# Part VII

# Constraints Satisfaction Techniques



## Introduction

- Constraint satisfaction is a general and powerful problem-solving paradigm that is applicable to a broad set of aeras, e.g.,
  - planning and scheduling
  - computer vision
  - patter recognition
  - etc.
- A constraint satisfaction problem (CSP) takes as input:
  1. A set of variables and their respective domains
  2. a set of constraints on the compatible values that variables may take
- The objective is to find a value for each variable within its domaines such that these values meet all the constraints



## CSP and Planning

- CSP can be use in planning in two different ways:
  1. Directly, by stating a planning problem as a CSP.
     - It is possible to follow an approach similar to that of SAT, i.e., to encode a planning problem into a CSP and to rely entirely on CSP tools for planning
  2. Indirectly, by using CSP techniques within approaches specific to planning
- The latter approach is more frequent



## Constraint Satisfaction Problems

## Constraint Satisfaction Problems

- A CSP over a finite domains is defined to be a triple $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where:
  - $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of $n$ variables
  - $\mathcal{D} = \{D_1, \ldots, D_n\}$ is the set of finite domains of the variables, $x_i \in D_i$
  - $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a finit set of constraints. A constraint $c_j$ of some arity $k$ restricts the possibles values of a subset of $k$ variables $\{x_{j1}, \ldots, x_{jk}\} \subseteq \mathcal{X}$. $c_j$ is defined as a subset of the cartesian product: $c_j \subseteq D_{j1} \times \ldots \times D_{jk}$, i.e., as the set of tuples of values allowed by this constraint for its variables : $\{(v_{j1}, \ldots, v_{jk}) \in D_{j1} \times \ldots \times D_{jk} \mid (v_{j1}, \ldots, v_{jk}) \in c_j\}$.



## Solution to a CSP

- A solution to a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a $n$-tuple $\sigma = (v1, \ldots, v_n)$ such that $vi \in D_i$ and the values of the varaibles $x_i = v_i$, for $1 \leq i \leq n$, meet all the constraints in $\mathcal{C}$. A CSP is consistant if such a solution $\sigma$ exists.
- A tuple $\sigma$ is a solution iff for every constraints $c_j \in \mathcal{C}$, the values specified in $\sigma$ for the variables $x_{j1}, \ldots, x_{jk}$ of $c_j$ correspond to a tuple $(v_{j1}, \ldots, v_{jk}) \in c_j$



## Constraints in a CSP

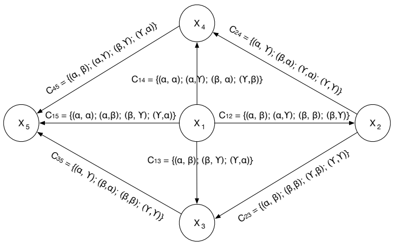
- Constraints in a CSP can be:
  1. Explicite: A explicite constraint lists the set of its allowed tuples or the complementary set of forbidden tuples, e.g., $x_i = v_i$
  2. Implicite: A implicite constraint use one or more relation symbols, e.g., $x_i \neq x_j k$
- There are two specific constraints:
  1. Universal which is satisfied by every tuple of values of its variables. In other words there is no constraint between its variables.
  2. Empty which forbids all tuples and cannot be satisfied



## Binary CSP

- Many popular combinatorial problems can be expressed as binary CSP
- A binary CSP is a CSP where all it constraints are binary relation
- A binary CSP can be represented as a constraint network, i.e., a graph in which each node is a CSP variable $x_i$ labeled by its domain $dD_i$, and each edge $(x_i, x_j)$ is labeled by the corresponding constraint on $x_i$ and $x_j$
- A binary CSP is symmetrical if for every constraints $c_{ij} \in \mathcal{C}$, the symmetrical relation $c'_{ij} \in \mathcal{C}$
- A unary constraint $c_i$ on a variable $x_i$ is simply subset of $D_i$, thus, one can replace $D_i$ with $c_i$ and remove this unary constraint

