Using Hard and Soft Rules to Define and Solve Optimization Problems

Barry O'Sullivan[1] Jacob Feldman[1,2]

[1]**Cork Constraint Computation Centre**
Department of Computer Science, University College Cork, Ireland
{b.osullivan|j.feldman}@4c.ucc.ie

[2]**OpenRules, Inc.**
New Jersey, USA
jacobfeldman@openrules.com

International Business Rules Forum
November 2009, Las Vegas, USA

Using Hard and Soft Constraints to Define and Solve Optimization Problems

Barry O'Sullivan[1] Jacob Feldman[1,2]

[1]**Cork Constraint Computation Centre**
Department of Computer Science, University College Cork, Ireland
{b.osullivan|j.feldman}@4c.ucc.ie

[2]**OpenRules, Inc.**
New Jersey, USA
jacobfeldman@openrules.com

International Business Rules Forum
November 2009, Las Vegas, USA

Acknowledgements

Financial Support

Supported by Science Foundation Ireland Grant 05/IN/I886.

science foundation ireland

fondúireacht eolaíochta éireann

Outline

1. Introduction to Constraint Programming
2. Quantitative Approaches
3. Qualitative Approaches
4. Wrap-up

Outline

1. Introduction to Constraint Programming
   - The Constraint Satisfaction Problem
   - Over-constrained CSPs
   - Overcoming Over-Constrainedness
2. Quantitative Approaches
3. Qualitative Approaches
4. Wrap-up

What is a Constraint Satisfaction Problem?

Example

variables and domains
$x_1 \in \{1, 2\}$
$x_2 \in \{0, 1, 2, 3\}$
$x_3 \in \{2, 3\}$

constraints
$x_1 > x_2$
$x_1 + x_2 = x_3$
$alldifferent(x_1, x_2, x_3)$

Solution

By backtrack search and constraint propagation: $x_1 = 2, x_2 = 1, x_3 = 3$

What is a Constraint Satisfaction Problem?

Example

variables and domains

$$x_1 \in \{1, 2\}$$
$$x_2 \in \{0, 1, 2, 3\}$$
$$x_3 \in \{2, 3\}$$

constraints

$$x_1 > x_2$$
$$x_1 + x_2 = x_3$$
$$alldifferent(x_1, x_2, x_3)$$

Solution

By backtrack search and constraint propagation: $x_1 = 2, x_2 = 1, x_3 = 3$

What happens when there are no solutions?

In practice, problems often have no solutions

variables and domains	$x_1 \in \{1, 2\}$ $x_2 \in \{2, 3\}$ $x_3 \in \{2, 3\}$
constraints	$x_1 > x_2$ $x_1 + x_2 = x_3$

Solution

There is no solution. Which is hardly useful in practice.

What happens when there are no solutions?

In practice, problems often have no solutions

variables and domains

$$x_1 \in \{1, 2\}$$
$$x_2 \in \{2, 3\}$$
$$x_3 \in \{2, 3\}$$

constraints

$$x_1 > x_2$$
$$x_1 + x_2 = x_3$$

Solution

There is no solution. Which is hardly useful in practice.

What happens when there are no solutions?

In practice, problems often have no solutions

variables and domains
$$x_1 \in \{1,2\}$$
$$x_2 \in \{2,3\}$$
$$x_3 \in \{2,3\}$$

constraints
$$x_1 > x_2$$
$$x_1 + x_2 = x_3$$

Solution

There is no solution. Which is hardly useful in practice.

Some non-solutions might be regarded as reasonable

x_1	x_2	x_3	comment
1	2	2	all constraints violated
1	2	3	**first constraint violated only (minimum violation)**
1	3	2	all constraints violated
1	3	3	all constraints violated
2	2	2	all constraints violated
2	2	3	all constraints violated
2	3	2	all constraints violated
2	3	3	all constraints violated

Back to the real-world....

This trivial example can be transferred to a real-world problem

A rules-based loan origination system rejects a student request for $30K loan instead of relaxing its hard rules and offering a $29.3K loan to the same student.

From Rules to Constraints

While BR methodologies do not offer a practical solution, we may look at the Constraint Programming (CP) that has an extensive experience in dealing with real-life over-constrained problems

Soft Constraints as Hard Optimisation Constraints [10]

Cost-based approach [8]

- Introduce a cost variable for each soft constraint
- This variable represents some violation measure of the constraint
- Optimize aggregation of all cost variables (e.g., their sum, or max)

In this way:

- Soft global constraints become hard optimization constraints
- The cost variables (z_1 and z_2) can be used in (meta-)constraints, e.g. $(z_1 > 0) \implies (z_2 = 0)$
- **Example:** if a nurse worked extra hours in the evening she cannot work next morning
- We can apply classical constraint programming solvers

Example of a measured constraint violation [10]

Example

- $x \in [9000, 10000]$
- $y \in [0, 20000]$
- $x \leq y$

- Let's make the constraint $x \leq y$ soft by introducing a 'cost' variable $z \in [0, 5]$ that represents the amount of violation, as the gap between x and y.
- Suppose that we impose $z \in [0, 5]$.
- By looking at the bounds of x and y, we can immediately deduce that $y \in [8995, 20000]$.

