Search, propagation, and learning in sequencing and scheduling problems

Mohamed Siala

Christian Artigues	LAAS-CNRS Toulouse
Fahiem Bacchus	University of Toronto
Christian Bessiere	LIRMM Montpellier
Hadrien Cambazard	G-SCOP & Grenoble INP
Emmanuel Hebrard	LAAS-CNRS Toulouse
George Katsirelos	INRA Toulouse
Christine Solnon	INSA Lyon

Context

Sequencing and Scheduling: the organization in time of of operations subject to capacity and resource constraints.

Mon 5	Tue 6	Wed 7	Thu 8	Fri 9	Sat 10	Sun 11	Mon 12	Tue 13	Wed 14	Thu 15	Fri 16	Sat 17	Sun 18
CD	EIA		EIA	LIK	NER		CD	EIA		EIA	LIK	NER	
LIK	LD		SS	EIF		NOO	LIK	SSN		SS	EIF		NOO
	CD	EIA	EIB	EIA	LIK	CD	EIA		EIA	LIK	NER		
	LIK	SSN		Conference L	EIF	LIK	SSN		Conference L	EIF		NOO	
CD	EIA												
LIK	EIA		EIA			NOO	CD	LIK		EIA	LIK	NER	
CD	EIA	LIK	EIA	LIK	NER		LIK		LIK	On Call	EIF		NOO
LIK	SSN		SS	EIF		NOO		CD					
		CD					CD	LIK	CD	EIA	LIK	LIK	
CD	EIA	LIK	EIA	LIK	NER		LIK	CD	LIK	Conference L	EIF		NOO
LIK	SSN	CD	Conference L	EIF			CD	EIA	CD	EIA	LIK	CD	
CD	EIA		EIA	LIK	NER		LIK	SSN		FL	EIF	LIK	NOO
LIK	LU		MLH									CD	
CD	EIA		EIA		NER		CD	EIA		EIA	LIK		
LIK	SSN		EIA				LIK	SSN		Conference L	EIF		NOO
	CD	EIA		EIA	LIK								
	LIK	SSN		Conference L	EIF		NOO						
CD	EIA		EIA	LIK	NER		CD	EIA		EIA	LIK	NER	
LIK	SSN		EIA	EIF		NOO	LIK	SSN		MLM	EIF		NOO
CD	EIA		EIG	LIK	NER		CD	EIA		EIA	LIK	NER	
LIK	LD		MLM	EIF		NOO	LIK	SSN		SST	EIF		NOO

PhD Context

- Combinatorial (optimization) problems
- Constraint satisfaction and optimization

- Laboratory: LAAS-CNRS, Toulouse
- Research Team: ROC
- Supervision: Christian Artigues, and Emmanuel Hebrard
- Funding:

Thesis overview

Constraint Programming: Search $\oplus$ Propagation

Thesis overview

Constraint Programming: Search $\oplus$ Propagation $\oplus$ **Learning**

Thesis overview

Constraint Programming: Search $\oplus$ Propagation $\oplus$ **Learning**

All these aspects are important and must all be taken into account in order to design efficient solution methods

Outline

Definition

A constraint is a finite relation

Definition

A constraint is a finite relation

Definition

A `constraint network` (CN) is defined by a triplet $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where

- $\mathcal{X} = [x_1, \ldots, x_n]$: finite set of variables
- $\mathcal{D}$: a domain for $\mathcal{X}$
- $\mathcal{C}$: finite set of constraints

Definition

A constraint is a finite relation

Definition

A `constraint network` (CN) is defined by a triplet $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where

- $\mathcal{X} = [x_1, \ldots, x_n]$: finite set of variables
- $\mathcal{D}$: a domain for $\mathcal{X}$
- $\mathcal{C}$: finite set of constraints

- Constraint Satisfaction Problem (CSP): deciding whether a constraint network has a solution or not
- CSP is NP-Hard in general

Definition

A constraint is a finite relation

Definition

A `constraint network` (`CN`) is defined by a triplet $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where

- $\mathcal{X} = [x_1, \ldots, x_n]$: finite set of variables
- $\mathcal{D}$: a domain for $\mathcal{X}$
- $\mathcal{C}$: finite set of constraints

- Constraint Satisfaction Problem (`CSP`): deciding whether a constraint network has a solution or not
- CSP is NP-Hard in general

- Complete backtracking algorithms

Search

Search

- Search: decisions to explore the search tree

Search

- Search: decisions to explore the search tree
- Search in $\mathrm{CP}=$ variable ordering $+$ value ordering

Search

- Search: decisions to explore the search tree
- Search in CP= variable ordering + value ordering
- Standard or customized