## Binary CSP Example




- A solution but not the only one to this CSP is the tuple $(\alpha, \gamma, \beta, \gamma, \alpha)$ which satisfies all eight constraints:

$$(\alpha, \gamma) \in c_{12} \quad (\alpha, \beta) \in c_{13} \quad (\alpha, \gamma) \in c_{14} \quad (\alpha, \alpha) \in c_{15}$$
$$(\gamma, \beta) \in c_{23} \quad (\gamma, \gamma) \in c_{24} \quad (\beta, \alpha) \in c_{35} \quad (\gamma, \alpha) \in c_5$$

- The other possible solutions are: $(\alpha, \beta, \beta, \alpha, \beta)$, $(\alpha, \gamma, \beta, \alpha, \beta)$ and $(\beta, \gamma, \gamma, \alpha, \gamma)$



## CSP Properties (1/3)

- Two CSPs $\mathcal{P}$ and $\mathcal{P}''$ on the same set of variables $\mathcal{X}$ are equivalent if they have the same set of solutions
- A value $v$ in a domain $D_i$ is redundant if it does not appear in any solution
  - For instance, $\gamma$ is redundant in $D_1$ and $\alpha$ redundant in $D_2$
- A tuple in a constraint $c_j$ is redundant if it is not an element of any solution
  - For instance, pair $(\beta, \beta)$ in $c_{12}$ is redundant and $(\alpha, \gamma)$ in $c_{13}$
- If all values a domain of if all tuples in a constraint are redundant, then the CSP problem is not consistant



## CSP Properties (2/3)

- A CSP is minimal if it has no redundant values in the domains of $\mathcal{D}$ and no redundant tuples in the constraints of $\mathcal{C}$
- A set of constraints $\mathcal{C}$ is consistent with a constraint $c$ iff the following holds: when $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is consistent, then $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{c\})$ is also consistent
  - For instance, the constraint $c_{25} = \{(\alpha, \alpha), (\beta, \beta), (\gamma, \gamma)\}$ is consistent with our CSP. It leaves the tuples $(\alpha, \beta, \beta, \alpha, \beta)$ and $(\beta, \gamma, \gamma, \alpha, \gamma)$ as solutions to $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{c_{25}\})$



## CSP Properties (3/3)

- A set of constraints $\mathcal{C}$ entails a constraint $c$, denoted $\mathcal{C} \models c$, iff the CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is equivalent to $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{c\})$, i.e., have the same set of solutions.
  - For instance, the constraint $c_25 = \{(\alpha, \alpha), (\beta, \beta), (\gamma, \gamma)\}$ is not entails by $\mathcal{C}$ because it reduces the set of solution: the two tuples $(\alpha, \gamma, \beta, \gamma, \alpha)$ and $(\alpha, \gamma, \beta, \alpha, \beta)$ are not consistent with $c_25$.
- A constraint $c \in C$ is redundant iff the CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is equivalebt to $(\mathcal{X}, \mathcal{D}, \mathcal{C} - \{c\})$
  - For instance, the constraints $c_13$ is redundant

## CSP Properties and so what ...

- Given a CSP, one may be interested in addressing:
  1. a resolution problem, i.e., finding a solution tuple
  2. Checking CSP consistency, i.e., checking if a solution exists is interested
  3. Filtering some redundant values or some redundant tuples from constraints is interested because the size of the problem
  4. Working with minimal CSP by removing every redundant values and tuples
- Problems:
  1. Checking CSP consistency is NP-complete
  2. Resolution and minimal reduction is NP-complete
- but ...
  - Checking CSP consistency could be approximated in polynomial time
  - Filtering is polynomial



# Planning problem as CSPs

## Planning problem as CSPs

- We will introduce a technique for encoding a bouded planning problem $P$ into a constraints satisfaction problem $P'$
- This encoding as the following properties:
  - Given $P$ and a constant integer $k$, there is a one to one mapping between the set of solution of $P$ of length $\leq k$ and the set of solution of $P'$
  - From a solution of the CSP problem $P'$, if any, the mapping provides a solution plan to the planning problem $P$
  - If $P'$ has no solution, then there is no plan of length $\leq k$ for the problem $P$
- The encoding will not use classical representation but instead state variable representation that is more convenient to compact encoding into CSPs.