BR and CP Integration

What are meta-constraints?

CP defines meta-constraints that convert soft constraints to hard optimization constraints

How are they defined?

These meta-constraints are usually defined by subject-matter experts (not programmers!) and thus can be expressed in business rules.

Integration

So, it is a natural to integrate BR and CP in a such way when:

- BR define a problem (or sub-problems)
- CP solves the problem

BR and CP Integration

What are meta-constraints?

CP defines meta-constraints that convert soft constraints to hard optimization constraints

How are they defined?

These meta-constraints are usually defined by subject-matter experts (not programmers!) and thus can be expressed in business rules.

Integration

So, it is a natural to integrate BR and CP in a such way when:

- BR define a problem (or sub-problems)
- CP solves the problem

BR and CP Integration

What are meta-constraints?

CP defines meta-constraints that convert soft constraints to hard optimization constraints

How are they defined?

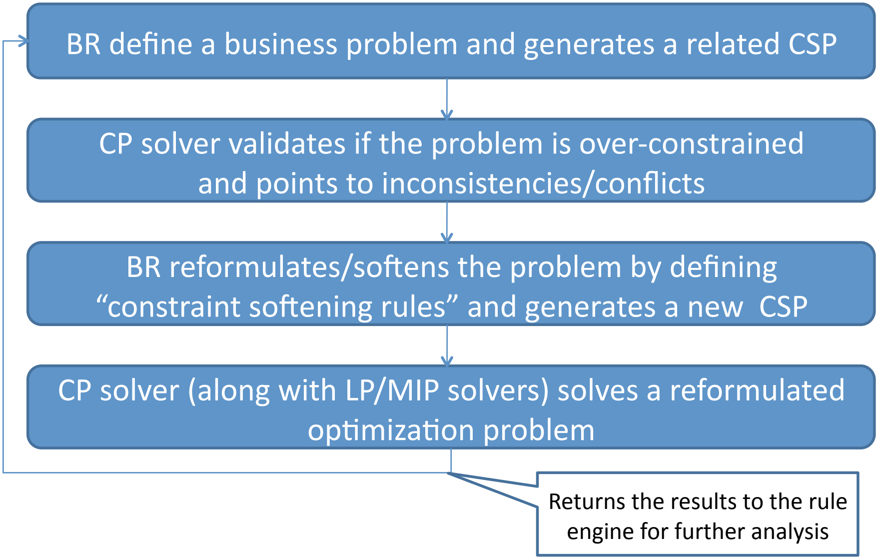
These meta-constraints are usually defined by subject-matter experts (not programmers!) and thus can be expressed in business rules.

Integration

So, it is a natural to integrate BR and CP in a such way when:

- BR define a problem (or sub-problems)
- CP solves the problem

BR and CP Integration



Example "Balancing Financial Portfolio"

Example

The "target" portfolio is defined as a currently active set of rules that directs a shape of every particular portfolio

Rules Violations

Fluctuation of stock prices makes stock allocation rules being almost always "a little bit" violated

Objective

keep portfolio as close as possible to the "target" portfolio

Example: Portfolio Management Rules

Rules void allocationRules(Portfolio portfolio)		
IF Selection Criteria	THEN Set Allocation Percent	
	Min	Max
Financial Sector	18	24
Utilities	19	25
Technology Sector	13	17
Retail Sector	6	
Pharmaceutical Sector	7	15
European except UK		10
Cash	5	

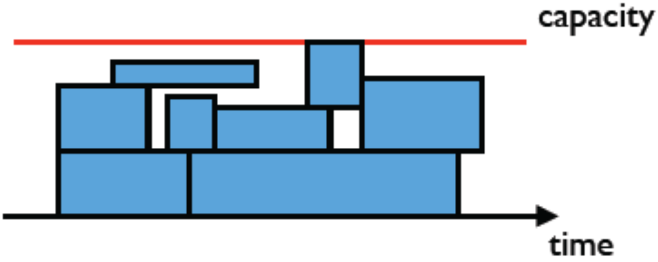
Example: Softening the Rules

Rules void allocationRules(Portfolio portfolio)						
IF Selection Criteria	THEN Set Allocation Percent		Set Rule Properties			
	Min	Max	Hard/Soft	Importance	Maximal Violation	Possible to Exceed
Financial Sector	18	24	Soft	8	1	
Utilities	19	25	Soft	7	0.5	
Technology Sector	13	17	Soft	9	3	Yes
Retail Sector	6		Hard			
Pharmaceutical Sector	7	15	Soft	5	2	
European except UK		10	Soft	6	4	Yes
Cash	5		Hard			

Typical Scheduling Constraints

Example

Given set of activities, each with processing time, resource consumption, earliest start time and latest end time, assign an execution time to each activity so that a given resource does not exceed its capacity



Softening Scheduling Constraints

Violation measures

- Number of late activities
- Acceptable overcapacity of resource
- Use of overtime
- Overuse of skills
- Worker preferences

