Introduction to Constraint Programming

Combinatorial Problem Solving (CPS)

Enric Rodríguez-Carbonell (based on materials by Javier Larrosa)

February 11, 2020

Constraint Satisfaction Problem

- A constraint satisfaction problem (CSP) is a tuple (X, D, C) where:
  - $X = \{x_1, x_2, \ldots, x_n\}$ is the set of variables
  - $D = \{d_1, d_2, \ldots, d_n\}$ is the set of domains
    (d_i is a finite set of potential values for x_i)
  - $C = \{c_1, c_2, \ldots, c_m\}$ is a set of constraints

- For example: $x, y, z \in \{0, 1\}, x + y = z$ is a CSP where:
  - Variables are: x, y, z
  - Domains are: $d_x = d_y = d_z = \{0, 1\}$
  - There is a single constraint: $x + y = z$

Constraints

- A constraint C is a pair (S, R) where:
  - $S = (x_{i_1}, ..., x_{i_k})$ are the variables of C (scope)
  - $R \subseteq d_{i_1} \times ... \times d_{i_k}$ are the tuples satisfying C (relation)
- According to this definition: $x + y = z$ in the CSP $x, y, z \in \{0, 1\}, x + y = z$ is short for

$$((x, y, z), \{(0, 0, 0), (1, 0, 1), (0, 1, 1)\})$$

- A tuple $\tau \in d_{i_1} \times ... \times d_{i_k}$ satisfies C iff $\tau \in R$
- The arity of a constraint is the size of its scope
  - Arity 1: unary constraint (usually embedded in domains)
  - Arity 2: binary constraint
  - Arity 3: ternary constraint
  - ...
- This corresponds to the extensional representation of constraints

Constraints

- But constraints are usually described more compactly: intensional representation
- A constraint with scope S is determined by a function

$$\prod_{x_i \in S} d_i \longrightarrow \{\text{true}, \text{false}\}$$

- Satisfying tuples are exactly those that give true
- In the example: $x + y = z$
- Unless otherwise stated, we will assume that evaluating a constraint takes time linear in the arity
- This is usually, but not always, true

Solution

- Given a CSP with variables $X = \{x_1, x_2, \ldots, x_n\}$, domains $D = \{d_1, d_2, \ldots, d_n\}$ and constraints C, a solution is an assignment of values $(x_1 \mapsto \nu_1, \cdots, x_n \mapsto \nu_n)$ such that:
  - Domains are respected: $\nu_i \in d_i$
  - The assignment satisfies all constraints in C
- Solving a CSP consists in finding a solution to it
- Other related problems:
  - Finding all solutions
  - Finding a best solution wrt. an objective function (then we talk of a Constraint Optimization Problem)

Examples (I): Prop. Satisfiability

- Given a formula F in propositional logic, is F satisfiable?
- Variables are the atoms of the formula
- Variables have all domain $\{\mathrm{true}, \mathrm{false}\}$
- A single constraint: the evaluation of F must be 1

- Let F be $(p \vee q) \ \wedge \ (p \vee \neg q) \ \wedge \ (\neg p \vee q)$:

Examples (I): Prop. Satisfiability

- Given a formula F in propositional logic, is F satisfiable?
- Variables are the atoms of the formula
- Variables have all domain $\{\mathrm{true}, \mathrm{false}\}$
- A single constraint: the evaluation of F must be 1

- Let F be $(p \vee q) \ \wedge \ (p \vee \neg q) \ \wedge \ (\neg p \vee q)$:
  - Variables are p, q

Examples (I): Prop. Satisfiability

- Given a formula F in propositional logic, is F satisfiable?
- Variables are the atoms of the formula
- Variables have all domain $\{\text{true}, \text{false}\}$
- A single constraint: the evaluation of F must be 1

- Let F be $(p \vee q) \ \wedge \ (p \vee \neg q) \ \wedge \ (\neg p \vee q)$:
  - Variables are p, q
  - Domains are $d_p = d_q = \{\text{true}, \text{false}\}$

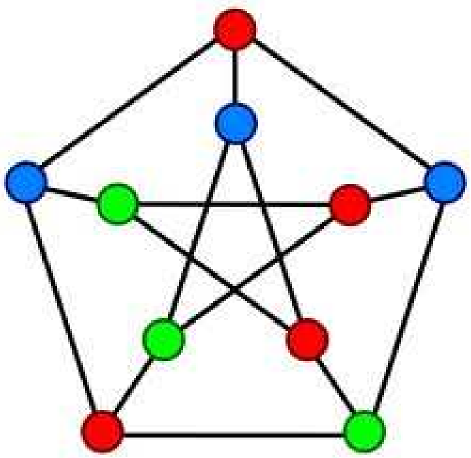
Examples (I): Prop. Satisfiability

- Given a formula F in propositional logic, is F satisfiable?
- Variables are the atoms of the formula
- Variables have all domain $\{\text{true}, \text{false}\}$
- A single constraint: the evaluation of F must be 1