## Reminders on State-Variable Representation (1/2)

- Recall that a state-variable representation for planning relies on the following elements:
  1. Constant symbols are partitioned into disjoint classes corresponding to the objects of the domain, e.g., the classes of robots, locations, etc.
  2. Object varaible symbols are typed varaibles: each ranges over a class or the union of classes of constants, e.g., $r \in robots$, $l \in location$, etc.
  3. State variable symbols are functions from the set if states and one or more sets of constants into a set of constants:

     rloc: robots $\times S \leftarrow$ locations

     rload: robots $\times S \leftarrow$ container $\cup \{$nil$\}$

     cpos: containers $\times S \leftarrow$ locations $\cup$ robots

  4. Relation symbols are rigid relation one the constrants that do not vary form state to state, e.g., adjacent(loc1,loc2), etc.

## Reminders on State-Variable Representation (2/2)

- A planning operator is a triple:
  $o = (\text{name}(o), \text{precond}(o), \text{effects}(o))$ where
  - $\text{precond}(o)$ is a set of expression that are condisions on stat-variables and on rigid relations
  - $\text{effects}(o)$ is a set of assignments of values to state variables
- The statement of a bounded planning problem is
  $P = (O, R, s_0; g, k)$ where $O$, $s_0$ and $g$ are as usual, $R$ is the set of rigid relations of the domain, and $k$is the length bound



## State-Variable Representation Example

- Consider a simplified version of the DWR domain with no pile and no cranes with three operators:
  1. $\text{move}(r, l, m)$
     *;; robot r at location l moves to an adjacent location m*
     precond: $\text{rloc}(r) = l, \text{adjacent}(l, m)$
     effects: $\text{rloc}(r) \leftarrow m$
  2. $\text{load}(c, r, l)$
     *;; robot r load container c at location l*
     precond: $\text{rloc}(r) = l, \text{cpos}(c) = l, \text{rload}(r) = \text{nil}$
     effects: $\text{rload}(r) \leftarrow c, \text{cpos}(c) \leftarrow r$
  3. $\text{unload}(c, r, l)$
     *;; robot r unload container c at location l*
     precond: $\text{rloc}(r) = l, \text{rload}(r) = c$
     effects: $\text{rload}(r) \leftarrow \text{nil}, \text{cpos}(c) \leftarrow l$



## Encoding a Planning Problem into CSP

- A bound planning problem $P = (O, R, s_0, g, k)$ in the state-variable representation is encoded into a CSP $P'$ in 4 steps:
  1. The definition of the CSP variables of $P'$
  2. The definition of the constraints of $P'$ encoding the initial state $s_0$ and the goal $g$
  3. The encoding of the actions that are instances of operators in $O$
  4. The encoding of the frame axioms



## Step 1: CSP Variables

- The CSP variables of $P'$ bounded by $k$ are defined as follows:
  - For each state variable $x_i$ of $P$ ranging over $D_i$ and for each $0 \leq j \leq k$, there is a CSP varaible of $P'$, $x_i(j, v_u, \ldots, v_w)$ whose domain is $D_i$
  - For each $P \leq j \leq k - 1$, ther eis a CSP varaibles of $P'$, denoted $\text{act}(j)$, whose domain is the set of all possible actions in the domain, in addition to a no-op action that has no preconditions and no effects, i.e., $\forall s, \gamma(s, noop) = s$. More formally:

$$\text{act}: \{0, ldots, k-\} \leftarrow D_{act}$$

$$D_{act} = \{a(v_u, \ldots, v_w) \text{ ground instance of } o \in O\} \cup \{\text{noop}\}$$

