05 - Constraints Programming

Artificial Intelligence

J. BUISINE[1]

[1]IT Student at ULCO Calais

February 22, 2018

Summary

1. Constraints programming
   - Constraints Satisfaction Problem
   - Constraints Optimization Problem
2. Inference
   - Domains filtering
   - Constraints propagation
3. Search strategy & Heuristics
   - Search strategy
   - BPRA Model
     - Branching & Heuristics
     - Look back
     - Look ahead

What is constraints programming ?

Constraints programming's aim is to propose a generic way for modeling problems based on constraints in order to resolve them.

Advantages

- Possess a formalism which makes easy the representation of many problems.
- Possess a vast set of algorithms and heuristics allowing to solve these problems.

Summary

What is a CSP ?

A mathematical problem where we look for states or for satisfying objects a number of constraints or of criteria.

- CSP is situated at the heart of the programming by constraints.
- Problem instance is represented by a network of constraints.
- CSP is known as a **NP-Complete** problem.

CSP components

A constraint satisfaction problem consists of three components, X, D, and C.

CSP components

A constraint satisfaction problem consists of three components, X, D, and C.

Variables

X is a set of variables, $\{X_1, ..., X_n\}$

CSP components

A constraint satisfaction problem consists of three components, X, D, and C.

Variables

X is a set of variables, $\{X_1, ..., X_n\}$

Domain variables

D is a set of domains, $\{D_1, ..., D_n\}$, one for each variable.

CSP components

A constraint satisfaction problem consists of three components, X, D, and C.

Variables

X is a set of variables, $\{X_1, ..., X_n\}$

Domain variables

D is a set of domains, $\{D_1, ..., D_n\}$, one for each variable.

Constraints

C is a set of constraints that specify allowable combinations of values.

Definition

The **variable** x is an unknown to whom we have to give a value among those of a set called current domain denote as follow $dom(x)$.

Definition

The **variable** x is an unknown to whom we have to give a value among those of a set called current domain denote as follow $dom(x)$.

Kind of domains

- $dom^{init}(x)$ **:** initial domain of the variable (before searching in the tree).
- $dom(x)$ **:** current domain of the variable (during the search in the tree at a specific node which is not the initial).

Definition

The **variable** x is an unknown to whom we have to give a value among those of a set called current domain denote as follow $dom(x)$.

Kind of domains

- $dom^{init}(x)$ **:** initial domain of the variable (before searching in the tree).
- $dom(x)$ **:** current domain of the variable (during the search in the tree at a specific node which is not the initial).

Search space

$\prod_{i=1}^{q} dom(x_i)$, represents the search space of the CSP.

Constraints definition

A **constraint** c consists of a pair $\langle scope, rel \rangle$.

- *scope* is a set of variable that participate in the constraint, noted $scp(c)$.
- *rel* is a relation that defines the values that those variables (called **tuples**) can take on, noted $rel(c)$. *rel* is generally a subset of the Cartesian product from variables of $scp(c)$.

Constraints definition

A **constraint** c consists of a pair $\langle scope, rel \rangle$.

- *scope* is a set of variable that participate in the constraint, noted $scp(c)$.
- *rel* is a relation that defines the values that those variables (called **tuples**) can take on, noted $rel(c)$. *rel* is generally a subset of the Cartesian product from variables of $scp(c)$.

Arity definition

The **arity** of a constraint c is the number of variable involved by c noted $|scp(c)|$.

Constraints definition

A **constraint** c consists of a pair $\langle scope, rel \rangle$.

- *scope* is a set of variable that participate in the constraint, noted $scp(c)$.
- *rel* is a relation that defines the values that those variables (called **tuples**) can take on, noted $rel(c)$. *rel* is generally a subset of the Cartesian product from variables of $scp(c)$.

Arity definition

The **arity** of a constraint c is the number of variable involved by c noted $|scp(c)|$.

- **unary** iff $|scp(c)| = 1$.
- **binary** iff $|scp(c)| = 2$.

Example

$dom(a) = \{1, 2, 3\}$
$dom(b) = \{0, 1\}$

$c_{ab} : \langle (a, b), \{(1, 0), (2, 0), (3, 1), (2, 1)\} \rangle$

Remark

This constraint is a **binary** constraint, so $|scp(c)| = 2$.

