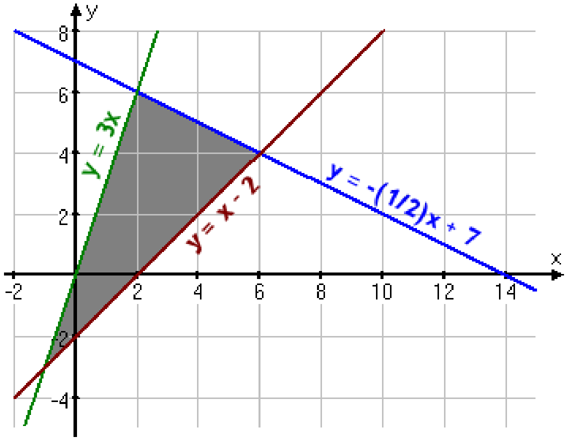
Constraint Satisfaction Problems

DV2557

Dr. Prashant Goswami
Assistant Professor, BTH (DIDA)
prashantgos.github.io

prashant.goswami@bth.se



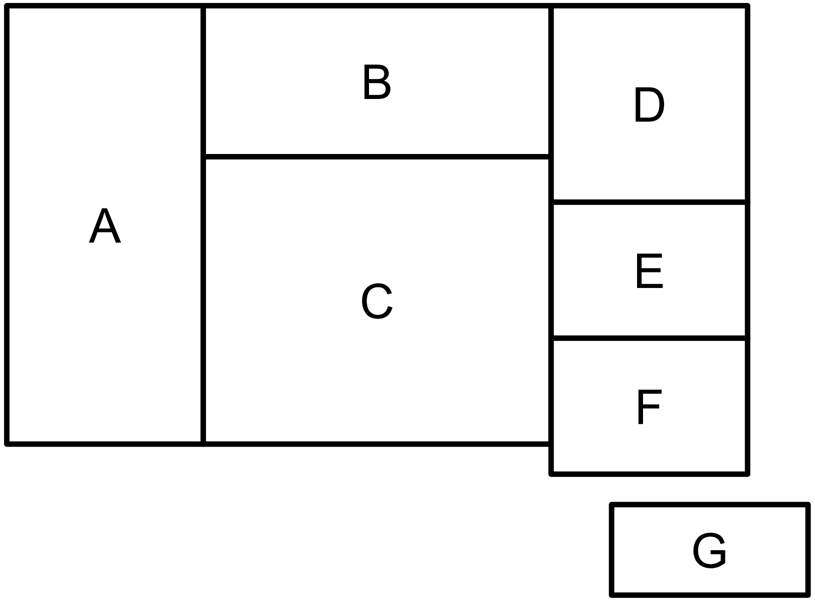
CSP

- Constraint Satisfaction Problems are special kinds of problems where we have:
  - A number of variables X_1, X_2, …, X_n
  - A set of constraints C_1, C_2, … , C_m
  - Each variable have a non-empty *domain* D_i of possible values
- A *state* is where some or all variables are assigned a value.

CSP

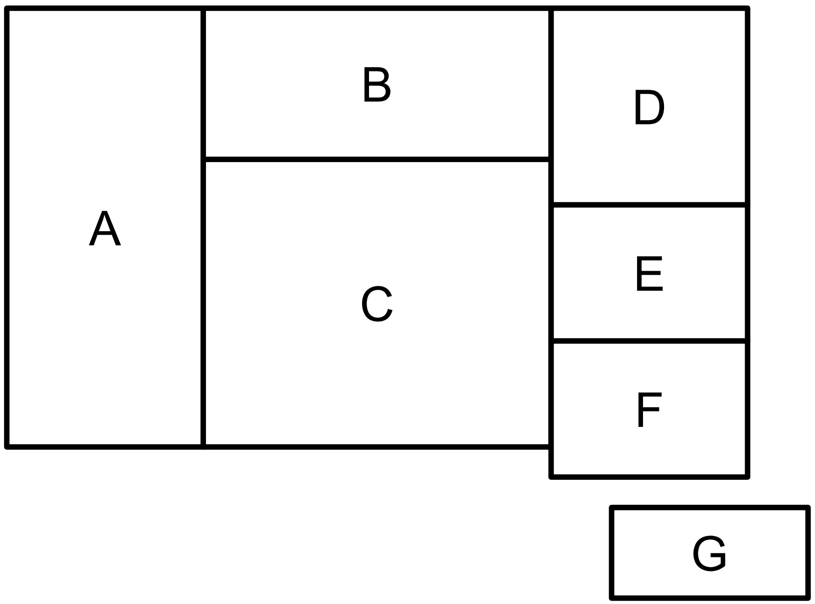
- An assignment that does not violate any constraint is called a *consistent/legal* assignment.
- A *complete* assignment is a state where all variables are assigned a value.
- A *solution* is a complete assignment that does not violate any rules.
- It is also possible to give a value of good or bad solutions using an *objective function*.

Example CSP



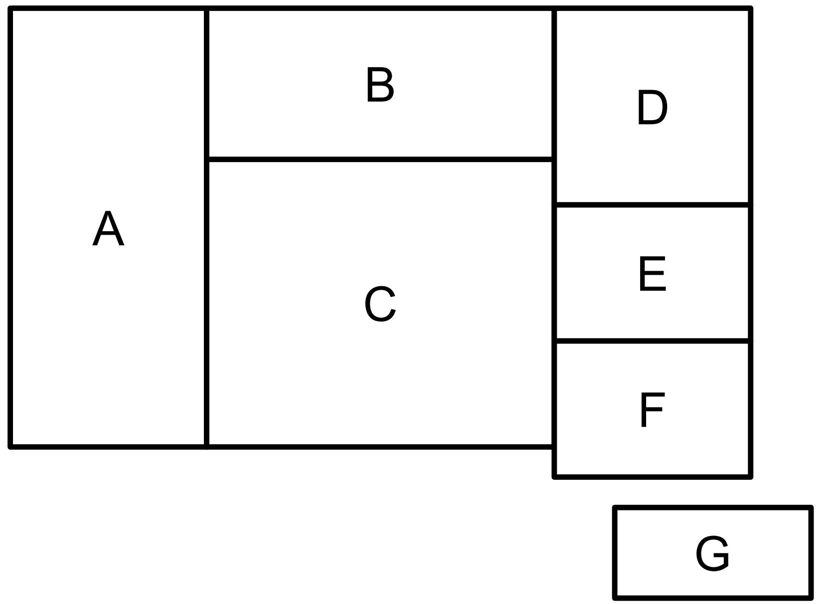
We have a number of regions A – G.

Example CSP



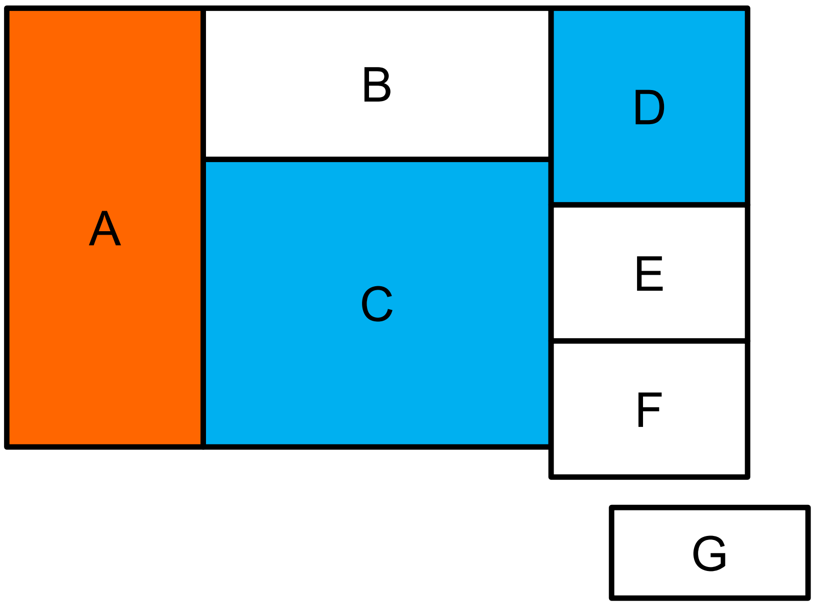
The problems is to give each region a color red, green or blue.