Search

- Search: decisions to explore the search tree
- Search in $\mathrm{CP}=$ variable ordering $+$ value ordering
- Standard or customized

Variable Ordering

'Fail-first' principle [Haralick and Elliott, 1980]:

"To succeed, try first where you are most likely to fail"

Search

- Search: decisions to explore the search tree
- Search in CP= variable ordering + value ordering
- Standard or customized

Variable Ordering

'Fail-first' principle [Haralick and Elliott, 1980]:

"To succeed, try first where you are most likely to fail"

Value Ordering

'succeed-first' [Geelen, 1992]:

Best chances leading to a solution

Propagation

Propagation

- Propagation: inferences based on the current state
- Constraint $\leftrightarrow$ a propagator
- Propagators are executed sequentially before taking any decision
- The level of pruning $\leftrightarrow$ local consistency

Propagation

- Propagation: inferences based on the current state
- Constraint $\leftrightarrow$ a propagator
- Propagators are executed sequentially before taking any decision
- The level of pruning $\leftrightarrow$ local consistency

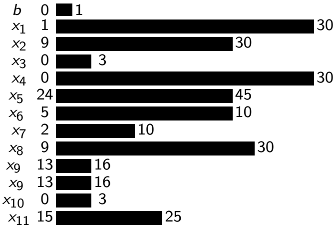
Arc Consistency

- Let $\mathcal{D}$ be a domain, and C be a constraint
- C is Arc Consistent (AC) iff for every x in the scope of C, for every value $v \in \mathcal{D}(x)$ there exists an assignment w in $\mathcal{D}$ satisfying C in which v is assigned to x

Learning

Learning

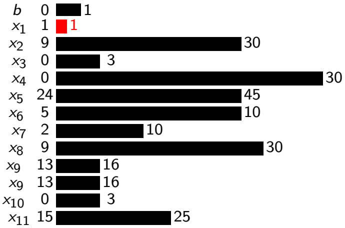
$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$



Learning

$$\begin{aligned}
&x_1 + x_7 \geq 4 \land \\
&x_2 + x_{10} \geq 11 \land \\
&x_3 + x_9 = 16 \land \\
&x_5 \geq x_8 + x_9 \land \\
&b \leftrightarrow (x_9 - x_4 = 14) \land \\
&b \rightarrow (x_6 \geq 7) \land \\
&b \rightarrow (x_6 + x_7 \leq 9) \land \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}$$

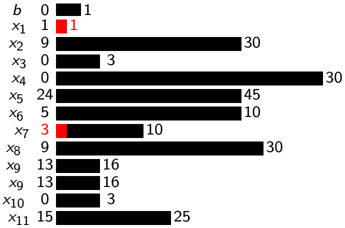
$[\![x_1 = 1]\!]$



Learning

$$x_1 + x_7 \geq 4 \wedge$$
$$x_2 + x_{10} \geq 11 \wedge$$
$$x_3 + x_9 = 16 \wedge$$
$$x_5 \geq x_8 + x_9 \wedge$$
$$b \leftrightarrow (x_9 - x_4 = 14) \wedge$$
$$b \rightarrow (x_6 \geq 7) \wedge$$
$$b \rightarrow (x_6 + x_7 \leq 9) \wedge$$
$$x_{11} \geq x_9 + x_{10}$$

$$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$$

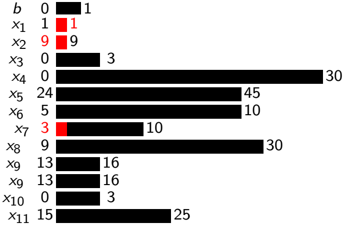


Learning

$$
\begin{aligned}
&x_1 + x_7 \geq 4 \wedge \\
&x_2 + x_{10} \geq 11 \wedge \\
&x_3 + x_9 = 16 \wedge \\
&x_5 \geq x_8 + x_9 \wedge \\
&b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
&b \rightarrow (x_6 \geq 7) \wedge \\
&b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}
$$

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!]$

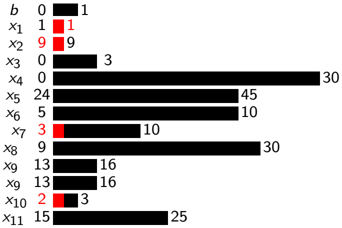


Learning

$$x_1 + x_7 \geq 4 \wedge$$
$$x_2 + x_{10} \geq 11 \wedge$$
$$x_3 + x_9 = 16 \wedge$$
$$x_5 \geq x_8 + x_9 \wedge$$
$$b \leftrightarrow (x_9 - x_4 = 14) \wedge$$
$$b \rightarrow (x_6 \geq 7) \wedge$$
$$b \rightarrow (x_6 + x_7 \leq 9) \wedge$$
$$x_{11} \geq x_9 + x_{10}$$