Intension

A constraint c is defined in intension when $rel(c)$ is implicitly described by a boolean formula.

Example

$c_{ab} : a > b + 10$ with $scp(c_{ab}) = \{a, b\}$

Extension

A constraint c is defined in extension when $rel(c)$ is implicitly described :

- positively by listing authorized tuples of c.
- negatively by listing forbidden tuples of c.

Example

Let consider variables a, b, c with $dom(a) = dom(b) = dom(c) = \{2, 3\}$, then extension constraint c can be defined negatively by :

$$c_{abc} : rel(c_{xyz}) \backslash \{(2, 2, 3), (3, 2, 3)\}$$

with $rel(c) = \{\ (2,\ 2,\ 2),\ (2,\ 2,\ 3),\ (3,\ 2,\ 3),\ (3,\ 3,\ 3)\ \}$

Global

A global constraint c is based on semantic and can concern an any number of variables.

Global

A global constraint c is based on semantic and can concern an any number of variables.

Example

QAP instance uses Integer as indexes to represent solution. This constraint can be represented by this global constraint :

$$c = allDifferent(x_i, ..., x_n)$$

Global

A global constraint c is based on semantic and can concern an any number of variables.

Example

QAP instance uses Integer as indexes to represent solution. This constraint can be represented by this global constraint :

$$c = allDifferent(x_i, ..., x_n)$$

Other global constraints

sum, table, notAllEqual, cardinality, lexicographic, or, and...

Map coloring example

Australia map coloring problem (P. Norvig, 2017)

- $X = \{WA, NT, SA, Q, NSW, V, T\}$
- $D_i = \{red, green, blue\}$
- $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$

Australia states list

- **WA** : Western Australia
- **NT** : Northern Territory
- **SA** : South Australia
- **Q** : Queensland
- **NSW** : New South Wales
- **V** : Victoria
- **T** : Tasmania

Constraint network

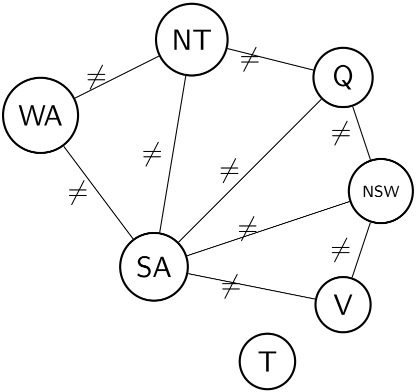
Definition

A constraint network (CN) is graph representation of CSP noted $P = (\mathcal{X}, \mathcal{C})$ where $\mathcal{X}$ is the set of variables and $\mathcal{C}$ the set of constraints.

Constraint network

Definition

A constraint network (CN) is graph representation of CSP noted $P = (\mathcal{X}, \mathcal{C})$ where $\mathcal{X}$ is the set of variables and $\mathcal{C}$ the set of constraints.



Definition

An instantiation A of the set $X = \{x_1, ..., x_k\}$ of k variables, is a set of couples (x_i, v_i) where $v_i \in dom^{init}(x_i)$ and x_i is unique.

Definition

An instantiation A of the set $X = \{x_1, ..., x_k\}$ of k variables, is a set of couples (x_i, v_i) where $v_i \in dom^{init}(x_i)$ and x_i is unique.

Kind of instantiation

- **Partial** instantiation is one that assigns values to only some of the variables.
- **Illegal** instantiation is one that does violate at least one constraint.
- **Complete** instantiation is one in which every variables is assigned.
- **Consistent** (legal) instantiation is one that does not violate any constraints.

Definition

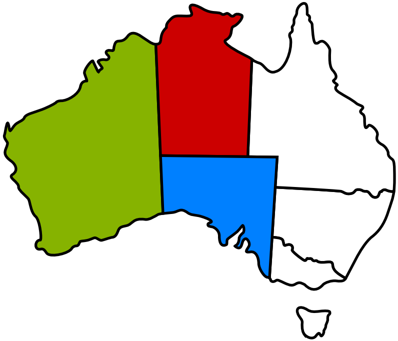
A solution of CSP is a **consistent** and **complete** instantiation.

