DECISION MAKING WITH CONSTRAINT PROGRAMMING

2023/2024

Second cycle degree/two year
Master in Computer Science
Dept of Computer Science and Engineering (DISI)
University of Bologna

Course Info

- **Lecturer**
  - Zeynep KIZILTAN (associate professor in AI and optimization).
  - Email: zeynep.kiziltan@unibo.it
  - Appointment at Teams, upon request by email.
- **Content**
  - Fundamentals of Constraint Programming (CP), a general-purpose AI-based approach to combinatorial decision making.
- **Prerequisites**
  - Basic computer science such as discrete mathematics, logic, algorithms and data structures, programming.
  - Prior knowledge on AI is not necessary.

Course Info

- **Timetable:** September 18 – December 5
  - 5 + 5 weeks
    - No lecture during the weeks of October 23 and 30.
  - Monday 11:00 – 13:00 (Aula Bombelli)
  - Tuesday 14:00 – 16:00 (Aula E1)
- **Lectures**
  - Theory and practice via programming exercises using personal laptops.
  - Lecture timing (please vote later in Virtuale):
    - A: Start on time, finish 10 mins before?
    - B: Start 10 mins later, finish on time?

Course Info

- **Teaching Tools**
  - Virtuale platform
    - Distribution of the course material (lecture slides, exercises, lecture recordings, resources, etc).
    - Communication between the students and the lecturer.
    - Discussion of anything related to the course.
    - Working on programing exercises **interactively**.
    - Exchange of feedback.
    - Participation to polls and informal quiz.

Course Info

- **Teaching Tools**
  - Virtuale platform
    - Participate now!
      - Enrollment: study programme and password (230901).
    - Add a profile photo for fast recognition.
    - Check your UniBo email **frequently**!
    - Check the course syllabus to program yourselves.
    - Activate notifications.
    - Material will be available before the lectures.
      - Take a look at them in advance.

Course Info

- **Exam**
  - Programming exercises.
    - To complete and submit following the exercise sessions.
      - Interactively with the lecturer via Virtuale.
      - Try and complete each one before the next exercise session.
    - **First deadline for completion:** November 1 (the first two exercises)
    - **Final deadline for completion**: December 19 (all the exercises)
  - Oral exam on the course contents.
    - January/February for those completed the exercises in December.
    - At a later time for those completed after December and by September.
    - Is not granted otherwise, need to repeat the course next year.
  - Final grade in equal parts.

Course Info

- **Programming Tool**
  - **MiniZinc**
    - a modeling language with interfaces to several CP (and other) solvers (https://www.minizinc.org/),
    - by Monash University in collaboration with Data61 and the University of Melbourne.
    - Free and well-documented.
    - Download it and start getting familiar with it.

Course Info

- **Tips**
  - Theory lectures are important to understand the practice and for the oral exam.
  - Practical sessions are important to correctly complete the exercises and for the oral exam.
  - Participation and engagement are vital!
    - Ask questions, don't be shy ☺
    - Follow the Virtuale page!
    - Use the forum for discussions and exchange of knowledge, rather than sending emails to the lecturer.
    - Answer questions, don't be humble ☺
    - Submit exercises on time, long before the deadlines to for modifications and resubmissions.

Course Info

- **FAQ**
  - Can I follow the course via the video recordings?
    - Sure, if necessary (health, work, uncancellable appointment etc) for some lectures.
    - Not recommended if you cannot participate at all.
      - Remember that you cannot get engaged ☹
  - I graduate in October, can I catch up?
    - Yes! Follow the Virtuale page.
  - I graduate in December, can I catch up?
    - No! Please take the course next year.

Introduce Yourself

- Send a message to the discussion forum under the topic “Hello!”.
  - Name & surname.
  - Exchange student or not.
  - Degree programme & year.
  - Bachelor background.
  - Followed the course previously?
  - Prior knowledge and experience with Mathematical Programming & CP.
  - Any particular situation? Especially for exchange students.

Introduction

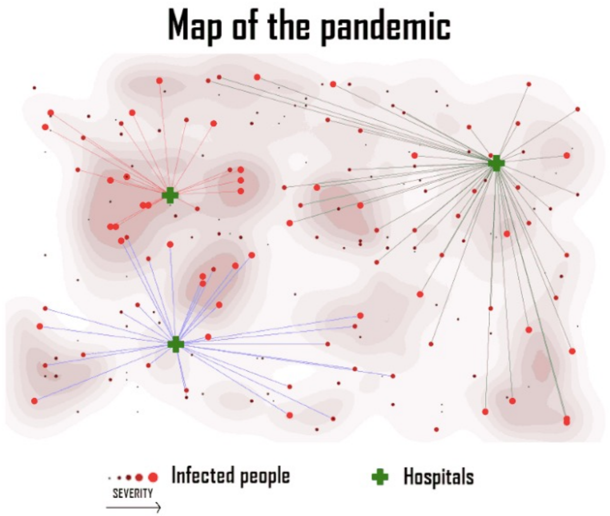
- Combinatorial Decision Making.
- Why with Constraint Programming (CP)?
- Overview of CP.
- Examples from MiniZinc.

Combinatorial Decision Making

- Decision making within many combinations of possibilities subject to restrictions = constraints.
  - Any solution (that meets all constraints).
  - Optimal solution (best solution according to an objective).
- Can appear under different names, e.g.,
  - combinatorial optimization.
  - constraint satisfaction/optimization.
- Common in our daily lives, business, industry and science.

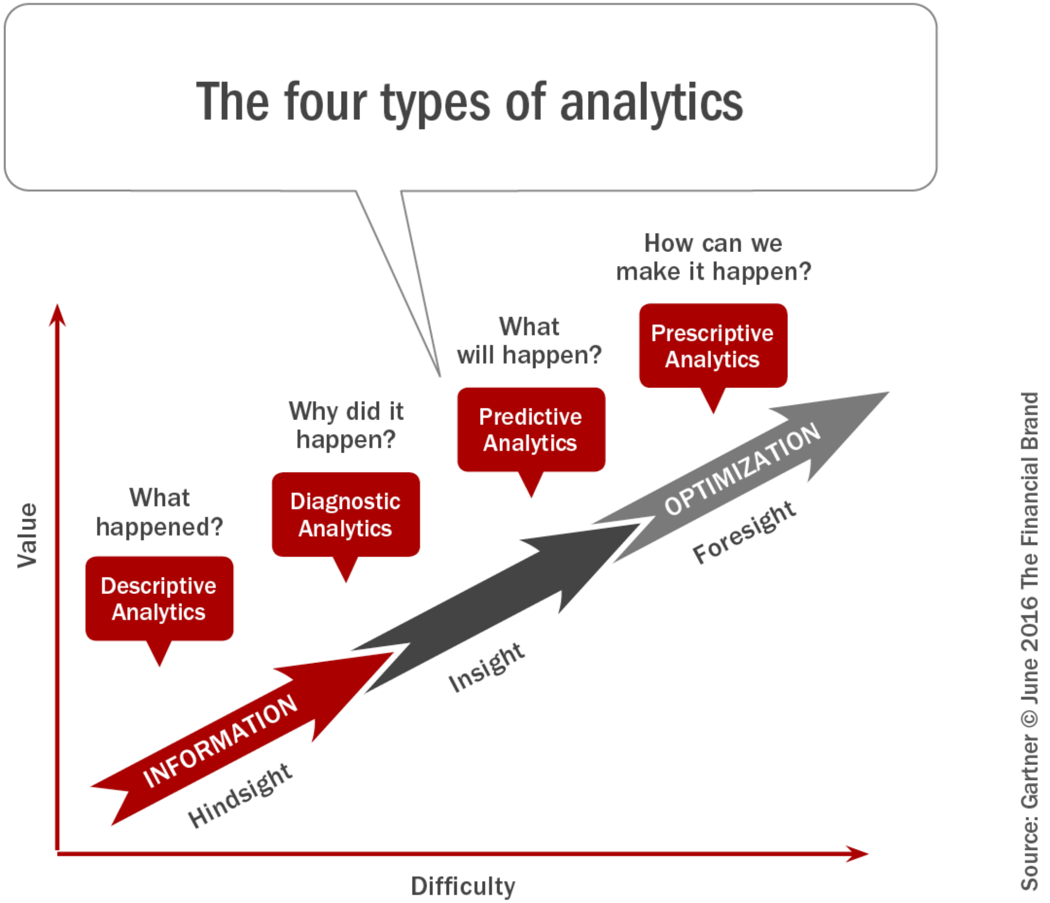
Hospitalization during the Pandemic

- Assign infected people to hospitals according to:
  - severity of illness,
  - patient age,
  - patient location,
  - hospital capacity,
  - hospital equipment, etc.



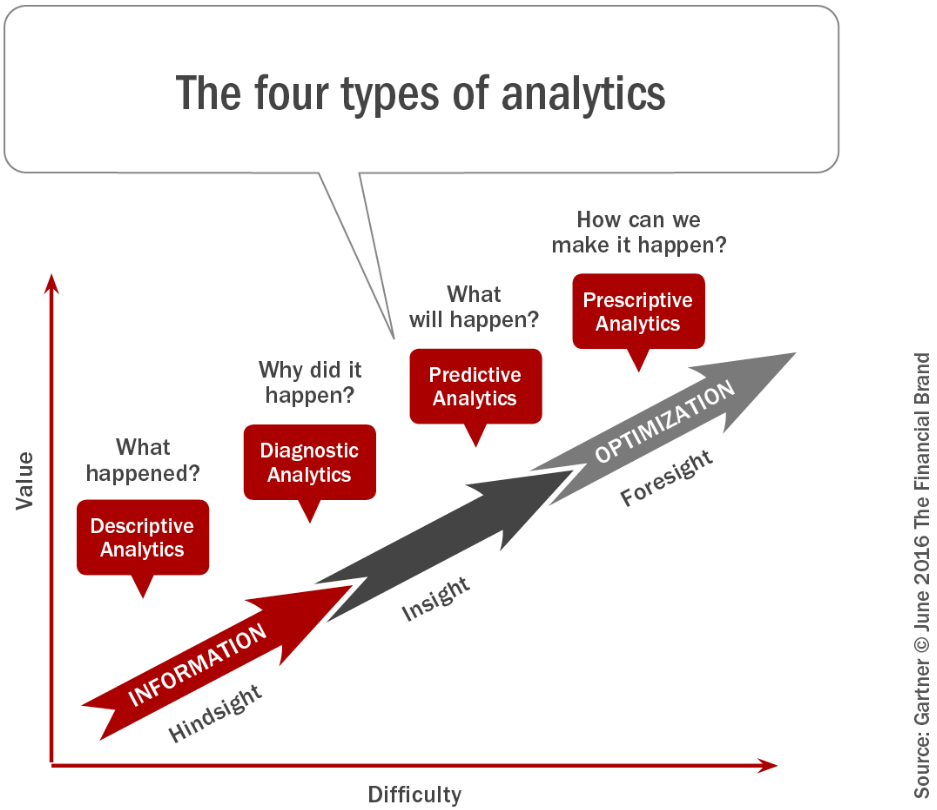
- An approach like neural networks is not suitable:
  - no historical data for training,
  - data cleaning and consolidation is time consuming,
  - a variety of architectures would need to be tested with lengthy training sessions.

Data Analytics



Data Analytics

- AI is not just for machine learning, but also for decision support.



Combinatorial Decision Making

- Properties
  - Computationally difficult (NP-hard in general).
  - Can only be solved by intelligent search.
  - Experimental in nature.
  - Finding good/optimal solutions can save time, $ and reduce environmental impact.
- Many solution techniques
  - Integer Linear Programming (ILP).
  - Boolean SATisfiability, SAT Modulo Theories (SMT).
  - Heuristic search methods (HS).
  - Constraint Programming (CP).

Popularity of Constraint Programming

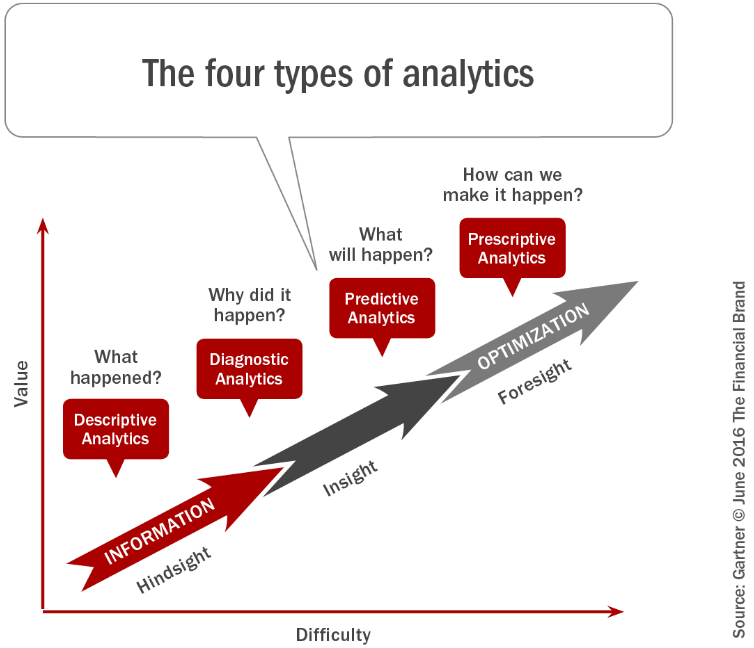
- An important and growing area of AI.
  - Universities, research centers and companies (such as IBM, Google) around the world contribute to the advancement of the state-of-the-art.
  - Many companies are applying CP successfully.
    - Including IBM, Google, Ericsson, Siemens, Renault, Oracle, Sap, Intel, Tacton.
- Technology of choice in logistics, scheduling, planning…
- A useful asset on the job market!

Example: Covid-19 Test Scheduling

- Ocado Retail Ltd, one of the world's biggest online-only grocery retail businesses.
- Employs over 15K people, many of them performing frontline roles such as packing in the warehouses, delivering orders, providing customer service in the call centers etc.
- With the pandemic, the company decided to test all frontline employees on a weekly basis, which required scheduling the employees at each site subject to various constraints.
  - Proved difficult to solve manually.
- Data Science team developed a CP-based solution, which was successfully used to schedule up to 3,500 employees across 4 sites (IFORS news, vol. 15, number 4, December 2020)

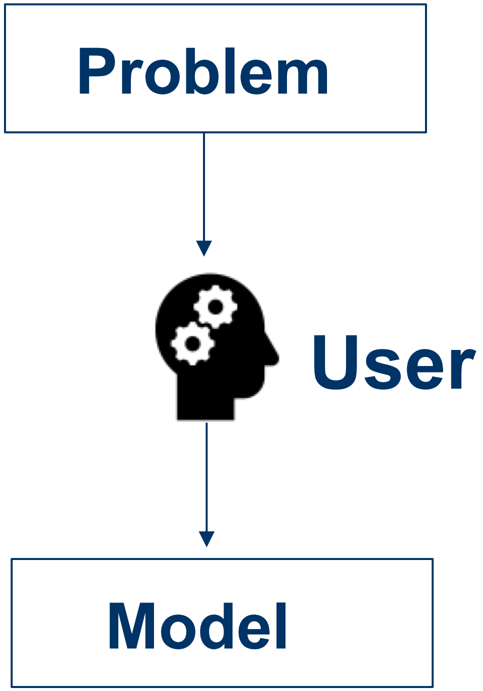
Example: London Bike Hiring

- AI is not just for machine learning, but also for decision support.
  - IBM® ILOG® CPLEX® Optimization Studio for London bike hiring scheme
  - ML to forecast and predict the movements of bikes, customer demand, customer behavior, maintenance time of bikes, …
  - Combinatorial optimization to decide how to move bikes to the stations in the best possible way and how many bikes to leave in each station.



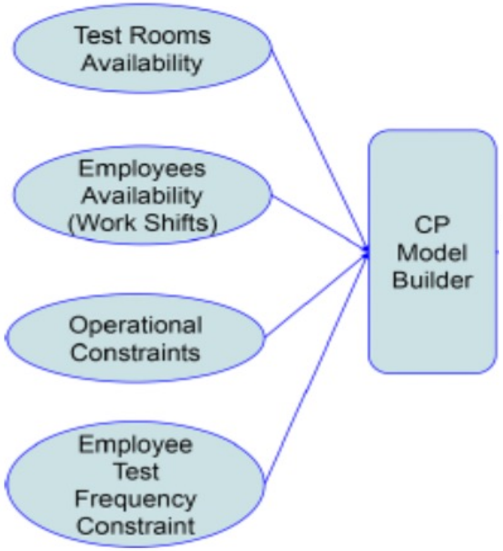
What is Constraint Programming?

- A declarative programing paradigm for stating and solving combinatorial optimization problems.
  - User models a decision problem by formalizing:
    - the unknowns of the decision → decision variables (X_i).
    - possible values for unknowns → domains ($D(X_i) = \{v_j\}$).
    - relations between the unknowns → constraints ($r(X_i, X_{i'})$).



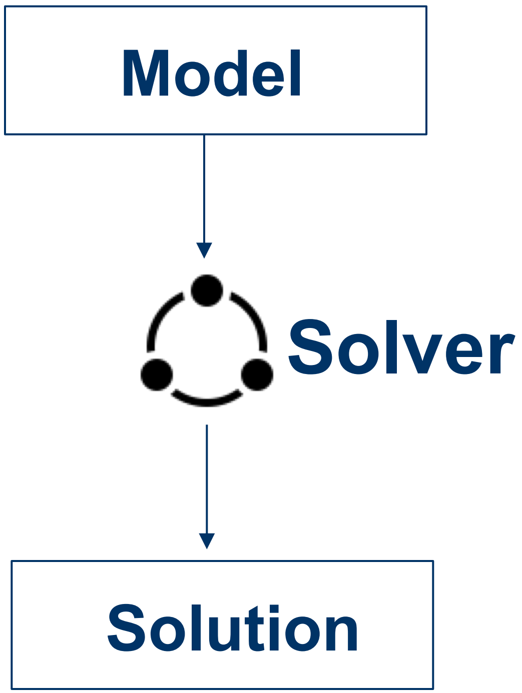
Covid-19 Test Scheduling

- When and where to test each employee?
- Availability Constraints
  - Testing room, tester, and employee availabilities.
- Frequency constraints
  - The spacing between tests performed on the same employee should be within given bounds.
- Operational constraints
  - Each employee should be tested within their working shift.
  - Only a limited share of employees from the same work area should be scheduled for a test on the same day.