- Let F be $(p \vee q) \ \wedge \ (p \vee \neg q) \ \wedge \ (\neg p \vee q)$:
  - Variables are p, q
  - Domains are $d_p = d_q = \{\text{true}, \text{false}\}$
  - Constraint is $(p \vee q) \ \wedge \ (p \vee \neg q) \ \wedge \ (\neg p \vee q) = \text{true}$

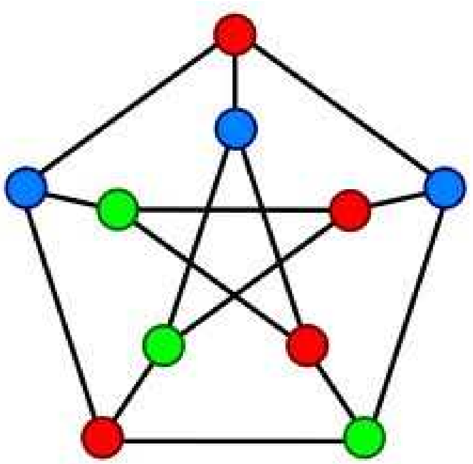
Examples (II): Graph Coloring

- Given a graph $G = (V, E)$ and $K > 0$ colors,
  can vertices be painted so that neighbors have different colors?



Examples (II): Graph Coloring

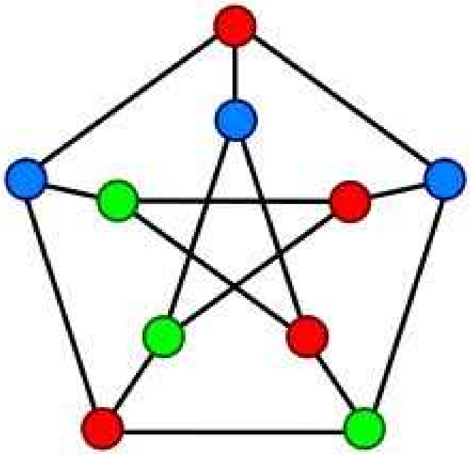
- Given a graph $G = (V, E)$ and $K > 0$ colors, can vertices be painted so that neighbors have different colors?



  - Variables are $\{c_v \mid v \in V\}$, the color for each vertex

Examples (II): Graph Coloring

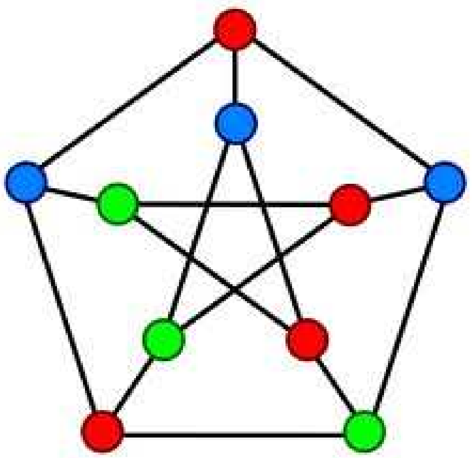
- Given a graph $G = (V, E)$ and $K > 0$ colors,
  can vertices be painted so that neighbors have different colors?



  - Variables are $\{c_v \mid v \in V\}$, the color for each vertex
  - Domains are $\{1, 2, \ldots, K\}$, the available colors

Examples (II): Graph Coloring

- Given a graph $G = (V, E)$ and $K > 0$ colors,
  can vertices be painted so that neighbors have different colors?