Real-life soft scheduling constraints (LILCO examples)

- Do not start new job less than x minutes before the end of the shift
- Unavailability tolerance (the same person “CAN” be in two different places at the same time)

Technical Approaches from Constraint Programming

Quantitative strategies

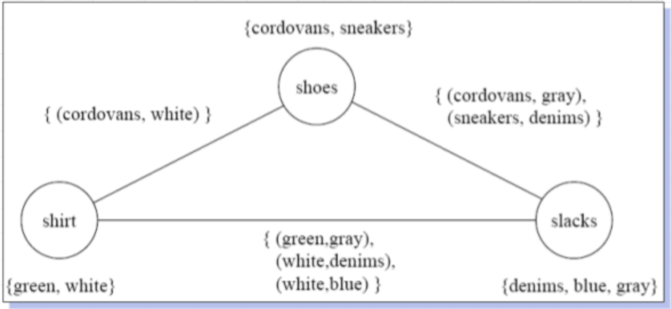
We can define a constraint violation cost and optimize an aggregated function defined on all cost variables.

Qualitative strategies

We can try to find explanations of conflicts or find a preferred relaxation.

Outline

Partial Constraint Satisfaction [3]



Principles of Relaxation

We can relax a problem by:

- Enlarging the domain of a variable
- Enlarging the set of values allowed by a constraint
- Remove a constraint
- Remove a variable

Adding values is enough:

- Add values to a domain
- Add values to a constraint
- Add all possible values to a constraint
- Add all possible values to a domain

Partial Constraint Satisfaction as Optimisation

Partial-order amongst problems

The partial-order defined over the set of problems is defined in terms of the set of solutions to those problems. Specifically,
$P_1 \leq P_2 \equiv sols(P_2) \subseteq sols(P_1)$.

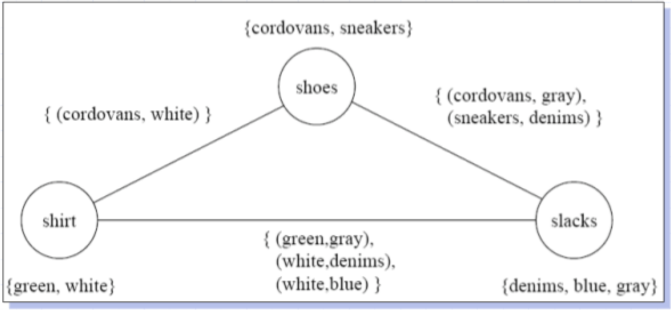
Minimise an Objective Function using Branch-and-Bound

Solution Subset – the number of solutions added.

Augmentation – the number of constraint augmentations.

Max-CSP – the number of constraints satisfied.

Partial Constraint Satisfaction [3]



- Buy a red shirt and augment the constraints so that it compatible with sneakers and denims.
- Solution: $\langle red, sneakers, denims \rangle$
- Metrics:
  - Solution subset distance $= 1$
  - Augmentation distance $= 3$
  - Max-CSP distance $= 1$

Hierarchical CSP [2]

Approach

- We associate a priority with each constraint, and compare solutions using a comparator based on the constraints that are satisfied.
- Find solutions that satisfy the most important constraints.

Example

Hard constraints: Constraint between shirt and slacks.
Strong constraints: Constraint between shoes and slacks.
Weak constraints: Constraint between shirt and shoes.

Solutions

$\langle green, cordovans, gray \rangle$, $\langle white, sneakers, denims \rangle$.

Definition of the Hierarchical CSP

- A constraint hierarchy is a (finite) multiset of constraints labelled with a strength/priority.
- Given a constraint hierarchy $H =_{def} \{H_0, H_1, \ldots, H_k\}$, the set of constraints in H_0 are the hard constraints, and for each other level H_i, its constraints are more important than those at any level $j > i$.
- A solution to a constraint hierarchy H will consist of valuations for variables in H, that satisfy best constraints in H respecting the hierarchy.
- Solutions are compared using a comparator

An Example Comparator

Locally Better

A valuation θ is locally better that another valuation σ if, for each of the constraints through some level $k-1$, the error after applying θ is equal to that after applying σ, and at level k the error is strictly less for at least one constraint and less than or equal for all the rest.

A HCSP Example

Example

Level	Constraints	
H_0	required	$cel \times 1.8 = fah - 32.0$
H_1	strong	$fah = 212$
H_2	weak	$cel = 0$

Solving the problem

$$S(H_0) = \{\ldots, \langle 0, 32\rangle, \langle 10, 50\rangle, \langle 100, 212\rangle, \ldots\}$$

$$S = \{\langle 100, 212\rangle\}$$

The pair $\langle 100, 212\rangle$ is locally-better wrt the other pairs in $S(H_0)$.