- Hence, the CSP variables are all the state variables of $P$, plus one varaible $\text{act}(j)$ whose value corresponds to the action carried out in state $j$

## Step 1: Example

- Let $P = (O, R, s_0, g)$ with
  - 3 operators move, load and unload
  - the constants: robot (r1), containers (c1, c2, c3) andlocations (l1, l2, l3)
  - $s_0$ = { rloc(r1) = l1, rload(r1) = nil,
    cpos(c1) = l1, cpos(c2) = l2, cpos(c3) = l2}
  - $g$ = {cpos(c1) = l2, cpos(c2) = l1 }
- Assume we are looking for a plan of at most $k = 4$ step. The coressponding CSP $P'$ has the following set of variables:
  - rloc($j$,r1) ∈ {l1,l2,l3}, for $0 \le j \le 4$
  - rload($j$,r1) ∈ {c1,c2,c3,nil}, for $0 \le j \le 4$
  - cpos($j$,$c$) ∈ {l1,l2,l3,r1}, for $c$ ∈ {c1,c2,c3} and for $0 \le j \le 4$
  - act($j$) ∈ {move(r1,l1,l2), ..., load(c1,r1,l1), ..., unload(c1,r1,l1), ...}, for $0 \le j \le 3$



## Step 2: Encoding of $s_0$ and $g$ as Constraints

- The encoding of the state $s_0$ and the goal $g$ into constraints follows directly from the definition of the CSP variables.
- Every state variable $x_i$ whose value in $s_0$ is $v_i$ is encoded into a unary constraint of the corresponding CSP variable for $j = 0$ of the form:

$$(xi(0) = v_i)$$

- Every state variable $x_i$ whose value is $v_i$ in the goal $g$ is encoded into a unary constraint of the corresponding CSP variable for $j = k$

$$(xi(k) = v_i)$$



## Step 2: Example

- The state $s_0$ of our example is translated into the following constraints:

  rloc(0,r1) = l1, rload(0,r1) = nill, cpos(0,c1) = l1, cpos(0,c2) = l2, cpos(0,c3) = l2
- The goal $g$ is translated into the following constraints:

  cpos(4,c1) = l2, cpos(4,c2) = l1



## Step 3: Encoding Actions as Constraints

- Let $a(vu, \ldots, v_w)$ be an actions such that the constants $v_u, \ldots, v_w$, then $\forall j, 0 \le j \le k-1$:
  - Every condition of the form $(x_i = v_i)$ in precond($a$) is translated into a constraint with a single tuple of the form:

$$(\text{act}(j) = a(v_u, \ldots, v_w), x_i(j) = v_i)$$

  - Every condition of the form $(x_i \in D_i')$ in precond($a$) is translated into a constraint corresponding to the set of pairs:

$$\{(\text{act}(j) = a(v_u, \ldots, v_w), x_i(j) = v_i) \mid v_i \in D_i'\}$$

  - Every assignment of the form $(x_i \leftarrow v_i)$ in effects($a$) is translated into a constraint with a single tuple:

$$(\text{act}(j) = a(v_u, \ldots, v_w), x_i(j+1) = v_i)$$

## Step 3: Example

- The move operator has only one condition and one effet

  move$(r, l, m)$

  *;; robot r at location l moves to an adjacent location m*

  precond: $\text{rloc}(r) = l, \text{adjacent}(l, m)$

  effects: $\text{rloc}(r) \leftarrow m$

- it is encoded into the following constraints:

$$\{(\text{act}(j) = \text{move}(r, l, m), \text{rloc}(j, r) = l) \mid \text{adjacent}(l, m) \wedge 0 \leq j \leq 3\}$$
$$\{(\text{act}(j) = \text{move}(r, l, m), \text{rloc}(j + 1, r) = m) \mid \text{adjacent}(l, m) \wedge 0 \leq j \leq 3\}$$