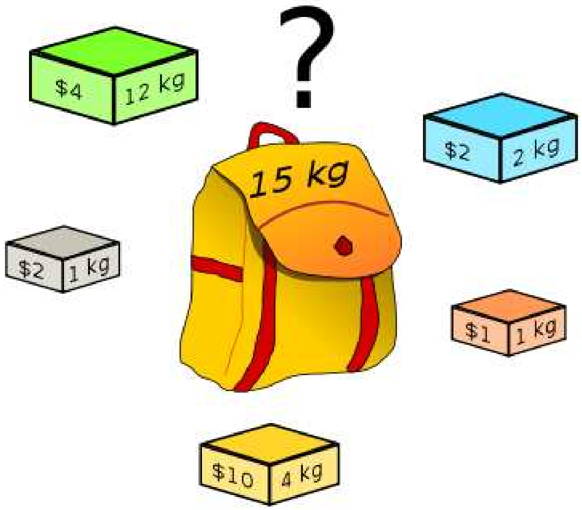


  - Variables are $\{c_v \mid v \in V\}$, the color for each vertex
  - Domains are $\{1, 2, \ldots, K\}$, the available colors
  - Constraints are: for each $(u, v) \in E$, $c_u \neq c_v$

Examples (III): Knapsack

- Given:
  - n items with weights w_i and values v_i
  - a capacity W
  - a number V,

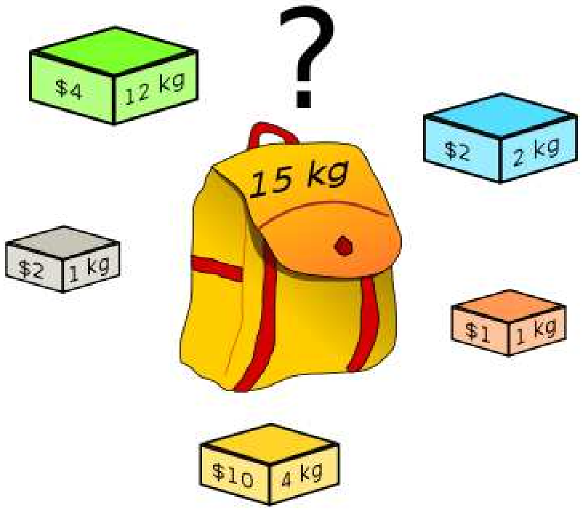
  is there a subset S of the items s.t. $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq V$?



Examples (III): Knapsack

- Given:
  - n items with weights w_i and values v_i
  - a capacity W
  - a number V,

  is there a subset S of the items s.t. $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq V$?

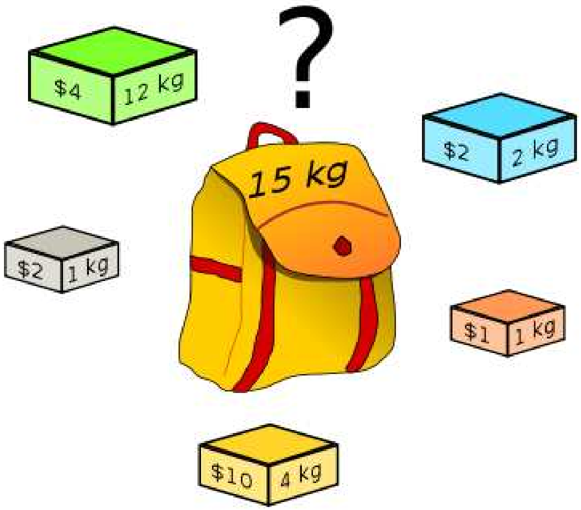


- Variables: n variables x_i meaning "item i is selected"

Examples (III): Knapsack

- Given:
  - n items with weights w_i and values v_i
  - a capacity W
  - a number V,

  is there a subset S of the items s.t. $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq V$?

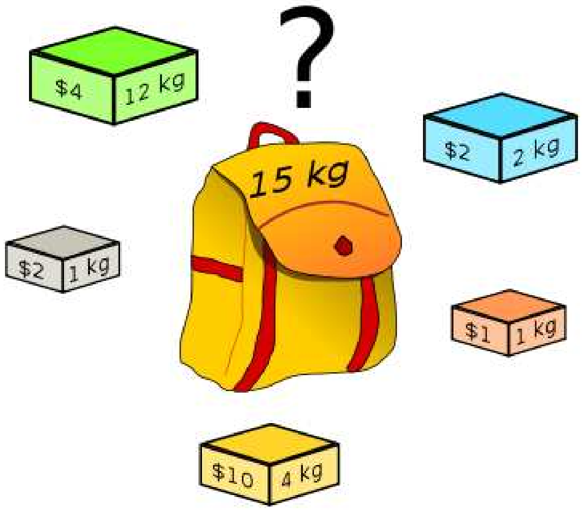


- Variables: n variables x_i meaning "item i is selected"
- Domains: $d_i = \{0, 1\}$

Examples (III): Knapsack

- Given:
  - n items with weights w_i and values v_i
  - a capacity W
  - a number V,

  is there a subset S of the items s.t. $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq V$?



