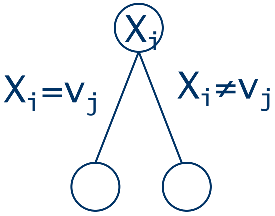
Search in CP

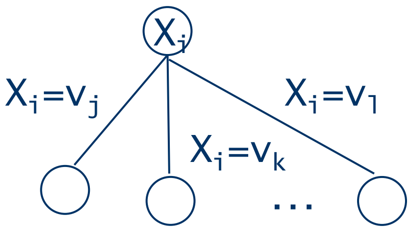
Constraint Solver

- Enumerates all possible variable-value combinations via a systematic backtracking tree search.
  - Guesses a value for each variable.

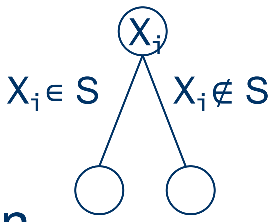
Backtracking Search Tree (BTS)

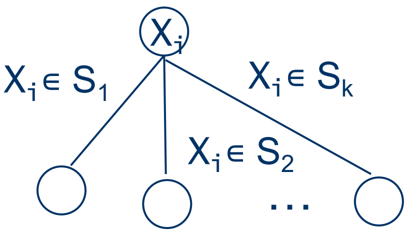
- Node → variable X_i
- Branch → decision on X_i
  - Labelling with single values from $D(X_i)$.





  - Domain partitioning of $D(X_i)$.

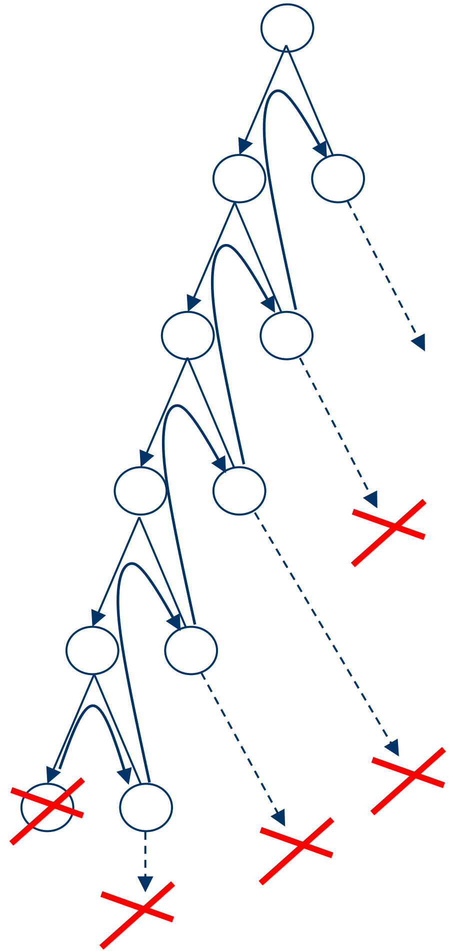




- X_i and (set of) values are chosen by the search heuristics.

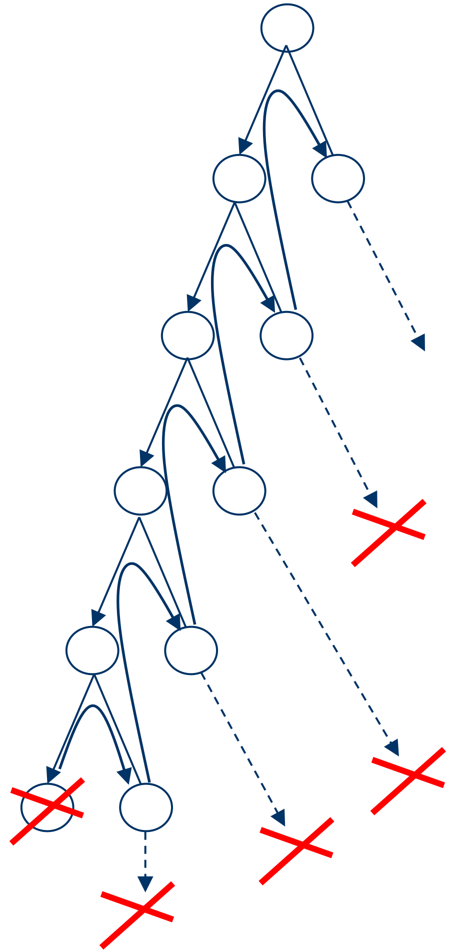
Backtracking Tree Search (BTS)

- Instantiates the variables sequentially.
- By default depth-first traversal.
- Whenever all the variables of a constraint is instantiated, checks the validity of the constraint.
  - In case of dead-end, retracts the most recently posted branching decision (chronological backtracking).
- Systematic search.
  - Eventually finds a solution or proves unsatisfiability.
  - Complexity $O(d^n)$, exponential!



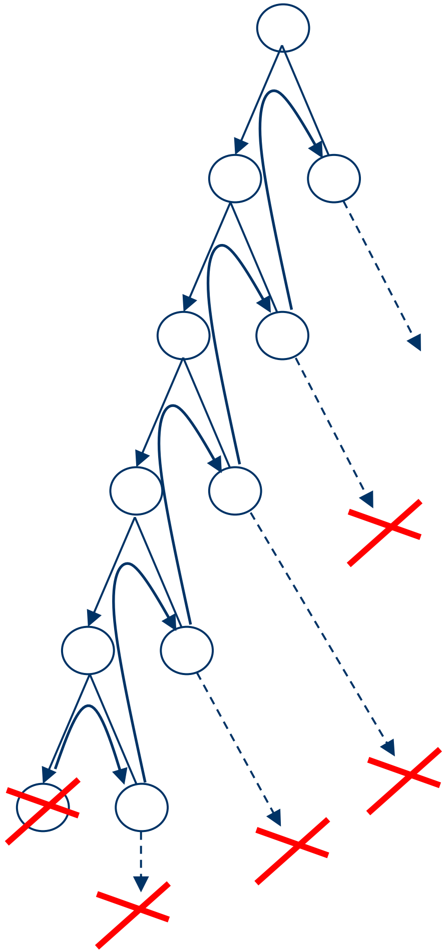
Constraint Solver

- Instantiates the variables sequentially.
- By default depth-first traversal.
- ~~Whenever all the variables of a constraint is instantiated, checks the validity of the constraint.~~
  - ~~In case of dead-end, retracts the most recently posted branching decision (chronological backtracking).~~
- Systematic search.
  - Eventually finds a solution or proves unsatisfiability.
  - Complexity $O(d^n)$, exponential!



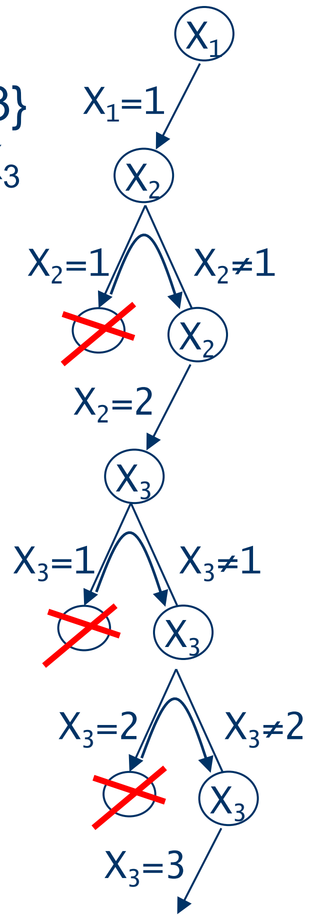
Constraint Solver

- Instantiates the variables sequentially.
- By default depth-first traversal.
- Examines the constraints to remove inconsistent values from the domains of the future (unexplored) variables, via propagation.
  - Shrinks the domains of the future variables.
  - The propagation mechanism propagates all the constraints before search, and only the necessary ones at each search decision.
- Systematic search.
  - Eventually finds a solution or proves unsatisfiability.
  - Complexity $O(d^n)$, exponential!



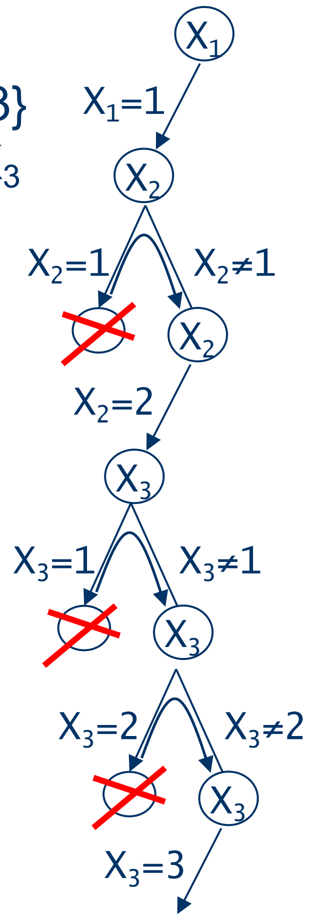
BTS

$D(X_1) = \{1,2\}$
$D(X_2) = D(X_3) = \{1,2,3\}$
$C_1: X_1 \neq X_2$ $C_2: X_2 < X_3$



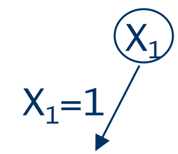
BTS interleaved with Propagation

$D(X_1) = \{1,2\}$
$D(X_2) = D(X_3) = \{1,2,3\}$
C_1: $X_1 \neq X_2$ C_2 : $X_2 < X_3$



