen randomly)
  - such technique can hardly find a solution
  - so it needs some guidance
- Random walk can be combined with the heuristic guiding the search via probability distribution:
  - *p*: probability of using the random walk (noise setting)
  - *(1-p)* : probability of using the heuristic guide
  - Min-conflicts random walk

Tabu search

- Being trapped in local minimum **can be seen as cycling**
- **How to avoid cycles** in general?
  - **Remember already visited states** and do not visit them again
    - memory consuming (too many states)
  - It is possible to **remember just a few last states**
    - Prevents "short" cycles
  - **Tabu list** = **a list of forbidden states**
    - Tabu list **has a fix length *k*** (**tabu tenure**)
      - **"old" states** are **removed** from the list when a **new state is added**
    - State included in the **tabu list** is forbidden (it is tabu)

Constraint weighting

- Can help **concentrate** the search on **important constraints**

- Each constraint is given a numeric weight (initially all 1)

- At each step of the search
  - We choose a **variable/value pair** to change
    with **lowest total weight** of all violated constraints

  - **Weights** are then **adjusted**
    by **incrementing the weight** of **each constraint violated** by the current assignment

Local search in real-world problems

- It can be **used** for **scheduling problems** in online setting when the problem changes
- **A week's airline schedule**
  - may involve thousands of **flights**
  - may involve tens of thousands of **personnel assignments**
  - **bad weather** at one airport can make the **schedule infeasible**
  - **To repair** the schedule with a **minimum number of changes**
    - a local search algorithm starting from the current schedule
  - A **backtracking search** with the new set of constraints usually requires
    - much **more time**
    - might find a solution with **many changes** from the **current schedule**

CONSTRAINT SATISFACTION PROBLEMS – PART VII

Chapter 6

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search
- Constraint propagation
- Local search for CSPs
- Structure of the problem

Structure of the problems

- **Problem structure** (constraint graph) can be **used** to **find solutions quickly**
- To deal with real world problems →
  decompose them **into independent subproblems**
- **Example** - map coloring problem

  Tasmania is **not connected** to the mainland

  → coloring Tasmania and coloring the mainland ar **independent subproblems**

  → **any solution** for the mainland **combined** with **any solution** for Tasmania yields **a solution** for the map

Structure of the problems

- **Independence** can be obtained by finding connected components of the constraint graph
  - **Each component** corresponds to a **subproblem** CSP_i
  - If assignment S_i is a solution of CSP_i $\rightarrow$

    $\mathbf{U_i S_i}$ is a solution of $U_i\ CSP_i$

Structure of the problems

- **Why is decomposition important?**
  - Assume **n variables,** each variable has a domain with cardinality d
  - Assume c is a constant, **c < n**
  - Assume **each CSP$_i$** has **c variables** from the total of n → **n/c** subproblems
  - Each subproblem requires at most **d^c** work to solve it →
  - Total work is **$O(d^c\ n/c)$,** which is ***linear* in n**
  - **Without the decomposition**

    Total work is **$O(d^n)$,** which is ***exponential* in n**
  - **Example:**
    - Dividing a **Boolean CSP** with **80 variables** into **4 subproblems**
    - Worst-case solution time: from **the lifetime of the universe** to **<u>less than 1 second</u>**

The structure of the problems

- **Other graph structures** easy to solve: **trees**
  - A constraint graph is a **tree** when <u>any two variables</u> are connected by <u>only one path</u>

- Any **tree-structured CSP** can be solved **in time linear** in the number of variables

- A **CSP** is defined to be **directed arc-consistent** **(DAC)** **under an ordering** of variables $X_1, X_2, \ldots, X_n$ iff

  every X_i is arc-consistent **with each X_j** for **$j>i$**

Tree-structured CSP

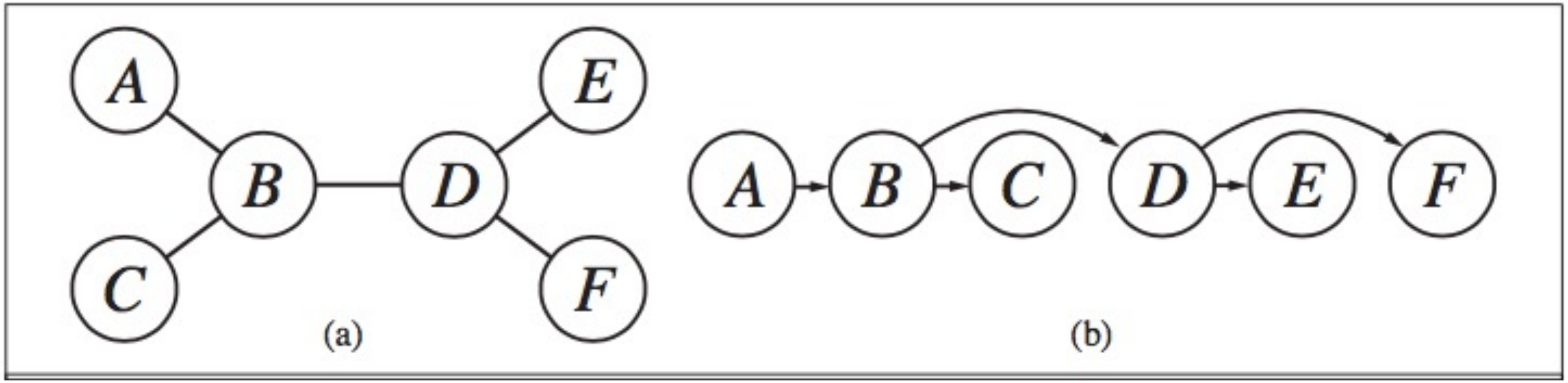
- Given **n** variables, with **d** values in each domain
  - If the **CSP graph is a tree**, it can be solved in $\boxed{\mathbf{O(nd^2)}}$