- Variables: n variables x_i meaning "item i is selected"
- Domains: $d_i = \{0, 1\}$
- Constraints: $\sum_{i=1}^{n} w_i x_i \leq W$, $\sum_{i=1}^{n} v_i x_i \geq V$

Complexity

- **Theorem**. Solving a CSP is an NP-complete problem

  Proof:

  - It is in NP, because one can check a solution in polynomial time
  - It is NP-hard, as there is a reduction e.g. from Prop. Satisfiability (which is known to be NP-complete)

- For any CSP, there are instances that require exp time
  Can we solve real life instances in reasonable time?

Constraint Programming

- Constraint programming (CP) is a general framework for modeling and solving CSP's:
    - Offers the user many kinds of constraints,
      which makes modeling easy and natural

      Check out the Global Constraint Catalogue at
      `https://sofdem.github.io/gccat/gccat/sec5.html`
      with more than 400 different types of constraints!

    - Provides solving engines for those constraints
      (CP toolkits: in this course, Gecode `http://www.gecode.org`)

Generate and Test

- How can we solve CSP's?
- 1st naïf approach: Generate and Test (aka Brute Force)
  - Generate all possible candidate solutions
    (assignments of values from domains to variables)
  - Test whether any of these is a true solution indeed

Generate and Test

- Example: *Queens Problem*. Given $n \geq 4$,
  put n queens on an $n \times n$ chessboard so that they don't attack each other

  Wlog, we can place one queen per row
  so that no two are in the same column or diagonal.

  - Variables: c_i, column of the queen of row i
  - Domains: all domains are $\{1, 2, \ldots, n\}$
  - Constraints: no two are in same column/diagonal

Q				
Q				
Q				
Q				
Q				

Q				
Q				
Q				
Q				
	Q			

Basic Backtracking

- Generate and Test is very inefficient
- 2nd approach to solving CSP's: Basic Backtracking
- The algorithm maintains a partial assignment that is consistent with the constraints whose variables are all assigned:
  - Start with an empty assignment
  - At each step choose a var and a value in its domain
  - Whenever we detect a partial assignment that cannot be extended to a solution, backtrack: undo last decision