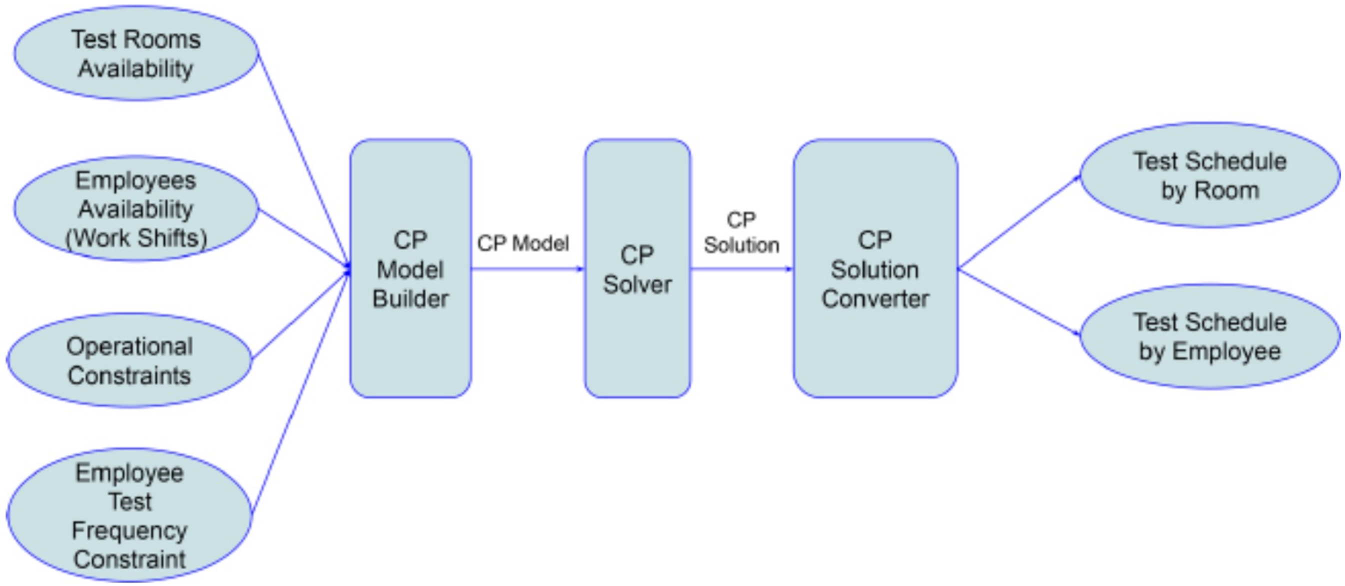


What is Constraint Programming?

- A declarative programing paradigm for stating and solving combinatorial optimization problems.
  - A constraint solver finds a solution to the model (or proves that no solution exists) by assigning a value to every variable ($X_i \leftarrow v_j$) via a search algorithm.



Covid-19 Test Scheduling



Why Constraint Programming?

- Sounds like Integer Linear Programming.
- CP provides a rich language for expressing constraints and defining search procedures.
  - Easy modelling.
    - Fast prototyping with a variety of constraints.
    - Easy to maintain programs.
    - Extensibility.
  - Easy control of search.
    - Experimentation with advanced search strategies.

Why Constraint Programming?

- Main focus on constraints and feasibility.
  - Constraints → reductions in the search space.
  - Of interest on tightly constrained problems.
  - More constraints mean more domain reductions, making the problem easier to solve.

Orthogonal and Complementary Approaches to Combinatorial Optimization

- **ILP**
  - Modeling with linear inequalities.
  - Numerical calculations.
  - Focus on objective function and optimality.
    - Bounding → elimination of suboptimal assignments.
  - Exploits global structure.
    - Relaxations, cutting planes, and duality theory.

- **CP**
  - Rich language for modeling and search procedures.
  - Logical processing.
  - Focus on constraints and feasibility.
    - Propagation → elimination of infeasible assignments.
  - Exploits local structure.
    - Domain reductions based on individual constraints.

Strengths of CP

- Success on irregular problems!
  - Timetabling, sequencing, scheduling allocation, rostering, etc.
  - Contain messy constraints non-linear in nature.
  - Contain multiple disjunctions which result in poor information returned by a linear relaxation of the problem.

Weaknesses and Opportunities of CP

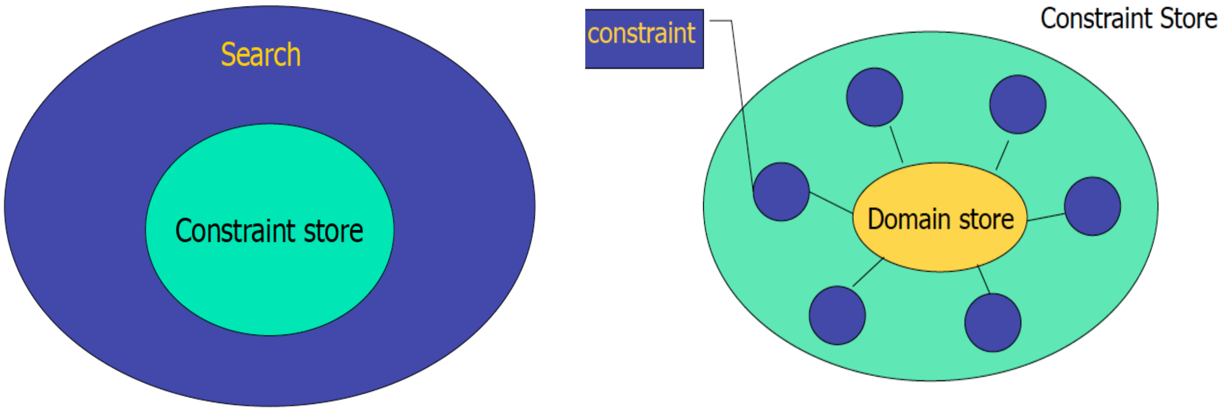
- Optimality
  - CP: no special focus on objective function and optimality ☹
  - ILP: scales up on loosely constrained optimization problems.
  - HS: is effective in finding quickly good-quality solutions.
- Best optimality approaches are often hybrids of CP, ILP and HS.
  - CP is a suitable framework for hybridization ☺

Overview of CP

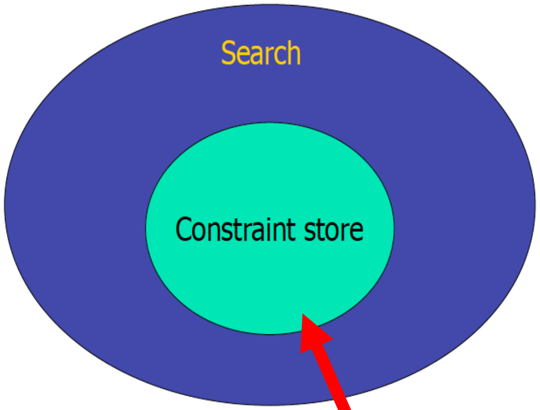
Constraint Solver

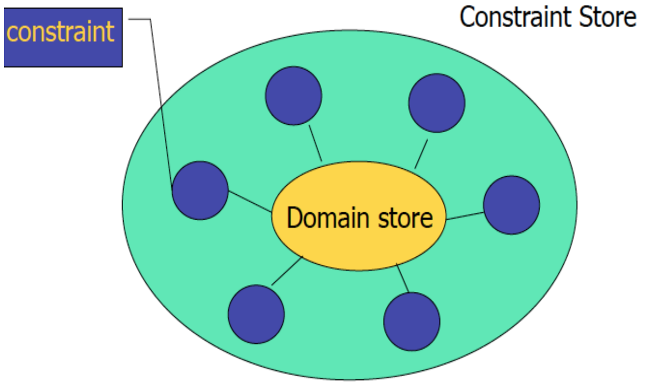
- Enumerates all possible variable-value combinations via a systematic backtracking tree search.
  - Guesses a value for each variable.
- During search, examines the constraints to remove incompatible values from the domains of the future (unexplored) variables, via propagation.
  - Shrinks the domains of the future variables.

Constraint Programming



Constraint Programming





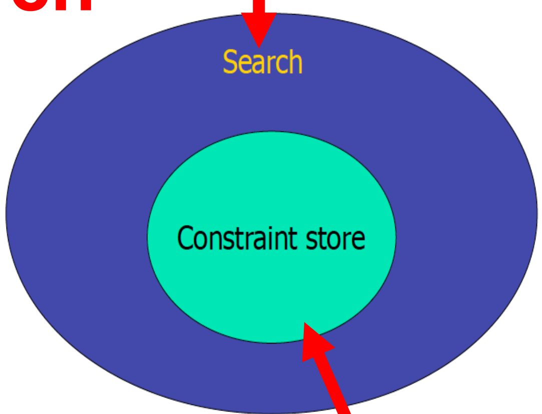
Modelling

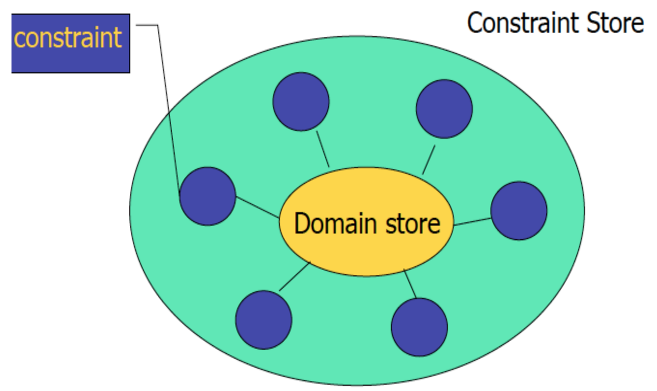
User expresses the problem

Constraint Programming

Solver uses a backtracking tree search algorithm to guess a value for each variable

Search





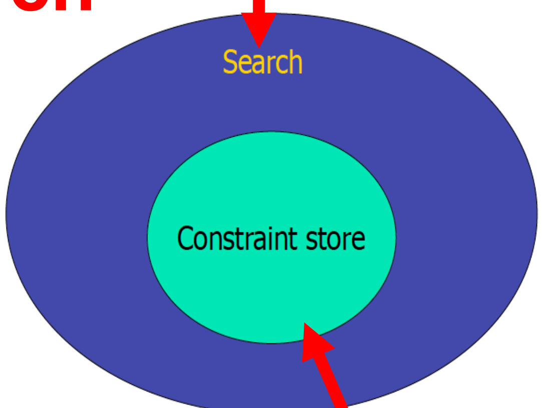
Modelling

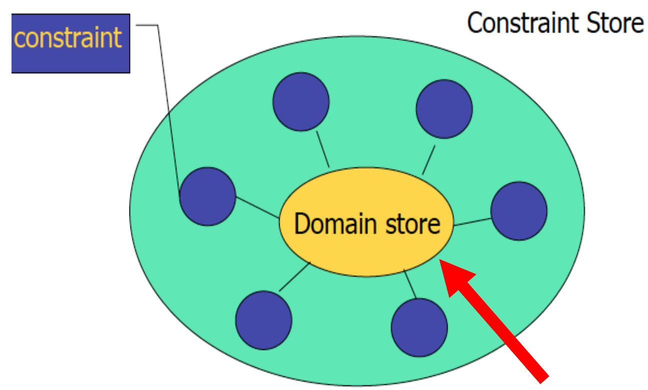
User expresses the problem

Constraint Programming

Solver uses a backtracking tree search algorithm to guess a value for each variable

Search





Propagation

Solver uses algorithms to examine each constraint to reduce the domains of the future variables

Modelling

User expresses the problem

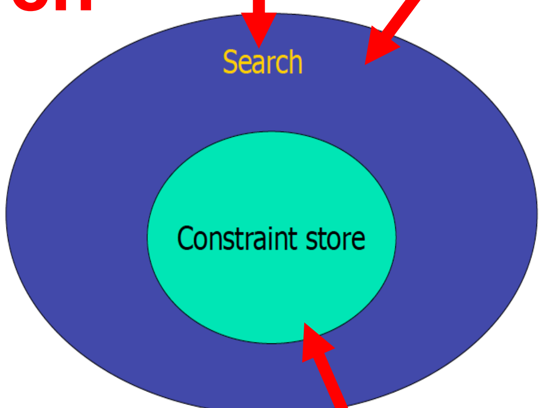
Constraint Programming

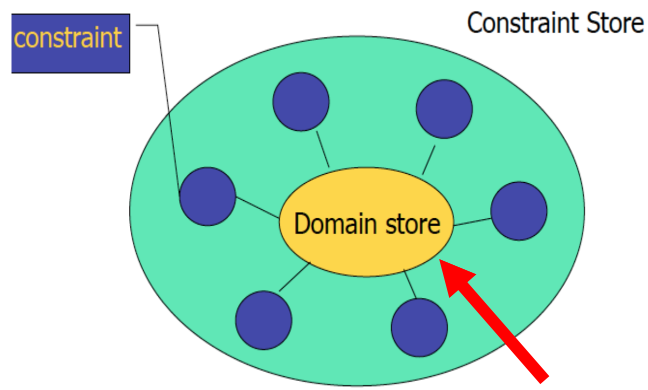
Solver uses a backtracking tree search algorithm to guess a value for each variable

Search

Solver exploits the current search state and problem specific knowledge to guide the search

Search heuristics





Modelling

User expresses the problem

Propagation

Solver uses algorithms to examine each constraint to reduce the domains of the future variables

Dual Role of a Model

- Captures combinatorial substructures.
- Enables solver to reduce the search space.
  - Constraints act as propagation algorithms.
  - Variables' domains act as communication mechanism.

Search and Propagation

- Search decisions and propagation are interleaved.

Propagation

↓

$X_i \leftarrow v_j$

↓

Propagation

↓

$X_{i'} \leftarrow v_{j'}$

↓

Propagation

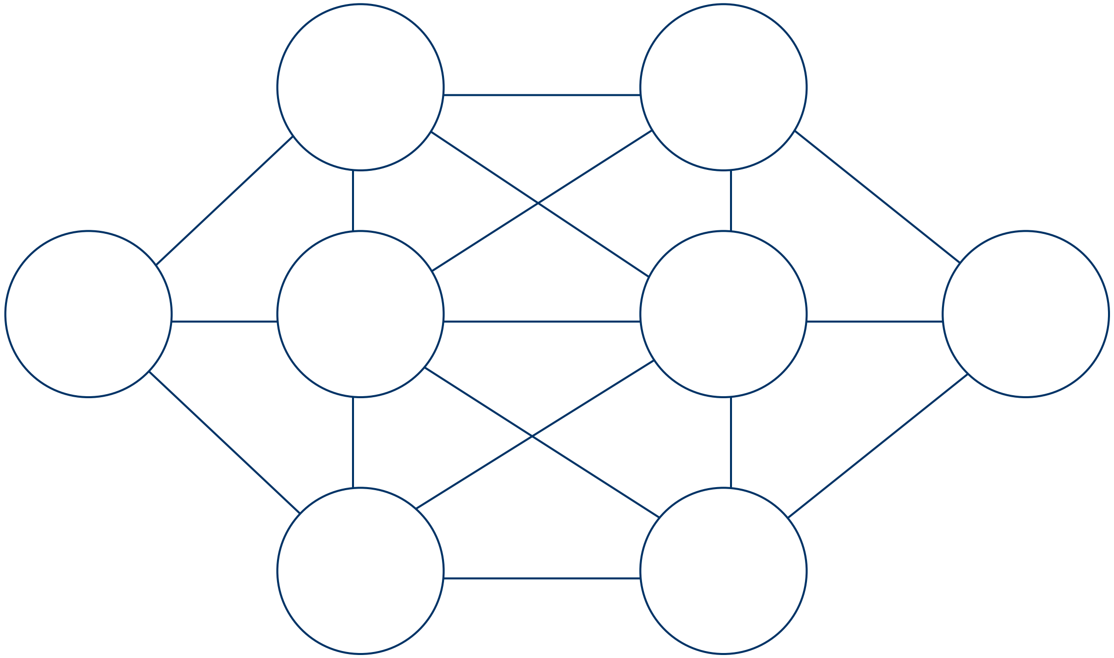
Expectation from CP

- Declarative programming
  - The user declaratively models the problem.
  - An underlying solver returns a solution with its default search.

Reality in CP

- Modelling is critical!
  - The user often has to use advanced modelling techniques for strong propagation.
- Default search of the solver is usually not enough!
  - The user often has to program the search strategy (search algorithm, search heuristics,…)

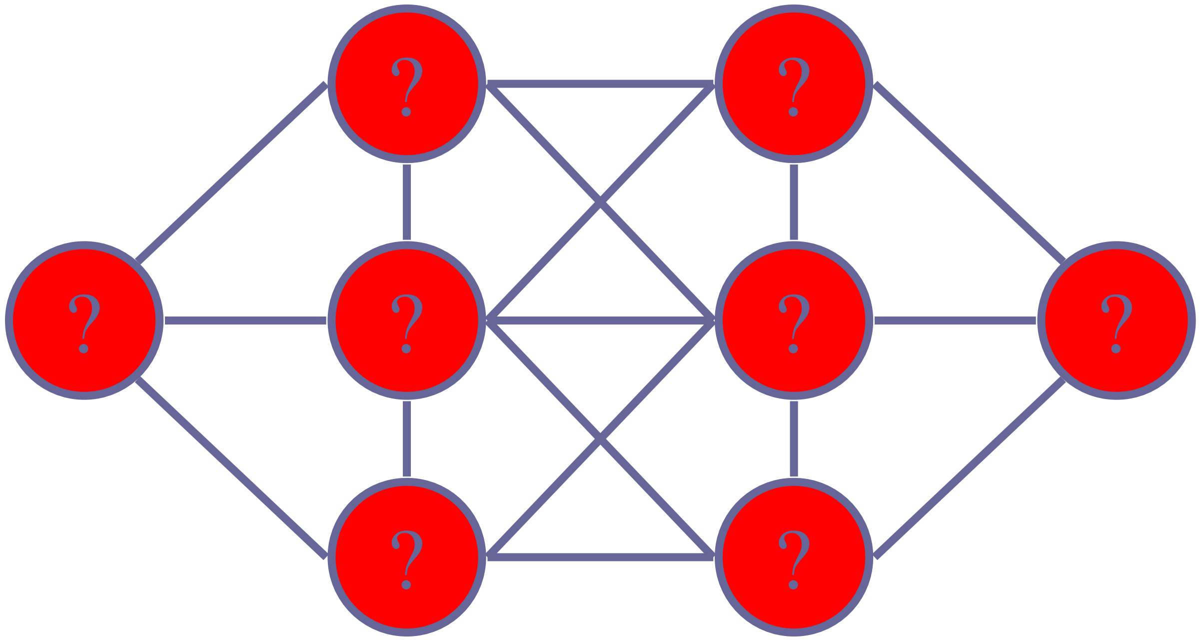
A Puzzle



Place a different number in each node (1 to 8) such that adjacent nodes cannot take consecutive numbers

A Puzzle

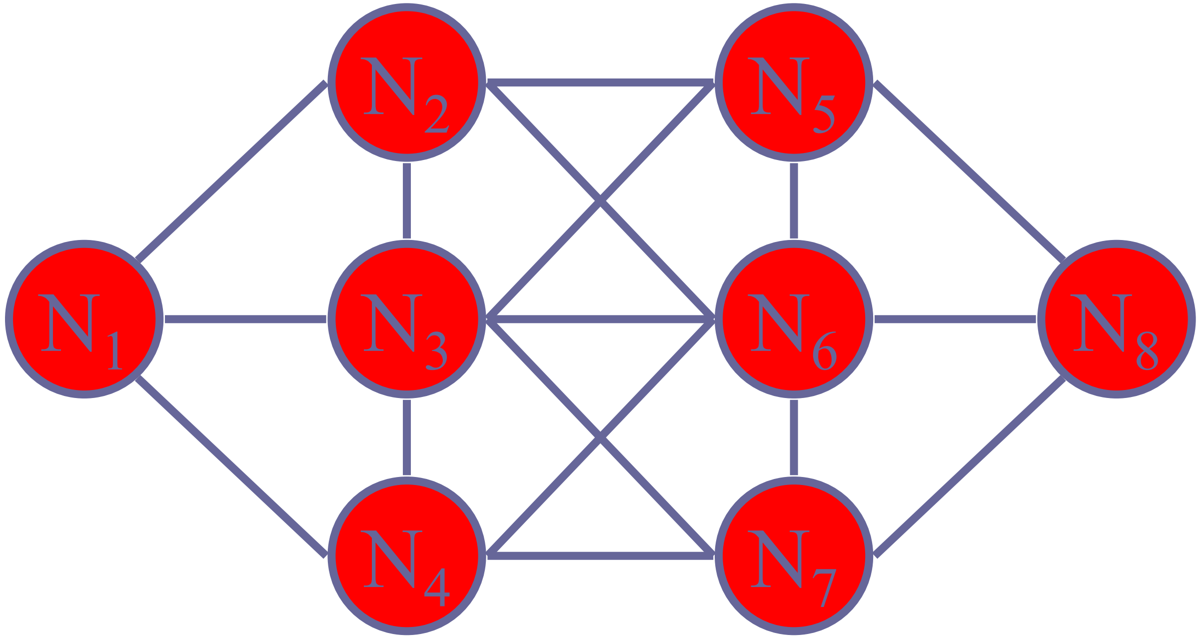
- Place numbers 1 through 8 on nodes, s.t.:
  - each number appears exactly once;
  - no connected nodes have consecutive numbers.