Generalised Soft Constraints [1]

- We can define soft constraint problems as $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ where:
- A is the set of all possible 'scores' of our constraints: **0** and **1** are the worst and best 'scores', respectively;
- $+$ compares solutions, and $\times$ combines constraints
- Examples:

⋆ Crisp CSP: $\langle \{false, true\}, \vee, \wedge, false, true \rangle$;
⋆ Fuzzy CSP: $\langle [0, 1], max, min, 0, 1 \rangle$;
⋆ Probabilistic CSP: $\langle [0, 1], max, \times, 0, 1 \rangle$;
⋆ Weighted CSP: $\langle \mathcal{R}, min, +, 0, +\infty \rangle$.

Outline

An industrial example

Example

In November 2003, a configuration client had the problem that constraint propagation in their configurator was failing for a system described by $300,000$ constraints.

How do we debug this?

There are $2^{300,000}$ possible causes, but in our example, only 8 of the constraints were sufficient to produce the failure, but there are still $> 10^{39}$ combinations of possibilities.

After this talk you will know how to ...

Identify these 8 constraints after only 270 consistency checks!

An industrial example

Example

In November 2003, a configuration client had the problem that constraint propagation in their configurator was failing for a system described by $300,000$ constraints.

How do we debug this?

There are $2^{300,000}$ possible causes, but in our example, only 8 of the constraints were sufficient to produce the failure, but there are still $> 10^{39}$ combinations of possibilities.

After this talk you will know how to ...

Identify these 8 constraints after only 270 consistency checks!

An industrial example

Example

In November 2003, a configuration client had the problem that constraint propagation in their configurator was failing for a system described by $300,000$ constraints.

How do we debug this?

There are $2^{300,000}$ possible causes, but in our example, only 8 of the constraints were sufficient to produce the failure, but there are still $> 10^{39}$ combinations of possibilities.

After this talk you will know how to ...

Identify these 8 constraints after only 270 consistency checks!

Where can I apply what I learn?

1. Product Configuration
2. Test Generation
3. Recommender Systems
4. Case-based Reasoning Systems
5. Knowledge-based Systems
6. Software Product Lines
7. Debugging
8. Can you think of any others?

Classic Setting

Two Categories of Constraints

- *background constraints* expressing the connections between the components of the “product”, that cannot be removed
- *user constraints* interactively stated by the user when deciding on options (= a query)

Consistency

- A set of constraints is *consistent* if it admits a solution.
- The background constraints are assumed to be consistent.
- The “solubility” of a set of constraints refers to the number of solutions it is consistent with.

Classic Setting

Two Categories of Constraints

- *background constraints* expressing the connections between the components of the “product”, that cannot be removed
- *user constraints* interactively stated by the user when deciding on options (= a query)

Consistency

- A set of constraints is *consistent* if it admits a solution.
- The background constraints are assumed to be consistent.
- The “solubility” of a set of constraints refers to the number of solutions it is consistent with.

Terminology

Explanations

- **Conflict**: an inconsistent subset of U: show one cause of inconsistency.
- **Relaxation**: a consistent subset of U: show one possible way of recovering from it

Optimality – sort of

- A relaxation is **maximal** when *no constraint can added* while remaining consistent.
- A conflict is **minimal** when *no constraint can be removed* while remaining inconsistent.

Terminology

Explanations

- **Conflict**: an inconsistent subset of U: show one cause of inconsistency.
- **Relaxation**: a consistent subset of U: show one possible way of recovering from it

Optimality – sort of

- A relaxation is **maximal** when *no constraint can added* while remaining consistent.
- A conflict is **minimal** when *no constraint can be removed* while remaining inconsistent.

Example explanation tasks

Configuration as a CSP

- A “product” is fully specified by some constraints
- Several options are available to the user
- The user expresses his preferences as constraints

Explanations

When preferences conflict:

Conflict: show a set of conflicting preferences

Relaxation: show a set of feasible preferences

Example explanation tasks

Configuration as a CSP

- A “product” is fully specified by some constraints
- Several options are available to the user
- The user expresses his preferences as constraints

Explanations

When preferences conflict:

Conflict show a set of conflicting preferences

Relaxation show a set of feasible preferences

Conflicts, Arguments, and Counterarguments (I)

Assumption

The *propagation capability* of a constraints solver can be described by operator Π mapping a set of given constraints to a set of deduced constraints. (e.g. arc consistency deduces constraints of form $x \neq v$)

Conflicts, Arguments, and Counter-arguments (II)

Conflict

For given set of constraints $\mathcal{X}$ + background $\mathcal{B}$:

- **Π-conflict:** subset X of $\mathcal{X}$ such that $\Pi(\mathcal{B} \cup X)$ contains an inconsistency.
- **minimal Π-conflict:** no proper subset is a conflict
- **preferred Π-conflict:** culprits are chosen according to a total order
- **global conflict:** Π is complete (i.e. achieves global consistency)

Arguments and Counter-Arguments

(counter-)argument for ϕ: add $\neg\phi$ (ϕ) to $\mathcal{B}$ + find conflict

Conflicts, Arguments, and Counter-arguments (II)

Conflict

For given set of constraints $\mathcal{X}$ + background $\mathcal{B}$:

- Π**-conflict:** subset X of $\mathcal{X}$ such that $\Pi(\mathcal{B} \cup X)$ contains an inconsistency.
- **minimal** Π**-conflict:** no proper subset is a conflict
- **preferred** Π**-conflict:** culprits are chosen according to a total order
- **global conflict:** Π is complete (i.e. achieves global consistency)

Arguments and Counter-Arguments

(counter-)argument for ϕ: add $\neg\phi$ (ϕ) to $\mathcal{B}$ + find conflict

Which Explanations?