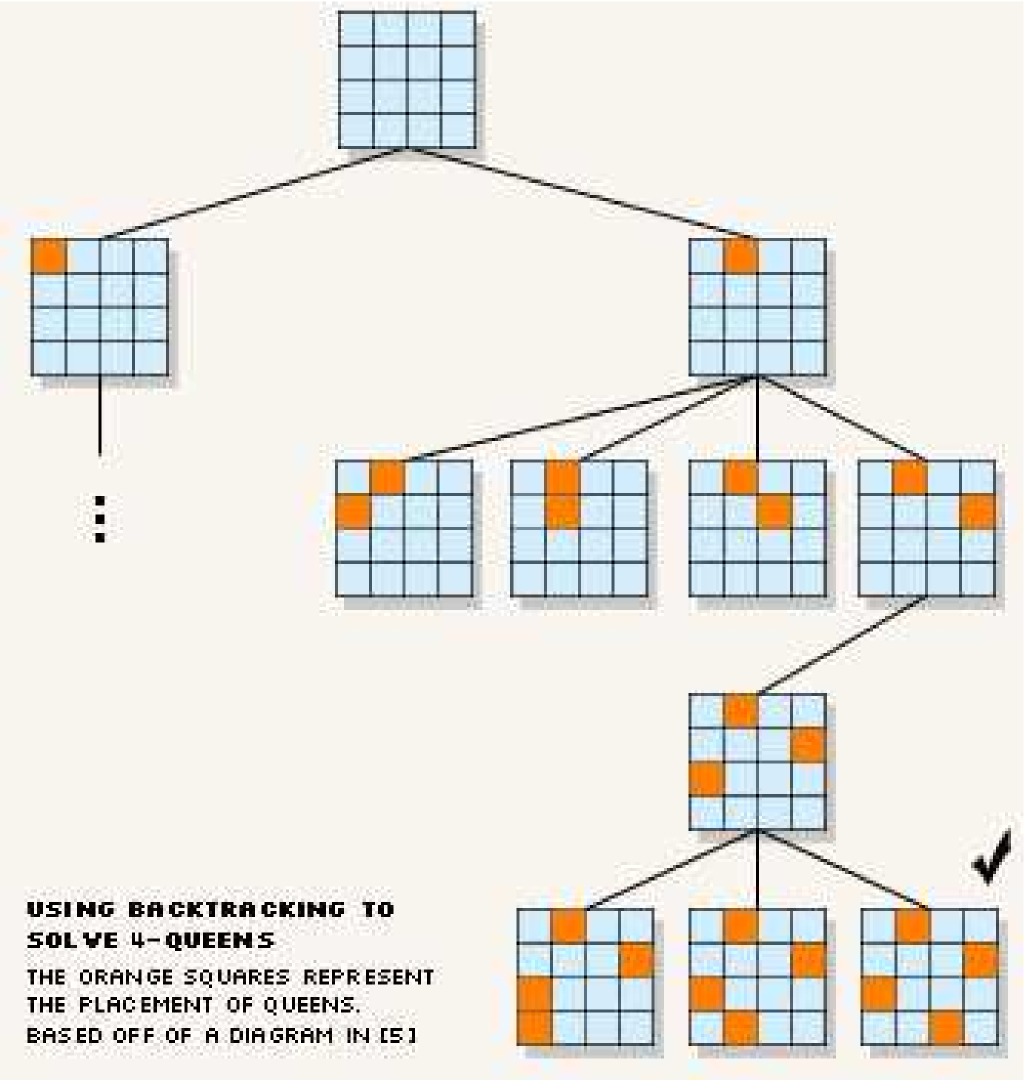
Basic Backtracking

- We can solve the problem by calling `backtrack(x1)`:

```
function backtrack(variable X) returns bool
  for all a in domain(X) do
    val(X) := a
    if compatible(X, assigned)
      assigned := assigned ∪ {X}
      if no next(X)                  then return TRUE
      else if backtrack(next(X)) then return TRUE
      else assigned := assigned - {X}
  return FALSE

function compatible(variable X, set A) returns bool
  for all constraint C with scope in A∪{X} and not in A do
     // Let A be {Y1, ..., Yn}
     if (val(X), val(Y1),..., val(Yn)) don't satisfy C then
        return FALSE
  return TRUE
```

Basic Backtracking



Basic Backtracking

- The set of all possible partial assignments forms a search tree:
  - The root corresponds to the empty assignment
  - Each edge corresponds to assigning a value to a var
  - For each node, there are as many children
    as values in the domain of the chosen variable
  - Generate and Test corresponds to visiting each of the leaves until a solution is found
  - Complexity: $O(m^n \cdot e \cdot r)$
    - $n =$ no. of variables
    - $m =$ size of the largest domain
    - $e =$ no. of constraints
    - $r =$ largest arity
  - Basic Backtracking performs a depth-first traversal
  - Complexity: the same, as in the worst case we need to visit all leaves
  - But in practice it works much better than Generate and Test

Basic Backtracking

- Problems with backtracking
  - Inconsistencies may be found late, after a lot of useless work

    If $x_1 \mapsto a$ is incompatible with $x_n \mapsto \text{anything}$,
    then BT explores the subtree rooted at $x_1 \mapsto a$
    (if x_1 can take m values, this subtree is $\frac{1}{m}$ of the whole search tree!)
    to realize that no solution can be found

  - The right backtracking point may not be the last decision

Basic Backtracking

					Q				
		Q							
Q									
									Q
						Q			
								Q	

Basic Backtracking

					Q				
		Q		X	X	X			
Q	X	X	X		X		X		
X	X	X		X	X			X	Q
X	X	X			X	Q		X	X
X		X	X		X	X	X	Q	X
X		X		X	X	X	X	X	X
X		X	X		X	X		X	X
X		X		X	X	X		X	X
X	X	X	X	X	X	X	X	X	X

Propagation

- CP approach: prune search tree a priori by removing values from the domains that can't appear in any solution

```
while (solution not found) do
    assign values to some of the variables
    propagate with constraints to prune other domains
    if (found inconsistency) undo last decision
```

- Smaller search tree, at the cost of more time per node
- There exist different kinds of propagation with different tradeoffs between pruning power and cost in time

Propagation

Q				
X	X			
X		X		
X			X	
X				X

Propagation

Q				
X	X	Q		
X	X	X	X	
X		X	X	X
X		X		X

Propagation

Q				
X	X	Q		
X	X	X	X	Q
X		X	X	X
X		X		X

Propagation

Q				
X	X	Q		
X	X	X	X	Q
X	Q	X	X	X
X	X	X		X

Propagation

Q				
X	X	Q		
X	X	X	X	Q
X	Q	X	X	X
X	X	X	Q	X

No search!