Propagation

C_2 : $D(X_2) = \{1,2,\cancel{3}\}$, $D(X_3) = \{\cancel{1},2,3\}$

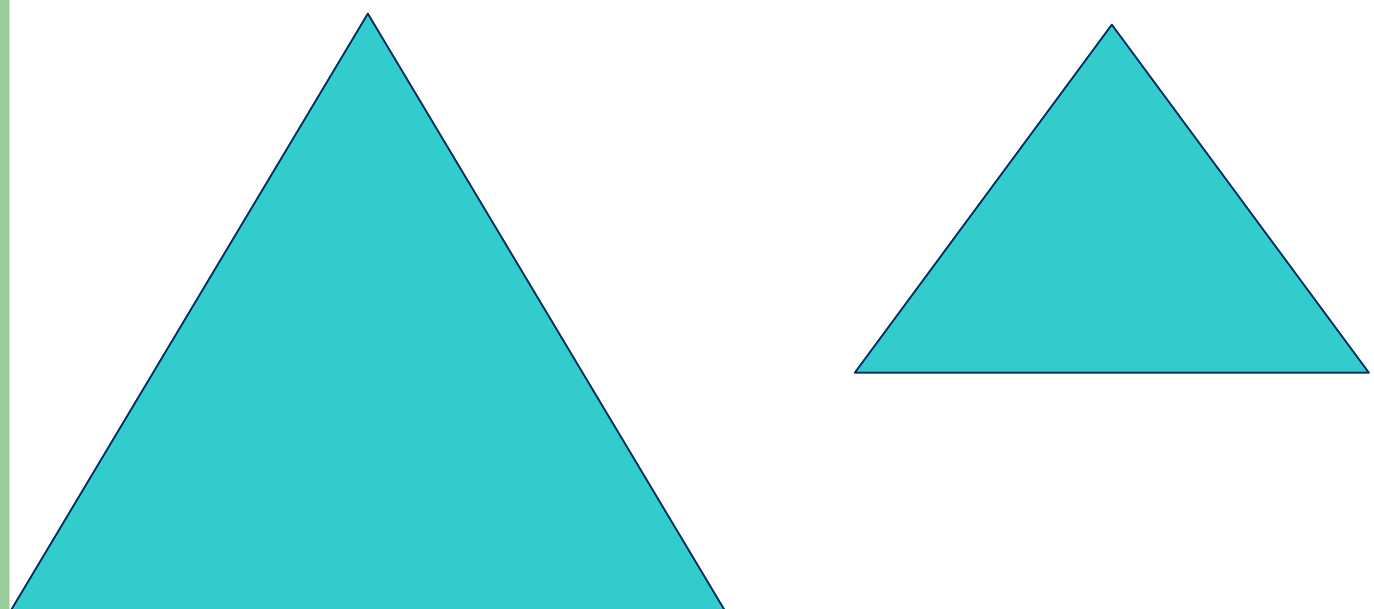


Propagation

C_1 : $D(X_2) = \{\cancel{1},2\}$
C_2 : $D(X_3) = \{\cancel{2},3\}$

BTS interleaved with Propagation

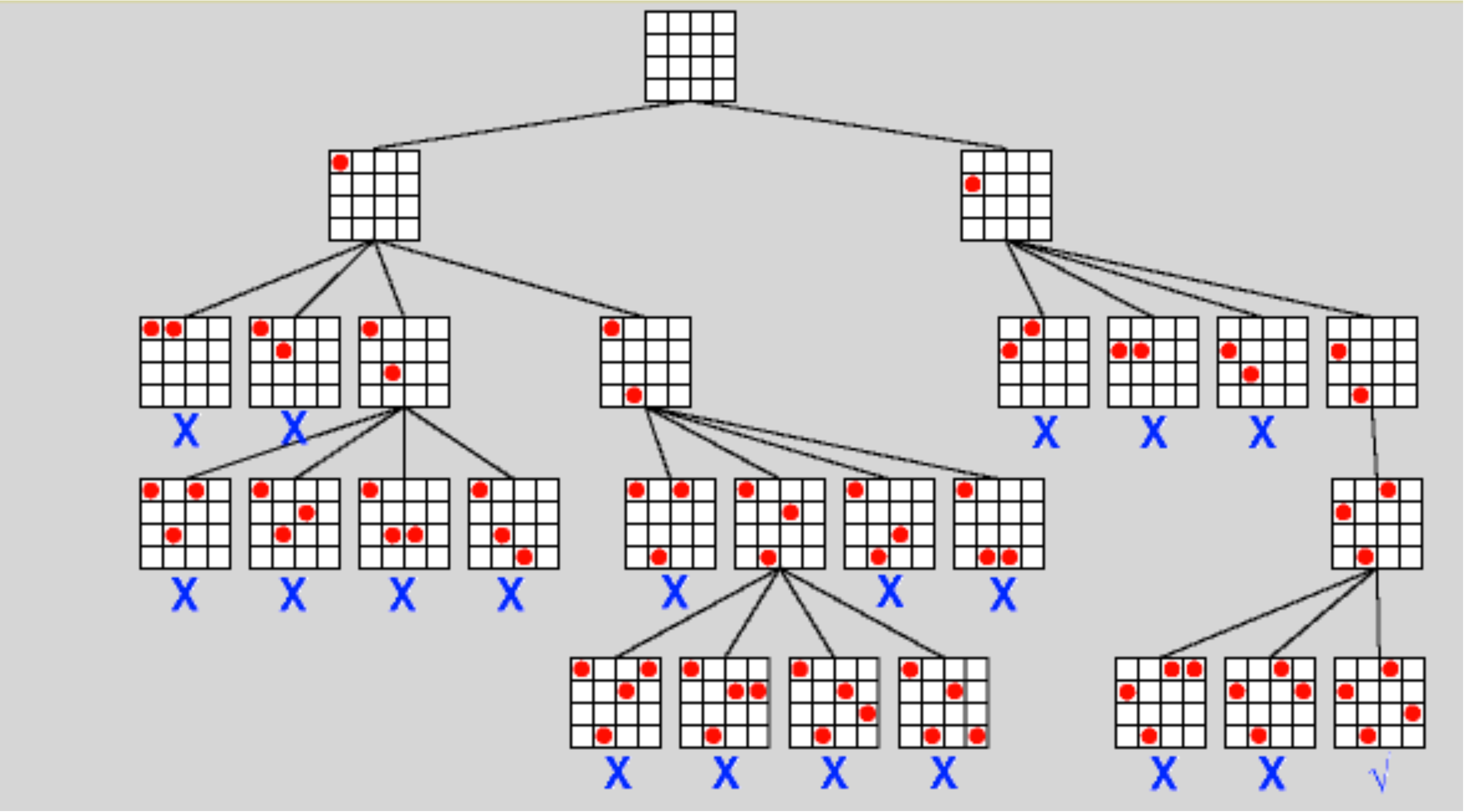
Search vs Search & Propagation



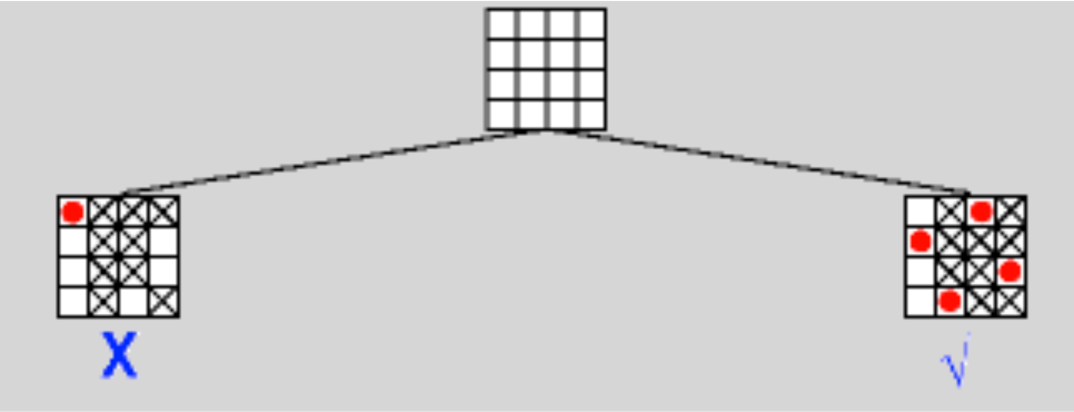
Exponential size

Reduction in the search tree size

BTS for 4-Queens



BTS + AC Propagation



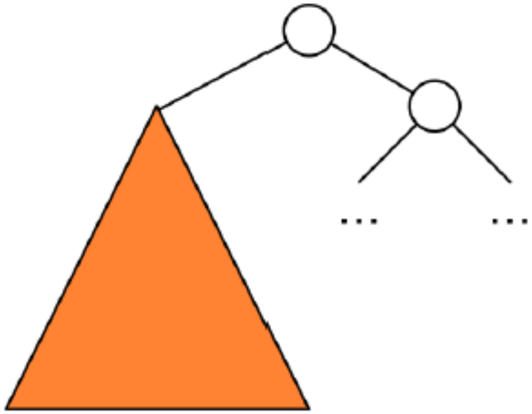
Search Heuristics

- Guide the search decisions.
  - Which variable next? Which value(s) next?
- Problem specific vs generic heuristics.
- Static heuristics
  - Order is known before search. E.g.,
    - X_1, X_2, …X_n, exploring the domains in increasing order.
  - Low cost.
- Dynamic heuristics
  - Order is decided during dynamically during search.
  - Considers the current search state.

Search Heuristics

THAT HAS A SOLUTION

- For feasible problems, choose variables and values that are likely to yield a solution.
  - In general, no guarantee of feasibility.
- What if we make a mistake?
  - Infeasible sub-problem!
  - We need to explore the whole sub-tree before backtracking!
  - We should explore the sub-tree as quickly as possible.



Heuristics for Infeasible Problems

- Fail-first (FF) principle: Try first where you are most likely to fail so as to maximize propagation.
- How do we know if a CSP is feasible or not?
- Trade-off:
  - choose next the variable that is most likely to cause failure;
  - choose next the value that is most likely to be part of a solution (least constrained value).
- Main focus on Variable Ordering Heuristics (VOHs).
  - To backtrack from an infeasible sub-problem, we need to explore all the values in the domain of a variable.

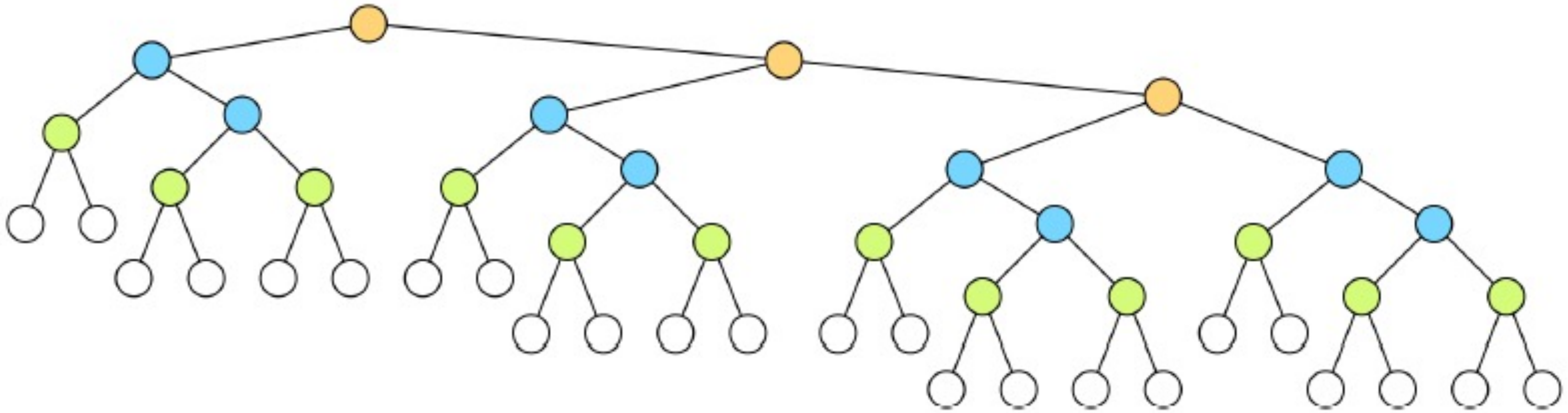
Generic Dynamic VOHs based on FF

- Minimum domain
  - Choose next the variable with minimum domain size.
  - Idea: minimize the search tree size.