Example CSP



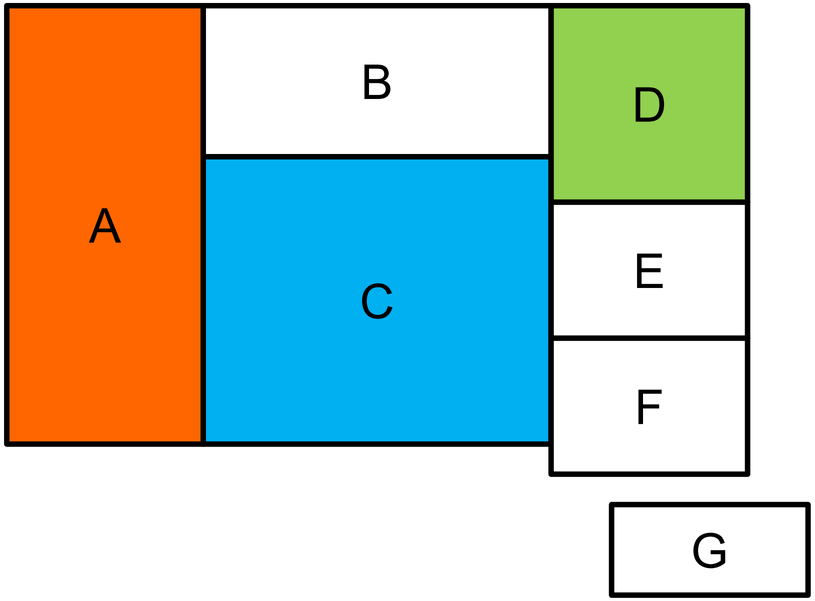
And the constraint is that two adjacent regions cannot have the same color.

Example CSP



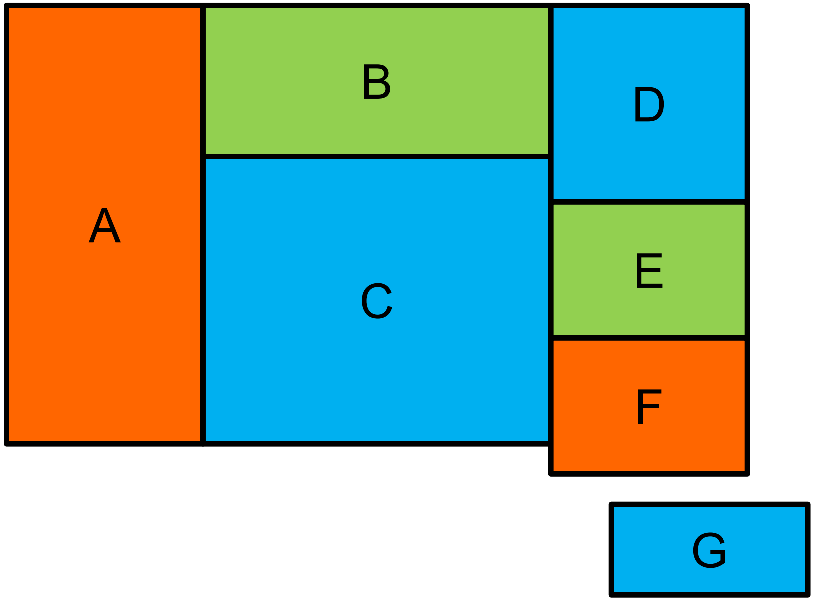
State (non-consistent).
C and D violate the constraint.

Example CSP



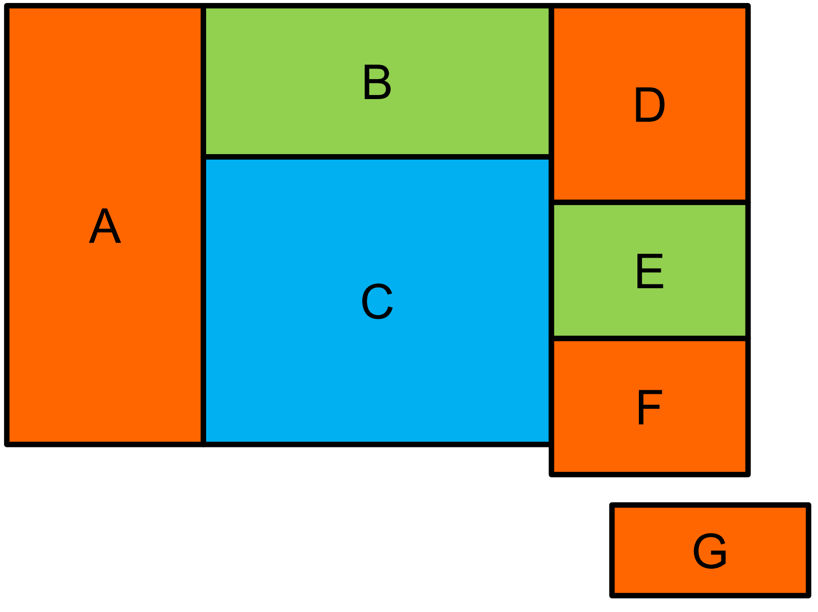
State (consistent).
All constraints are satisfied.

Example CSP



Complete assignment (non-consistent).
C and D violate the constraint.

Example CSP

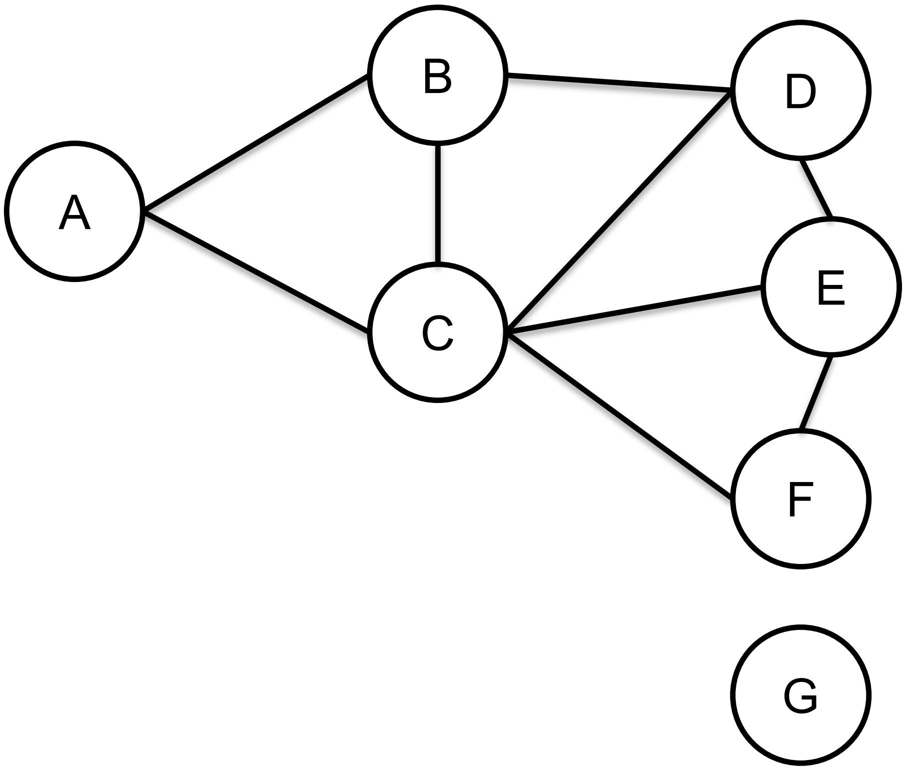


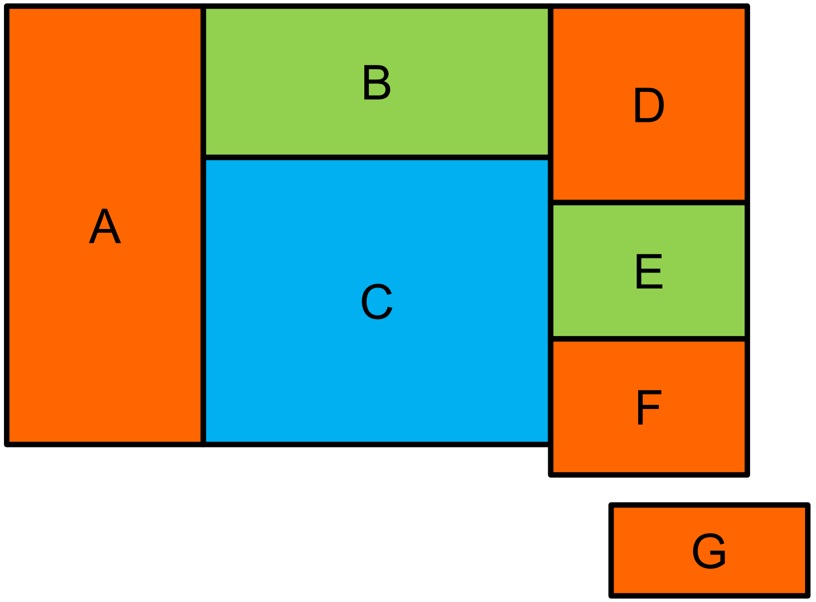
Solution.
All constraints are satisfied.