Modelling

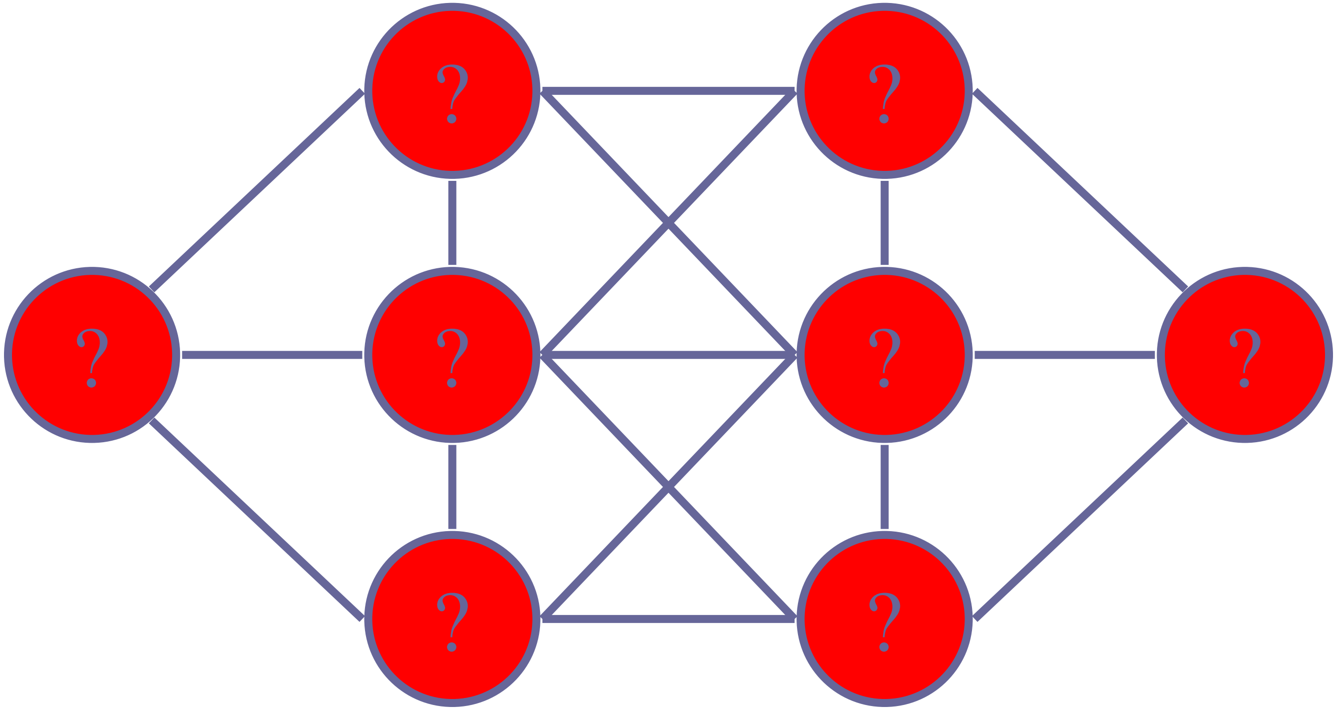
- Variables: $N_1 \ldots N_8$ that represent the nodes
- Domains: the set of values {1,2,3,4,5,6,7,8} that $N_1..N_8$ can take
- Constraints: for all $i < j$ s.t. N_i and N_j are adjacent $|N_i - N_j| > 1$

  for all $i < j$ $N_i \neq N_j$



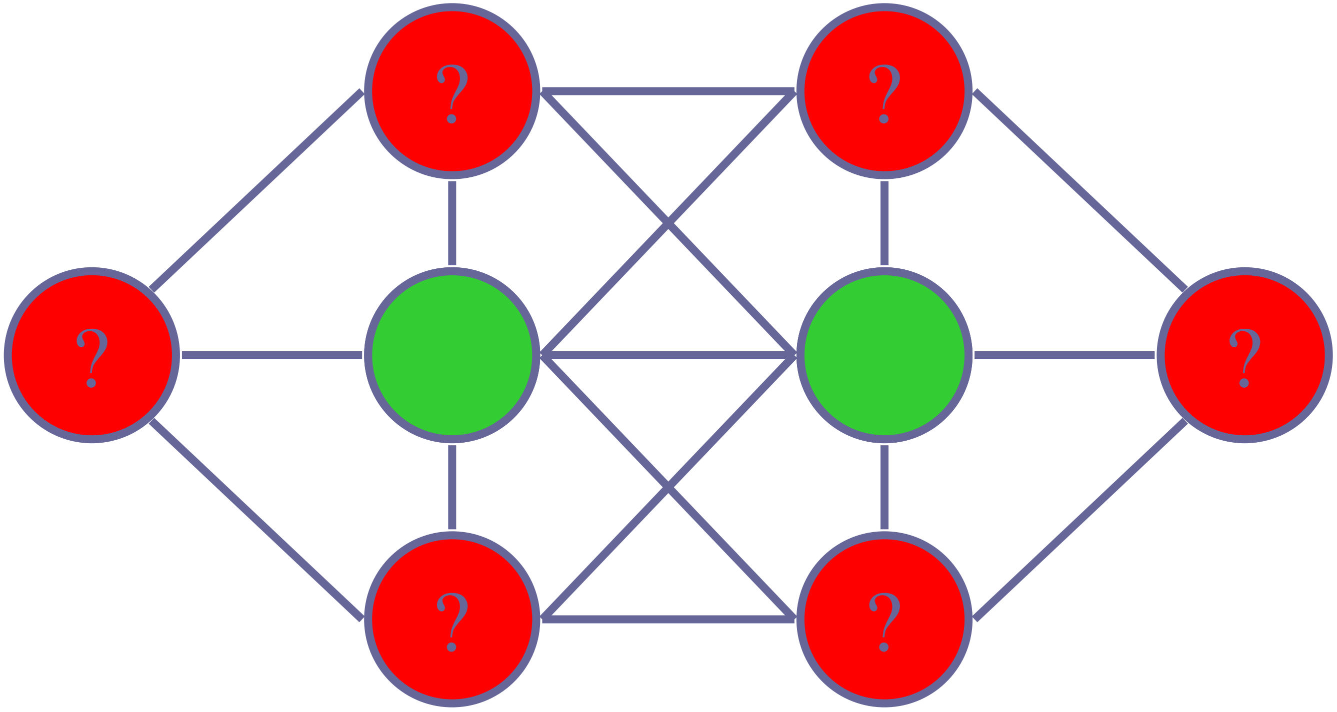
Backtracking Search + Heuristics

- Guess a value for a variable!



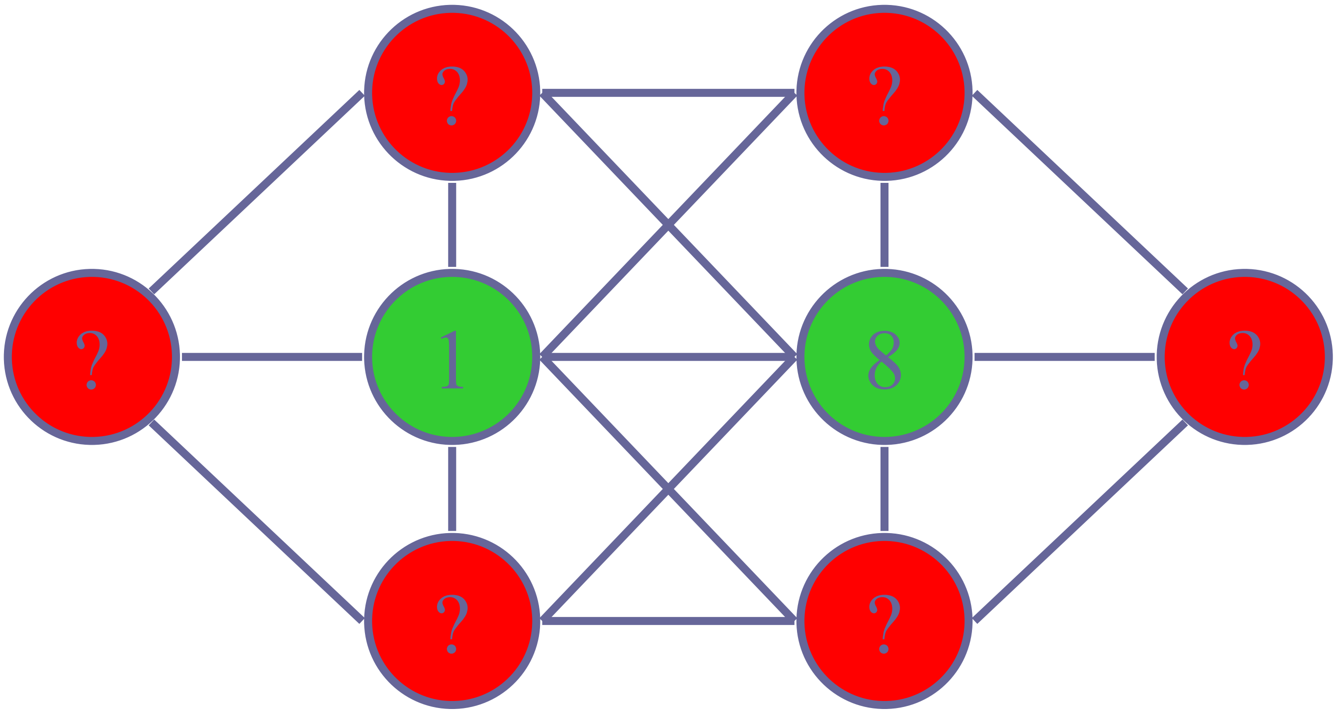
Backtracking Search + Heuristics

- Guess a value for a variable!
  - We start with the hardest variables.



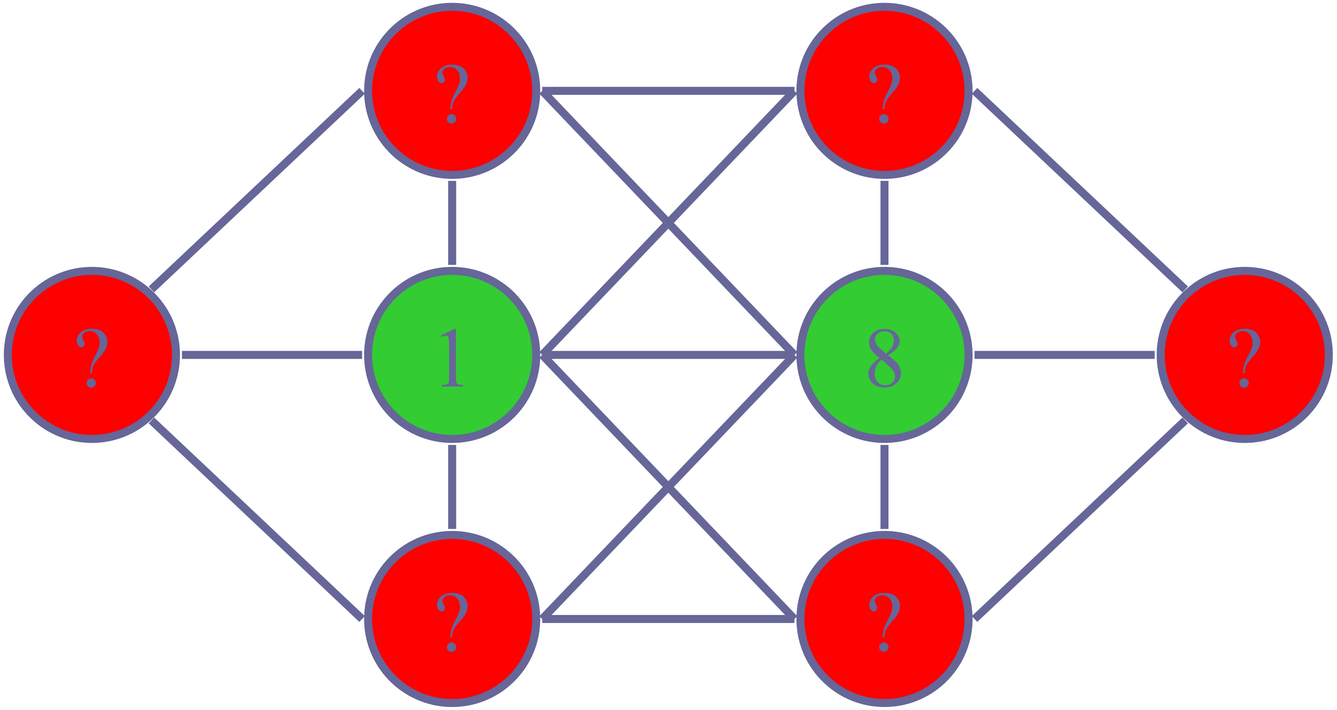
Backtracking Search + Heuristics

- Guess a value for a variable!
  - We assign them the safest values.

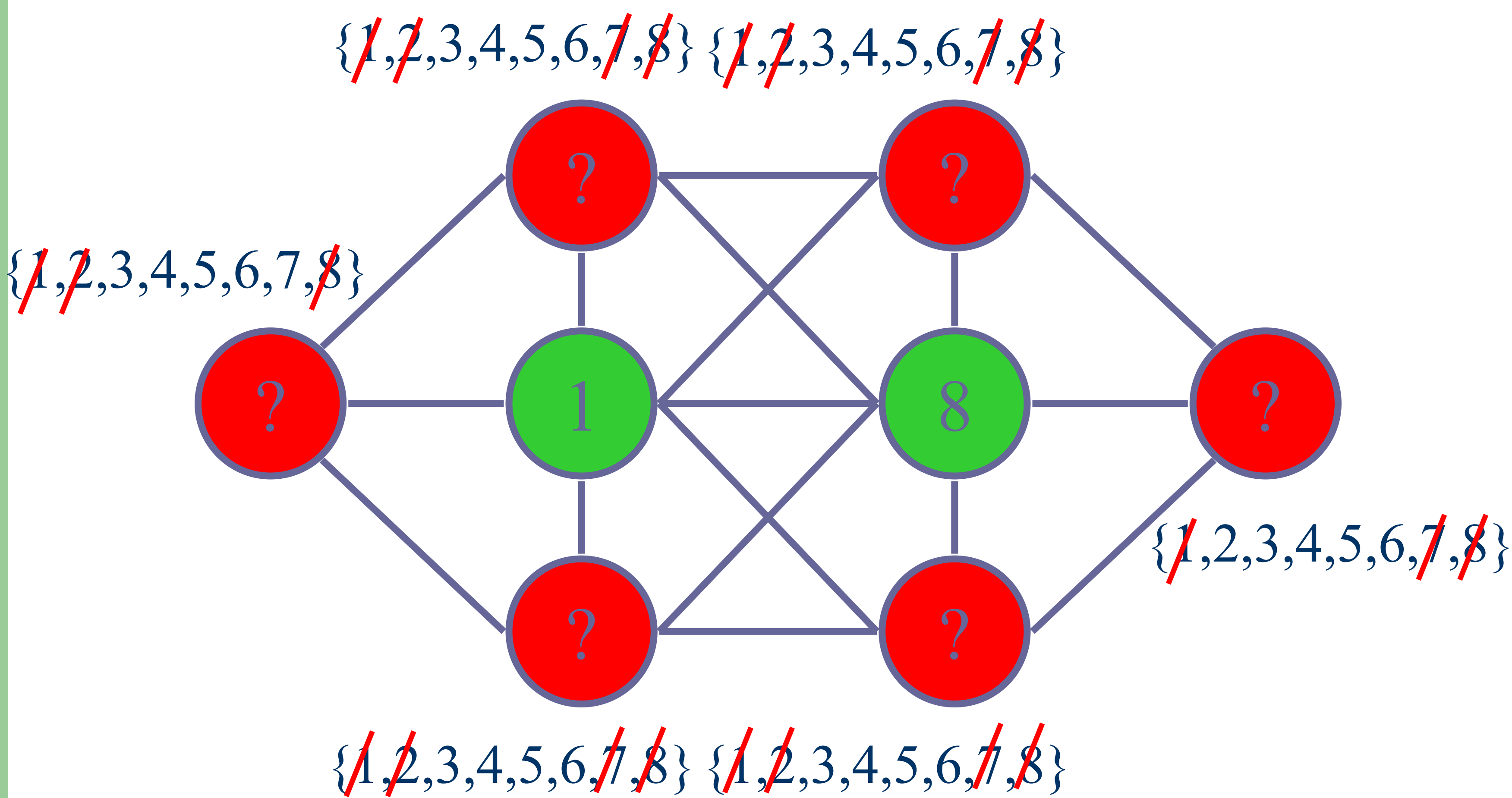


Propagation

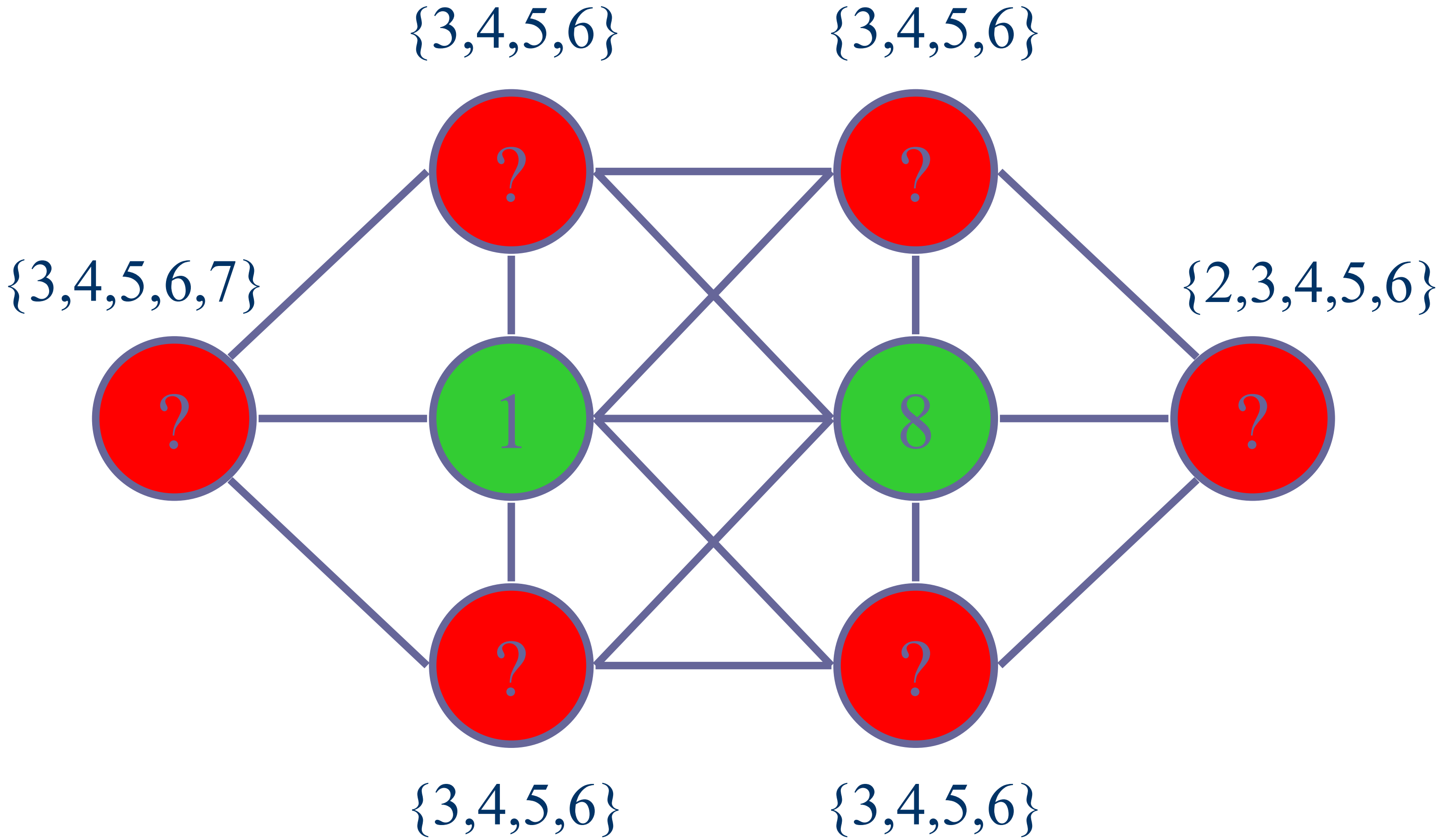
- We now examine the constraints.