Minimum Domain Heuristic

- Consider the order X_1, X_2, X_3.

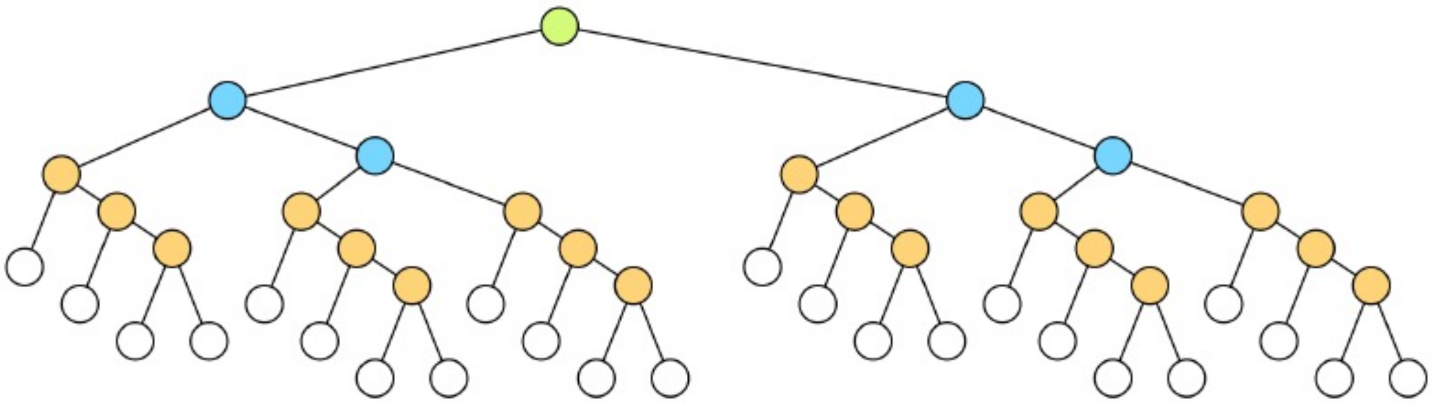
$X_1 \in \{0, 1, 2, 3\}$, $X_2 \in \{0, 1, 2\}$, $X_3 \in \{0, 1\}$



Minimum Domain Heuristic

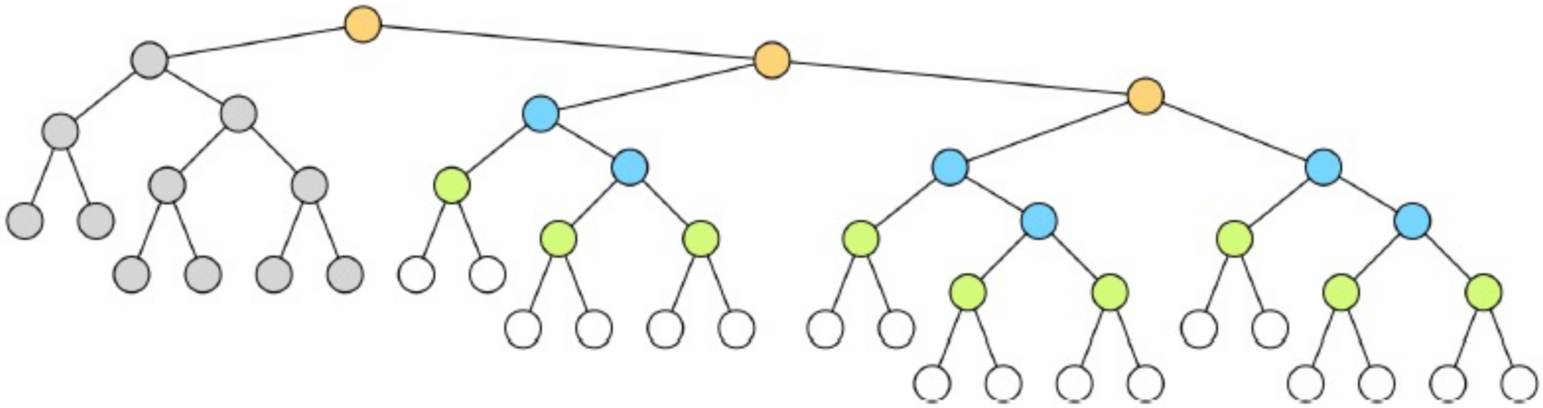
- Consider the order X_3, X_2, X_1.

$X_3 \in \{0, 1\}$, $X_2 \in \{0, 1, 2\}$, $X_1 \in \{0, 1, 2, 3\}$



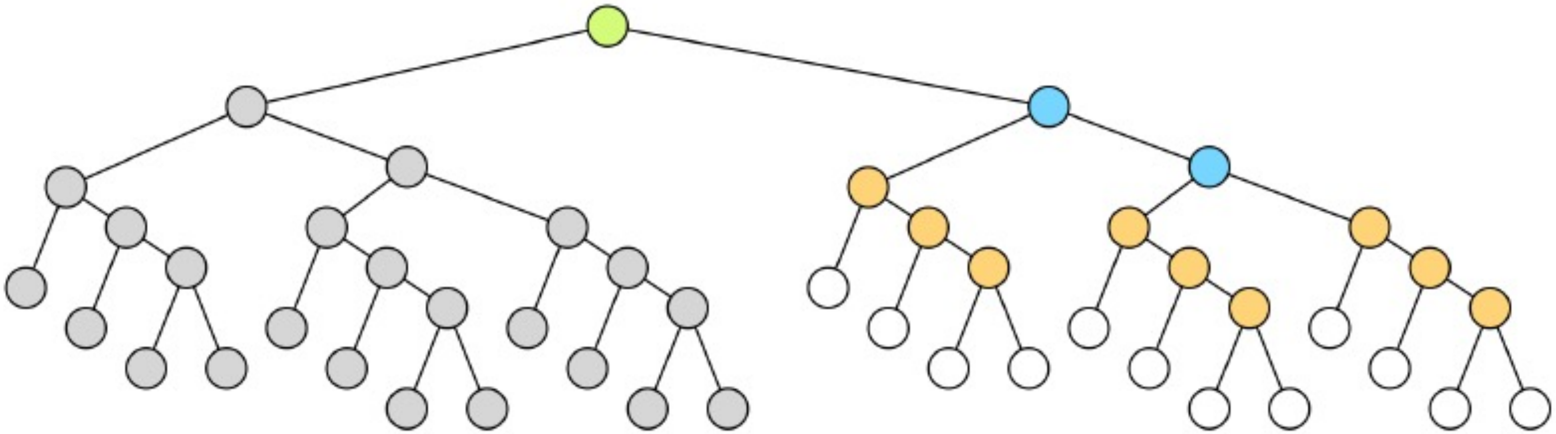
Minimum Domain Heuristic

- If propagation prunes a value at depth 1…



Minimum Domain Heuristic

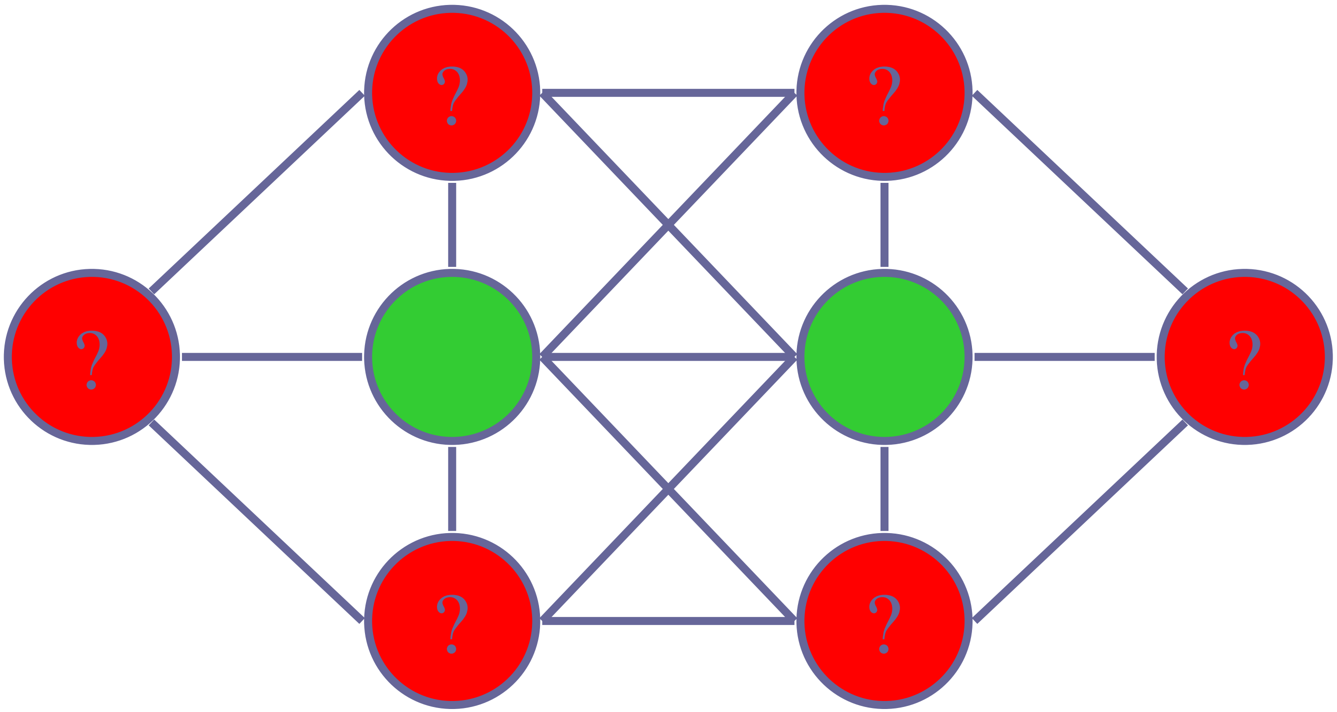
- …the effect is much stronger with the second ordering!