Tree-structured CSPs

- **To solve** a tree-structured CSP:
  - Pick **any variable** to be the **root** of the tree
  - Choose an **ordering of the variables** such that each variable appears after its parent in the tree
    - This kind of ordering is called **topological sort**
  - **Make** this graph **directed arc-consistent** ($O(nd^2)$)
  - Follow the list of **variables** starting **from the root** and **choose any** remaining **value**

DAC guarantees that for any value we choose for the parent, there will be a valid value left to choose for the child

Tree-structured CSPs



a) **The constraint graph** of a **tree-structured CSP**

b) **Linear ordering** of the variables **consistent with the tree** with A as the root
This is known as a topological sort of the variables

Tree CSP solver

function TREE-CSP-SOLVER(csp) **returns** a solution, or failure
inputs: csp, a CSP with components X, D, C

n ← number of variables in X
assignment ← an empty assignment
root ← any variable in X

X ←TOPOLOGICALSORT(X, root)

for j = n **down to** 2 **do**
  MAKE-ARC-CONSISTENT(PARENT(Xj), Xj)
  if it cannot be made consistent **then return** failure

for i = 1 **to** n **do**
  assignment [Xi] ← any consistent value from Di
  if there is no consistent value **then return** failure

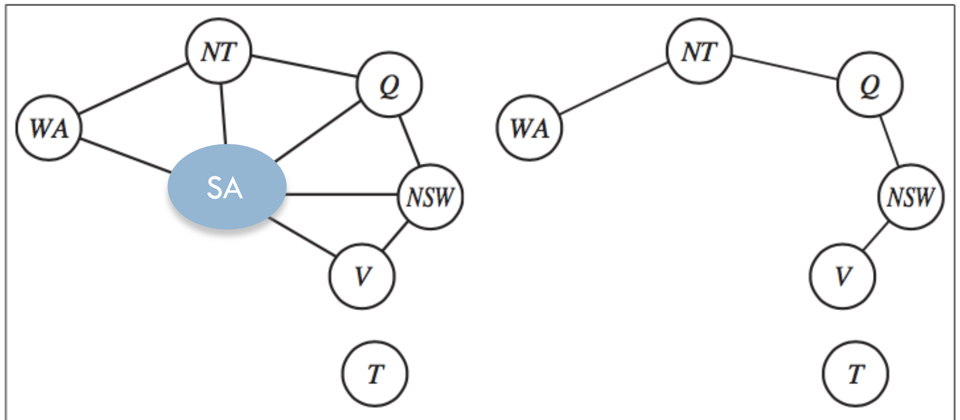
return assignment

TOPOLOGICALSORT
each variable appears **after** **its parent** in the tree

Backtrack is not required
We can move linearly through the variables

Amost tree-structured

- Idea: Reduce the graph structure to a tree
  assigning values to some variables
- **Example**
  - Consider the constraint graph for Australia
  - If we could delete South Australia, the graph would become a tree



(a) (b)

- We can do this by fixing a value for SA and deleting from the domains of the other variables any values that are inconsistent with the value chosen for SA

Amost tree-structured

Cutset Conditioning

- Choose **a subset S** of the CSPs **variables** such that the constraint graph **becomes a tree** <u>after removal of **S**</u> (**S** is called a **cycle cutset**)

- **For each possible assignment** of variables **in S** that **satisfies all** constraints on **S**
  - **Remove** from the domains of the remaining variables **any values** that are **inconsistent** with the assignment for **S**
  - If the **remaining CSP** has **a solution**, return it <u>together</u> with the **assignment for S**

CONSTRAINT SATISFACTION PROBLEMS – PART VIII

Chapter 6

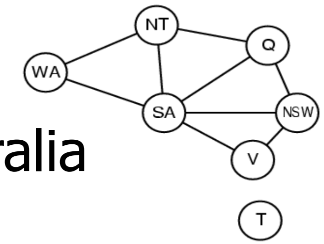
Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search
- Constraint propagation
- Local search for CSPs
- Structure of the problem