Example CSP

- Define the problem:
  - Each region is a variable A, B, …, G
  - The domain of each variable is the set {red, green, blue}
  - The constraint(s) is that each variable must have a distinct value.
- Example: allowable combinations for two adjacent regions are the pairs:
  - {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

Define a *Constraint Graph*





Each node corresponds to a variable.
Each arc corresponds to constraints.

We can define this as a standard search problem:

- Initial state:
  - The empty assignment {}
- Successor function:
  - A value can be assigned to any unassigned value as long as the constraints are met.
- Goal test:
  - The assignment is complete and consistent.
- Path cost:
  - A constant cost (1) for each step.

Standard search problem

- If we have n variables, every solution must be at depth n.
- The maximum depth of the tree cannot exceed n (all variables are assigned).
- Therefore, depth first search is popular for CSPs.

Local search algorithms

- The path to a solution is irrelevant.
- The only thing that matters is the solution.
- Therefore local search algorithms such as simulated annealing can be used.

CSPs and Breadth First Search

- Suppose we have *n* variables with *d* possible values.
- The branching factor at the root node is then *nd*.
- At the next level it is *(n-1)d*.
- A complete tree will then contain $n!*d^n$ leaves.

CSPs and Breadth First Search

- This is quite much, considering we have only d^n possible complete assignments.
- This occurs because all CSPs are *commutative*.
- In a commutative problem the order of application of any set of actions has no effect on the outcome.
  - A=red, B=green, C=blue
  - B=green, C=blue, A=red
  - C=blue, A=red, B=green

Reach the same state!

CSPs and Breadth First Search

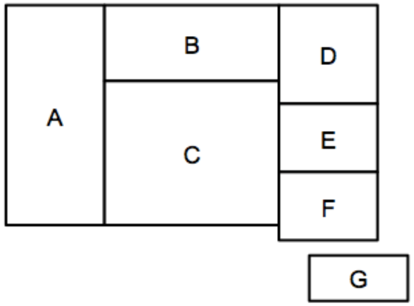
- BFS considers possible assignments for only a single variable at each node.
- A solution is to backtrack at each node, and see if all constraints are met for each possible value of the current node.
- This reduces the number of leaf nodes to d^n

Backtracking Search

- Backtracking Search is a depth-first search with backtracking to check constraints.
- This is best implemented using a recursive algorithm.

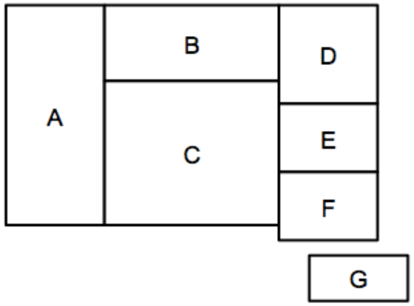
Backtracking Search (BS)

```
function Result RecursiveBS(currentAssignment, constraints)
   if (currentAssignment.isComplete())
   {
      return currentAssignment.result();
   }
   foreach (value in possibleValues)
   {
      if (value.isConsistentWith(constraints))
      {
         assignment.add(variable = value);
         result = RecursiveBS(currentAssignment, constraints);
         if(result is solution) return result;
      }
   }
   return Result.Failure;
}
```

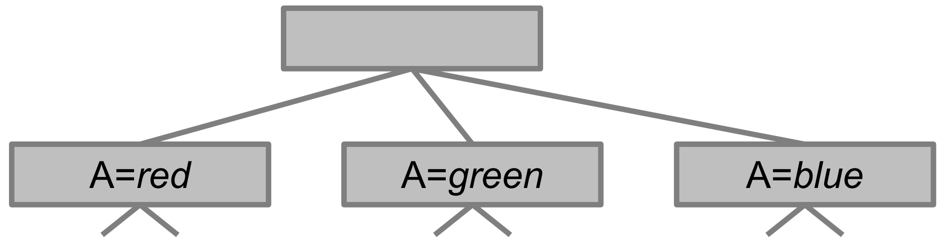


Example: BS

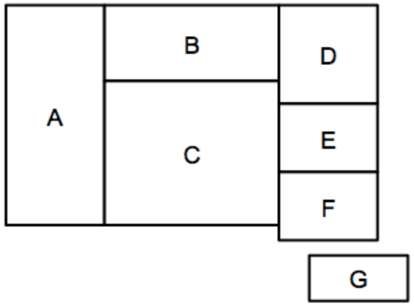
Start with an empty assignment.
We choose to begin at variable A