Propagation

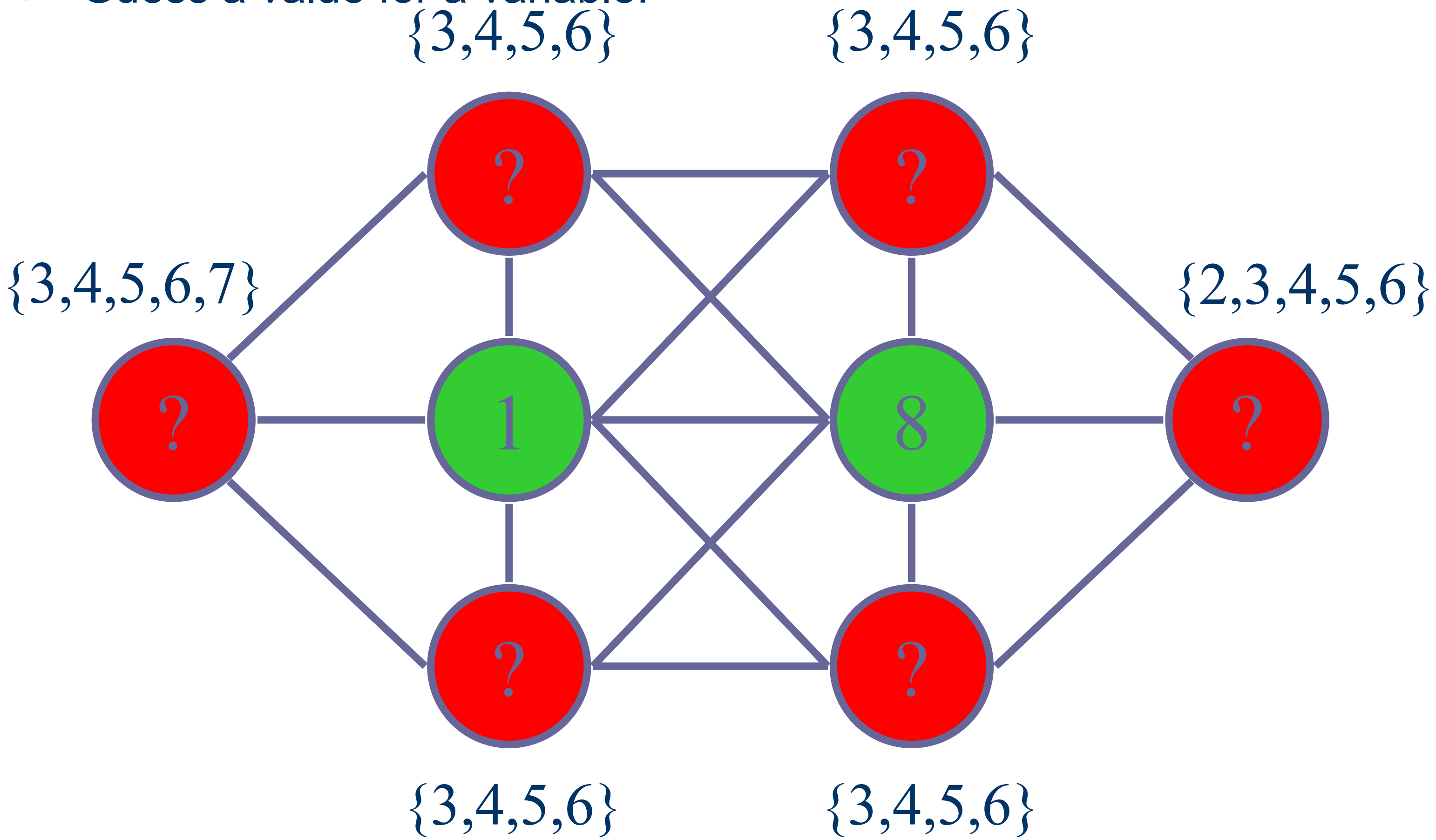


Propagation



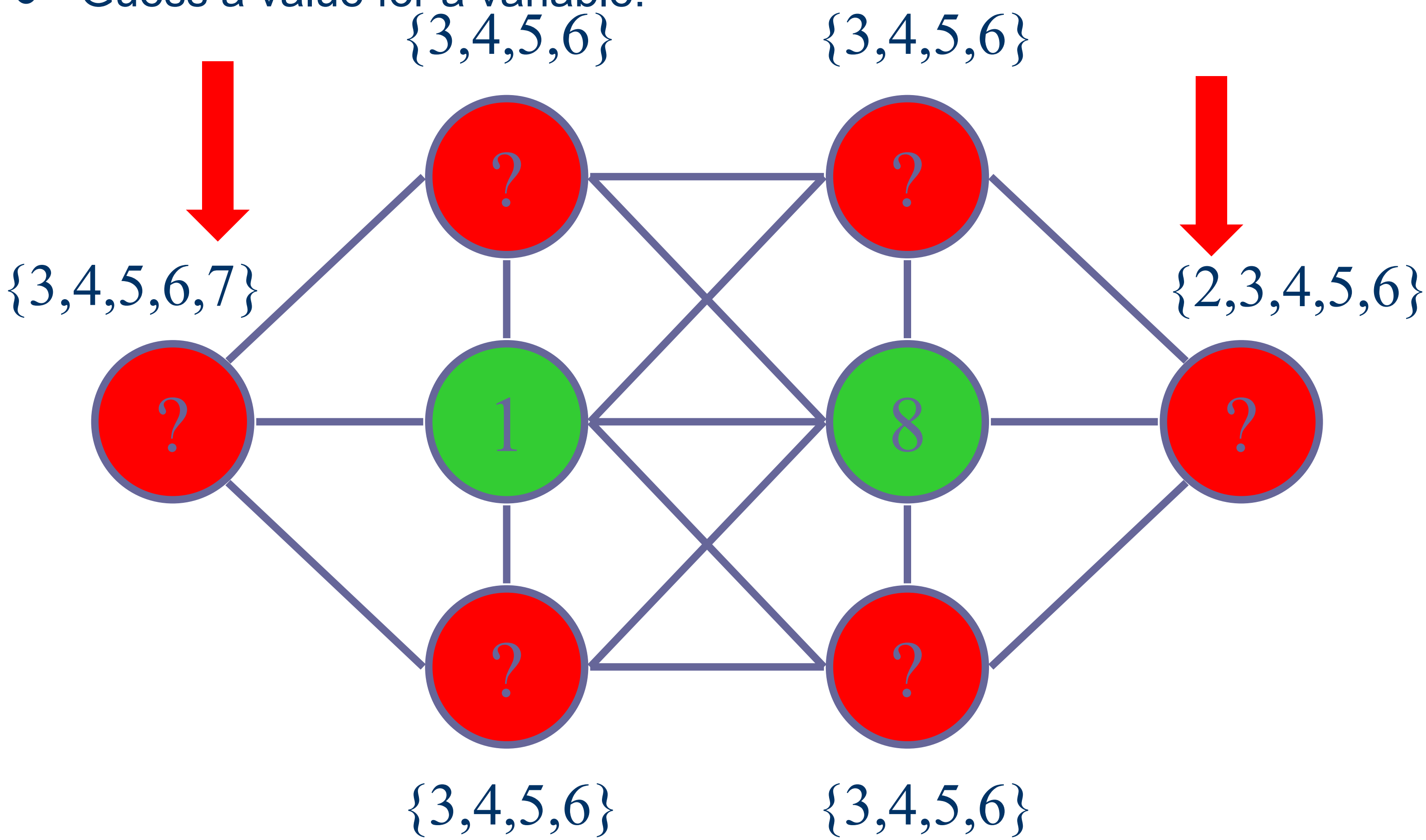
Backtracking Search + Heuristics

- Guess a value for a variable!

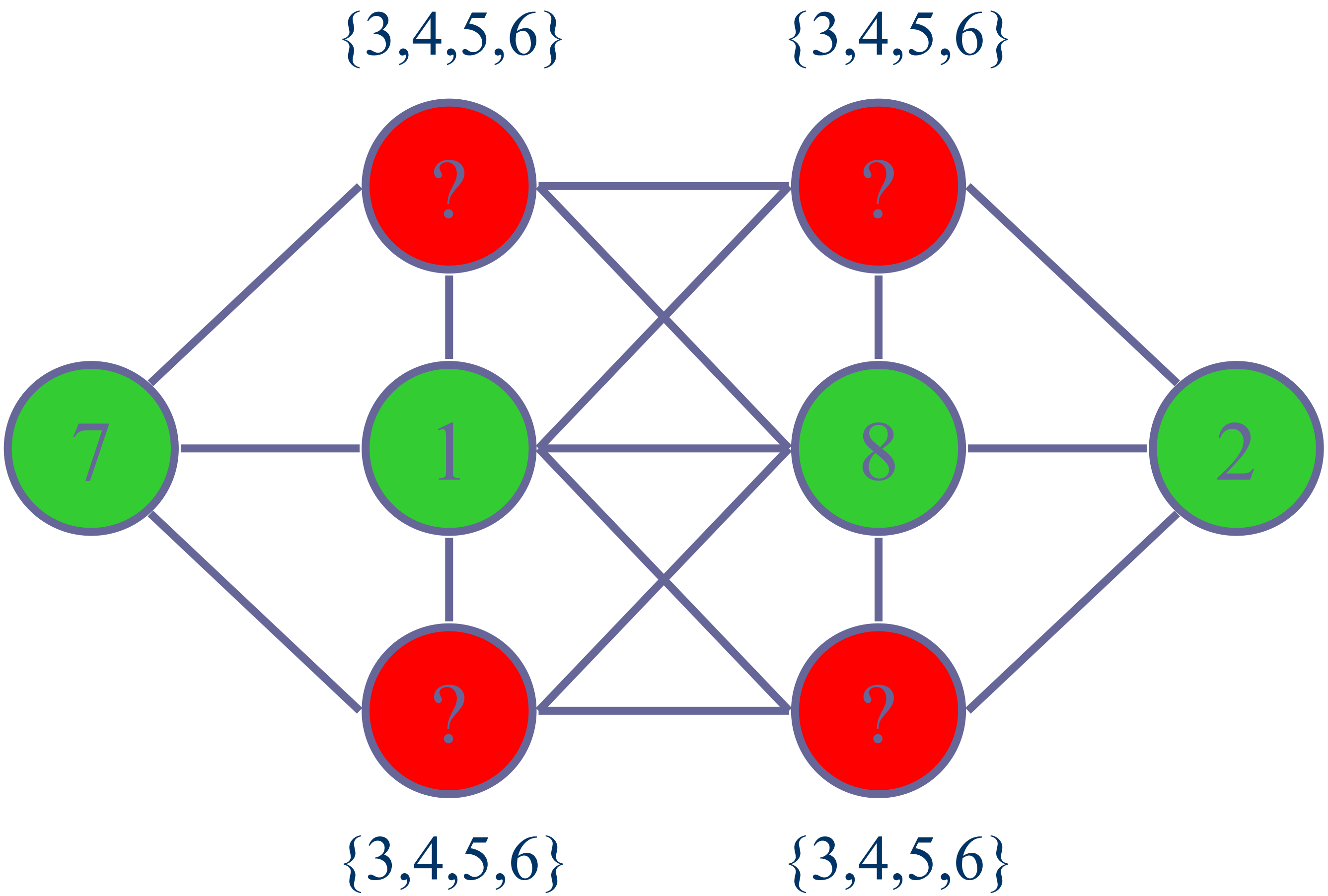


Backtracking Search + Heuristics

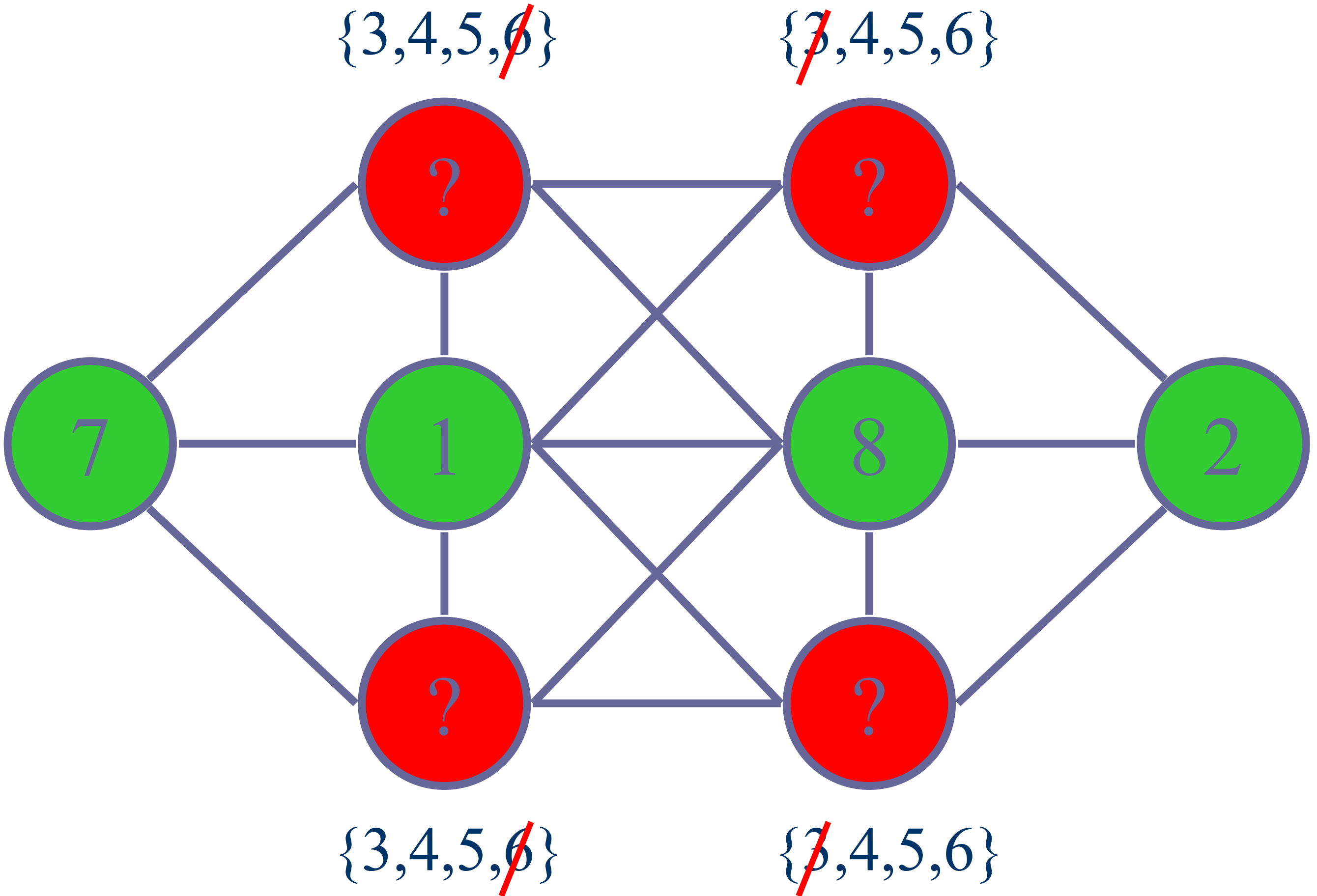
- Guess a value for a variable!