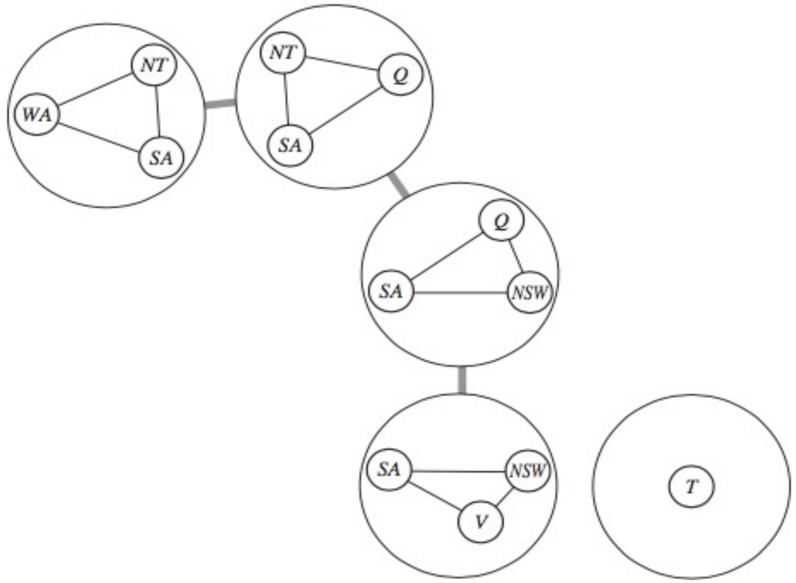
Structure of the problem
Another method: tree decomposition

- **Decompose problem** into a **set of connected sub-problems**, where **two sub-problems** are connected when they **share a constraint**

- **Solve sub-problems** independently and **combine solutions**



Example: A <u>tree decomposition</u> of the constraint graph for Australia



Another method: tree decomposition

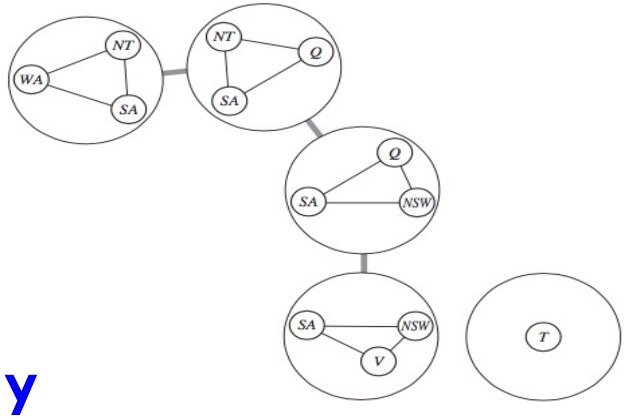
A **tree decomposition** must satisfy the following conditions:

- Every variable of the original CSP appears in at least one sub-problem
- If two variables are connected by a constraint in the original CSP →

  they must appear with their constraint in at least one subproblem
- If a variable appears in some subproblems →

  it must have the same value in every subproblem

Another method: tree decomposition



- We solve each sub-problem **independently**
- If a sub-problem has no solution → entire problem has **no solution**
- If we can solve all the subproblems → we construct a **global solution**
  - Consider **each sub-problem** as **new "mega-variable"**
    - **Domain** of each mega-variable: **all the solutions** to the sub-problem
  - Then, **solve the constraints** that connect the subproblems by using **tree CSP solver** to find an **overall solution** with identical values for the same variable

Tree width

- A constraint graph allows for several tree decompositions
- Aim: to select decomposition with the suproblems as **small** as possible

- **Tree width** of a tree decomposition: **s - 1**

  where **s** is the **size of largest sub-problem**

- **Tree width** of a graph is

  the **minimum** tree width among all its tree decompositions

Tree width

- If
  - a graph has tree-width w
  - we know the corresponding tree decomposition

  Then we can solve the problem in $\boxed{O(nd^{w+1})}$

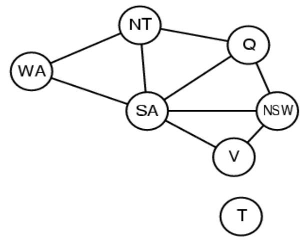
- **CSPs** with constraint graph with a **bounded tree width** can be solved in **polynomial time**

- **Finding a tree decomposition** with **minimal tree width** is **NP-hard** (but some heuristic methods work well in practice)

Symmetry breaking

- So far: **structure** of the **constraint graph**
- Now: **structure** in the **values** of variables

- **Example**: **map-coloring problem** with **n colors**
  - ∀**solution**, **n! solutions** formed by **permuting the color names**
  - Australia map:
    - **WA, NT , SA** must all have **different colors**
    - But there are **3! = 6** ways to assign three colors to three regions
    - This is called **value symmetry**



- **To reduce** the search space: **symmetry-breaking** constraint
  - We impose an **arbitrary ordering constraint, NT < SA < WA** that requires the three values to be in alphabetical order →
  - **One** of the n! solutions is possible: {NT = blue, SA = green , WA = red }

Summary

- **CSPs** are a special kind of **search problem:**
  - **states** are value assignments
  - **goal test** is defined by constraints
- **Backtracking** = DFS with one variable assigned per node. Other **intelligent** backtracking techniques possible
- **Variable/value ordering heuristics** can help dramatically
- **Constraint propagation prunes** the search space
- **Tree structure** of CSP graph **simplifies problem** significantly
- **CSPs** can also be solved using **local search**