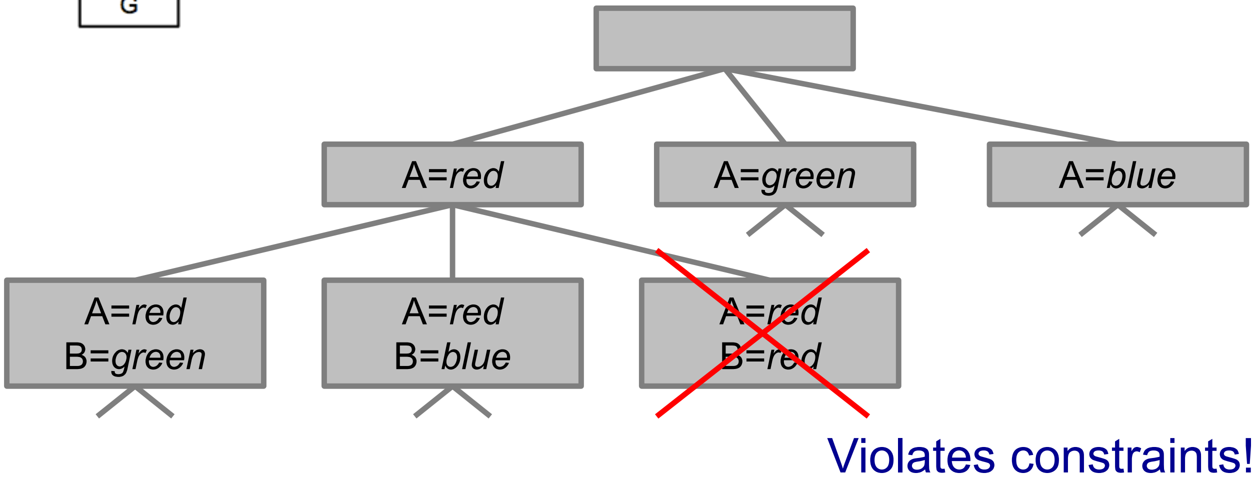
Example: BS



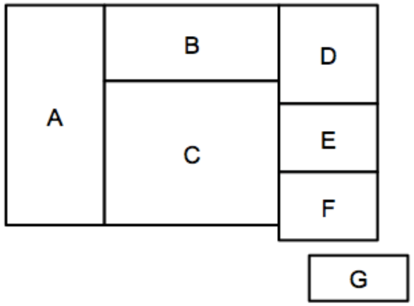
Example: BS



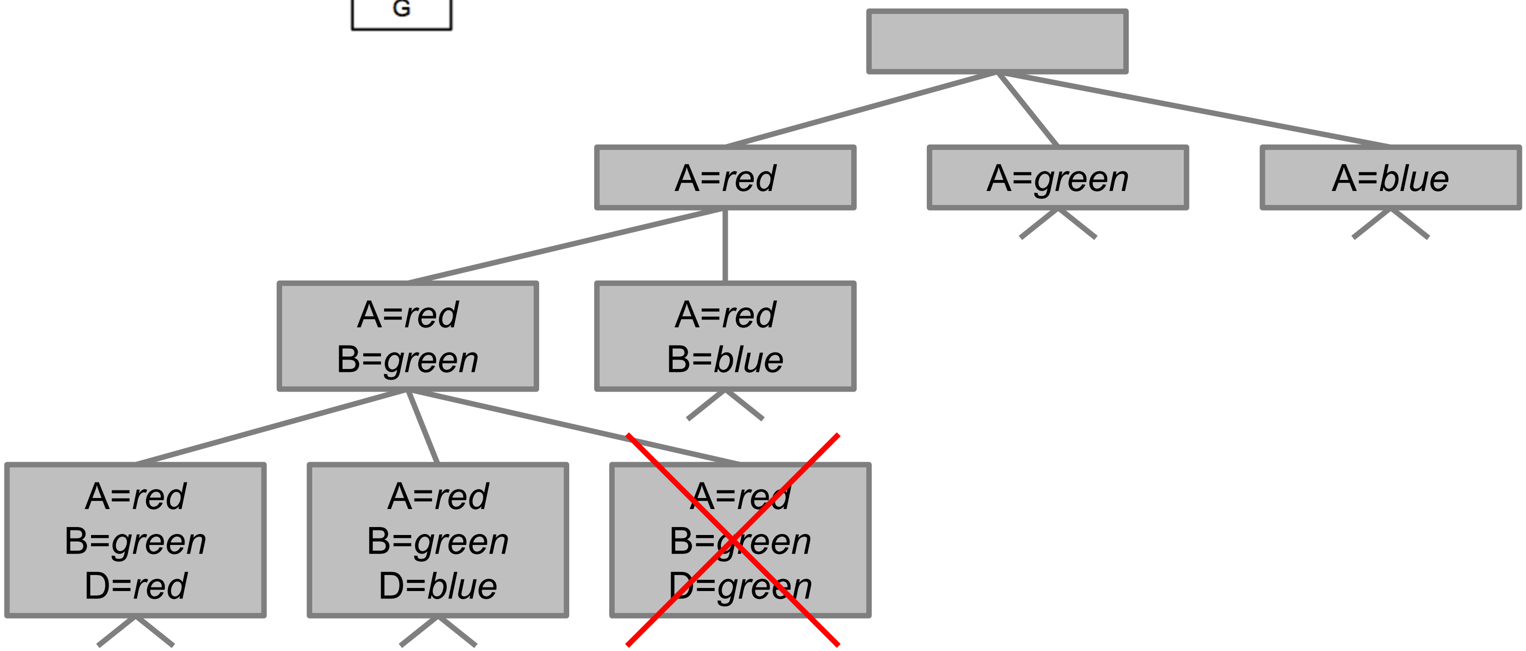
A
B
C
D
E
F
G



A=*red*
A=*green*
A=*blue*
A=*red*
B=*green*
A=*red*
B=*blue*
A=*red*
B=*red*
Violates constraints!



Example: BS



Violates constraints!

… and so on, until we reach depth 7 (7 variables).

Backtracking Search

- BS is an uninformed search algorithm.
- And as we know from informed/un-informed search, uninformed algorithms are usually not feasible for larger problems.
- Luckily there are some heuristics we can use for CSP problems!

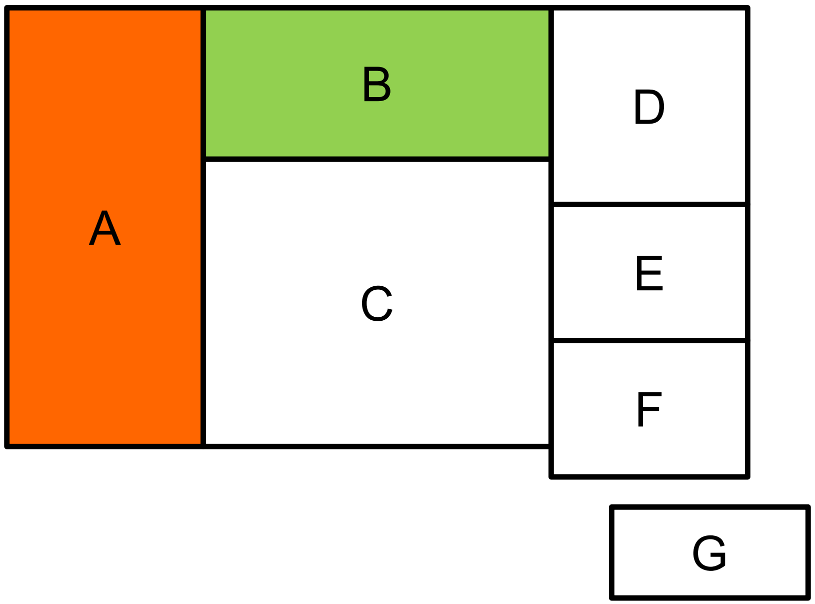
Heuristics for CSP

- There are three questions we can address:
  - Which variable should we assign next, and in what order should its values be tried?
  - What are the implications of the current assignment on other unassigned variables?
  - When a path fails (i.e. no legal values to assign), how can we avoid this failure in other search branches?

Minimum Remaining Values (MRV)

- The next variable to assign is the variable with the fewest legal values.
- By doing this we pick the variable that is most likely to cause a failure soon, thereby pruning the search tree.

Minimum Remaining Values (MRV)

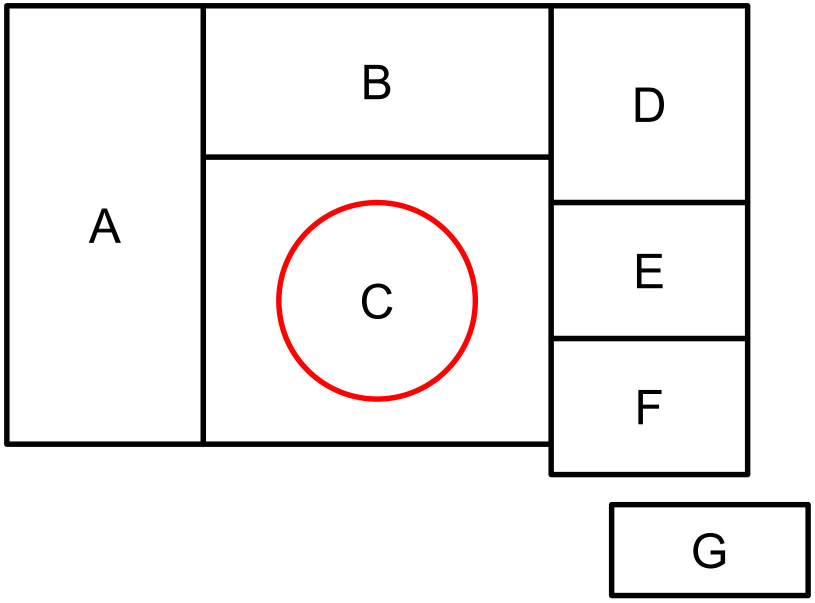


- Next to choose is C because it has the fewest possible values:
  - C: 1
  - D: 2
  - E: 3
  - F: 3
  - G: 3

Degree Heuristic

- MRV does not help in choosing which variable to start with.
  - All regions have three possible values.
- For the first assignment we can instead use Degree Heuristic.
- It means choosing the variable with the largest number of constraints on other variables.

Degree Heuristic

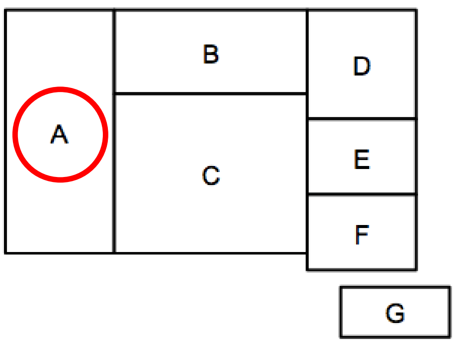


- According to DH we should start with C:
  - A: 2
  - B: 3
  - C: 5
  - D: 3
  - E: 3
  - F: 2
  - G: 0

Forward Checking

- Forward Checking optimizes the resources needed for constraint checking.
- Once we have assigned a value to a variable, the non-valid values for adjacent variables (through constraints) are deleted from their domain.

Forward Checking

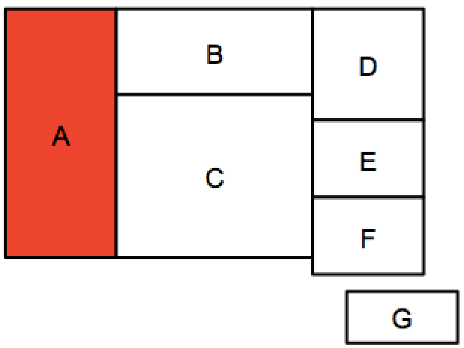


Variable	Domain	Assignment
A		

Variable	Domain	Assignment
B		

Variable	Domain	Assignment
C		

Forward Checking



Variable	Domain	Assignment
A		

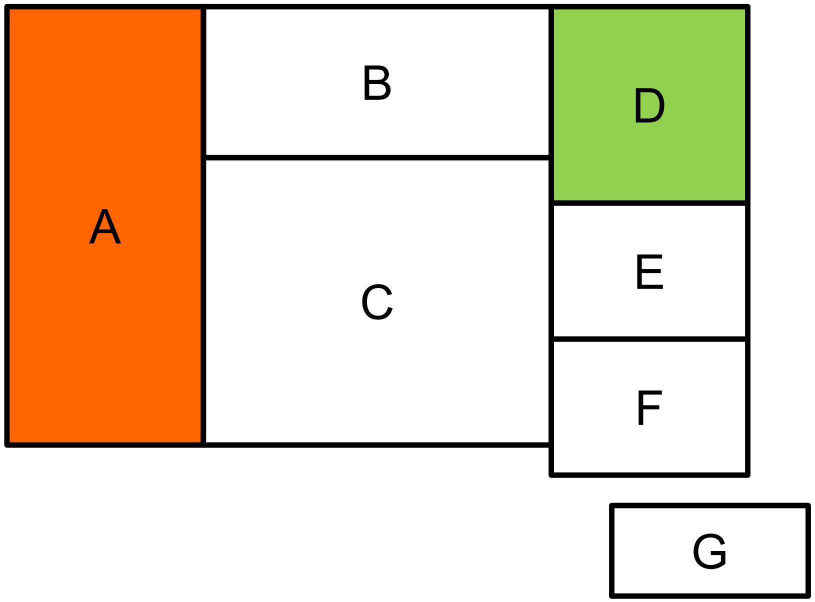
Variable	Domain	Assignment
B		

Variable	Domain	Assignment
C		

- If we assign red to A, B and C cannot be red and therefore red is removed from their domains.
- When expanding B or C, we don't need to backtrack to check if red is a legal value.