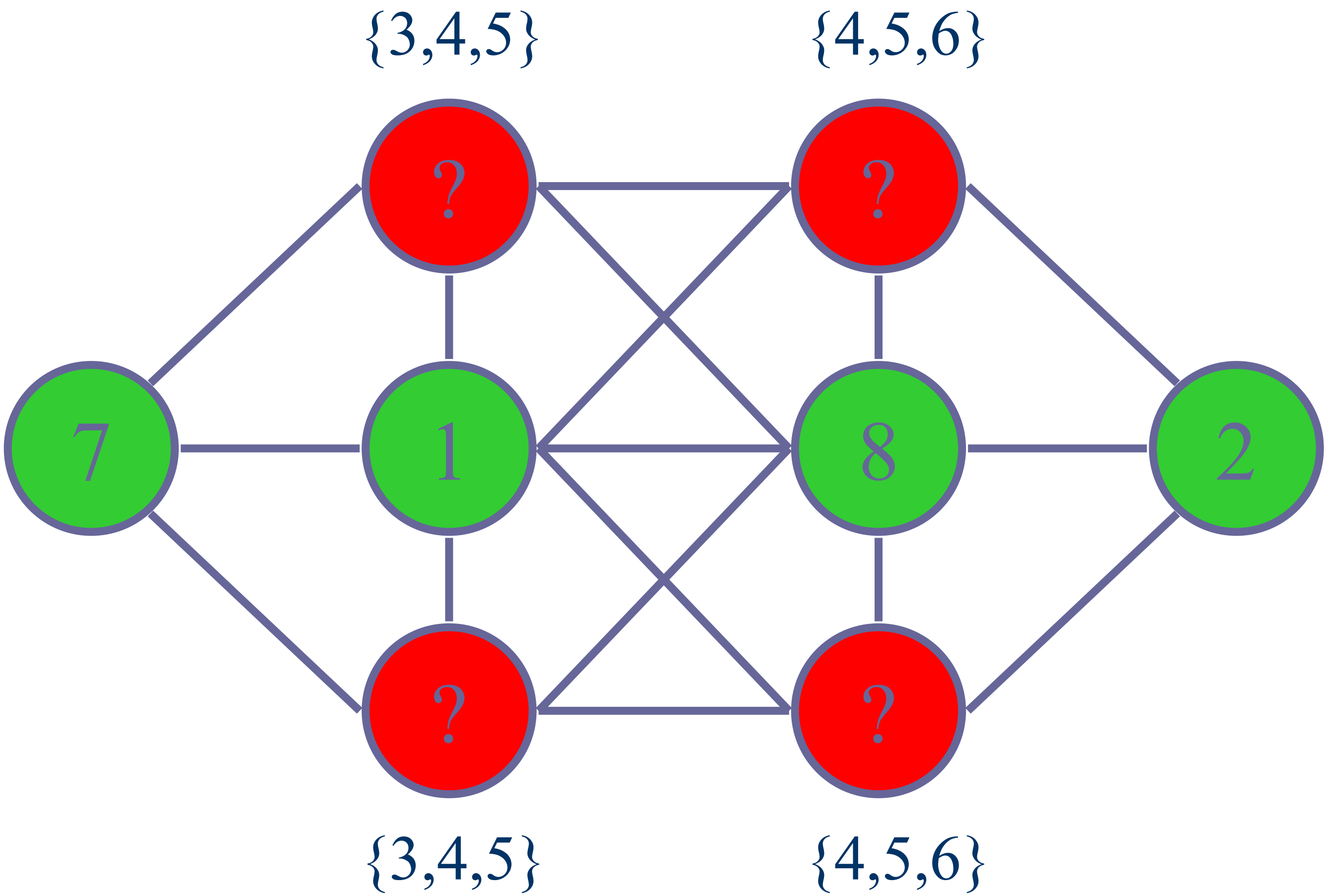
Backtracking Search + Heuristics



Propagation

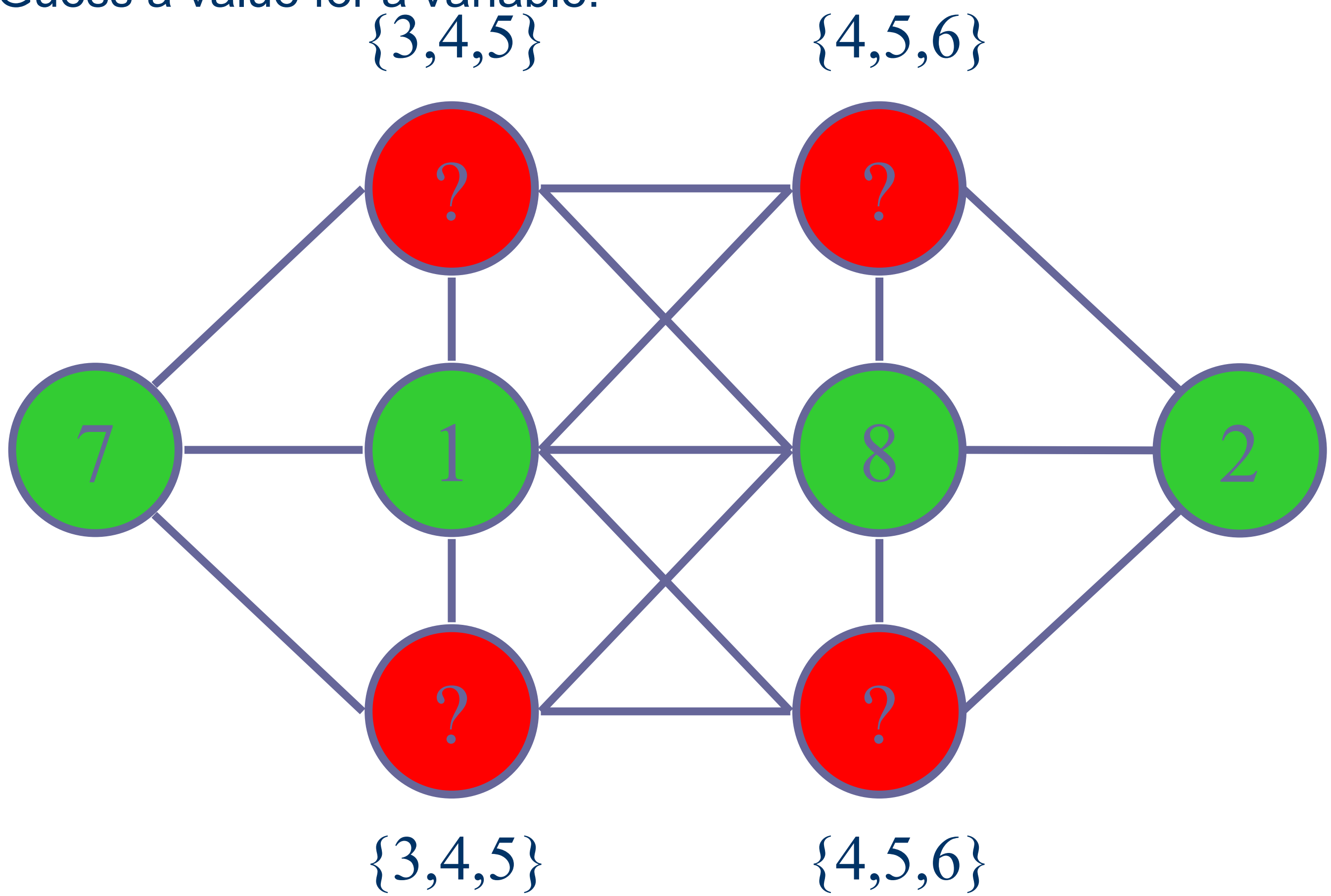


Propagation

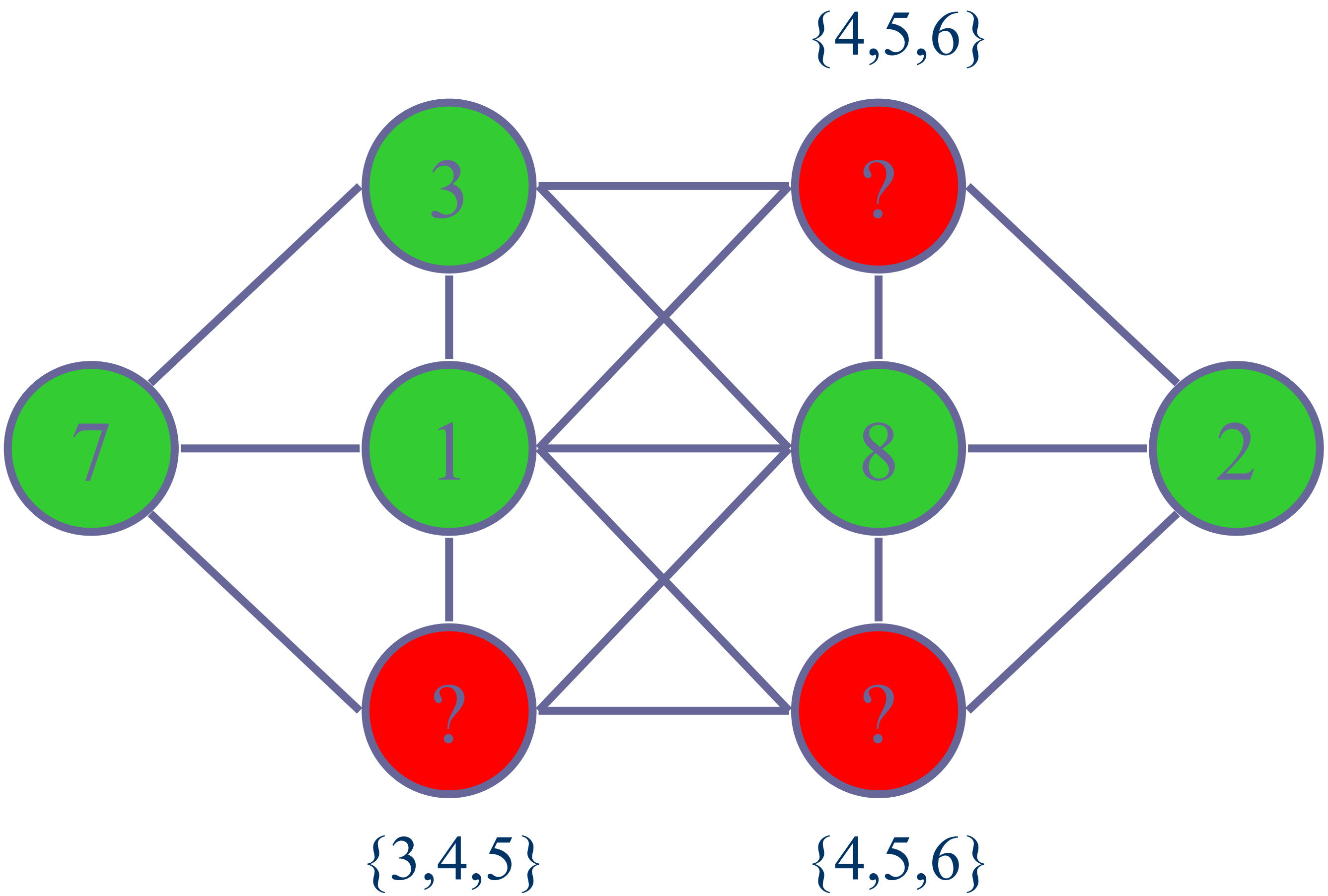


Backtracking Search + Heuristics

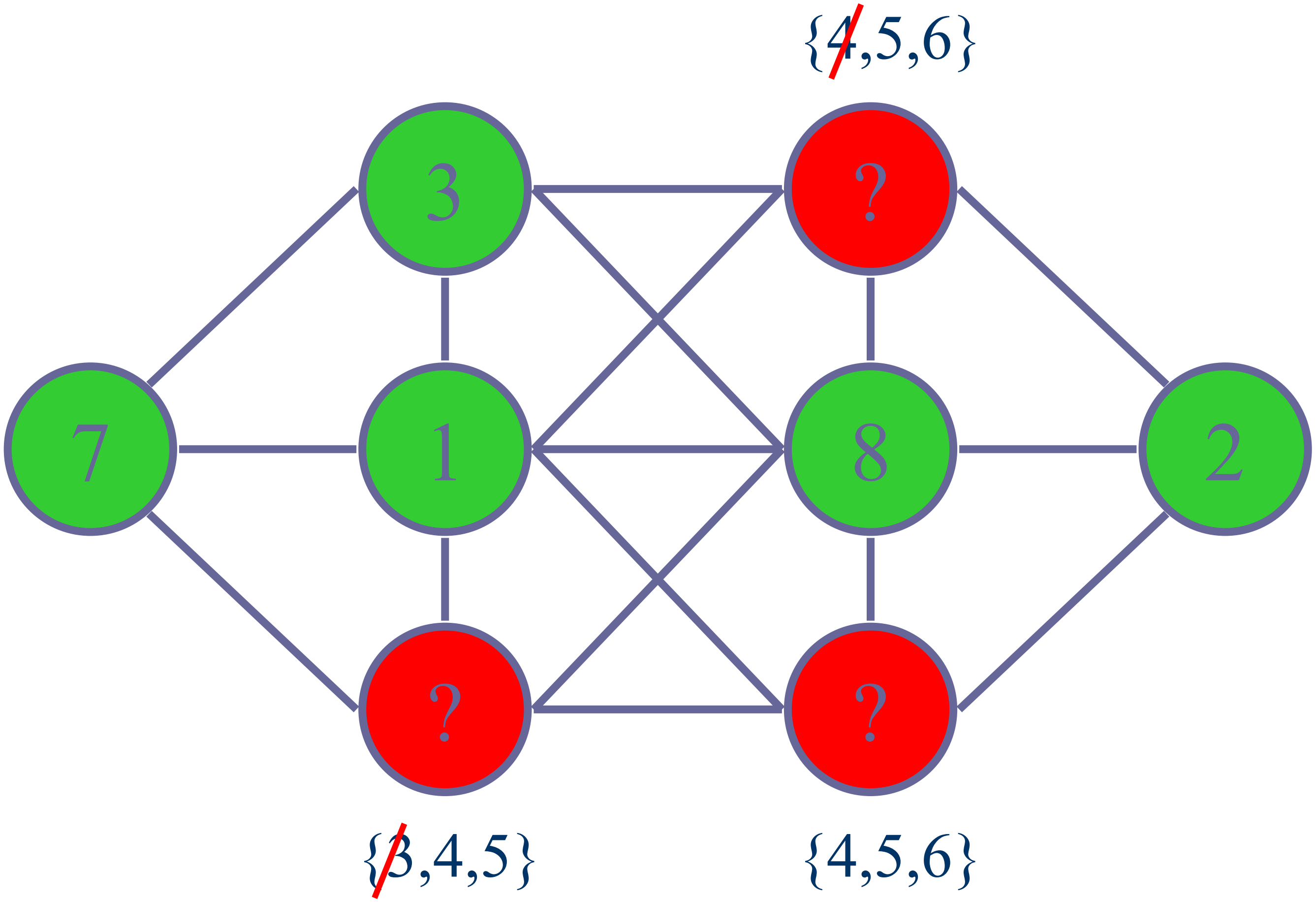
- Guess a value for a variable!