Generic Dynamic VOHs based on FF

- Minimum domain
  - Choose next the variable with minimum domain size.
  - Idea: minimize the search tree size.
- Most constrained (max degree)
  - Choose next the variable involved in most number of constraints.
  - Idea: maximize constraint propagation.

Most Constrained Variables



Generic Dynamic VOHs based on FF

- Minimum domain
  - Choose next the variable with minimum domain size.
  - Idea: minimize the search tree size.
- Most constrained (max degree)
  - Choose next the variable involved in most number of constraints.
  - Idea: maximize constraint propagation.
- Combination
  - Minimize domain size / degree

Weighted Degree Heuristic

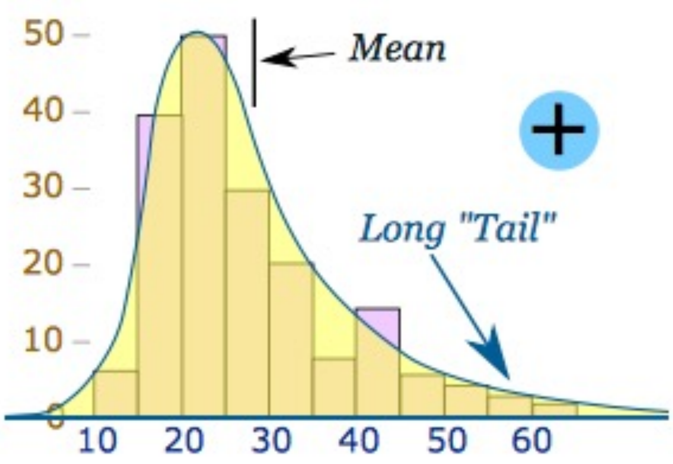
- Constraints are weighted.
  - Initially set to 1.
- During the propagation of a constraint c, its weight w(c) is incremented by 1 if the constraint fails.
- The weighted degree of a variable X_i:

$$w(X_i) = \sum_{c \ s.t.\ X_i \in X(c)} w(c)$$

- Domain over weighted degree heuristic (domWdeg):
  - Choose the variable X_i with minimum $|D(X_i)| / w(X_i)$.

Heavy Tail Behaviour

- Given a collection of instances of a problem, we often observe some exceptionally hard instances that take exceptionally longer time to solve.
  - Large impact on the runtime distributions for a given set of instances.



Heavy Tail Behaviour

- Not a characteristic of the instance!
  - The same behaviour is observed if we run several times the same instance while varying some parameter (like the variable ordering) of the solver.
  - For some combination instance + solver parameters, we get trapped into an exponential subtree.
- Intuitive reason:
  - If we make a mistake early during search, we get stuck in a subtree.
    - Remember the puzzle example!
  - Different heuristics lead to “bad” mistakes on different instances.
- Observation: such mistakes are seemingly random.

Heavy Tail Behaviour

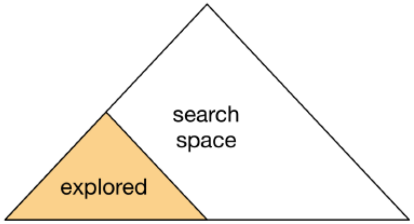
- Randomization
  - Add some randomized parameter in search. E.g.,
    - Pick (some) variables/values at random.
    - Break ties randomly.
  - Given the same random seed the solver will explore the same tree, however it will never explore two identical subproblems in the same way.

Heavy Tail Behaviour

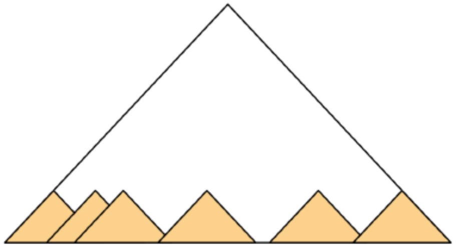
- Restarting
  - Restart the search, after certain amount of resources are consumed.
    - Usually in the form of search steps, such as the number of visited nodes.
  - In the subsequent runs, search differently.
    - Introduce randomization.
    - Learn from the accumulated experiences of previous runs.

Heavy Tail Behaviour

- Randomization + restarts eliminates the huge variance in solver performance.
- Without randomization + restarts



- With randomization + restarts



Restart Strategies

- Constant restart
  - Restart after using L resources.
- Geometric restart
  - Restart after L resources, with the new limit α*L.
  - Ends up being L, α*L, α^{2*}L, α^{3*}L, …
- Luby restart
  - Restart after s[i]*L resources where s[i] is the i^{th} number in the Luby sequence = [1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, …], which repeats two copies of the sequence ending in 2^i before adding the number 2^{i+1}.

domWdeg & Restarts

- domWdeg heuristic works well with restart.
  - Collected fail counts are carried over to subsequent runs.
- domWdeg combined with random choice of values can be very effective!

Constraint Optimization Problems (COPs)

- CSP enhanced with an optimization criterion, e.g.:
  - minimum cost;
  - shortest distance;
  - fastest route;
  - maximum profit.
- Formally, **<X,D,C,f>** where **f** is the formalization of the optimization criterion as an objective function. Goal: minimize **f** (maximize –**f**).

Branch & Bound Algorithm

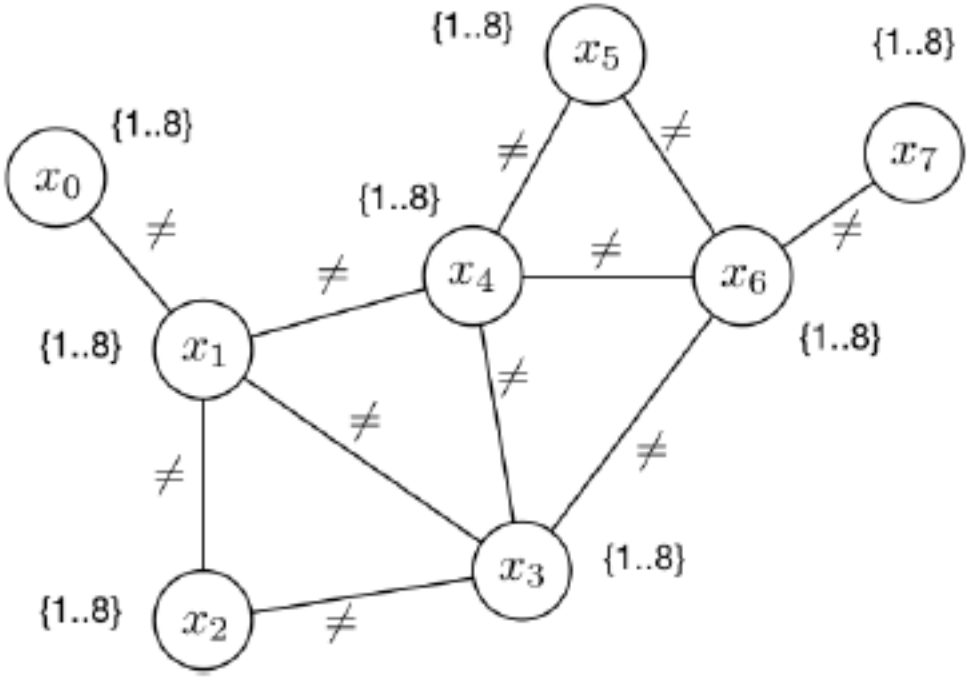
- Solves a sequence of CSPs to solve a COP and incorporates bounding in the search.
- How?
  - Each time a feasible solution is found, posts a new bounding constraint which ensures that a future solution must be better than it.
  - Backtracks to the last decision and looks for a new solution with the additional bounding constraint, using the same search tree.
  - Repeats until infeasible: the last solution found is optimal.

Optimal Map Colouring

- What is the minimum number of colours to colour a map?
- Variables and Domains
  - X_i for each of n regions with domain {1,…,n}
- Constraints
  - $X_i \neq X_j$ for each neighbour region i and j
- Objective function
  - $f(X) = \max(X_i)$
- Objective
  - min $f(X)$

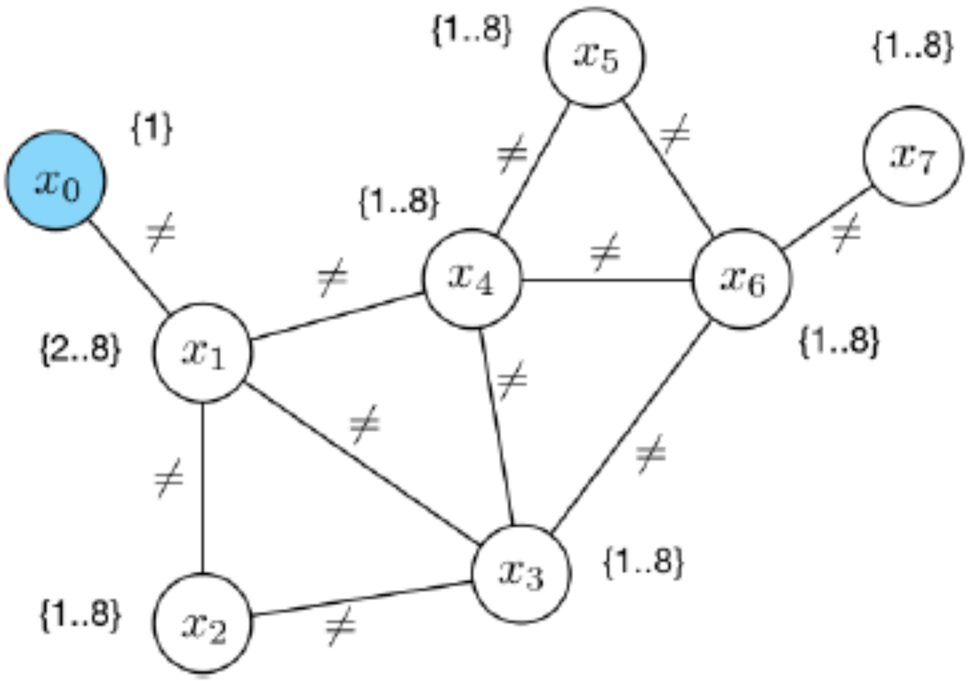
Solving Optimal Map Colouring with Branch & Bound