Forward Checking

Forward Checking does not detect everything!

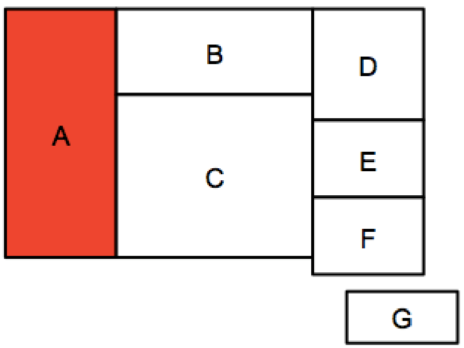


Variable	Domain	Assignment
B		

Variable	Domain	Assignment
C		

Both B and C cannot be blue. FC does not look far enough ahead to detect this!

Constraint Propagation



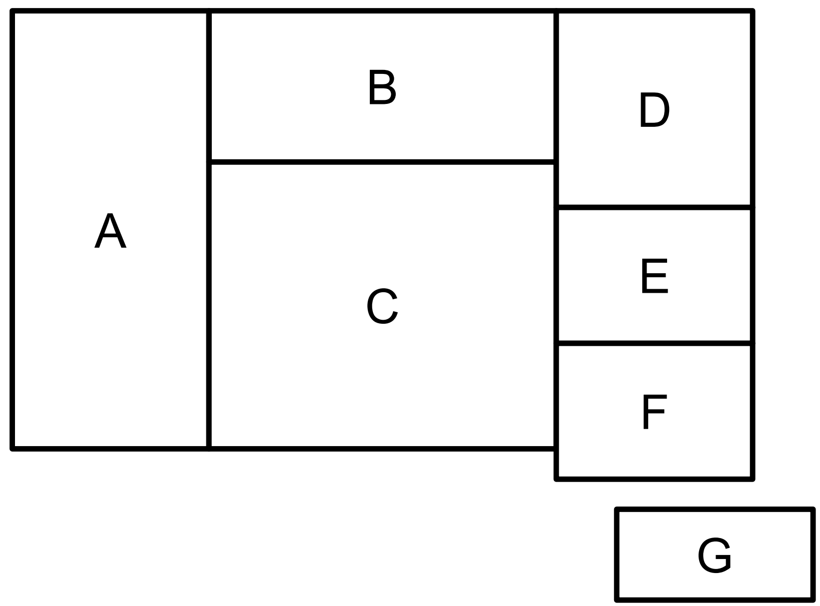
- When assigning red to A Forward Checking propagates constraints to B and C.
- But as we saw we also need to propagate a constraint to D.
- This is done using Constraint Propagation algorithms such as AC-3.
- We will not dig into more details about this.

Backtracking

- The Backtracking approach is very simple:
  - If a branch fails, we go back to the most recent decision point.
- This is not always the best approach.
- Consider the following example:

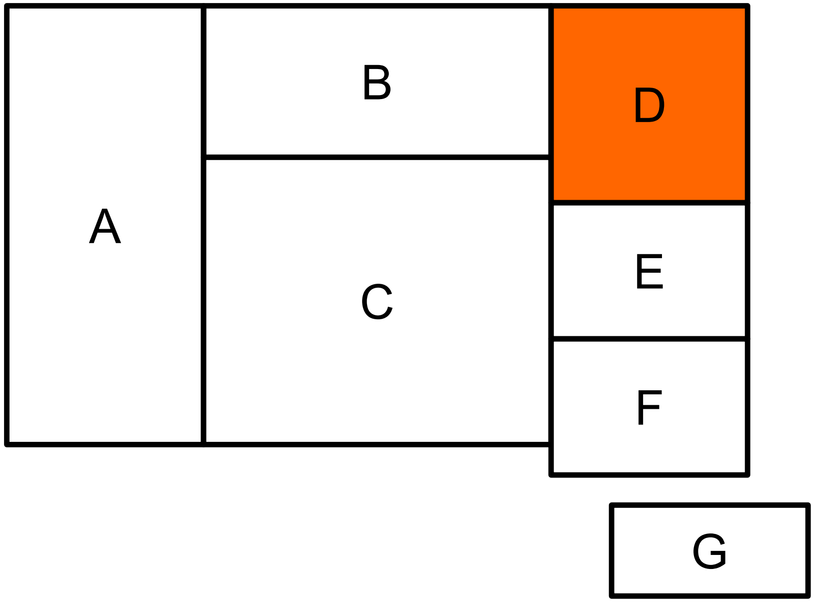
Backtracking fails

Variable assignment order:
D, E, F, G, C, A, B



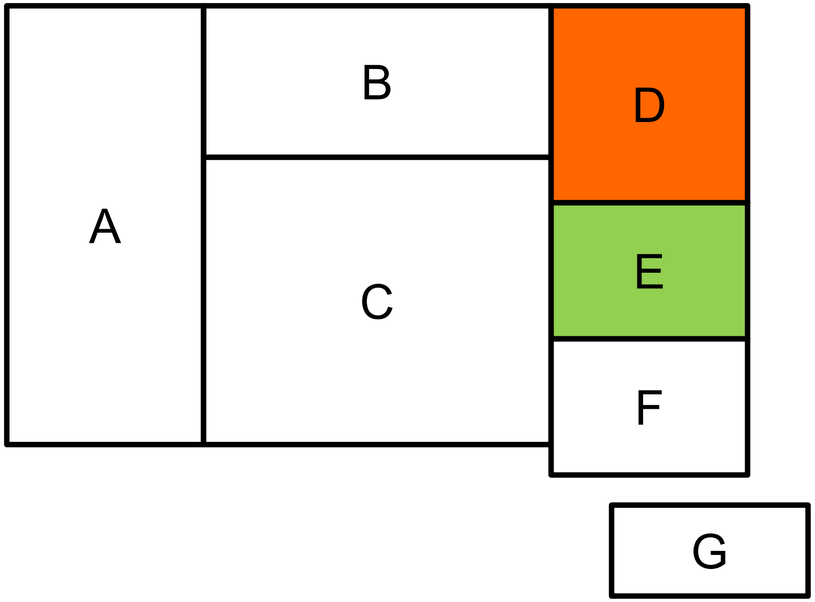
Backtracking fails

Variable assignment order:
D, E, F, G, C, A, B



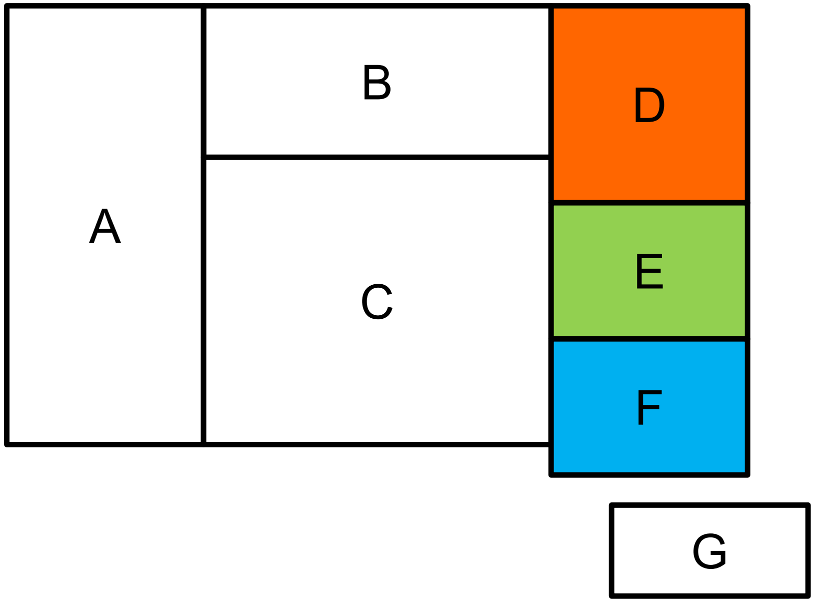
Backtracking fails

Variable assignment order:
D, E, F, G, C, A, B



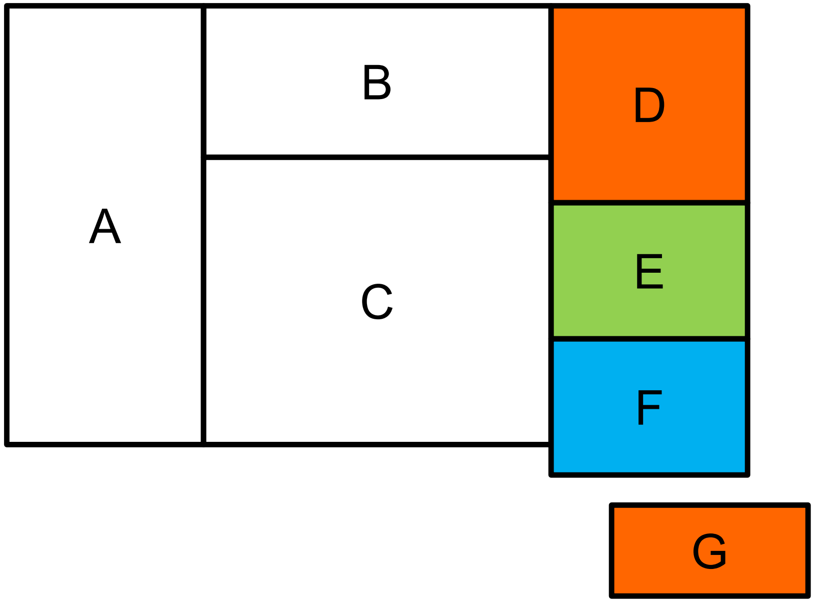
Backtracking fails

Variable assignment order:
D, E, F, G, C, A, B



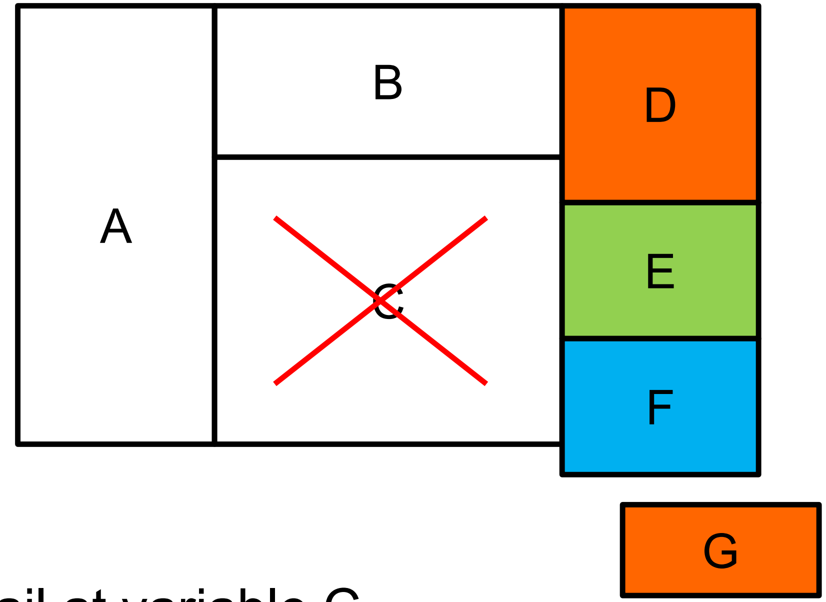
Backtracking fails

Variable assignment order:
D, E, F, G, C, A, B



Backtracking fails

Variable assignment order:
D, E, F, G, C, A, B



- We fail at variable C.
- Backtracking tells us to go back to the previous variable G.
- However, changing color of G doesn't help us at all!

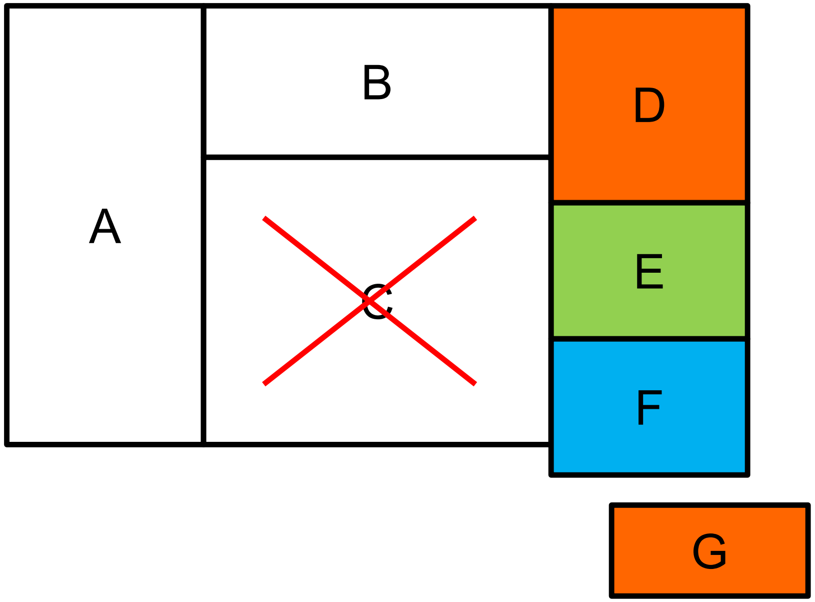
Conflict Set

- A better approach is go back to the variables that caused the problem.
- These are called the *conflict set*.

- Let’s go back to the example:

Conflict Set

Variable assignment order:
D, E, F, G, C, A, B



- The conflict set for C is {D, E, F}.
- We backtrack to the most recent variable in the conflict set, in this case F.
- Changing color of F makes much more sense!
- We call this process *Backjumping*.

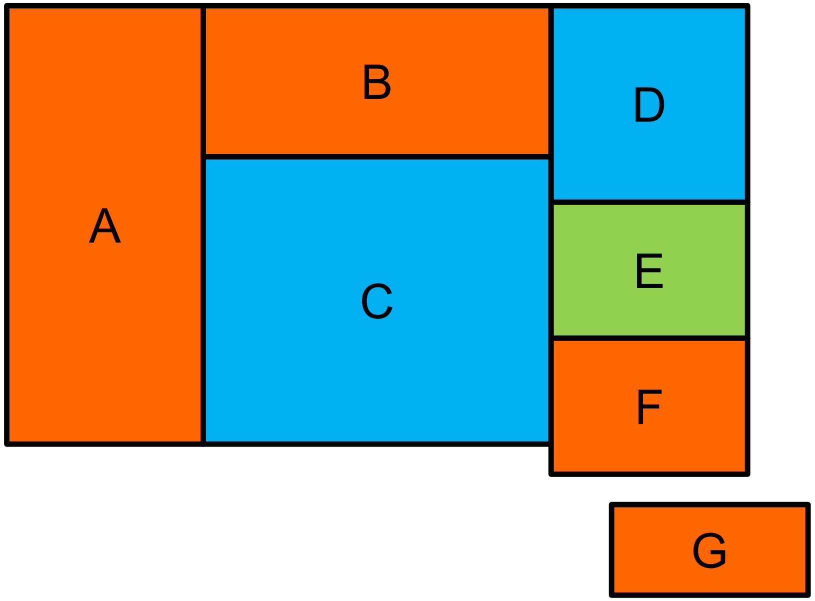
Local-Search algorithms

- It turns out that local search algorithms are very effective for many CSPs.
- A local search algorithm makes a local change to a variable and see how it turns out.
- The most common algorithm is the Min-conflicts.

Min-Conflicts

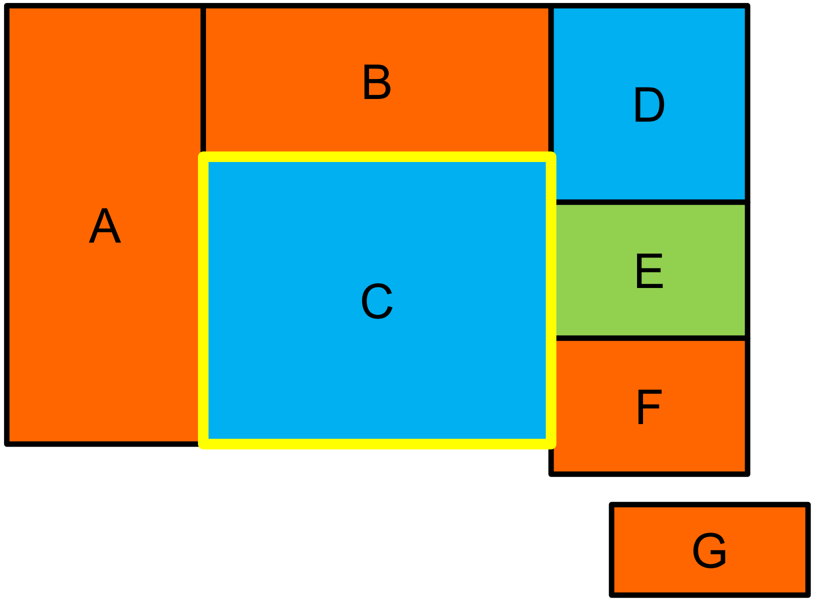
- Start with assigning a random value to each variable.
- In every iteration, select a random variable to update.
- Update the variable to the value that causes the least number of conflicts with other variables.
- Continue until we are done or we reach a max number of iterations.