Example

A customer wants station-wagon with options:

1. requirement r_1: roof racks ($\$500$)
2. requirement r_2: CD-player ($\$500$)
3. requirement r_3: extra seat ($\$800$)
4. requirement r_4: metal color ($\$500$)
5. requirement r_5: luxury version ($\$2600$)

Total budget for options is $\$3000$

User requirements cannot be satisfied

Which requirements are in conflict?

Which Explanations?

Example

A customer wants station-wagon with options:

1. requirement r_1: roof racks ($\$500$)
2. requirement r_2: CD-player ($\$500$)
3. requirement r_3: extra seat ($\$800$)
4. requirement r_4: metal color ($\$500$)
5. requirement r_5: luxury version ($\$2600$)

Total budget for options is $\$3000$

User requirements cannot be satisfied

Which requirements are in conflict?

An Arbitary Explanation

Maintain explanations during propagation

r_1	roof racks	$c \geq 500$	$\{r_1\}$
r_2	CD-player	$c \geq 1000$	$\{r_1, r_2\}$
r_3	extra seat	$c \geq 1800$	$\{r_1, r_2, r_3\}$
r_4	metal color	$c \geq 2300$	$\{r_1, r_2, r_3, r_4\}$
r_5	luxury version	$c \geq 4900$	$\{r_1, r_2, r_3, r_4, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_1, r_2, r_3, r_4, r_5, b\}$

explanation: $\{r_1, r_2, r_3, r_4, r_5, b\}$

This explanation is not minimal (irreducible)!

The user may retract constraints unnecessarily.

An Arbitary Explanation

Maintain explanations during propagation

r_1	roof racks	$c \geq 500$	$\{r_1\}$
r_2	CD-player	$c \geq 1000$	$\{r_1, r_2\}$
r_3	extra seat	$c \geq 1800$	$\{r_1, r_2, r_3\}$
r_4	metal color	$c \geq 2300$	$\{r_1, r_2, r_3, r_4\}$
r_5	luxury version	$c \geq 4900$	$\{r_1, r_2, r_3, r_4, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_1, r_2, r_3, r_4, r_5, b\}$

explanation: $\{r_1, r_2, r_3, r_4, r_5, b\}$

This explanation is not minimal (irreducible)!

The user may retract constraints unnecessarily.

An Arbitary Explanation

Maintain explanations during propagation

r_1	roof racks	$c \geq 500$	$\{r_1\}$
r_2	CD-player	$c \geq 1000$	$\{r_1, r_2\}$
r_3	extra seat	$c \geq 1800$	$\{r_1, r_2, r_3\}$
r_4	metal color	$c \geq 2300$	$\{r_1, r_2, r_3, r_4\}$
r_5	luxury version	$c \geq 4900$	$\{r_1, r_2, r_3, r_4, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_1, r_2, r_3, r_4, r_5, b\}$

explanation: $\{r_1, r_2, r_3, r_4, r_5, b\}$

This explanation is not minimal (irreducible)!

The user may retract constraints unnecessarily.

Minimal Explanation

Some other propagation order

r_4	metal color	$c \geq 500$	$\{r_4\}$
r_5	luxury version	$c \geq 3100$	$\{r_4, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_4, r_5, b\}$

explanation: $\{r_4, r_5, b\}$

Minimal - Good!

The **explanation is minimal**, since any proper subset is consistent.

Minimal Explanation

Some other propagation order

r_4	metal color	$c \geq 500$	$\{r_4\}$
r_5	luxury version	$c \geq 3100$	$\{r_4, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_4, r_5, b\}$

explanation: $\{r_4, r_5, b\}$

Minimal - Good!

The **explanation is minimal**, since any proper subset is consistent.

Minimal Explanation

Some other propagation order

r_4	metal color	$c \geq 500$	$\{r_4\}$
r_5	luxury version	$c \geq 3100$	$\{r_4, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_4, r_5, b\}$

explanation: $\{r_4, r_5, b\}$

Minimal - Good!

The **explanation is minimal**, since any proper subset is consistent.

Finding a Minimal Conflict

Example

Step	Activated constraints	Result	Partial conflict
1.	ρ_1	no fail	$\{\}$
2.	ρ_1 ρ_2	no fail	$\{\}$
3.	ρ_1 ρ_2 ρ_3	no fail	$\{\}$
4.	ρ_1 ρ_2 ρ_3 ρ_4	no fail	$\{\}$
5.	ρ_1 ρ_2 ρ_3 ρ_4 ρ_5	fail	$\{\rho_5\}$
6.	ρ_5	no fail	$\{\rho_5\}$
7.	ρ_5 ρ_1	fail	$\{\rho_1, \rho_5\}$

rePlayXplain: Detect culprit and replay

Modified example

Requested options 1,2,3,4,7 cost $100\$$ each; requested options 5,6,8 cost $800\$$ each; budget is 2200.