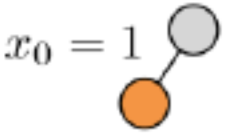
$$\max_{i=0..7}(x_i) \in \{1..8\}$$



Solving Optimal Map Colouring with Branch & Bound

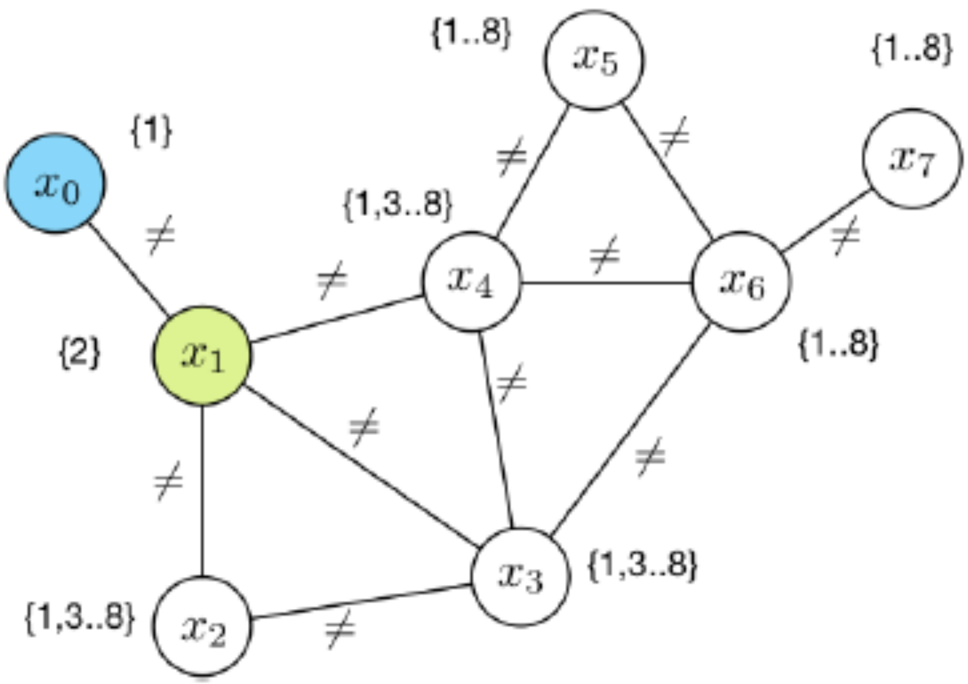
$$\max_{i=0..7}(x_i) \in \{1..8\}$$

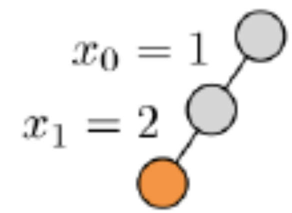




Solving Optimal Map Colouring with Branch & Bound

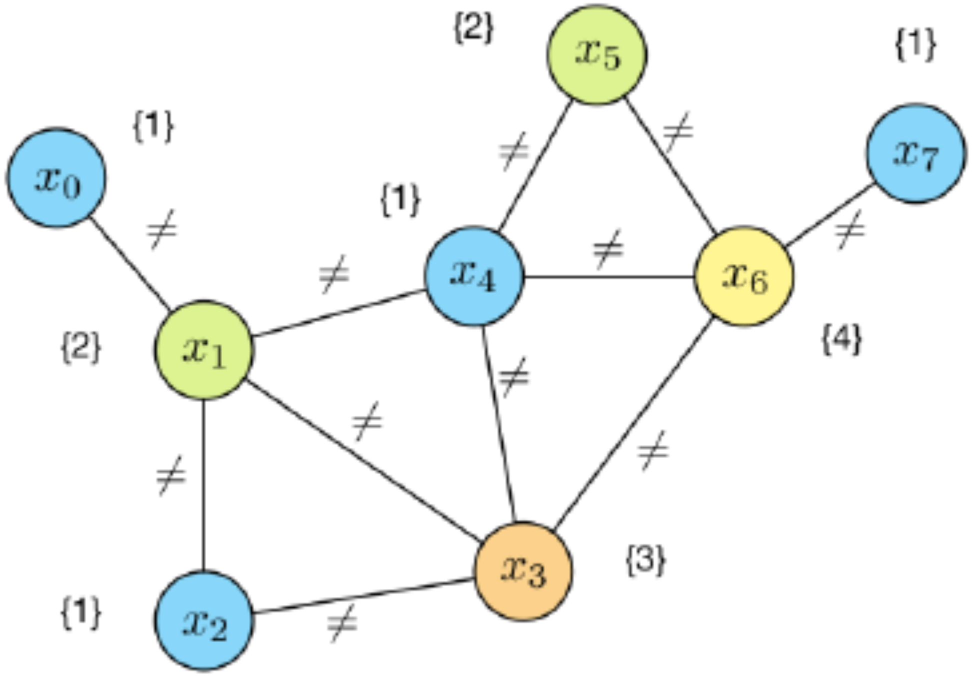
$$\max_{i=0..7}(x_i) \in \{2..8\}$$

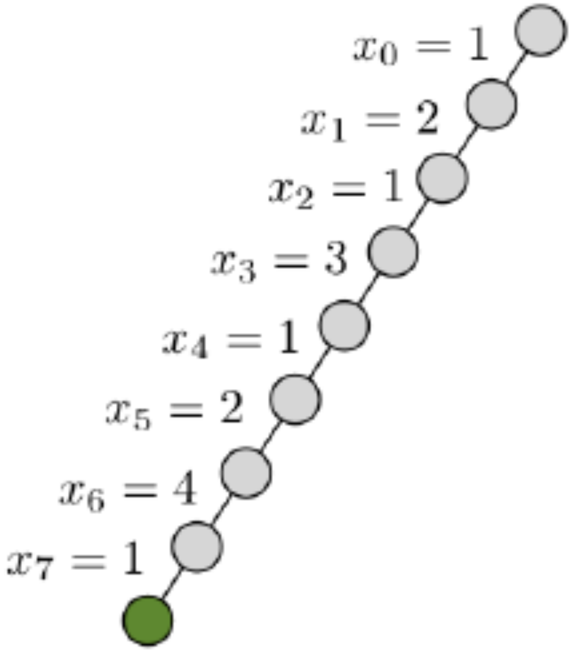




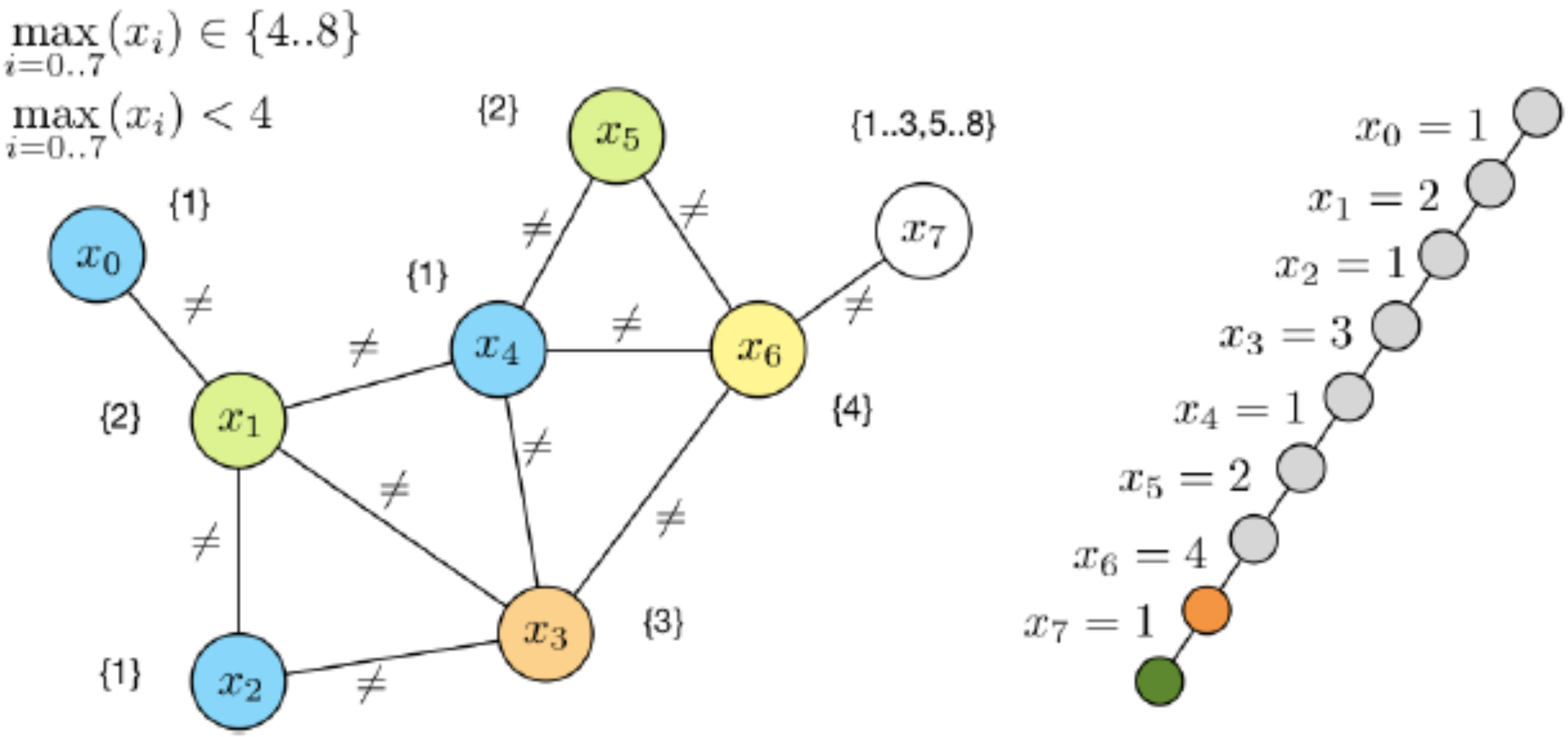
Solving Optimal Map Colouring with Branch & Bound