## Step 3: Example

- The load operator has 3 conditions and two effets

  load$(c, r, l)$

  *;; robot r load container c at location l*

  precond: $\text{rloc}(r) = l, \text{cpos}(c) = l, \text{rload}(r) = \text{nil}$

  effects: $\text{rload}(r) \leftarrow c, \text{cpos}(c) \leftarrow r$

- it is encoded into the following constraints:

$$\{(\text{act}(j) = \text{load}(c, r, l), \text{rloc}(j, r) = l) \mid 0 \leq j \leq 3\}$$
$$\{(\text{act}(j) = \text{load}(c, r, l), \text{rload}(j, r) = \text{nil}) \mid 0 \leq j \leq 3\}$$
$$\{(\text{act}(j) = \text{load}(c, r, l), \text{cpos}(j, c) = l) \mid 0 \leq j \leq 3\}$$
$$\{(\text{act}(j) = \text{load}(c, r, l), \text{rload}(j + 1, r) = c) \mid 0 \leq j \leq 3\}$$
$$\{(\text{act}(j) = \text{load}(c, r, l), \text{cpos}(j + 1, c) = r) \mid 0 \leq j \leq 3\}$$



## Step 4: Encoding Frame Axioms as Constraints

- A frame axiom constraint says that any state variable that is invariant for an action
- A frame axiom is encoded into a ternary constraint involving 3 state variable bu of in state $j$ and $j + 1$
- More precisely for every action $a(v_u, \ldots, v_w)$ and every state variable $x_i$ that is invariant for $a$, we have a constraint with the following set of triples:

$$\{(\text{act}(j) = a(v_u, \ldots, v_w), x_i(j) = v_i, x_i(j + 1 = v_i) \mid v_i \in D_i)\}$$

- Note that every state variable is invariant for no-op action.



## Step 4: Example

- Two state variables are invariant for the action move: rload and cpos
- The frame axioms for this operator are the following for $0 \leq j \leq 3$:

$$\{(\text{act}(j) = \text{move}(r, l, m), \text{rload}(j, r) = v), \text{rload}(j + 1, r) = v) \mid v \in D_{rload}\}$$
$$\{(\text{act}(j) = \text{move}(r, l, m), \text{cpos}(j, c) = v), \text{rload}(j + 1, r) = v) \mid v \in D_{cpos}\}$$

where

- $D_{rload} = \{c1, c2, c3, nil\}$
- $D_{cpos} = \{l1, l2, l3, r1\}$

## Plan extraction

- We have encoded a planning problem $P$ and an integer $k$ into a CSP $P'$
- Let assume that a CSP solver return a tuple $\sigma$ as a solution of $P'$ or failure if $P'$ has no solutions
- The tuple $\sigma$ gives a value to every CSP variable in $P'$, in particular the action act($j$)
- Let these values in $\sigma$ be: $\text{act}(j) = a_{j+1}$, for $0 \leq j \leq k-1$
- Each $a_j$ is an action of $P$ and the sequence $\pi = \langle a_1, \ldots, a_k \rangle$ is a valid plan of $P$ that possiblt includes no-op action.



## Analysis of the CSP encoding

- SAT and CSP encoding are very similar
  - SAT encoding needs complete exclusion axioms, i.e., one action per step
  - State encoding is simpler due to state-variable representation
  - SAT encoding prevented can be considered for CSP encoding
- CSP encoding require $m = k(n+1) - 1$ CSP variables where $n$ is the number of state and $k$ the bound on the plan length
- Planning problem with a bound is psace- or nexptime-complete where as CSP and SAT are np-complete
  - This blowup results in the exponential number of boolean variable for SAT
  - For CSP, the number of variables is linear in the size of the problem but the total size of the CSP is exponential, i.e., $d = \Pi_{i=1}^{i=m} |D_i|$, where $D_i$ is the domain of the CSP variables $x_i$
- CSP solver with ternary constraints are less efficient