Definition

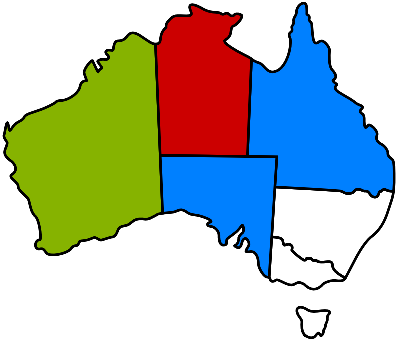
A solution of CSP is a **consistent** and **complete** instantiation.

Remark

A constraint network, a graph representation of a CSP, is satisfiable iff a **solution** exists.



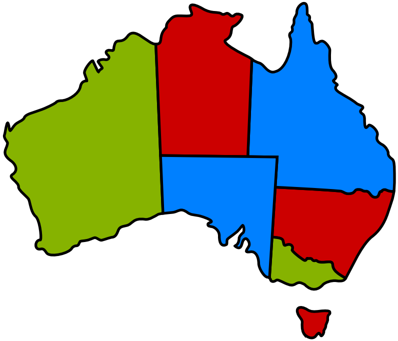
(a) Partial and consistent instantiation



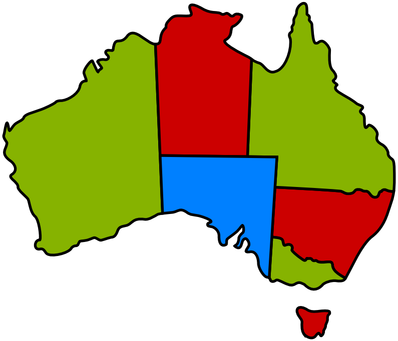
(b) Partial and illegal instantiation

Constraints Satisfaction Problem

Map coloring examples



(c) Complete and illegal instantiation (nogood)



(d) Complete and consistent instantiation (solution)

Summary

Definition of COP

An instance P of the Constraint Optimization Problem (COP) is composed of:

- a finite set of variables, denoted by $vars(P)$
- a finite set of constraints, denoted by $ctrs(P)$, such that $\forall c \in ctrs(P); scp(c) \subseteq vars(P)$
- an objective function o, also denoted by $obj(P)$, to be minimized or maximized.

Definition of COP

An instance P of the Constraint Optimization Problem (COP) is composed of:

- a finite set of variables, denoted by $vars(P)$
- a finite set of constraints, denoted by $ctrs(P)$, such that $\forall c \in ctrs(P); scp(c) \subseteq vars(P)$
- an objective function o, also denoted by $obj(P)$, to be minimized or maximized.

Remark

The goal is to find **optimal** solution by comparing its score.

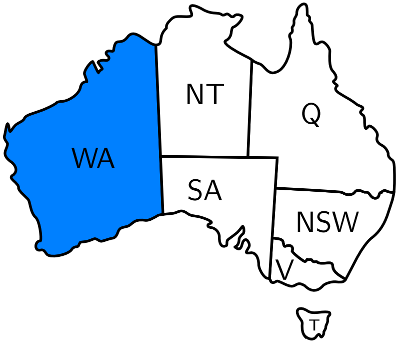
Summary

Definition

A conclusion reached on the basis of evidence and reasoning.

Definition

A conclusion reached on the basis of evidence and reasoning.



Example

$$WA = blue \implies NT \neq blue$$
$$WA = blue \implies SA \neq blue$$

$$dom(NT) = \{red, green\}$$
$$dom(SA) = \{red, green\}$$

Quick list

- **AC (Arc Consistency)** : all inconsistent values are identified and removed.
- **BC (Bounds Consistency)** : only bounds inconsistent values are identified and removed.
- ...

Domains filtering

Kind of filters

Quick list

- **AC (Arc Consistency)** : all inconsistent values are identified and removed.
- **BC (Bounds Consistency)** : only bounds inconsistent values are identified and removed.
- ...

Remark

Filters enable to **reduce** the search space (by cutting tree branch) to find a solution in reasonable time.