$$\max_{i=0..7}(x_i) \in \{4\}$$





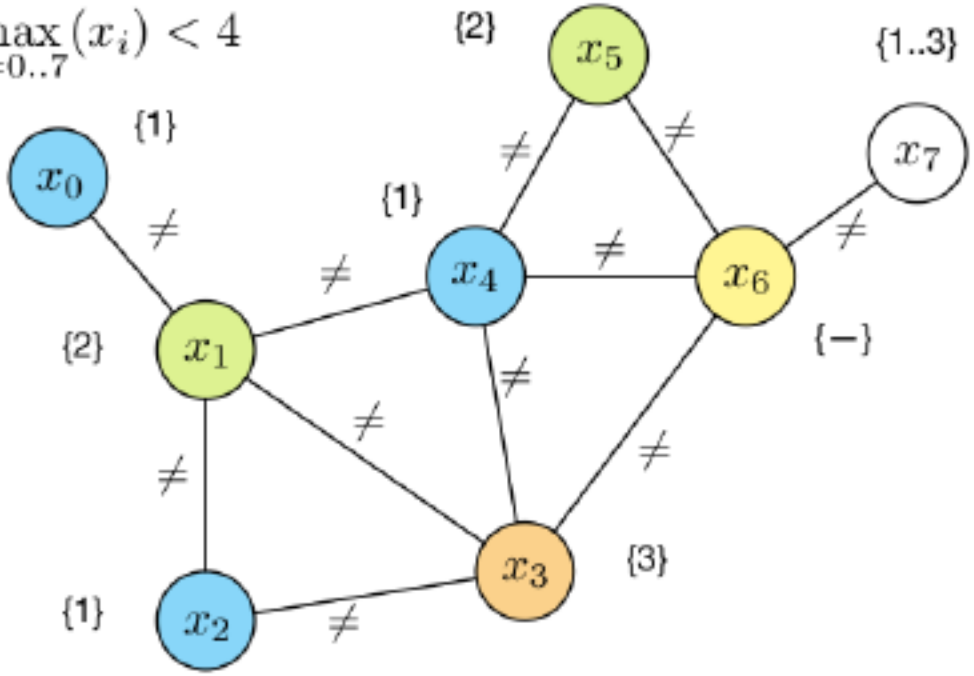
Solving Optimal Map Colouring with Branch & Bound