# CSP techniques and Algorithms

## CSP techniques et Algorithms

- We will present mains algorithms to
  1. solve CSP
  2. filter its domains and constraints

## Search Algorithms for CSPs

**Algorithm (Backtrack($\sigma, \mathcal{X}$))**

**if** $\mathcal{X}$ = *emptyset* **then return** $\sigma$
*Select any variable* $x_i \in \mathcal{X}$
**foreach** $v_j \in \sigma$ **do**
  $D_i \leftarrow D_i \cap \{v \in D_i \mid (v, v_i) \in c_{ij}\}$
**end**
**if** $D_i$ = *emptyset* **then return** failure
*nondeterministically choose* $v_i \in D_i$
`Backtrack` $(\sigma.(vi), \mathcal{X} - \{xi\})$

- This algorithme is sound and complete
- It runs in time $O(n^d)$ for $d = max_i\{|D_i]\}$
- Practically, it performance depends on the heuristics used for ordering the variables and the for choosing their values



## Heuristics for CSP Search Algorithms

- Heuristics for variables ordering rely on the idea that a backtrack done early in the search tree is less costly than a deep backtrack
  - Thus, it is interested to chose the most constraint variable, i.e., the variable $x_i$ with the smaller domain $|D_i$
- Heuristics for the choice of values apply the opposite principle preferring the least constraining value $v_i$ for a variable $x_i$.
  - This done by computing the number of paires in constraints $c_{ij}$ in which $v_i$ appears. The value $v_i$ chosen is the most frequent



## Filtering Techniques

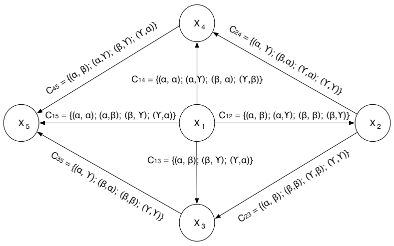
- Despite good heuristics, the resolution of a CSP remains in general a costly combinatorial problem
- It is possible to test the consistency of CSP with fast algorithms that provide a necessary but not sufficient condition of consistency
- These algorithms address the filtering problem introduce earlier, i.e., removing redundante values from domains or redundant tuples from constraints
- Filtering techniques rely on a contraint propagation operation
  - Propaging a constaint on a varaible $x$ consits of computing its local effects on varaibles adjacent to $x$ in the constraint network, removing redundant values and tuples
  - This removal in turn lead to new constraints that need to be propagated until a fixe-point is reached



## Arc Consistency

- A straightforward filter, called arc consistency, consists of removing from a domain $D_i$ any value that does not satisfy constraints $c_{ij}$ involving $x_i$
- Such value is redundant because it necessary violates a constraint
- A naive algorithm for arc consistency is to perform an iteration over all pairs of variables $(x_i, x_j)$, $i \neq j$ with the 2 following updates:
  1. $D_i \leftarrow \{v \in D_i \mid \exists v' \in D_j : (v, v') \in c_{ij}\}$
  2. $D_j \leftarrow \{v' \in D_j \mid \exists v \in D_i : (v, v') \in c_{ij}\}$
- If after propagation a domain is empty, the CSP is said to be inconsistent
- Otherwise the CSP is said to be arc-consistent or 2-consistent
- Note a Arc-consistent CSP is not necessary consistent

## Arc Consistency Example




- Filtering the variable $(x_1, x_2)$ reduces the domaines of $D_1 = \{\alpha, \beta\}$ and $D_1 = \{\beta, \gamma\}$ because no pair in $c_{12}$ starts with a $\gamma$ or end with an $\alpha$