Min-Conflicts example



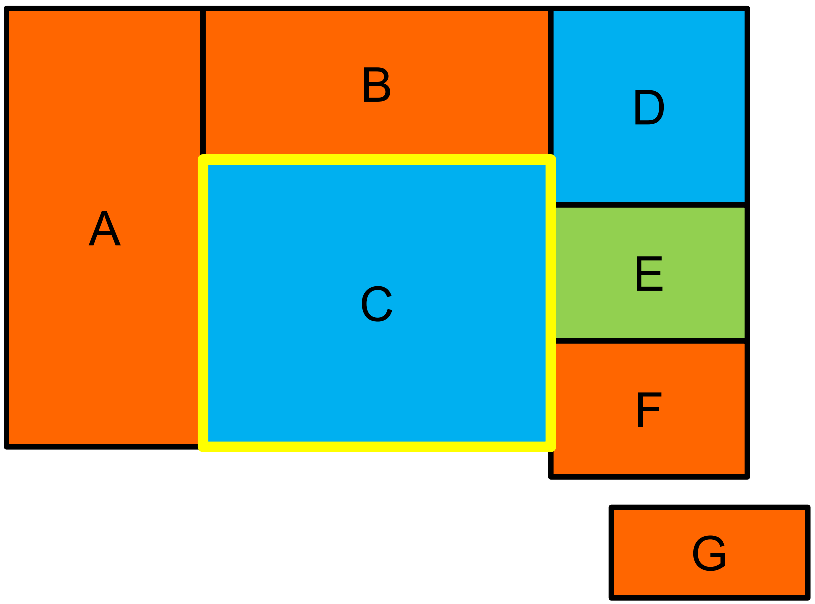
Initial, random assignment.

Min-Conflicts example



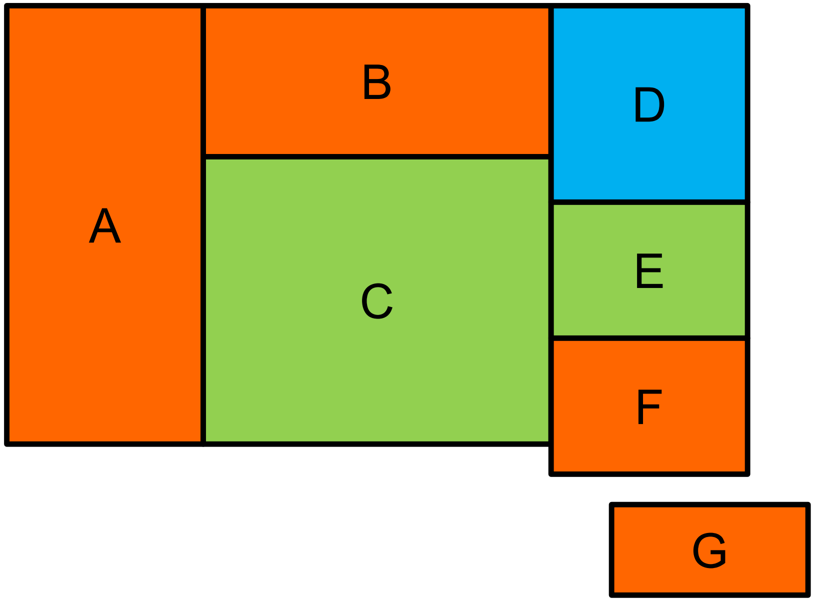
Select a random variable to update
–> C

Min-Conflicts example



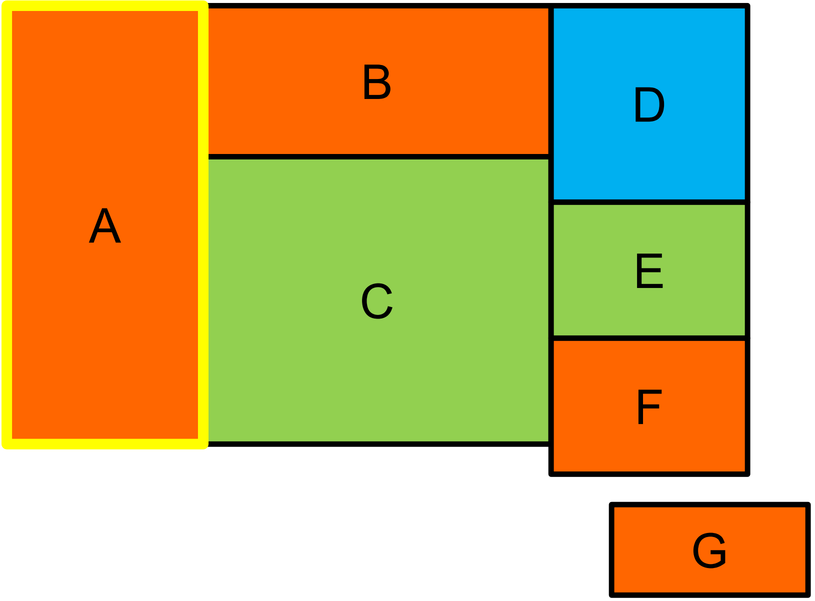
Check minimum number of conflicts for each color of C:
Red: 3
Green: 1
Blue: 1

Min-Conflicts example



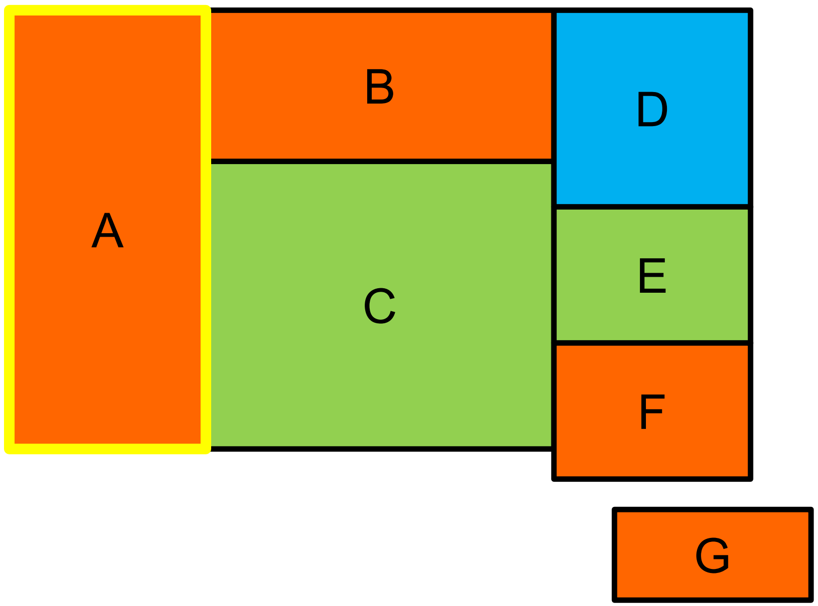
Update C to green (or blue).

Min-Conflicts example



Select a new random variable to update
–> A

Min-Conflicts example



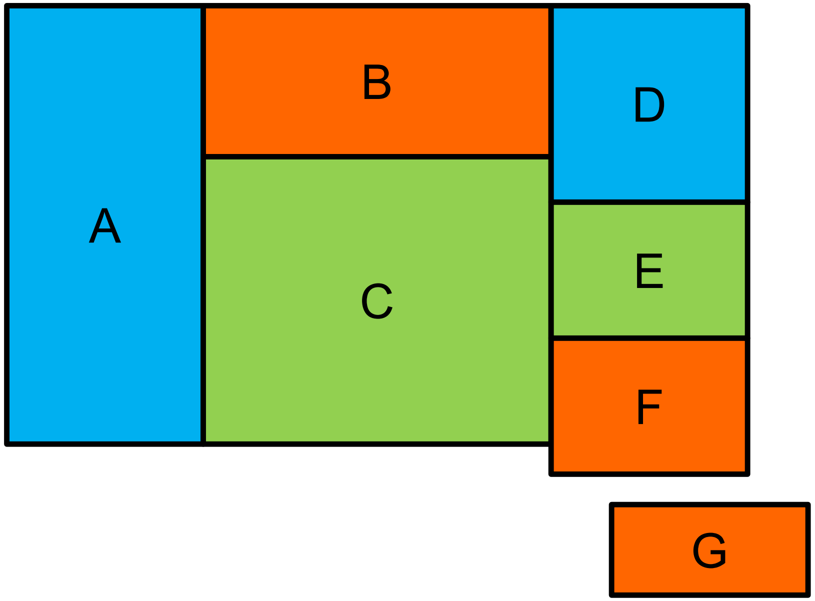
Check minimum number of conflicts for each color of A:

Red: 1

Green: 1

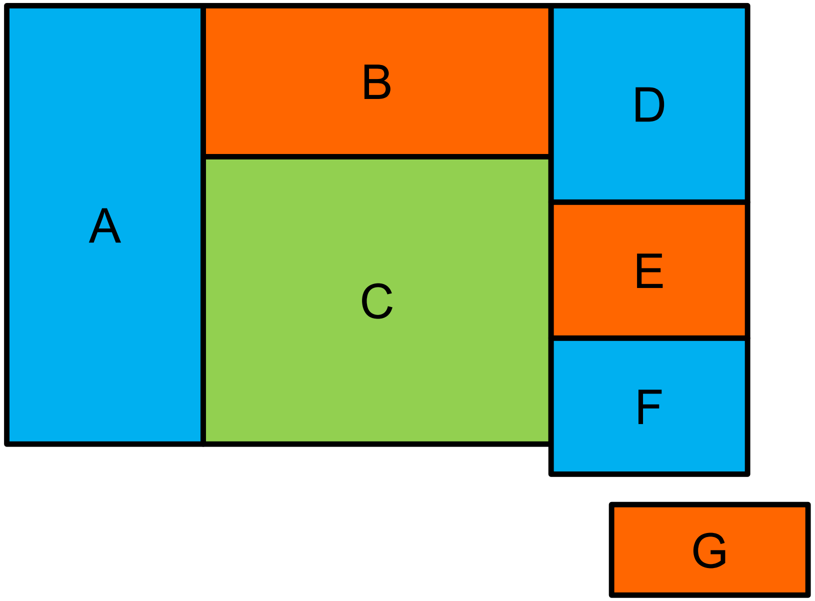
Blue: 0

Min-Conflicts example



Update A to blue.

Min-Conflicts example



… and so on until we have a solution!

Min-Conflicts

- An interesting property of Min-Conflicts is that it is roughly independent of problem size.
- Therefore it is very suitable even for very hard problems.
- A drawback is the initial placement. A bad placement can increase the searchtime a lot.

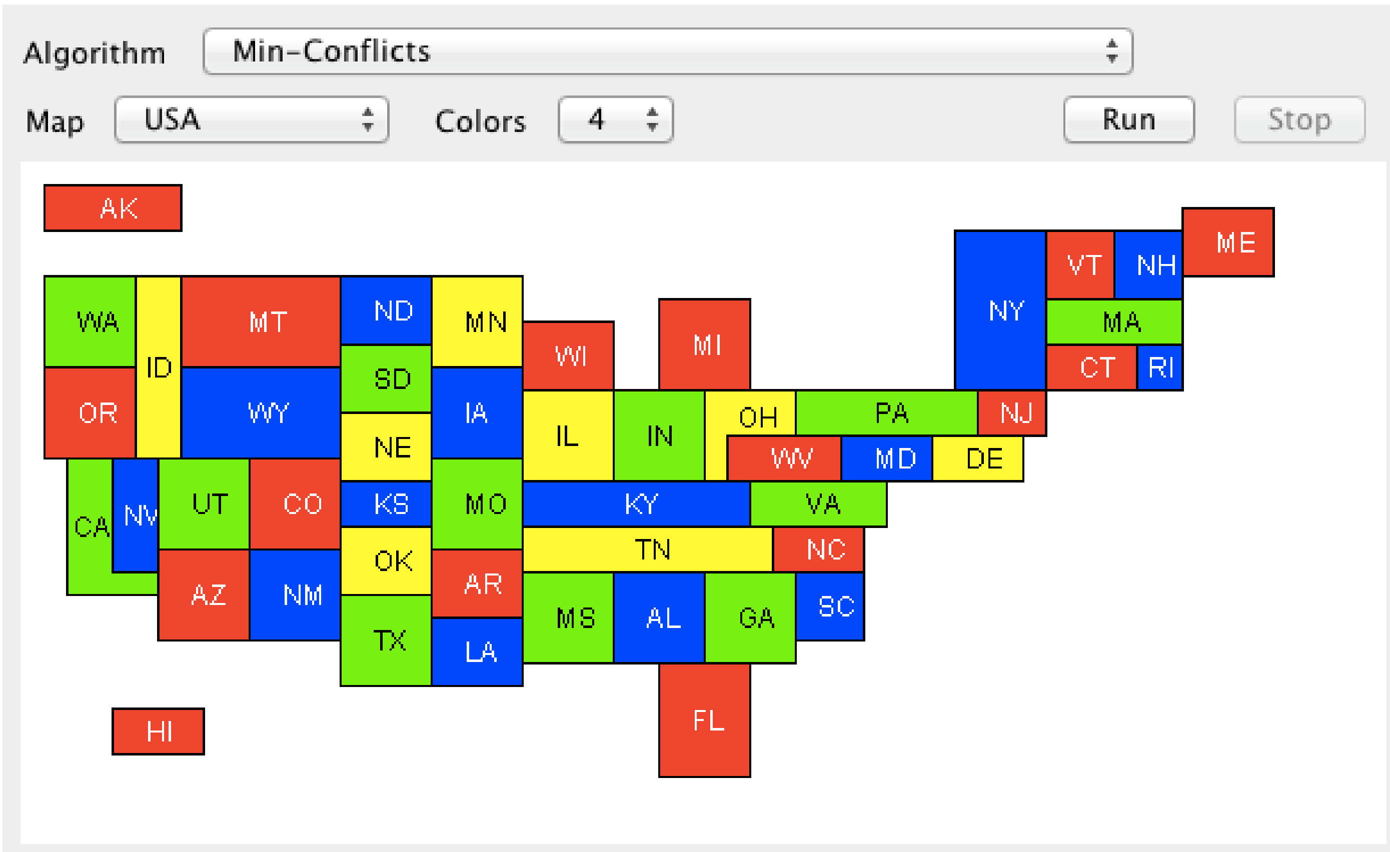
A better placement strategy

- Start with a randomly selected variable.
- For each remaining variable, assign the value with the least number of conflicts.
  - Min-conflicts strategy.

Min-Conflicts Applications

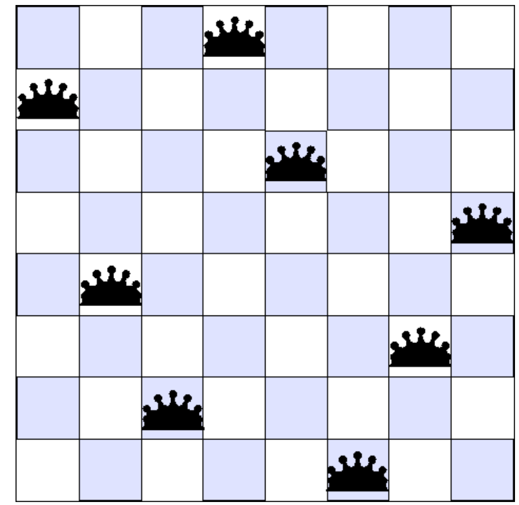
- Min-Conflicts have been used with success in many scheduling problems.
- It is for example very effective if a change requires updating the whole schedule.
- It was for example used by NASA to schedule the Hubble Telescope, reducing the scheduling time from weeks to around 10 minutes!

Test tool


Algorithm
Min-Conflicts
Map
USA
Colors
4
Run
Stop
AK
WA
ID
MT
ND
MN
WI
MI
NY
VT
NH
ME
MA
CT
RI
OR
WY
SD
IA
NE
IL
IN
OH
PA
NJ
WV
MD
DE
CA
NV
UT
CO
KS
MO
KY
VA
OK
TN
NC
AZ
NM
AR
TX
LA
MS
AL
GA
SC
FL
HI

Other CSPs

- The coloring problem is a quite simple type of CSP.
- There are types of CSPs that are far more complex.
- One other common CSP problem mentioned in the book is the n-queen problem.
  - Place n queens on a chess board.
  - They should be placed so that no queen can attack another queen.



8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5

Other CSPs

- Both the n-queens and the coloring problem have finite domains, i.e. all variables are discrete and have a fixed range.
- CSPs with continuous domains are far more complex.
- If the constraints are linear on integer variables the CSP can be solved.
- If the constraints are non-linear, no general algorithm exists.

That was all for this lecture

http://etc.ch/MLJ7

Acknowledgements

Dr. Johan Hagelbäck

Linnæus University

johan.hagelback@lnu.se

http://aiguy.org