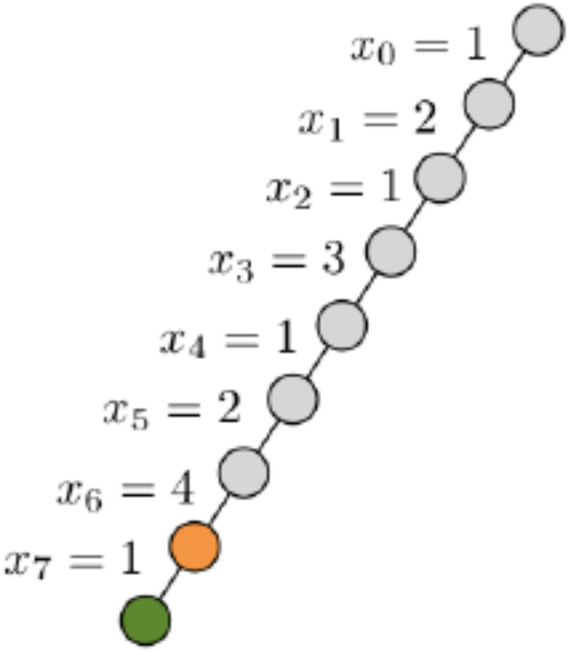


Solving Optimal Map Colouring with Branch & Bound

$$\max_{i=0..7}(x_i) \in \{-\}$$

$$\max_{i=0..7}(x_i) < 4$$

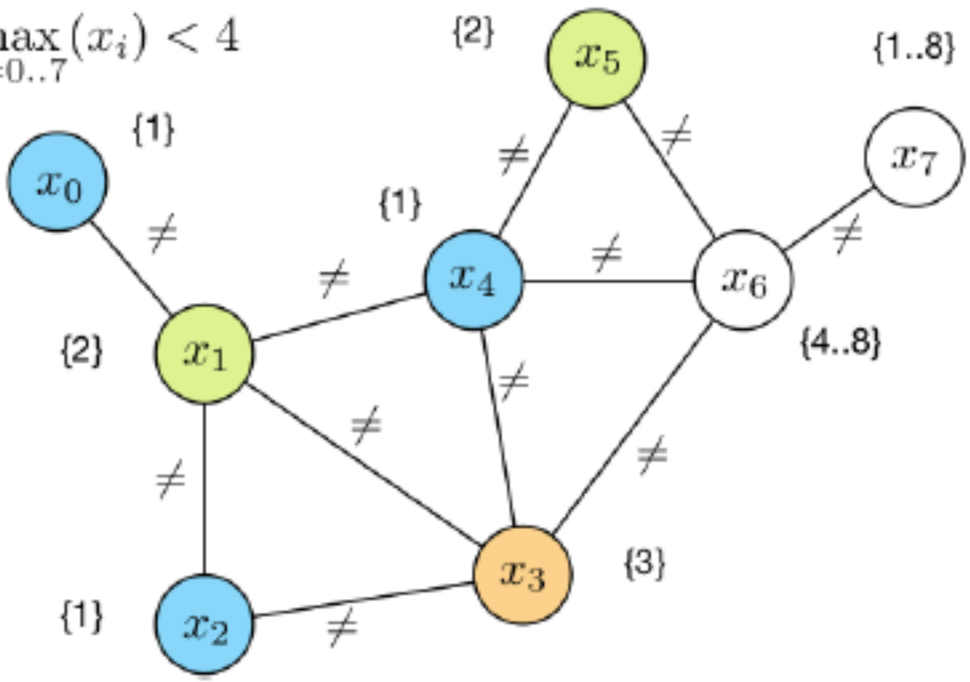


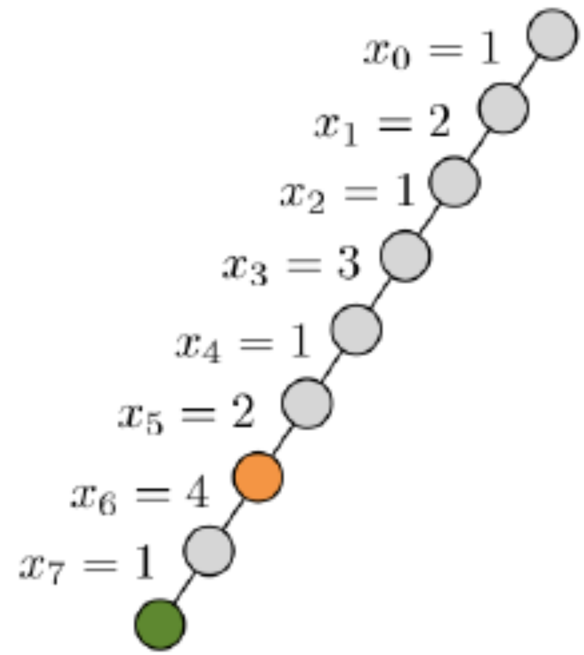


Solving Optimal Map Colouring with Branch & Bound

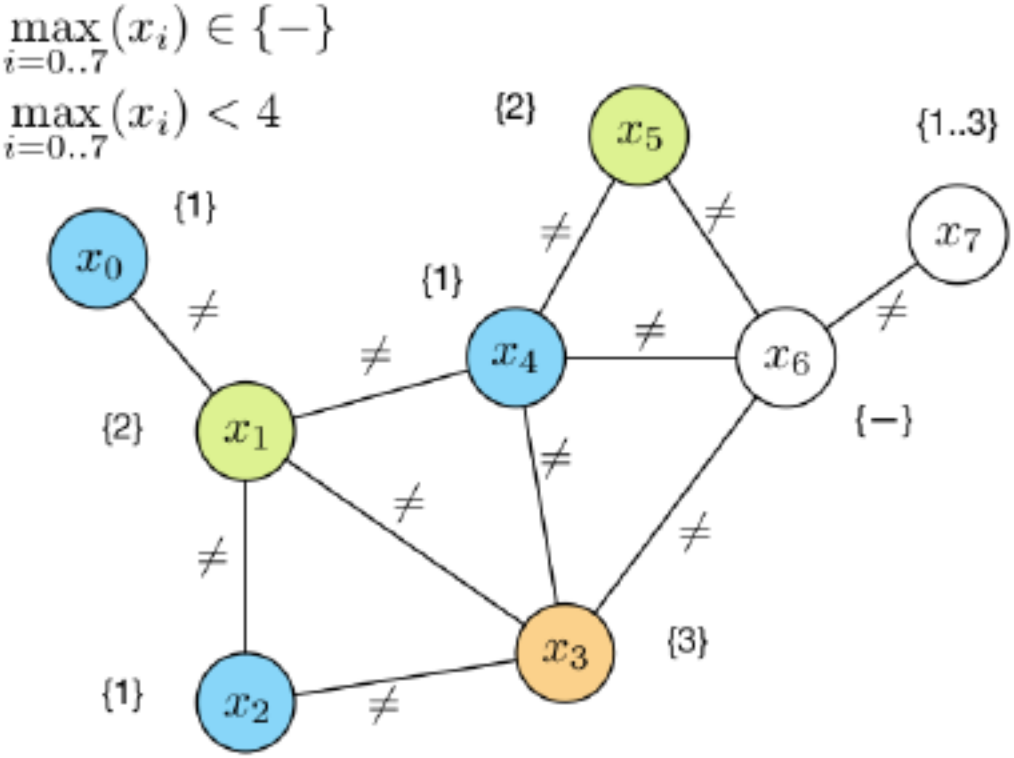
$$\max_{i=0..7}(x_i) \in \{4..8\}$$

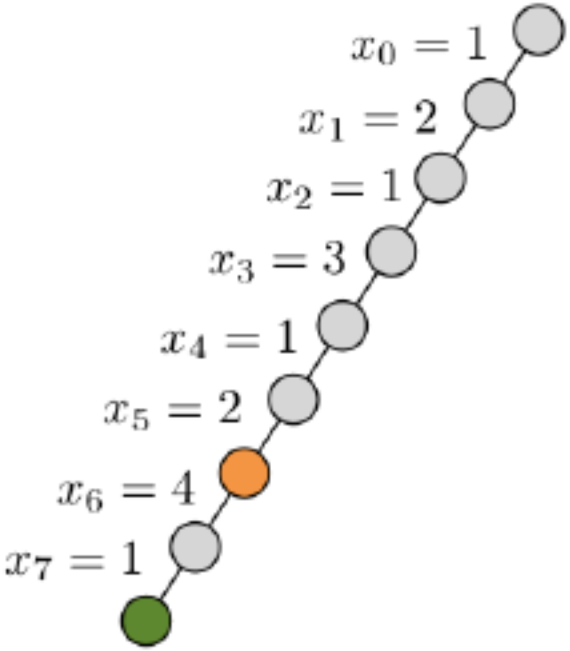
$$\max_{i=0..7}(x_i) < 4$$



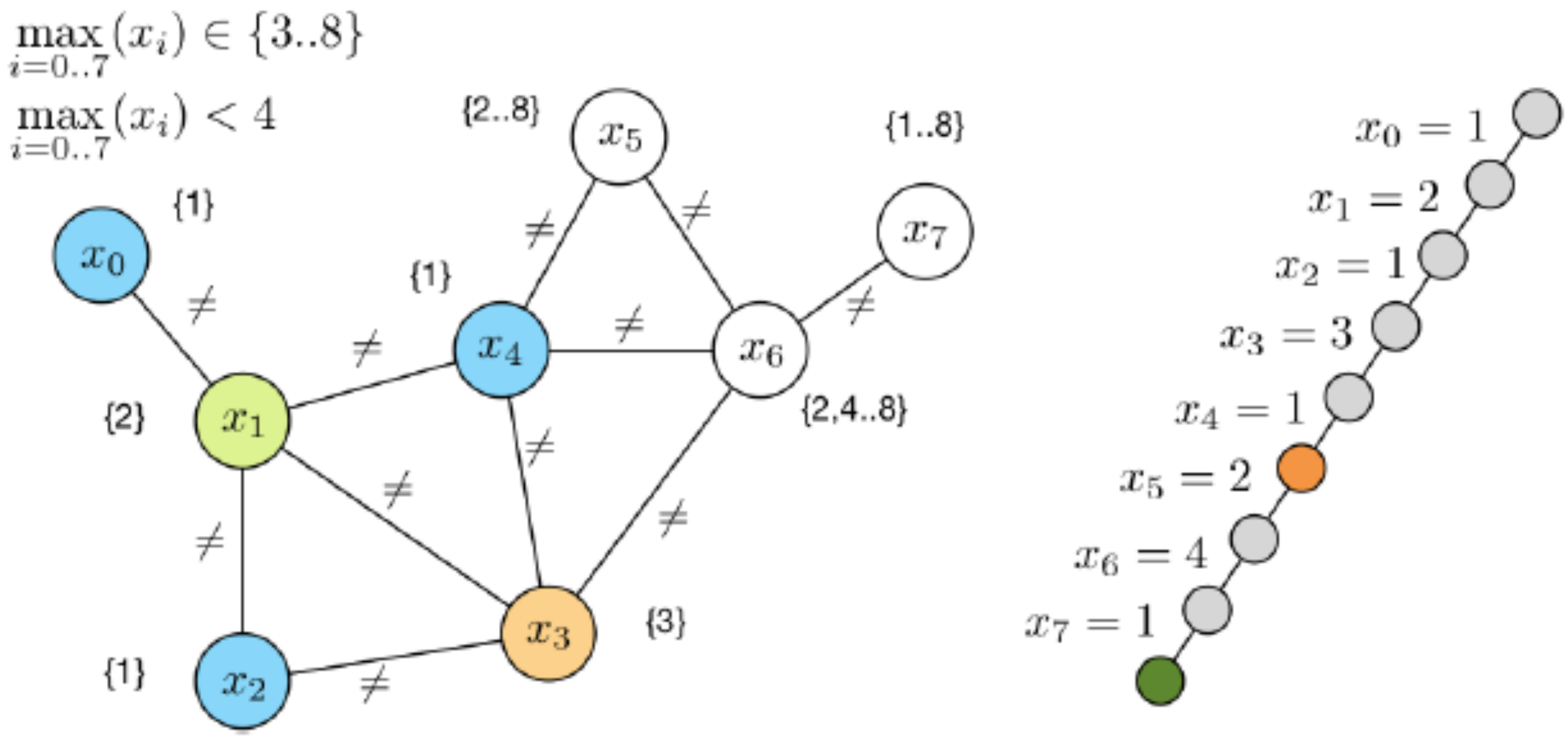


Solving Optimal Map Colouring with Branch & Bound





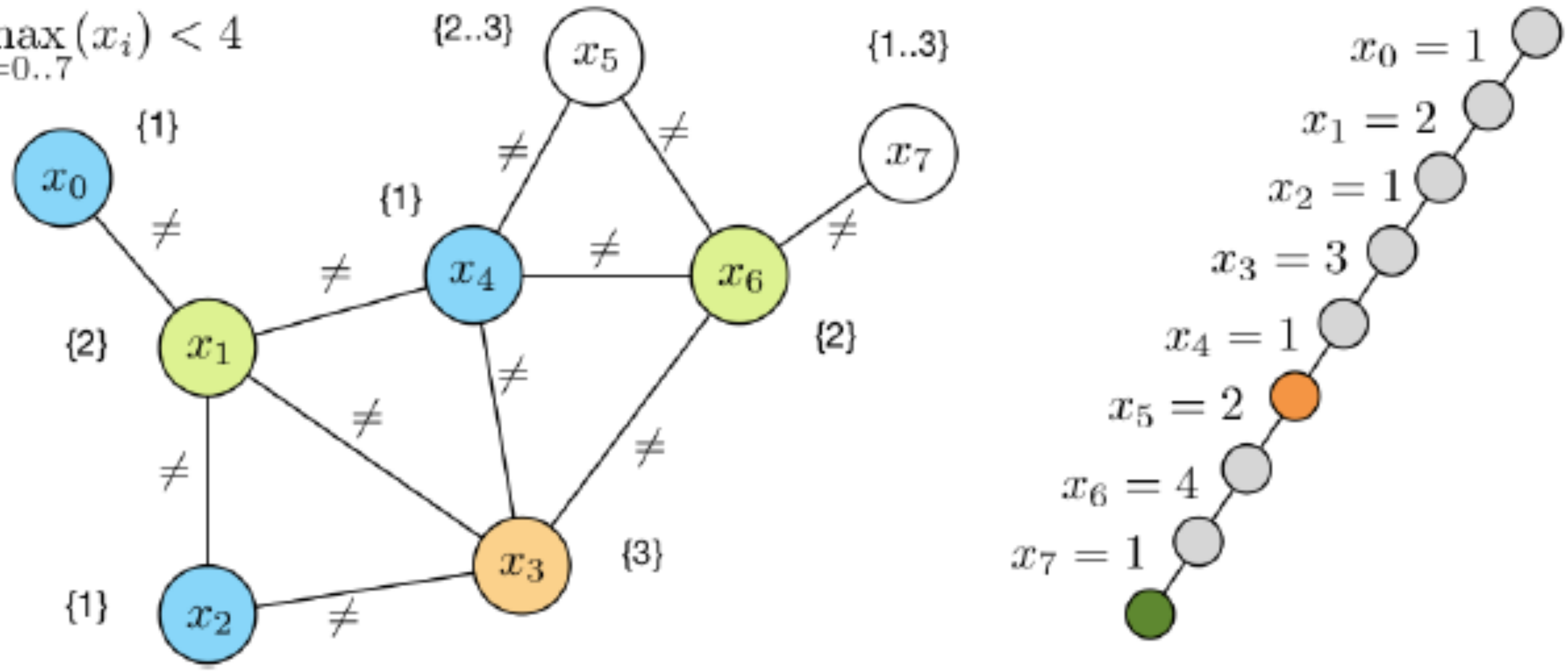
Solving Optimal Map Colouring with Branch & Bound



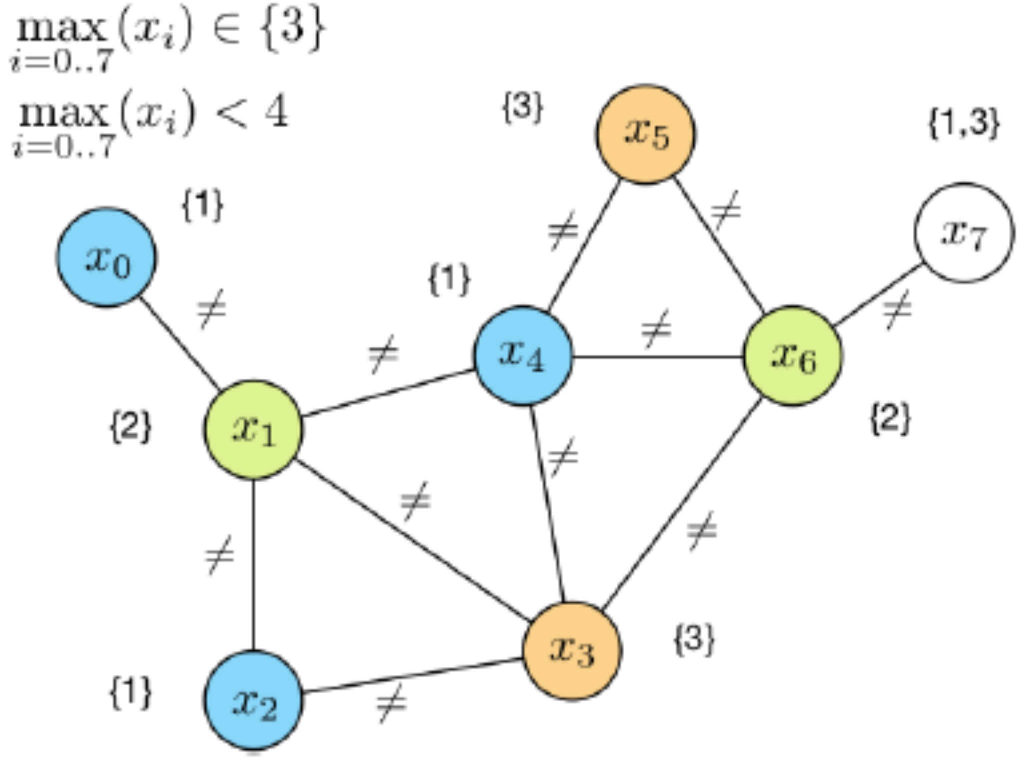
Solving Optimal Map Colouring with Branch & Bound