## A better Arc Consistency Algorithm

**Algorithm (AC3($L$))**

**while** $L \neq \emptyset$ **do**
  *Select any pairs* $(x_i, x_j)$ *in* $L$ *and remove it from* $L$
  $D \leftarrow \{v \in D_i \mid \exists v' \in D_j : (v, v') \in c_{ij}$
  **if** $D \neq D_i$ **then**
    $D_i n \leftarrow D$
    $L \leftarrow L \cup \{(x_i, x_k), (x_k, x_i) \mid \exists c_{ik} or c_{ki} \in \mathcal{C}, k \neq k\}$
  **end**
**end**

- AC3 keeps a list $L$ of pairs of variables whose domains have to be filtered
- AC3 runs in time $O(md^2)$, where $m = |C|$ and $d = max_i\{D_i\}$



## Path Consistency

- A more thorough filter is path consistency
- It consists of testing all triples of variables $x_i$, $x_j$ and $x_k$ checking they have values that meet the 3 constraints $c_{ij}$, $c_{jk}$ and $c_{ik}$
- A pair of values $(v_i, v_j)$ can be part of a solution if it meets the constraints $c_{ij}$ and if a value $v_k j$ for $x_k$ such that $(v_i, v_k)$ meets $c_{ik}$ and $(v_k, v_j)$ meets $c_{kj}$
- In other words, the two constrains $c_{ik}$ and $c_{kj}$ entail by transitivity a constraint on $c_i j$
- Let us define a composition operation between constraints, denote $\bullet$:

$$c_{ik} \bullet c_{kj} = \{(v, v'), v \in D_i, v' \in D_j | \\ \exists x \in D_k : (v, w) \in c_{ik} and (w, v') \in c_{kj}\}$$



## Path Consistency Filtering Operation

- Let us define a composition operation between constraints, denote $\bullet$:

$$c_{ik} \bullet c_{kj} = \{(v, v'), v \in D_i, v' \in D_j | \\ \exists x \in D_k : (v, w) \in c_{ik} and (w, v') \in c_{kj}\}$$

- The composition $c_{ik} \bullet c_{kj}$ defines a constraint from $x_i$ to $x_j$ enrailed by the 2 constraints $c_{ik}$ and $c_{kj}$.
- A pair $(v_i, v_j)$ has met $c_{ij}$ as well as the composition $c_{ik} \bullet c_{kj}$ for every $k$ otherwise it is redundant
- The following filtering operation is:

$$c_{ij} \leftarrow c_{ij} \cap [c_{ik} \bullet c_{kj}], \forall k \neq i, j$$

## Path Consistency Algorithm

**Algorithm (PC($\mathcal{C}$))**

**repeat**
  **foreach** $k : 1 \leq k \leq n$ **do**
    **foreach** *pair* $i, j : 1 \leq i \leq j \leq n, i \neq k, j \neq k$ **do**
      $c_{ij} \leftarrow c_{ik} \cap [c_{ik} \bullet c_{kj}]$
      **if** $c_{ij} = \emptyset$ **then return** inconsistent
    **end**
  **end**
**until** *until stabilization of all constraints in* $\mathcal{C}$

- A constraints network arc-consistence may not stay arc-consistent after a call to PC
- It is possible to maintaining both with the filtering operation

$$c_{ij} \leftarrow c_{ij} \cap [c_{ik} \bullet c_{kk} \bullet c_{kj}], \text{for all triples including } i = j$$



## Local Search techniques and Hybrid Approaches

- Local search presented in the cours on SAT are applicable to CSP solving
- We have to define a neighborhood method
- This approaches are not complet but may be very efficient



# To go further

## Further readings

R. Barták, M. Salido, F. Rossi:
**New trends in constraint satisfaction, planning, and scheduling: a survey.**
Knowl. Eng. Rev. 25(3): 249-279 (2010)

R. Dechter
**Constraint Processing**
Morgan Kaufmann, 2003