1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.	19.	20.	21.	22.
R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_8	R_8	R_8	R_8	R_8	R_8	R_8	R_8	R_8	R_8	R_8	R_8	R_8	R_8
	R_2	R_2	R_2	R_2	R_2	R_2	R_2		R_1	R_1	R_1	R_1	R_1	R_1	R_6	R_6	R_6	R_6	R_6	R_6	R_6
		R_3	R_3	R_3	R_3	R_3	R_3			R_2	R_2	R_2	R_2	R_2		R_1	R_1	R_1	R_1	R_1	R_5
			R_4	R_4	R_4	R_4	R_4				R_3	R_3	R_3	R_3			R_2	R_2	R_2	R_2	
				R_5	R_5	R_5	R_5					R_4	R_4	R_4				R_3	R_3	R_3	
					R_6	R_6	R_6						R_5	R_5					R_4	R_4	
						R_7	R_7							R_6						R_5	
							R_8														
							F							**F**						**F**	**F**
							R_8							R_6						R_5	

Add available constraints to CP Solver one after the other;
*when failure (**F**) occurs new culprit is detected;*
backtrack to initial state + add culprit there

QuickXplain: Detect culprit and divide

1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.
R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_8	R_8	R_8
	R_2	R_2	R_2	R_2	R_2	R_2	R_2	R_2	R_2	R_2	R_2	R_2		R_6	R_6
		R_3	R_3	R_3	R_3	R_3	R_3	R_3	R_3	R_3	R_3	R_3			R_5
			R_4	R_4	R_4	R_4	R_4	R_4	R_4	R_4	R_4	R_4			
				R_5	R_5	R_5	R_5	R_8	R_8	R_8	R_8	R_8			
					R_6	R_6	R_6		R_5	R_5	R_6	R_6			
						R_7	R_7			R_6		R_5			
							R_8								
							F			**F**		**F**			**F**
							R_8			R_6		R_5			

Divide conflict detection problem into 2 subproblems when culprit is detected:

1. keep all constraint of first subproblem when solving second subproblem;
2. add culprits of second subproblem when solving first subproblem.

Unnecessary Retractions

Use explanation for finding a solution

1. user submits requirements $r_1, \ldots, r_5 + b$
2. response: failure due to $\{r_4, r_5, b\}$
3. user prefers luxury (r_5) to metal color (r_4), so removes r_4
4. response: failure due to $\{r_3, r_5, b\}$
5. user prefers extra seats (r_3) to luxury (r_5), so removes r_5
6. response: **success**

The retraction of r_4 is no longer justified.

Can we avoid unnecessary retractions?

Unnecessary Retractions

Use explanation for finding a solution

1. user submits requirements $r_1, \ldots, r_5 + b$
2. response: failure due to $\{r_4, r_5, b\}$
3. user prefers luxury (r_5) to metal color (r_4), so removes r_4
4. response: failure due to $\{r_3, r_5, b\}$
5. user prefers extra seats (r_3) to luxury (r_5), so removes r_5
6. response: **success**

The retraction of r_4 is no longer justified.

Can we avoid unnecessary retractions?

Unnecessary Retractions

Use explanation for finding a solution

1. user submits requirements $r_1, \ldots, r_5 + b$
2. response: failure due to $\{r_4, r_5, b\}$
3. user prefers luxury (r_5) to metal color (r_4), so removes r_4
4. response: failure due to $\{r_3, r_5, b\}$
5. user prefers extra seats (r_3) to luxury (r_5), so removes r_5
6. response: **success**

The retraction of r_4 is no longer justified.

Can we avoid unnecessary retractions?

Unnecessary Retractions

Use explanation for finding a solution

1. user submits requirements $r_1, \ldots, r_5 + b$
2. response: failure due to $\{r_4, r_5, b\}$
3. user prefers luxury (r_5) to metal color (r_4), so removes r_4
4. response: failure due to $\{r_3, r_5, b\}$
5. user prefers extra seats (r_3) to luxury (r_5), so removes r_5
6. response: **success**

The retraction of r_4 is no longer justified.

Can we avoid unnecessary retractions?

Unnecessary Retractions

Use explanation for finding a solution

1. user submits requirements $r_1, \ldots, r_5 + b$
2. response: failure due to $\{r_4, r_5, b\}$
3. user prefers luxury (r_5) to metal color (r_4), so removes r_4
4. response: failure due to $\{r_3, r_5, b\}$
5. user prefers extra seats (r_3) to luxury (r_5), so removes r_5
6. response: **success**

The retraction of r_4 is no longer justified.

Can we avoid unnecessary retractions?

Unnecessary Retractions

Use explanation for finding a solution

1. user submits requirements $r_1, \ldots, r_5 + b$
2. response: failure due to $\{r_4, r_5, b\}$
3. user prefers luxury (r_5) to metal color (r_4), so removes r_4
4. response: failure due to $\{r_3, r_5, b\}$
5. user prefers extra seats (r_3) to luxury (r_5), so removes r_5
6. response: **success**

The retraction of r_4 is no longer justified.