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \longrightarrow [\![x_{10} \geq 2]\!]$



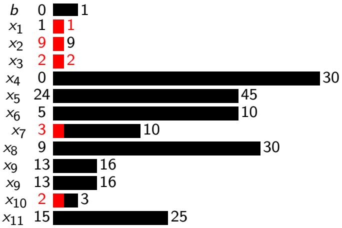
Learning

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \longrightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!]$

$$
\begin{aligned}
& x_1 + x_7 \geq 4 \wedge \\
& x_2 + x_{10} \geq 11 \wedge \\
& x_3 + x_9 = 16 \wedge \\
& x_5 \geq x_8 + x_9 \wedge \\
& b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
& b \rightarrow (x_6 \geq 7) \wedge \\
& b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



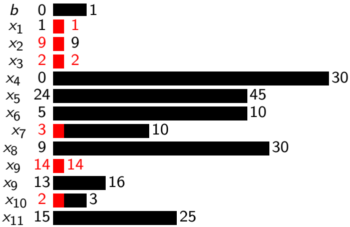
Learning

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \longrightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!] \longrightarrow [\![x_9 = 14]\!]$

$$
\begin{array}{l}
x_1 + x_7 \geq 4 \wedge \\
x_2 + x_{10} \geq 11 \wedge \\
x_3 + x_9 = 16 \wedge \\
x_5 \geq x_8 + x_9 \wedge \\
b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
b \rightarrow (x_6 \geq 7) \wedge \\
b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
x_{11} \geq x_9 + x_{10}
\end{array}
$$



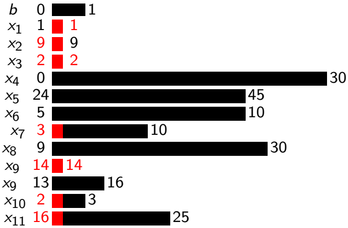
Learning

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \longrightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!] \longrightarrow [\![x_9 = 14]\!] \rightarrow [\![x_{11} \geq 16]\!]$

$$
\begin{aligned}
&x_1 + x_7 \geq 4 \wedge \\
&x_2 + x_{10} \geq 11 \wedge \\
&x_3 + x_9 = 16 \wedge \\
&x_5 \geq x_8 + x_9 \wedge \\
&b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
&b \rightarrow (x_6 \geq 7) \wedge \\
&b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



Learning

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$

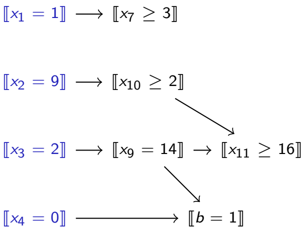
$[\![x_2 = 9]\!] \longrightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!] \longrightarrow [\![x_9 = 14]\!] \rightarrow [\![x_{11} \geq 16]\!]$

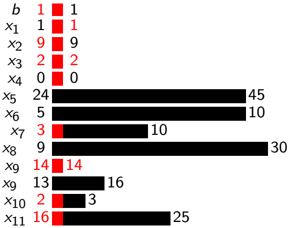
$[\![x_4 = 0]\!]$

$$
\begin{aligned}
& x_1 + x_7 \geq 4 \wedge \\
& x_2 + x_{10} \geq 11 \wedge \\
& x_3 + x_9 = 16 \wedge \\
& x_5 \geq x_8 + x_9 \wedge \\
& b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
& b \rightarrow (x_6 \geq 7) \wedge \\
& b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$

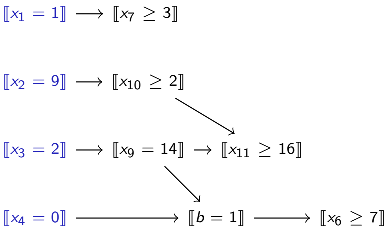
Learning



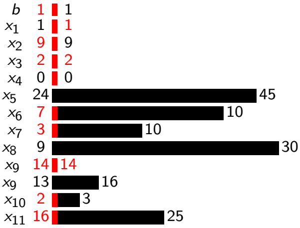
$$
\begin{aligned}
& x_1 + x_7 \geq 4 \wedge \\
& x_2 + x_{10} \geq 11 \wedge \\
& x_3 + x_9 = 16 \wedge \\
& x_5 \geq x_8 + x_9 \wedge \\
& b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
& b \rightarrow (x_6 \geq 7) \wedge \\
& b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