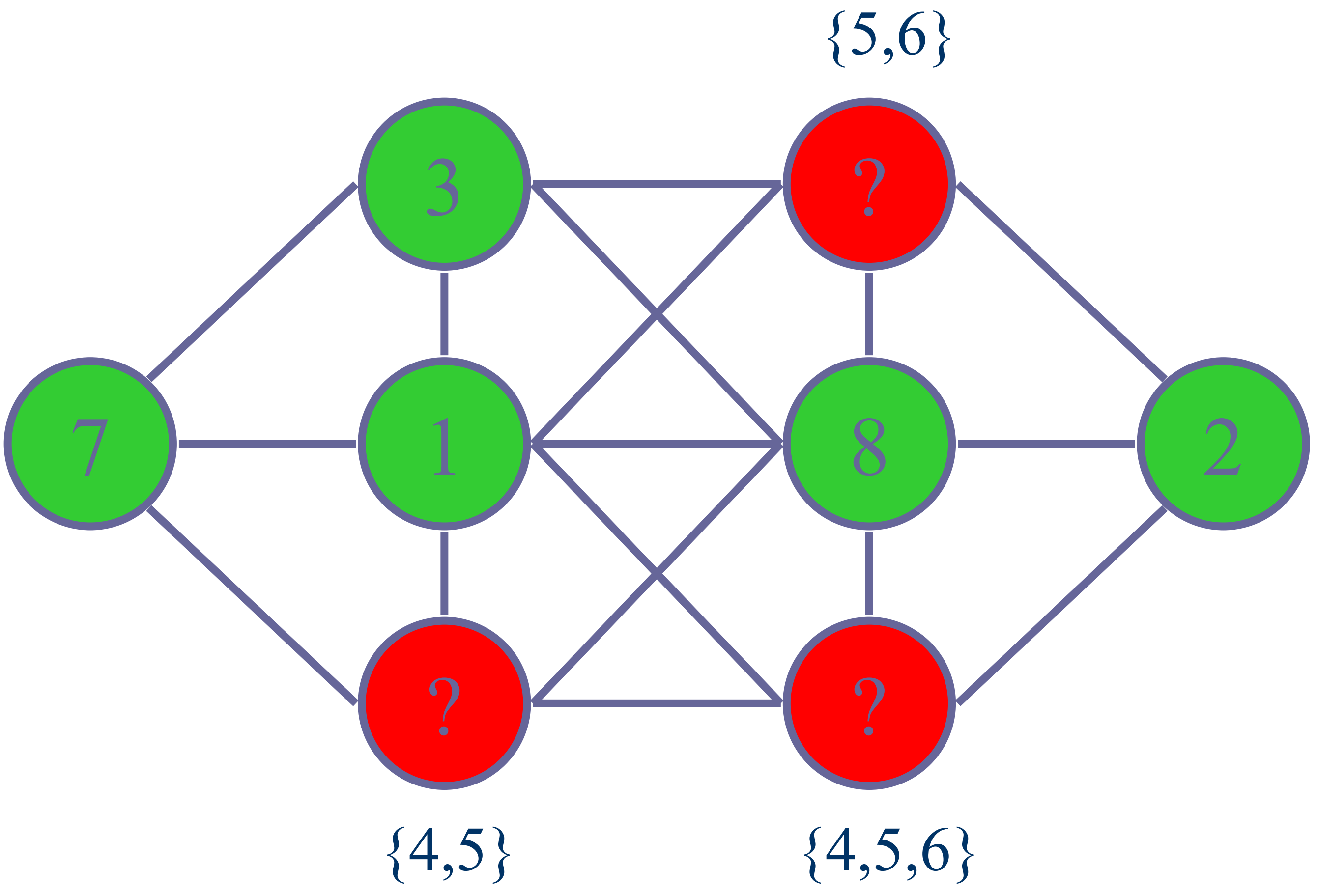
Backtracking Search + Heuristics



Propagation

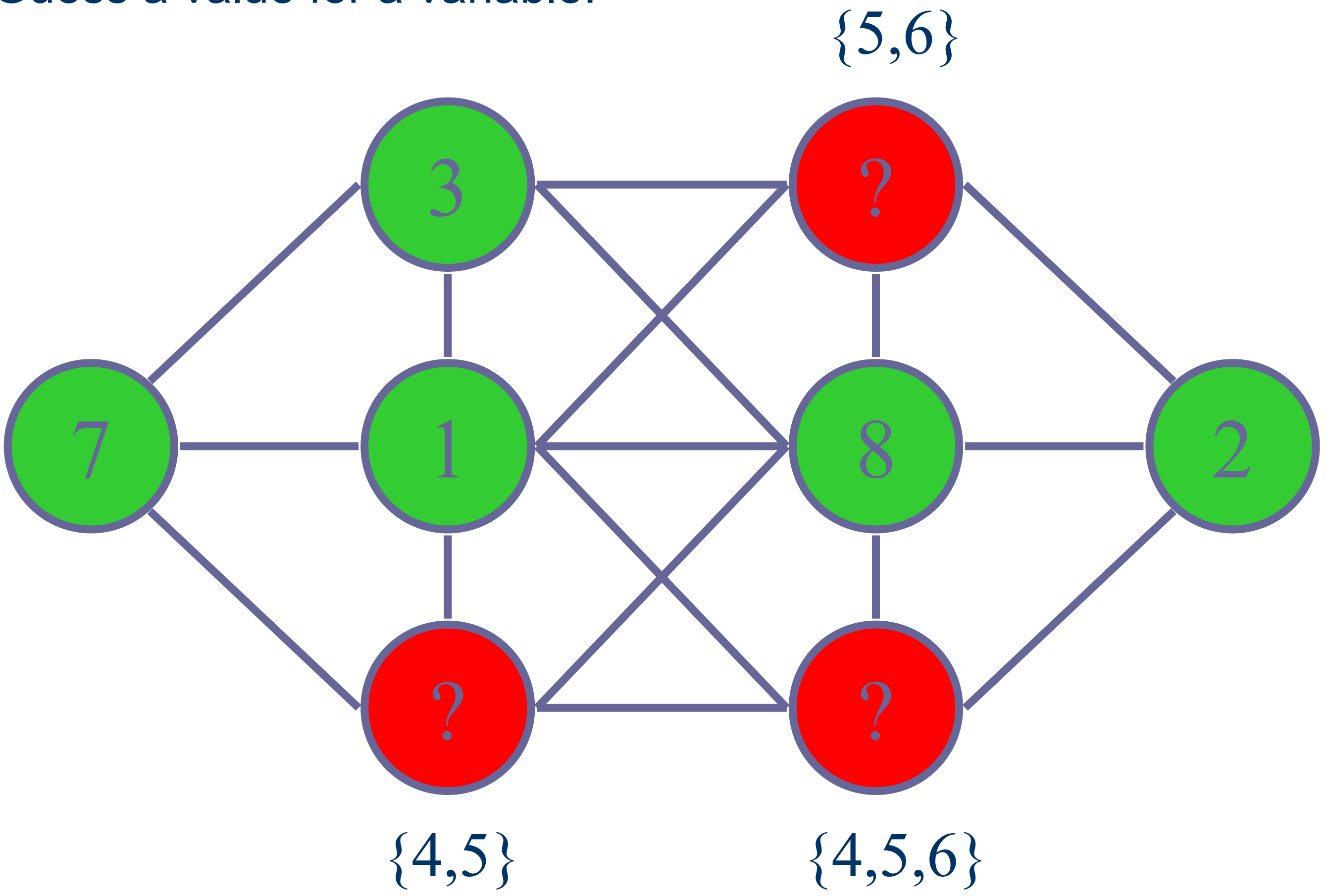


Propagation

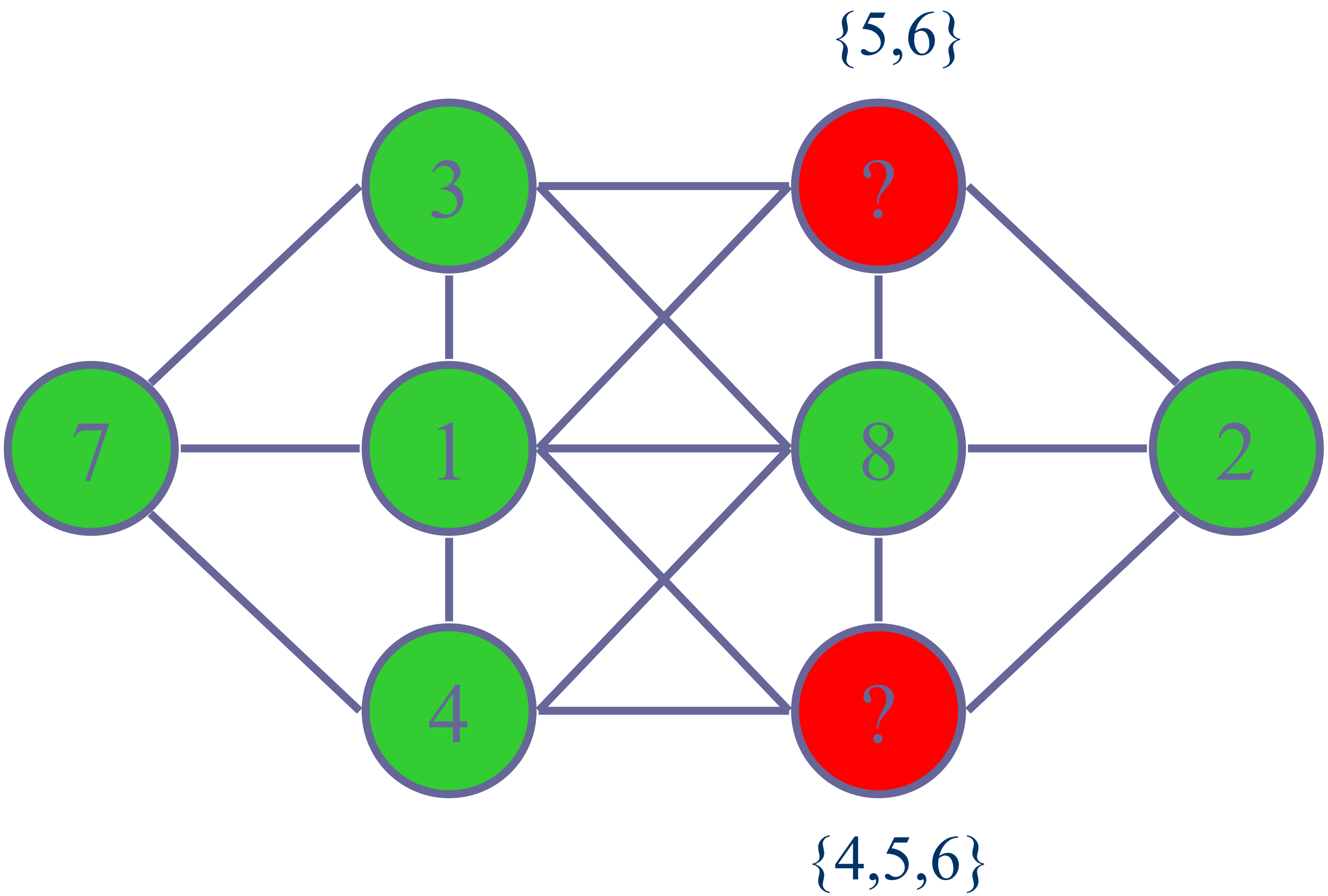


Backtracking Search + Heuristics

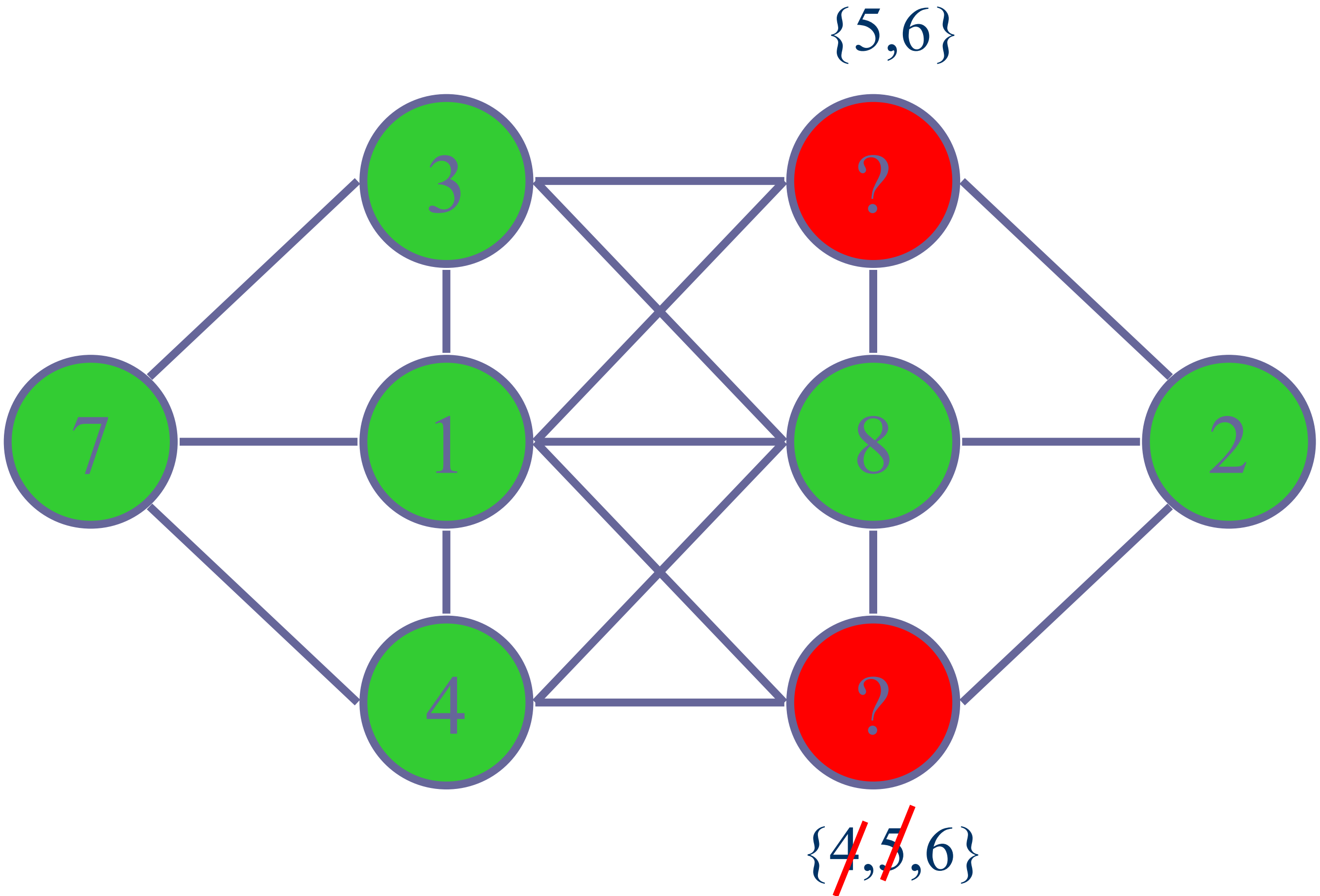
- Guess a value for a variable!