Can we avoid unnecessary retractions?

Unnecessary Retractions

Use explanation for finding a solution

1. user submits requirements $r_1, \ldots, r_5 + b$
2. response: failure due to $\{r_4, r_5, b\}$
3. user prefers luxury (r_5) to metal color (r_4), so removes r_4
4. response: failure due to $\{r_3, r_5, b\}$
5. user prefers extra seats (r_3) to luxury (r_5), so removes r_5
6. response: **success**

The retraction of r_4 is no longer justified.

Can we avoid unnecessary retractions?

Preferred Explanation

Again another propagation order

r_3	metal color	$c \geq 800$	$\{r_3\}$
r_5	luxury version	$c \geq 3300$	$\{r_3, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_3, r_5, b\}$

explanation: $\{r_3, r_5, b\}$

Explanation is preferred

Its worst element r_5 can safely be retracted

Preferred Explanation

Again another propagation order

r_3	metal color	$c \geq 800$	$\{r_3\}$
r_5	luxury version	$c \geq 3300$	$\{r_3, r_5\}$
b	total budget	$c \leq 3000$	$\{b\}$
		failure	$\{r_3, r_5, b\}$

explanation: $\{r_3, r_5, b\}$

Explanation is preferred

Its worst element r_5 can safely be retracted

Preferences between Constraints [5]

Intuitive statements with simple semantics

- preferences between constraints

```
prefer(luxury version, metal color)
prefer(extra seat, luxury version)
```

- groups of constraints
  - `equipment` contains requirements for roof racks, extra seat
  - `look` contains requirements for metal color, seat material
- preferences between groups

```
prefer(equipment, look)
```

The Tasks

Overconstrained problem with preferences

- background B
- constraints $C := \{c_1, \ldots, c_n\}$
- preferences P between the c_i's

such that $B \cup C$ is inconsistent

The tasks

- preferred relaxations
- preferred explanations

The Tasks

Overconstrained problem with preferences

- background B
- constraints $C := \{c_1, \ldots, c_n\}$
- preferences P between the c_i's

such that $B \cup C$ is inconsistent

The tasks

- preferred relaxations
- preferred explanations

Intuition behind the Approach

Preferred Conflicts

We use a preference-guided algorithm that successively adds most preferred constraints until they fail. It then backtracks and removes the least preferred constraints if this preserves the failure.

Preferred Relaxations

We remove the least preferred constraints from an inconsistent set until it is consistent.

Duality

Preferred conflicts explain why best elements cannot be added to preferred relaxations.

Intuition behind the Approach

Preferred Conflicts

We use a preference-guided algorithm that successively adds most preferred constraints until they fail. It then backtracks and removes the least preferred constraints if this preserves the failure.

Preferred Relaxations

We remove the least preferred constraints from an inconsistent set until it is consistent.

Duality

Preferred conflicts explain why best elements cannot be added to preferred relaxations.

Intuition behind the Approach

Preferred Conflicts

We use a preference-guided algorithm that successively adds most preferred constraints until they fail. It then backtracks and removes the least preferred constraints if this preserves the failure.

Preferred Relaxations

We remove the least preferred constraints from an inconsistent set until it is consistent.

Duality

Preferred conflicts explain why best elements cannot be added to preferred relaxations.

Algorithm QUICKXPLAIN [4]

Recursive decomposition à la QUICKSORT

1. If B is inconsistent then: $LexXplain(c_{\pi_1}, \ldots, c_{\pi_n})(B) = \emptyset$
2. If B is consistent and C is a singleton then:
   $LexXplain(c_{\pi_1}, \ldots, c_{\pi_n})(B) = C$
3. If B is consistent and C has more than one element then split at k
   1. let $C_k := \{c_{\pi_1}, \ldots, c_{\pi_k}\}$
   2. let E_2 be $LexXplain(c_{\pi_{k+1}}, \ldots, c_{\pi_n})(B \cup C_k)$
   3. let E_1 be $LexXplain(c_{\pi_1}, \ldots, c_{\pi_k})(B \cup E_2)$
   4. $LexXplain(c_{\pi_1}, \ldots, c_{\pi_n})(B) = E_1 \cup E_2$

Where to Split?

Effect

If a subproblem does not contain an element of the conflict then it can be solved by a single consistency check, namely $B \cup C_k$ or $B \cup E_2$

Strategy

Choose subproblems of same size to exploit this effect in a best way

#Consistency Checks

Between $\log_2 \frac{n}{k} + 2k$ and $2k \cdot \log_2 \frac{n}{k} + 2k$ (for conflicts of size k)

Consistency Checking

The cost of consistency checking

QUICKXPLAIN does multiple consistency checks that are NP-hard in general, but

- complexity is polynomial for tree-like CSPs
- approximations possible: trade time and optimality
- keep witnesses for success (= solution) and try them when adding constraints
- keep witnesses for failure (= critical search decisions) and try them when removing constraints

Compilation helps in practice

Most problems in practice give small compiled forms.