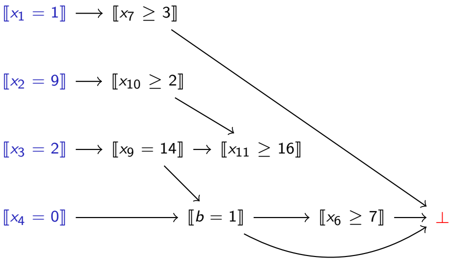
Learning



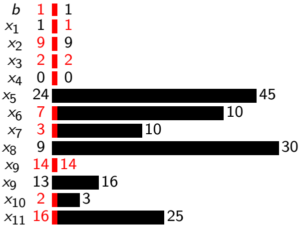
$$
\begin{aligned}
&x_1 + x_7 \geq 4 \wedge \\
&x_2 + x_{10} \geq 11 \wedge \\
&x_3 + x_9 = 16 \wedge \\
&x_5 \geq x_8 + x_9 \wedge \\
&b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
&b \rightarrow (x_6 \geq 7) \wedge \\
&b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



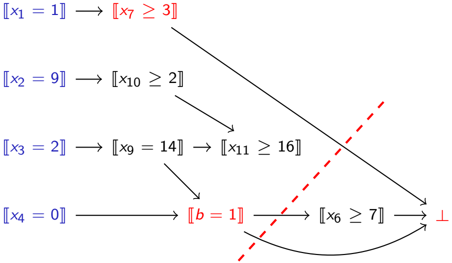
Learning



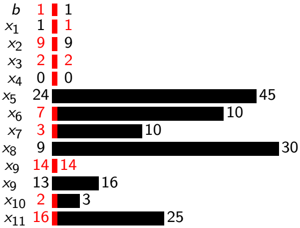
$$
\begin{aligned}
&x_1 + x_7 \geq 4 \wedge \\
&x_2 + x_{10} \geq 11 \wedge \\
&x_3 + x_9 = 16 \wedge \\
&x_5 \geq x_8 + x_9 \wedge \\
&b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
&b \rightarrow (x_6 \geq 7) \wedge \\
&b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



Learning

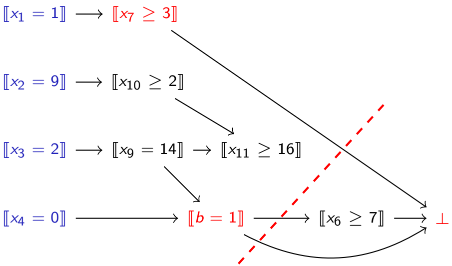


$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$

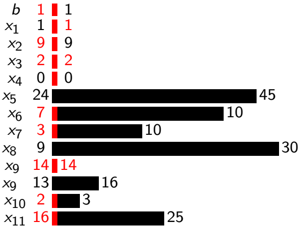


- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \bot$

Learning



$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$



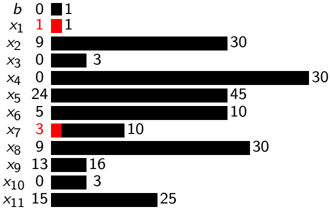
- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \bot$
- New clause: $[\![b \neq 0]\!] \vee [\![x_7 \leq 2]\!]$

Learning

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!]$

$$x_1 + x_7 \geq 4 \wedge$$
$$x_2 + x_{10} \geq 11 \wedge$$
$$x_3 + x_9 = 16 \wedge$$
$$x_5 \geq x_8 + x_9 \wedge$$
$$b \leftrightarrow (x_9 - x_4 = 14) \wedge$$
$$b \rightarrow (x_6 \geq 7) \wedge$$
$$b \rightarrow (x_6 + x_7 \leq 9) \wedge$$
$$x_{11} \geq x_9 + x_{10}$$

- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \bot$
- New clause: $[\![b \neq 0]\!] \vee [\![x_7 \leq 2]\!]$
- Backtrack to level 1

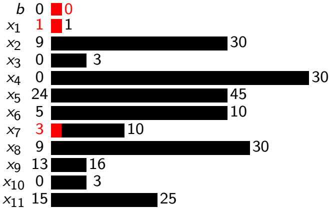


Learning

$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!] \longrightarrow [\![b = 0]\!]$

- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \bot$
- New clause: $[\![b \neq 0]\!] \vee [\![x_7 \leq 2]\!]$
- Backtrack to level 1
- Propagate the learnt clause