Definition

A constraint c is AC iff $\forall x \in scp(c)$, $\forall a \in dom(x)$, $\exists$ $x = a$ (a support) on c. Instantiation of $scp(c)$ which :

- is authorized by c.
- is valid, each values of the instantiation of $scp(c)$ are present in their respective $dom(x_i)$.
- contains $x = a$

Definition

A constraint c is AC iff $\forall x \in scp(c)$, $\forall a \in dom(x)$, $\exists\ x = a$ (a support) on c. Instantiation of $scp(c)$ which :

- is authorized by c.
- is valid, each values of the instantiation of $scp(c)$ are present in their respective $dom(x_i)$.
- contains $x = a$

Remark

To resume, c is AC iff after using **inference** heuristic, c always has valid and authorized value $\forall x_i \in scp(c)$. We can also tell that a variable is AC, a network is AC (every variables are AC).

k-consistency

A CSP is k-consistent if, for any $k - 1$ variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.

Example

Let consider variables a, b with $dom(a) = dom(b) = \{2, 3\}$ and binary constraint c_{ab} defined by :

$$c_{ab} : a \neq b$$

- $A = \{a = 2, b = 4\}$ is authorized but invalid $(4 \notin dom(b))$
- $A = \{a = 2, b = 2\}$ is not authorized but valid.
- $A = \{a = 2, b = 3\}$ is authorized and valid.

Domains filtering

Generalized Arc Consistent heuristic

Algorithm 1: filter(c : Constraint) : set of variables

```
X_reduce ← θ
foreach variable x ∈ scp(c) do
    foreach value a ∈ dom(x) do
        if ¬findSupport(c, x, a) then
            dom(x) ← dom(x)\{a}
            X_reduce ← X_reduce + x
        end
    end
end
return X_reduce
```

Domains filtering

Generalized Arc Consistent heuristic

Algorithm 2: filter(c : Constraint) : set of variables

$X_{reduce} \leftarrow \theta$
foreach *variable* $x \in scp(c)$ **do**
    foreach *value* $a \in dom(x)$ **do**
        if $\neg findSupport(c, x, a)$ **then**
            $dom(x) \leftarrow dom(x) \setminus \{a\}$
            $X_{reduce} \leftarrow X_{reduce} + x$
        end
    end
end
return X_{reduce}

Remark

Sometimes **revise** function is used for a specific check of a value from x. Hence, heuristic might depend of the kind of constraint (Simple Tabular Reduction for *table* constraint).

Summary

Constraints propagation explanation

Constraints propagation is a deduction process.

When a constraint c is checked and determined as Arc Consistency, we have to check all others constraints : $\exists x_i \in scp(c)$ such that $x_i \in scp(c_k) \backslash \{c\}$ where $k = |ctrs(P)|$.

This process determine quickly if a problem is satisfiable or not $(dom(x_i) = \theta)$.

Others consistency properties

- Path Consistency : a pair of values for a pair of variables is path-consistent iff it can be extended to a consistent instantiation of any third variable.
- Dual Consistency (Lecoutre, Cardon, and Vion, 2007) : iff $Y_b \in AC(P|_{X=a})$ and $X_a \in AC(P|_{Y=b})$.
- Singleton Arc Consistency : P is SAC iff $\forall i \in X, \forall a \in D_i$, the network $P|_{i=a}$ obtained by replacing D_i by the singleton a is not arc inconsistent.
- ...

These properties are stronger than AC which does check of constraints individually.

Constraints propagation

Generic algorithm

Algorithm 3: constraintsPropagation($P : (\mathcal{X}, \mathcal{C})$) : Boolean

$Q \leftarrow ctrs(P)$
while $Q \neq \theta$ **do**
  $c \leftarrow getCtr(P)$
  $ctrs(P) \leftarrow ctrs(P) \setminus \{c\}$
  $X_{reduce} \leftarrow filter(c)$
  if $\exists x \in X_{reduce}$ *such that* $dom(x) = \theta$ **then**
    return *false* // global inconsistency
  end
  foreach $c' \in ctrs(P)$ *such that* $(c' \neq c)$ *and* $(X_{reduce} \cap scp(c') \neq \theta)$ **do**
    // add c' to check because at least one variable of c is also involved in c'
    $Q \leftarrow Q + c'$
  end
end
return *true*