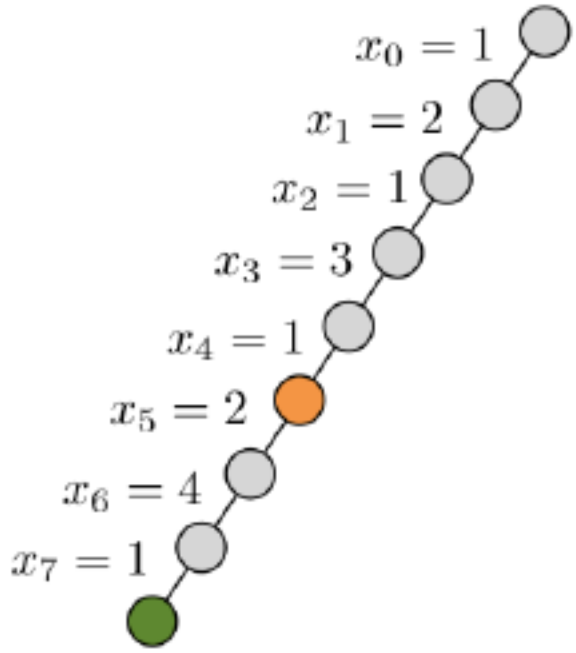
$$\max_{i=0..7}(x_i) \in \{3\}$$

$$\max_{i=0..7}(x_i) < 4$$



Solving Optimal Map Colouring with Branch & Bound

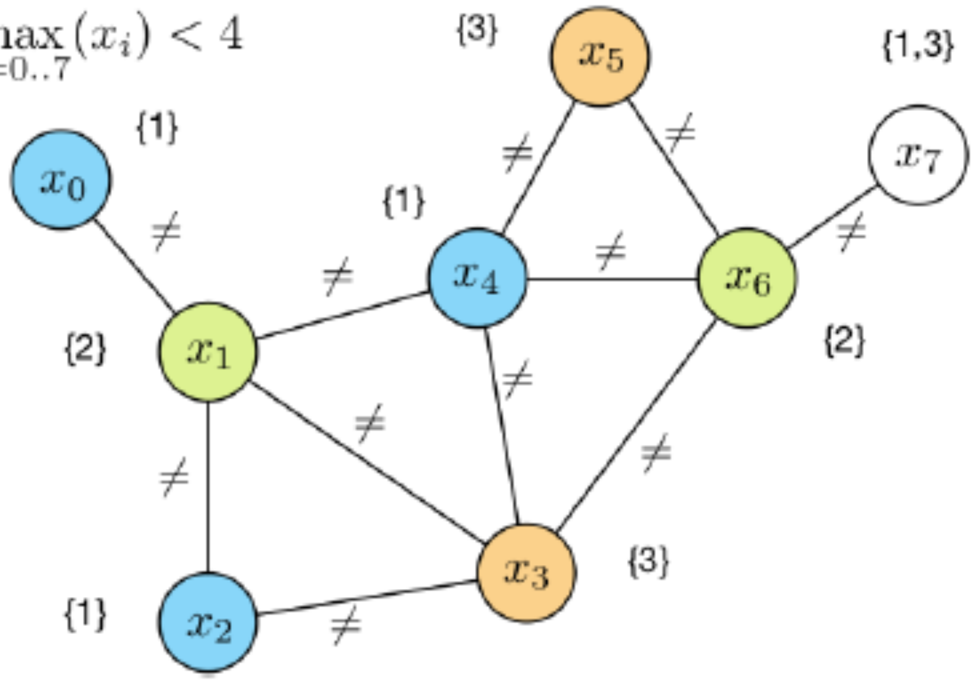


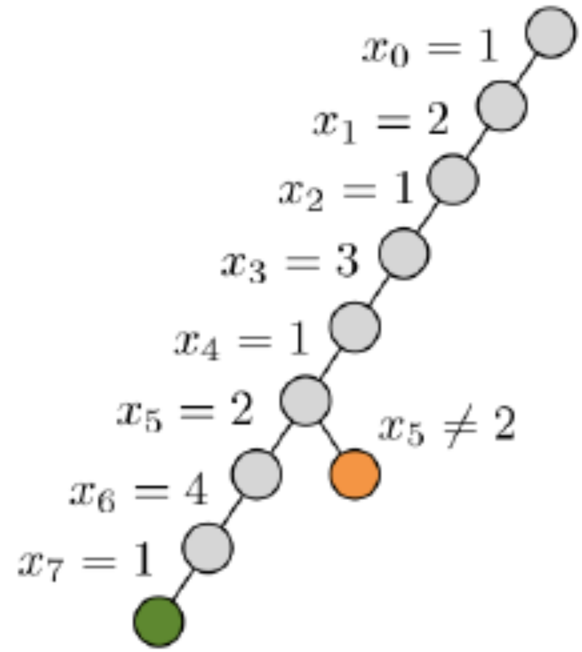


Solving Optimal Map Colouring with Branch & Bound

$$\max_{i=0..7}(x_i) \in \{3\}$$

$$\max_{i=0..7}(x_i) < 4$$

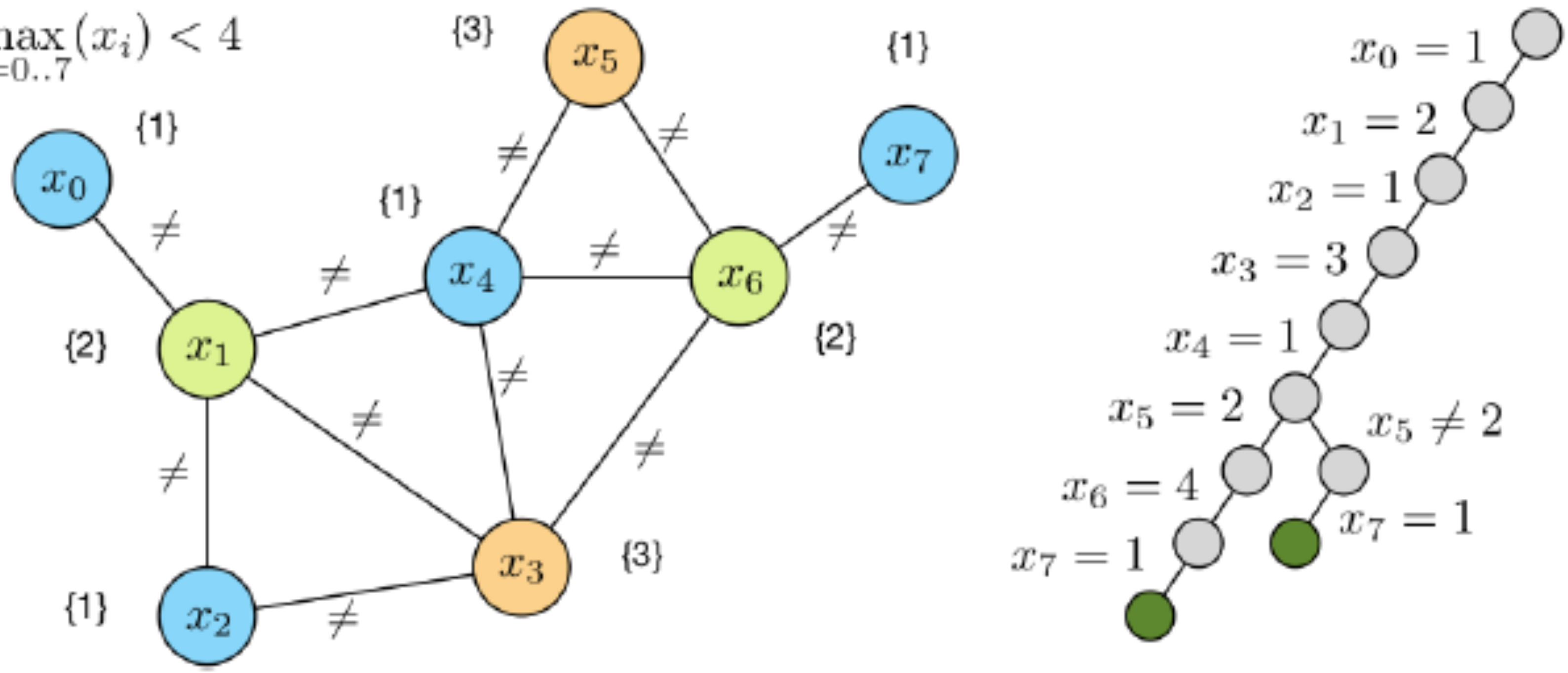




Solving Optimal Map Colouring with Branch & Bound

$$\max_{i=0..7}(x_i) \in \{3\}$$

$$\max_{i=0..7}(x_i) < 4$$

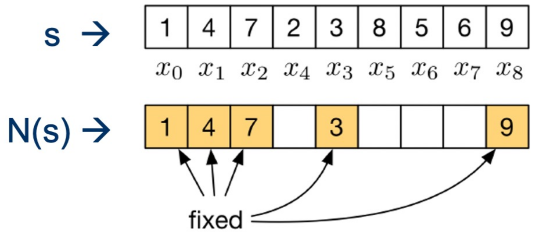


Complementary Strengths

- CP
  - + A generic complete approach with a focus on constraints and feasibility.
  - + Easy modelling and control of search.
  - – Poor in optimization with loose bounds on the objective function.
  - – Cannot scale to large optimization problems.
- Heuristic Search
  - + Scales to large optimization problems.
  - + Effective in finding good-quality solutions quickly.
  - – Neighborhoods are problem-specific.
  - – Constraints are handled inefficiently, i.e., often by penalizing infeasible assignments in the objective function.
  - – Finding an initial good solution and exploring a large neighborhood can be a challenge.

Large Neighbourhood Search

- A hybrid CP-HS method combining the benefits of both worlds.
- Use CP to find an initial solution s.
- Define a problem-independent generic large neighbourhood.
  - Given the initial solution s:
    - fix part of the variables to the values they have in s (called fragment);
    - relax the remaining variables.



- Explore the large neighbourhood with CP!
  - View the exploration of a neighbourhood as the solution of a sub-problem;
  - use propagation and advanced search techniques of CP to exhaustively and efficiently explore it.

Advantages over HS and CP

- Efficient neighbourhood exploration.
  - Thanks to propagation and advanced search techniques of CP.
- LNS is easier to develop than HS.
  - Easy and generic neighbourhood definition.
- More scalable than using only CP on the problem.
  - Subproblems are typically much smaller.
  - We can control the subproblem size.
  - The fixed-variables reduce the domain sizes.
  - Propagation works best when domains are small.

Advanced Search

- Other forms of search tree traversal.
  - Best-first search algorithms (limited discrepancy search, …)
- Sophisticated dynamic variable ordering heuristics.
  - Impact-based, activity-based, regrets, …
- Non-chronological backtracking.
  - Conflict-based, no-good learning, …
- Specific approaches for optimization problems.
  - Schedule-or-postpone search, ...
  - Integration of ILP models for obtaining lower bounds, cost-based propagation, …