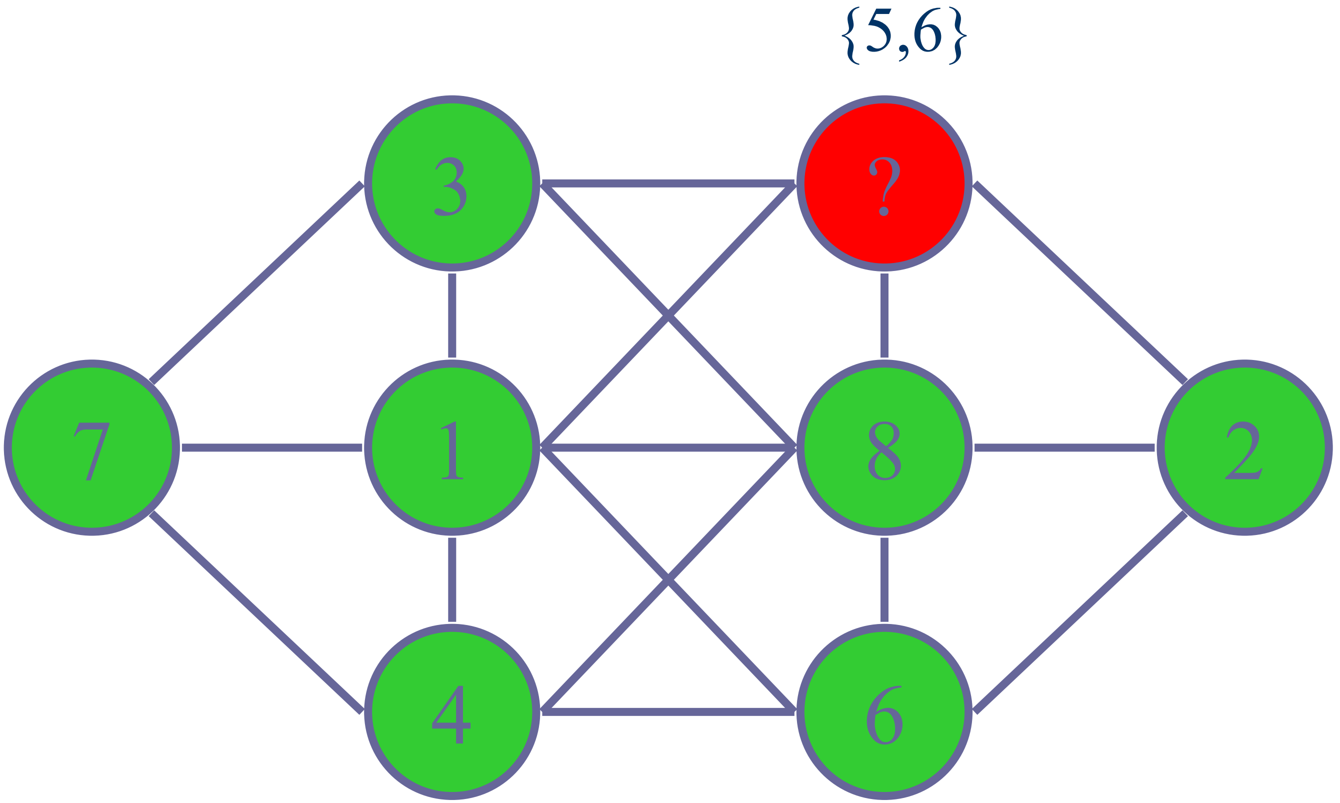
Backtracking Search + Heuristics



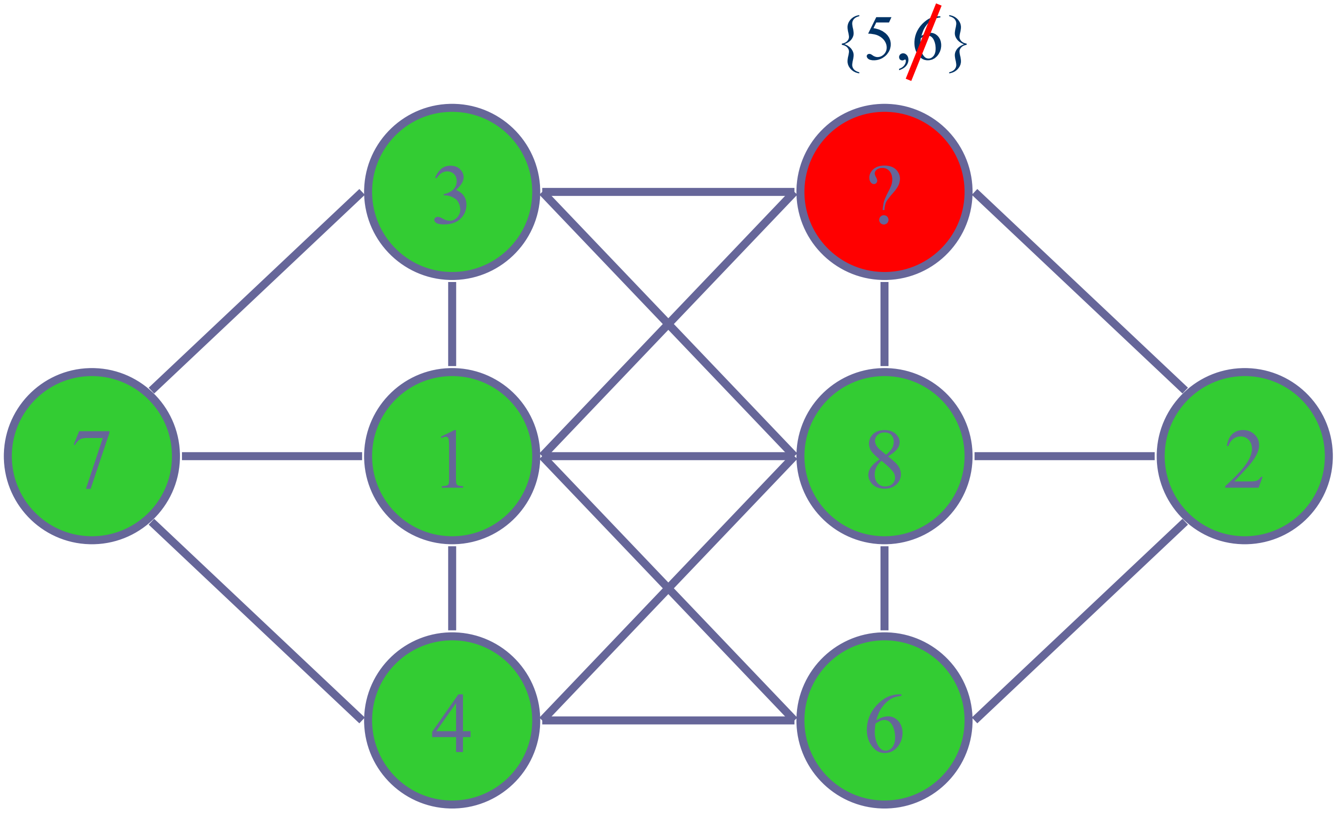
Propagation



Propagation

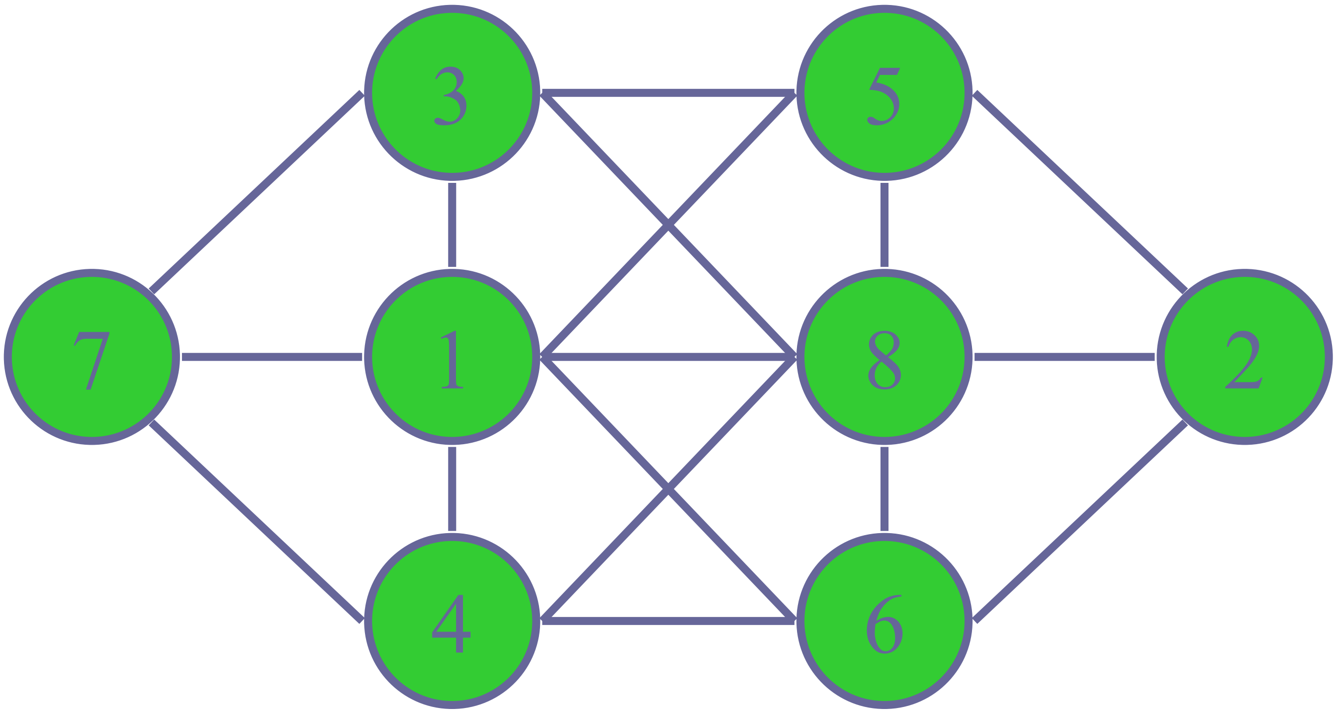


Propagation

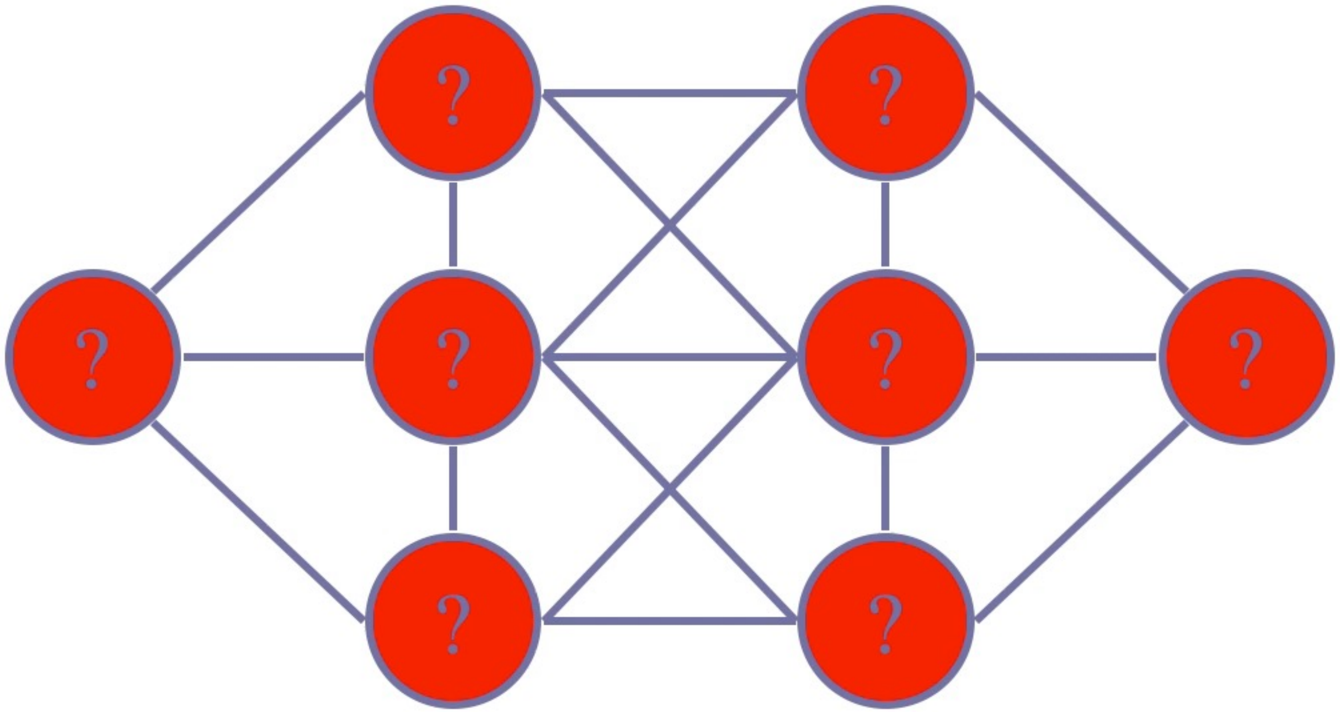


Solution

- 8 guesses, without any backtracking!

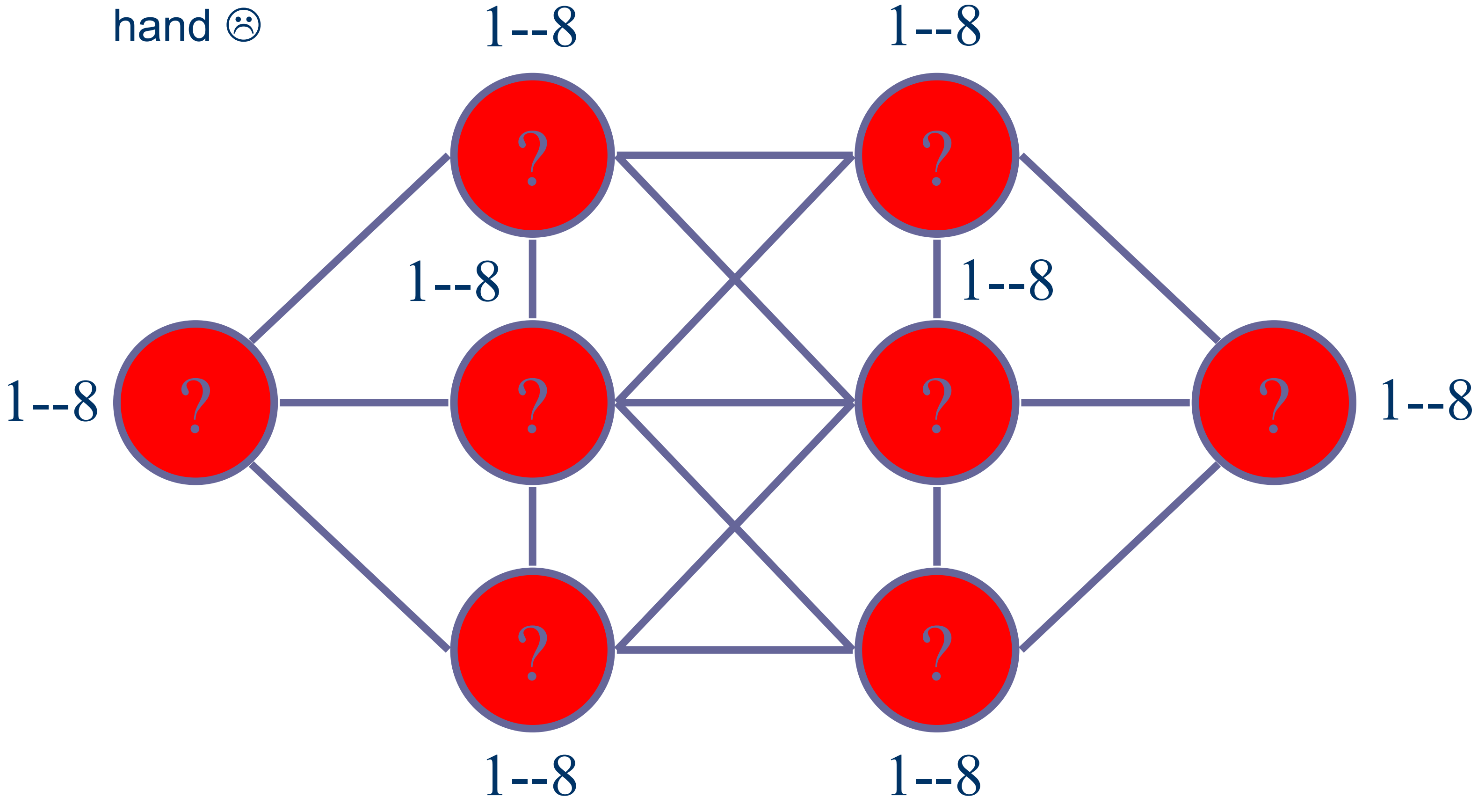


Backtracking Search without Heuristics



Backtracking

- Back to the beginning after 45 backtracks without any solution at hand ☹

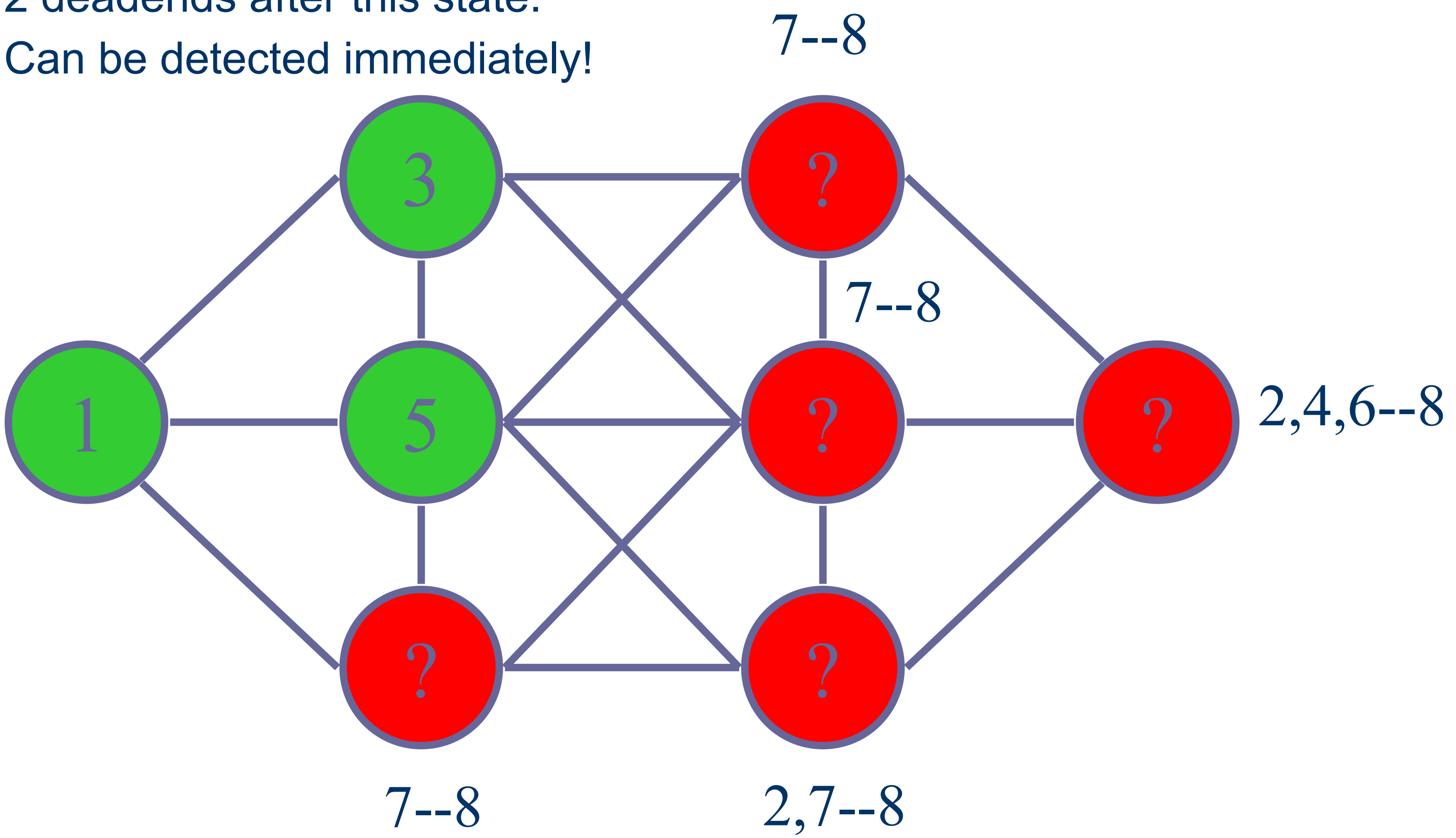


What’s going on?

- Bad choice of variables, bad assignment of values.
  - → Good heuristic choice is very important!
- Good heuristics are always possible?
  - Yes and no 🙃
- What can we do then?
  - Apply stronger form of propagation during search!

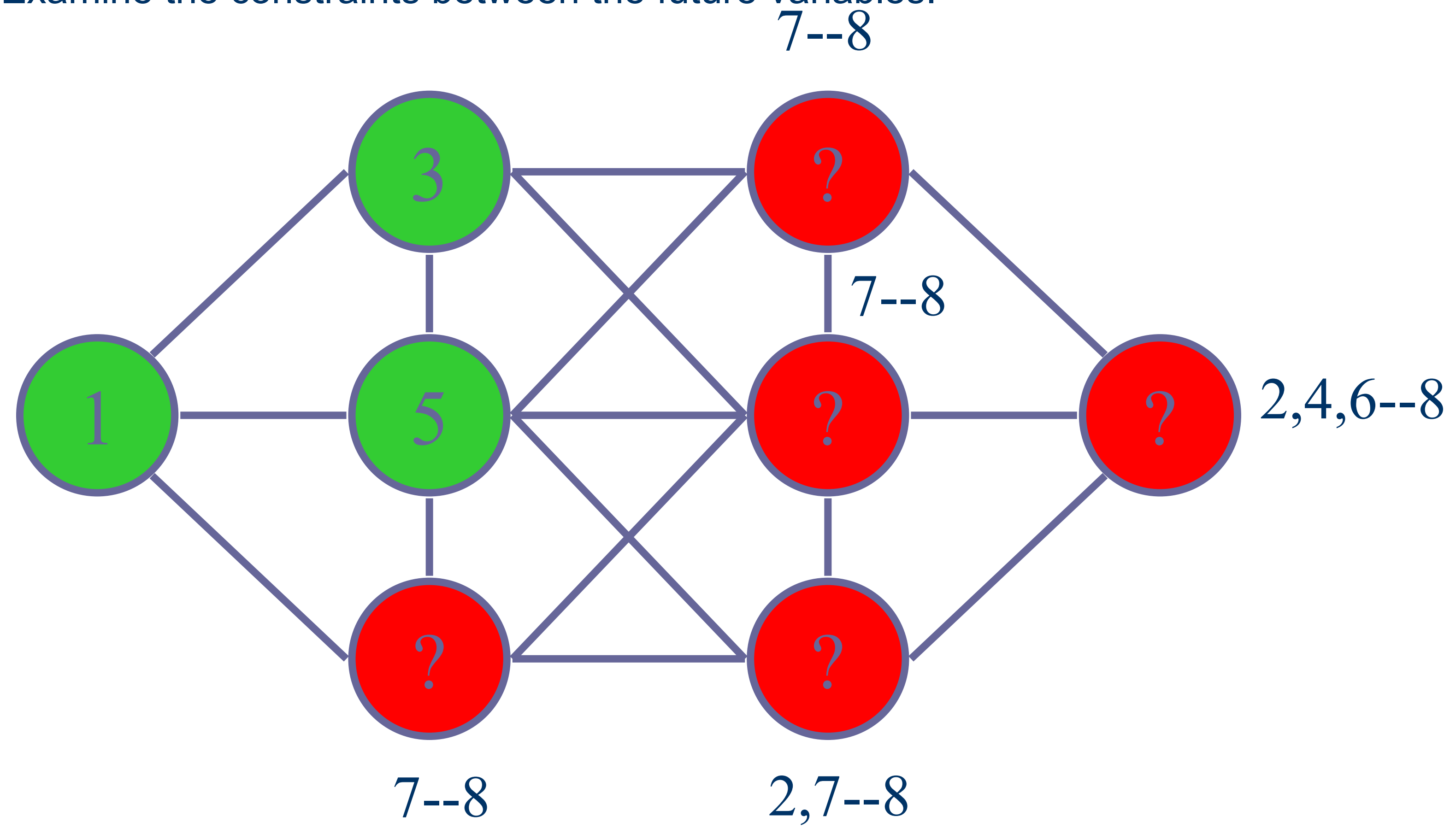
A State During Search

- 2 deadends after this state.
- Can be detected immediately!



A State During Search

- Examine the constraints between the future variables.

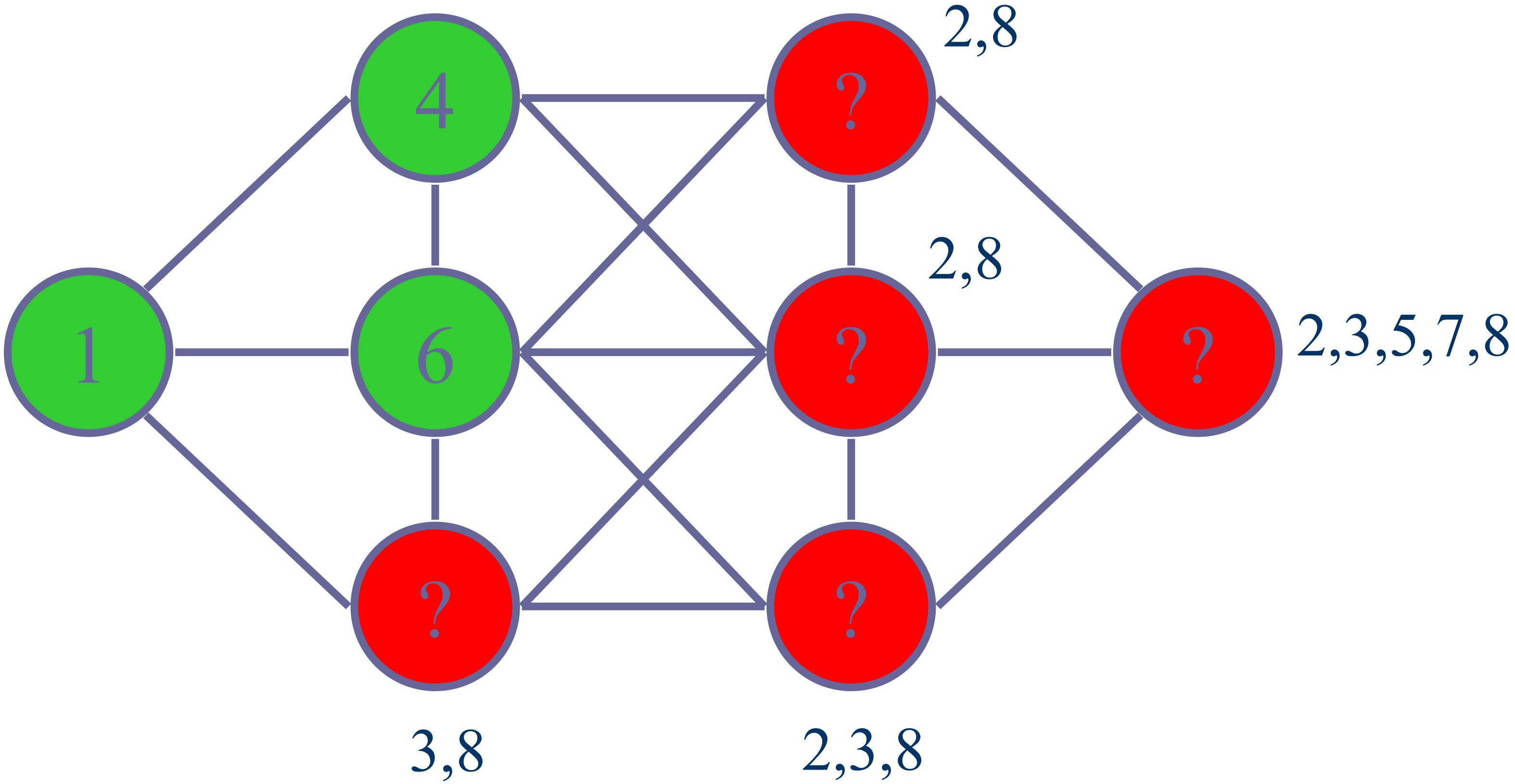


What's going on?

- Bad choice of variables, bad assignment of values.
  - → Good heuristic choice is very important!
- Good heuristics are always possible?
  - Yes and no 🙃
- What can we do then?
  - Apply stronger form of propagation during search!
- Is that all?
  - Better modelling can result in stronger form of propagation.