Algorithm 4: constraintsPropagation($P : (\mathcal{X}, \mathcal{C})$) : Boolean

```
1  Q ← ctrs(P)
2  while Q ≠ θ do
3      c ← getCtr(P)
4      ctrs(P) ← ctrs(P)\{c}
5      X_reduce ← filter(c)
6      if ∃x ∈ X_reduce such that dom(x) = θ then
7          return false // global inconsistency
8      end
9      foreach c' ∈ ctrs(P) such that (c' ≠ c) and (X_reduce ∩ scp(c') ≠ θ) do
10         // add c' to check because at least one variable of c is also involved in c'
11         Q ← Q + c'
12     end
13 end
14 return true
```

Remark

The **foreach** instruction at line 8 enables the constraint propagation process.

Kind of CP algorithm

Some constraints propagation algorithm exists like :

- Large grain (constraint-variable): AC3, AC2001/3.1, $AC3_d$, AC3.2/3.3, $AC3^{rm}$...
- Fine grain (constraint-variable-value) : AC4, AC6, AC7...

Summary

Search Strategy

Kind of search

Search space

$\prod_{i=1}^{q} dom(x_i)$, represents the search space of the CSP.

Search space

$\prod_{i=1}^{q} dom(x_i)$, represents the search space of the CSP.

Complete & Incomplete

We can differ two kinds of search in CSP :

- **Complete :** Cross the whole search space (cost & time consuming).
- **Incomplete :** local search with neighborhood principle.

The solution can be evaluated by $obj(P)$ function of COP.

Search space

$\prod_{i=1}^{q} dom(x_i)$, represents the search space of the CSP.

Complete & Incomplete

We can differ two kinds of search in CSP :

- **Complete :** Cross the whole search space (cost & time consuming).
- **Incomplete :** local search with neighborhood principle.

The solution can be evaluated by $obj(P)$ function of COP.

Local optima

Local search might give a **local optima** (e.g. hill-climbing, min-conflicts). In order to counter this and find **global optima**, we need to do a jump in the search space like *Simulated Annealing* or *Iterated Local Search* do for other problems.

Search Strategy

Strategy examples

Two search strategy algorithms are presented :

- **Generate and test :** complete method of resolution but naive and not optimized. Satisfiability of CSP can be proved and all solutions can be found. This method is not efficient at all.
- **BPRA (C. Lecoutre, 2009) :** *Branchement Propagation Retour-arrire Apprentissage*, a model which permits to combine techniques and heuristics (more efficient).

Summary

Model components

The BPRA model is composed of :

- **Branching :** way use to cross the tree (binary, not binary, depth, breadth).
- **Look-back :** manner to go backwards into the tree when a failure is encountered.
- **Propagation (look-ahead) :** kind of filter used (more or less strong consistencies)
- **Learning :** information kept during the cross (nogood encountered...), and manner to exploit this information.

Branching examples

We can enumerate different way of branching :

- **2-way branching :** use for binary branching.
- *d*-**way branching :** *d* is the number of variables. A variable unassigned is choose at each step.
- **Depth-first search :** first try to create an instantiation by cross tree in depth.
- **Breadth-first search :** less efficient than *Depth-first search* in terms of memory and solution found (long time before creating an instantiation).

Main principle

Based on the **fail-first** principle, *"To succeed, try at first where you have most luck to fail"*.

Main principle

Based on the **fail-first** principle, *"To succeed, try at first where you have most luck to fail"*.

Variable choice methods

- Static Variable Ordering (SVO)
- Dynamic Variable Ordering (DVO)
- Adaptive Variable Ordering

Definition

Static Variable Ordering (SVO) means that we take care of information of variables only before start the search. These information stayed statics during the search.

BPRA Model

Variable choice : Static Variable Ordering

Definition

Static Variable Ordering (SVO) means that we take care of information of variables only before start the search. These information stayed statics during the search.

SVO list

- **dom :** variable choice based on the size of the domain of variables (in increasing order).
- **lexico :** variable choice based on their name (orderly lexically).
- **deg (maxdeg) :** variable choice based on the degree (number of constraints where variable is involved) of the variable. This choice is in decreasing order.

Definition

Dynamic Variable Ordering (DVO) means that we take care of information of variables during all the search in a adaptive way.