Consistency Checking

The cost of consistency checking

QUICKXPLAIN does multiple consistency checks that are NP-hard in general, but

- complexity is polynomial for tree-like CSPs
- approximations possible: trade time and optimality
- keep witnesses for success (= solution) and try them when adding constraints
- keep witnesses for failure (= critical search decisions) and try them when removing constraints

Compilation helps in practice

Most problems in practice give small compiled forms.

How to use QuickXplain

- **Background:** effort is reduced by putting as many constraints as possible in the initial background
- **Preference order**: order of constraint uniquely characterizes the conflict found
- **Consistency checker**: time can be traded against minimality by an incomplete consistency checker, giving “anytime” behaviour

How to use QuickXplain

- **Background:** effort is reduced by putting as many constraints as possible in the initial background
- **Preference order**: order of constraint uniquely characterizes the conflict found
- **Consistency checker**: time can be traded against minimality by an incomplete consistency checker, giving “anytime” behaviour

How to use QuickXplain

- **Background:** effort is reduced by putting as many constraints as possible in the initial background
- **Preference order**: order of constraint uniquely characterizes the conflict found
- **Consistency checker**: time can be traded against minimality by an incomplete consistency checker, giving “anytime” behaviour

Applications of QuickXplain

- Configuration: B2B, B2C find conflicts between user requests.
- Constraint model debugging isolate failing parts of the constraint model.
- Rule verification find tests that make a rule never applicable.
- Benders decomposition.
- Diagnosis of ontologies.

Applications of QuickXplain

- Configuration: B2B, B2C find conflicts between user requests.
- Constraint model debugging isolate failing parts of the constraint model.
- Rule verification find tests that make a rule never applicable.
- Benders decomposition.
- Diagnosis of ontologies.

Applications of QuickXplain

- Configuration: B2B, B2C find conflicts between user requests.
- Constraint model debugging isolate failing parts of the constraint model.
- Rule verification find tests that make a rule never applicable.
- Benders decomposition.
- Diagnosis of ontologies.

Applications of QuickXplain

- Configuration: B2B, B2C find conflicts between user requests.
- Constraint model debugging isolate failing parts of the constraint model.
- Rule verification find tests that make a rule never applicable.
- Benders decomposition.
- Diagnosis of ontologies.

Applications of QuickXplain

- Configuration: B2B, B2C find conflicts between user requests.
- Constraint model debugging isolate failing parts of the constraint model.
- Rule verification find tests that make a rule never applicable.
- Benders decomposition.
- Diagnosis of ontologies.

Outline

1. Introduction to Constraint Programming
2. Quantitative Approaches
3. Qualitative Approaches
4. Wrap-up

Take-Home Messages

Integration

Close integration between business rules and constraint programming techniques is straightforward and meaningful.

Reasoning about Soft Constraints

There is a large body of work and software tools for reasoning about soft constraints in a variety of quantitative and qualitative settings.

Perspectives

We can view the integration as a basis for optimisation, but also as a basis for explanation generation.

Using Hard and Soft Rules to Define and Solve Optimization Problems

Barry O'Sullivan[1] Jacob Feldman[1,2]

[1]**Cork Constraint Computation Centre**
Department of Computer Science, University College Cork, Ireland
{b.osullivan|j.feldman}@4c.ucc.ie

[2]**OpenRules, Inc.**
New Jersey, USA
jacobfeldman@openrules.com

International Business Rules Forum
November 2009, Las Vegas, USA

Stefano Bistarelli, Ugo Montanari, and Francesca Rossi.
Semiring-based constraint satisfaction and optimization.
J. ACM, 44(2):201–236, 1997.

Alan Borning, Bjørn N. Freeman-Benson, and Molly Wilson.
Constraint hierarchies.
Lisp and Symbolic Computation, 5(3):223–270, 1992.

Eugene C. Freuder and Richard J. Wallace.
Partial constraint satisfaction.
Artif. Intell., 58(1-3):21–70, 1992.

Ulrich Junker.
Quickxplain: Preferred explanations and relaxations for over-constrained problems.
In *AAAI*, pages 167–172, 2004.

Ulrich Junker and Daniel Mailharro.
Preference programming: Advanced problem solving for configuration.
AI EDAM, 17(1):13–29, 2003.

David Lesaint, Deepak Mehta, Barry O'Sullivan, Luis Quesada, and Nic Wilson.
Personalisation of telecommunications services as combinatorial optimisation.
In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 1693–1698. AAAI Press, 2008.

David Lesaint, Deepak Mehta, Barry O'Sullivan, Luis Quesada, and Nic Wilson.
Solving a telecommunications feature subscription configuration problem.
In Stuckey [9], pages 67–81.

Thierry Petit, Jean-Charles Régin, and Christian Bessière.
Meta-constraints on violations for over constrained problems.
In *ICTAI*, pages 358–365. IEEE Computer Society, 2000.

Peter J. Stuckey, editor.
Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September

14-18, 2008. Proceedings, volume 5202 of *Lecture Notes in Computer Science*. Springer, 2008.

Willem van Hoeve.
Soft global constraints, 2009.