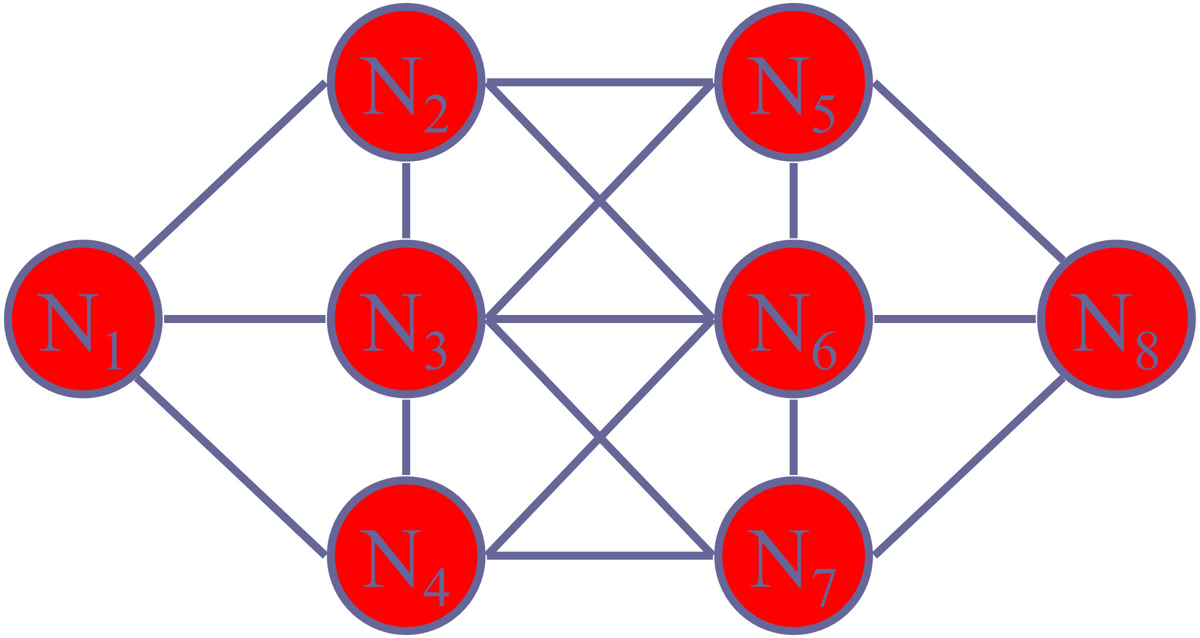
Another State

- Cannot detect the inconsistency of N_3= 6.
  - Future variables are fine wrt the constraints.



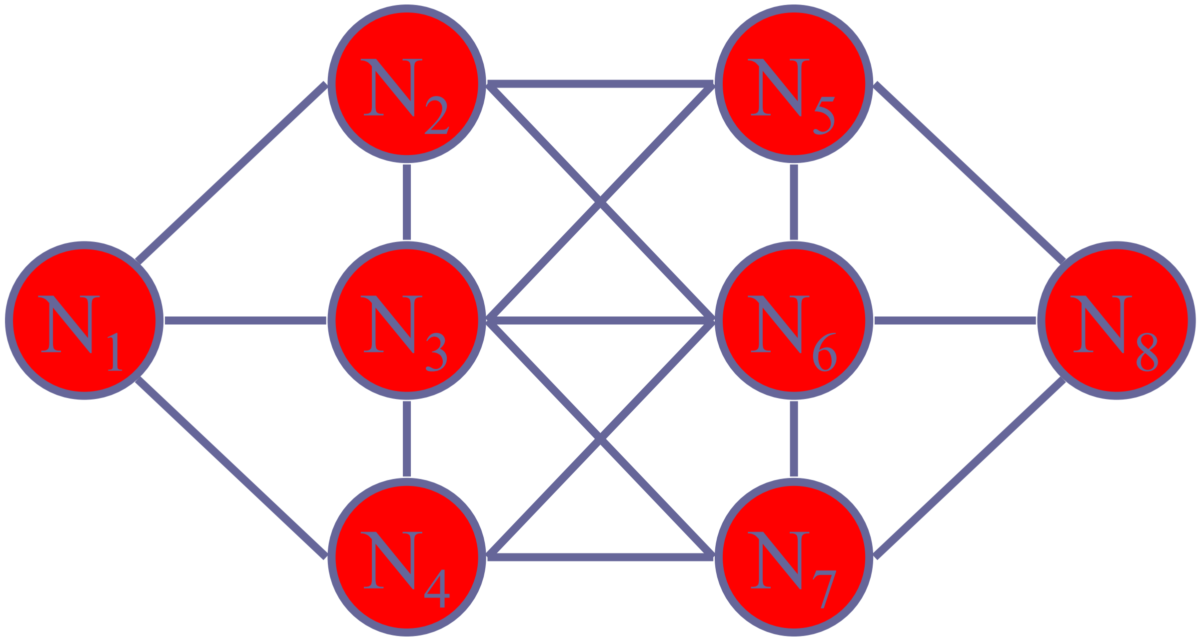
Initial Model

- Constraints:
  - for all i < j s.t. Ni and Nj are adjacent $|N_i - N_j| > 1$
  - for all i < j $N_i \neq N_j$



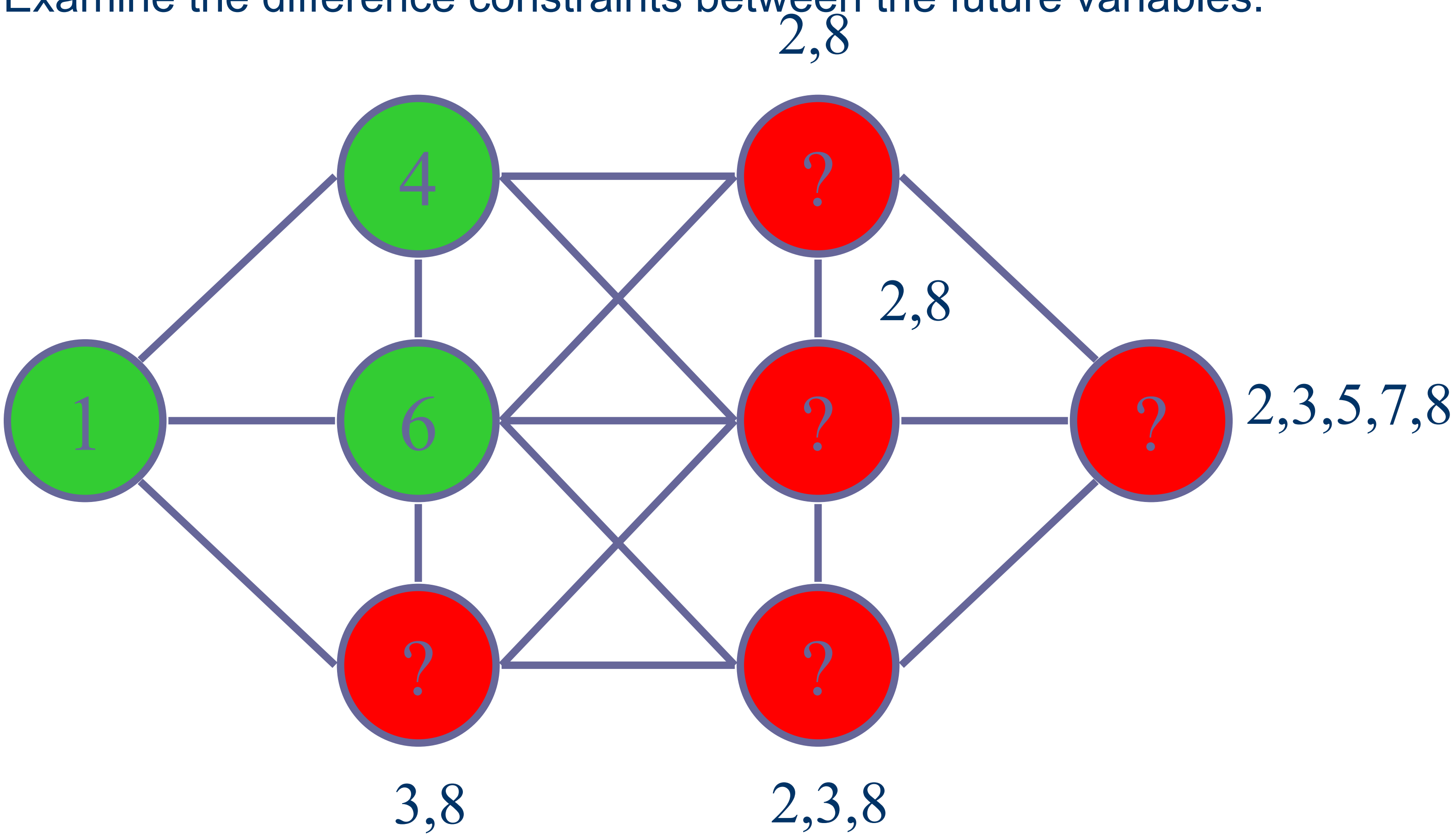
Better Model

- Constraints:
  - for all i < j s.t. Ni and Nj are adjacent $|N_i - N_j| > 1$
  - alldifferent($[N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8]$)



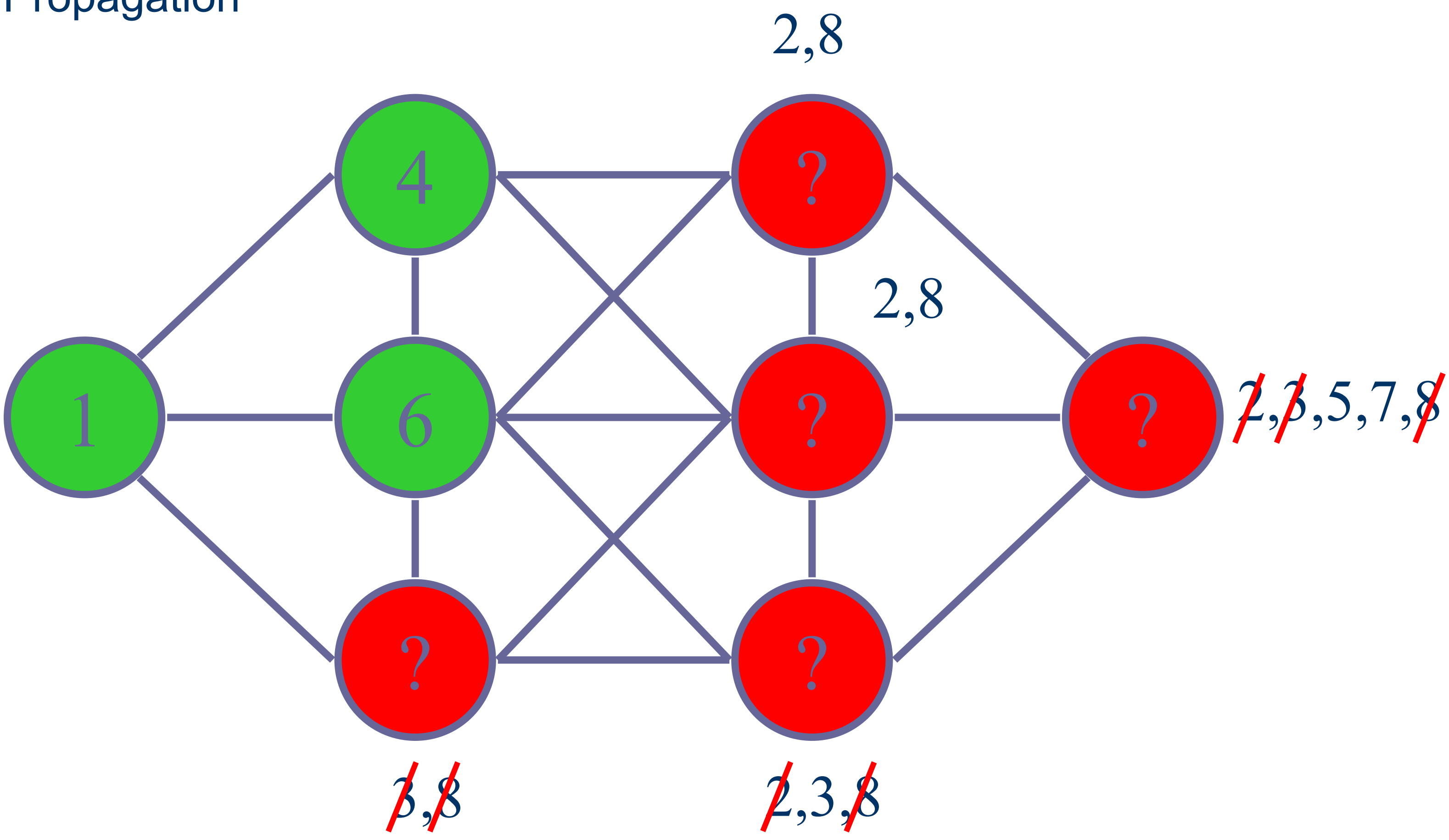
Another State

- Examine the difference constraints between the future variables.



Another State

- Propagation



Constraint Programming

- For an efficient CP solving, we need:
  - effective propagation algorithms;
  - a model with effectively propagating constraints;
  - effective search algorithm and heuristics.
- Attention!
  - Intelligent reasoning comes with a cost.
  - Need a good balance.

Constraint Programming

- Declarative programming, as in ILP:
  - the user models the problem;
  - an underlying search-based solver returns a solution.
- Computer programming:
  - the user needs to program a strategy to search for a solution:
    - search algorithm, heuristics, …
  - otherwise, solving process can be inefficient.

Examples from MiniZinc

Map Coloring

- What is the minimum number of colors needed to color the map?

Map Coloring

```
% number of colours available
int: nc = 6;

% variables mapping states to colours
var 1..nc: wa; var 1..nc: nt;
var 1..nc: sa; var 1..nc: q;
var 1..nc: nsw; var 1..nc: v;
var 1..nc: t;

% adjacent states have different
% colours

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;

% minimize the total number of colours used
solve minimize max([wa,nt,sa,q,nsw,v,t]);
```

Data

Variables & domains

Constraints

Search & objective

Crypto Arithmetic

```
    SEND
+   MORE
--------
= MONEY
```

```
    MINI
+   ZINC
--------
= ROCKZ
```

SEND + MORE = MONEY

```
include "alldifferent.mzn";

%variables for the digits
var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;

constraint              1000 * S + 100 * E + 10 * N + D
                      + 1000 * M + 100 * O + 10 * R + E
       = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;

output ["   \(S)\(E)\(N)\(D)\n",
        "+  \(M)\(O)\(R)\(E)\n",
        "= \(M)\(O)\(N)\(E)\(Y)\n"];
```

Variables & domains

Constraints

Search

SEND + MORE = MONEY

```
include "alldifferent.mzn";

%variables for the digits
var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;

constraint          1000 * S + 100 * E + 10 * N + D
                  + 1000 * M + 100 * O + 10 * R + E
       = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;

output ["   \(S)\(E)\(N)\(D)\n",
        "+  \(M)\(O)\(R)\(E)\n",
        "= \(M)\(O)\(N)\(E)\(Y)\n"];
```

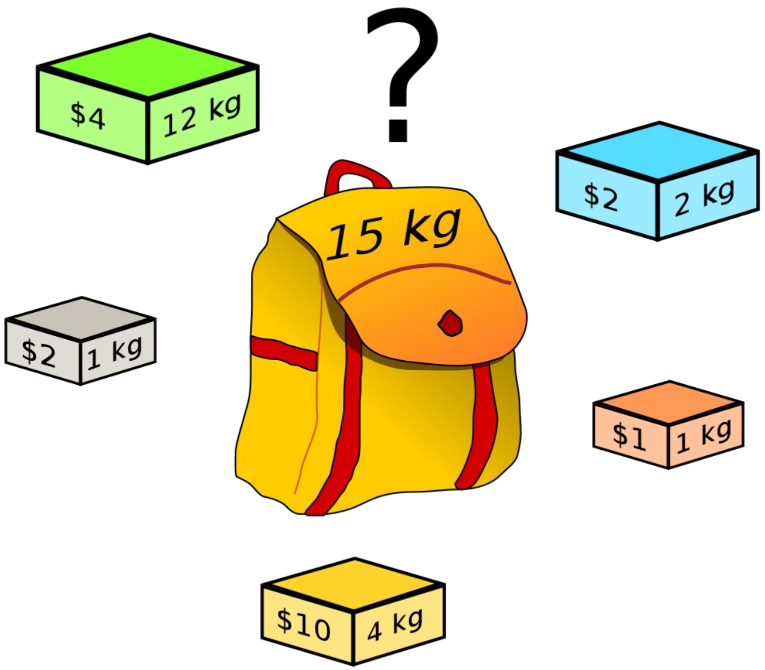
Variables & domains

Constraints

Search

Knapsack

- Given items, each with a weight and a value, determine which item and how many of it to pack in your knapsack without exceeding its capacity while maximizing your profit?



Knapsack

```
enum ITEM; %a set of items to pack
int: capacity; %knapsack capacity

array[ITEM] of int: profits; %item profits
array[ITEM] of int: weights; %item weights

array[ITEM] of var 0..1: knapsack; % a Bool. variable
                                   % for each item
var int: totalProfit; %objective function

constraint sum (i in ITEM) (weights[i]*knapsack[i]) <= capacity;
constraint totalProfit = sum (i in ITEM) (profits[i]*knapsack[i]);

solve maximize totalProfit;

output ["knapsack = \(knapsack)\n", "Total Profit = ", show(totalProfit)];
```

Data

Variables & domains

Cons.ts

Search & objective

Knapsack

```
enum ITEM; %a set of items to pack
int: capacity; %knapsack capacity

array[ITEM] of int: profits; %item profits
array[ITEM] of int: weights; %item weights

array[ITEM] of var 0..1: knapsack; % a Bool. variable
                                    % for each item
var int: totalProfit; %objective function

constraint sum (i in ITEM) (weights[i]*knapsack[i]) <= capacity;
constraint totalProfit = sum (i in ITEM) (profits[i]*knapsack[i]);

solve maximize totalProfit;

output ["knapsack = \(knapsack)\n", "Total Profit = ", show(totalProfit)];
```

Data

Variables & domains

Cons.ts

Search & objective

Task Assignment

```
int: n;
set of int: WORK = 1..n;
int: m;
set of int: TASK = 1..m;
array[WORK,TASK] of int: profit;

array[WORK] of var TASK: x;
array[WORK] of var int: px =
      [ profit[w,x[w]] | w in WORK ];
var int: obj = sum(w in WORK)(px[w]);

include "alldifferent.mzn";
constraint alldifferent(x);

ann: varselect = largest;
ann: valselect = indomain;

solve :: int_search(px, varselect, valselect, complete)
      maximize obj;

output ["obj = \(obj); x = \(x);\n"];
```

Data

Variables & domains

Constraints

Search & objective

Task Assignment

```
int: n;
set of int: WORK = 1..n;
int: m;
set of int: TASK = 1..m;
array[WORK,TASK] of int: profit;

array[WORK] of var TASK: x;
array[WORK] of var int: px =
      [ profit[w,x[w]] | w in WORK ];
var int: obj = sum(w in WORK)(px[w]);

include "alldifferent.mzn";
constraint alldifferent(x);

ann: varselect = largest;
ann: valselect = indomain;

solve :: int_search(px, varselect, valselect, complete)
      maximize obj;

output ["obj = \(obj); x = \(x);\n"];
```

Data

Variables & domains

Constraints

Search & objective