BPRA Model

Variable choice : Dynamic Variable Ordering

Definition

Dynamic Variable Ordering (DVO) means that we take care of information of variables during all the search in a adaptive way.

SVO list

- **dom (mindom) :** variable choice based on the size of the domain of variables (in increasing order and adaptive way).
- **ddeg (maxdeg) :** variable choice based on the degree (number of constraints where variable is involved) of the variable. This choice is in decreasing order and in adaptive way.
- **dom/ddeg :** ratio between dom and ddeg always in increasing order and adaptive way.
- **dom/ddeg :** first dom and if equality between two variables, ddeg is used to decide.

Definition

Adaptive Variable Ordering means that we take care of current state and other information learned (feedback gained during search).

Definition

Adaptive Variable Ordering means that we take care of current state and other information learned (feedback gained during search).

Adaptive list

- **wdeg (Boussemart et al., 2004) :** a *weight* is attached to each constraint. When filtering, we have failure (nogood) due to constraint c, *weight* variable of c is increased by 1. Variable choice is based on the sum of the constraints weights where the variable is involved (in descreasing order).
- **impact :** variable impact on others variables (during constraint propagation). The mean of impacts are exploited at the current state.
- **last conflict (Lecoutre, Sais, et al., 2006):** variable choice is based on the last variable which causes a failure.

Definition

After choosing a variable, we need to choose a value to assign to it. We must choose a value which will reach a solution quickly.

BPRA Model
Value choice

Definition

After choosing a variable, we need to choose a value to assign to it. We must choose a value which will reach a solution quickly.

Adaptive list

- **Succed-first :** first try a value which gives more luck to obtain a solution.
- **lexico :** default domain order.
- **min-conflicts :** value choice is based on total number of conflicts linked to this value (in increasing order).
- **impact :** in increasing order of their instantiation impact.

Look back definition

Look back is a way to back jump when a failure is encountered. Different levels of learning are done when a nogood solution is found.

Look back definition

Look back is a way to back jump when a failure is encountered. Different levels of learning are done when a nogood solution is found.

Levels of Back tracking

- **Standard BackTracking :** when a failure is encountered, a back jump is done, variables domains are restored but nothing is learned about failure.
- **Conflict-directed BackJumping :** use of explanations which give information about why we have failure at current state. A back jump is done with based on the recent explanation which give failure.
- **Dynamic BackTracking :** same as the previous but take care of keeping variables instantiations which do not have any impact of failure. We only change instantiation which is the reason of failure.

Look ahead definition

Look ahead is way of filter variables domains. We saw we have different levels of filtering search space (constraints propagation).

Look ahead definition

Look ahead is way of filter variables domains. We saw we have different levels of filtering search space (constraints propagation).

Levels of propagation

- **Backward checking :** only verify the consistency of the instantiation after a new value affected.
- **Forward Checking :** also called partial look ahead, only neighbors variables of the new variables assigned are checked.
- **Maintained AC :** the constraint propagation is always done at each node of the tree (AC is maintained).

Look ahead definition

Look ahead is way of filtering variables domains. We previously saw we have different levels of filtering search space (constraints propagation).

Remark

Two ways of filtering exist :

- Before searching in the tree (pre-processing filter).
- All during the search in the tree (at each node).

References I

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer (2002). "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2-3, pp. 235–256. DOI: 10.1023/A:1013689704352. URL: https://doi.org/10.1023/A:1013689704352.

Balafrej, Amine, Christian Bessière, and Anastasia Paparrizou (2015). "Multi-Armed Bandits for Adaptive Constraint Propagation". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 290–296. URL: http://ijcai.org/Abstract/15/047.

Boussemart, Frédéric et al. (2004). "Boosting Systematic Search by Weighting Constraints". In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, Valencia, Spain, August 22-27, 2004*, pp. 146–150.

C. Lecoutre (2009). *Constraint Networks: Techniques and Algorithms*.

References II

Lecoutre, Christophe, Stéphane Cardon, and Julien Vion (2007). "Conservative Dual Consistency". In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pp. 237–242. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-036.php.

Lecoutre, Christophe, Lakhdar Sais, et al. (2006). "Last Conflict Based Reasoning". In: *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, pp. 133–137.

P. Norvig, S. Russel & (2017). *Artificial Intelligence, A Modern Approach*.