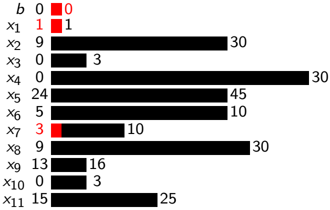


Learning

$[\![x_1 = 1]\!] \longrightarrow [\![x_7 \geq 3]\!] \longrightarrow [\![b = 0]\!]$

- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \bot$
- New clause: $[\![b \neq 0]\!] \vee [\![x_7 \leq 2]\!]$
- Backtrack to level 1
- Propagate the learnt clause
- Continue exploration

$$
\begin{array}{l}
x_1 + x_7 \geq 4 \wedge \\
x_2 + x_{10} \geq 11 \wedge \\
x_3 + x_9 = 16 \wedge \\
x_5 \geq x_8 + x_9 \wedge \\
b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
b \rightarrow (x_6 \geq 7) \wedge \\
b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
x_{11} \geq x_9 + x_{10}
\end{array}
$$



Learning in CP

- Hybrid CP/SAT
- Based on the notion of explanation
- Conflict Driven Clause Learning (CDCL) [Moskewicz et al., 2001]

Summary of the thesis

Summary of the thesis

Modern CP-Solvers may not underestimate any of the three aspects: search, propagation, and learning

Summary of the thesis

Modern CP-Solvers may not underestimate any of the three aspects: search, propagation, and learning

Contributions

- Search in car-sequencing

Summary of the thesis

Modern CP-Solvers may not underestimate any of the three aspects: search, propagation, and learning

Contributions

- Search in car-sequencing
- Propagation in a class of sequencing problems

Summary of the thesis

Modern CP-Solvers may not underestimate any of the three aspects: search, propagation, and learning

Contributions

- Search in car-sequencing
- Propagation in a class of sequencing problems
- Learning in car-sequencing

Summary of the thesis

Modern CP-Solvers may not underestimate any of the three aspects: search, propagation, and learning

Contributions

- Search in car-sequencing
- Propagation in a class of sequencing problems
- Learning in car-sequencing
- Revisiting lazy generation

Summary of the thesis

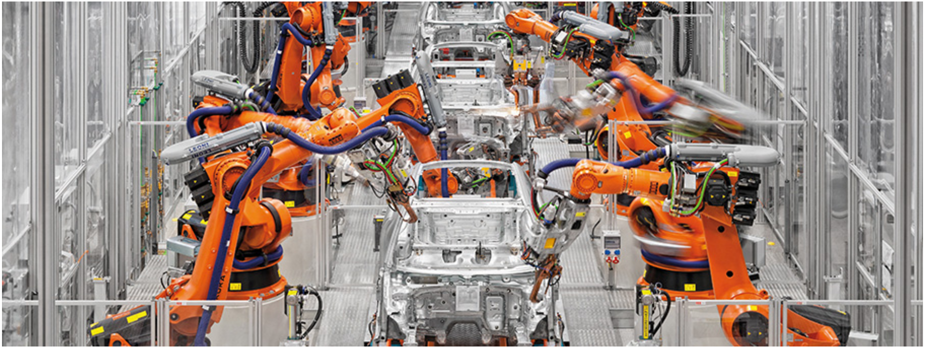
Modern CP-Solvers may not underestimate any of the three aspects: search, propagation, and learning

Contributions

- Search in car-sequencing
- Propagation in a class of sequencing problems
- Learning in car-sequencing
- Revisiting lazy generation
- Learning in disjunctive scheduling

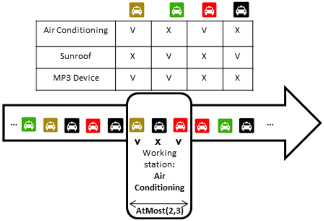
Outline

Car-Sequencing

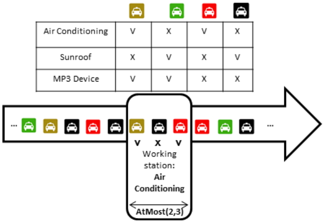


- ROADEF'05 challenge [Solnon et al., 2008]
- RENAULT

Problem Definition

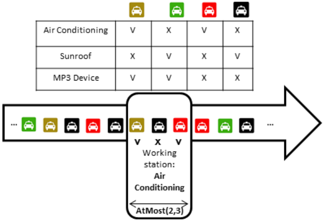


Problem Definition



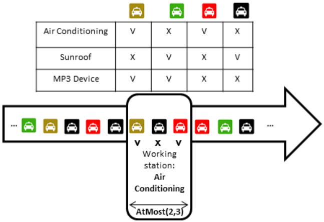
- A class of vehicles is defined by a set of options

Problem Definition



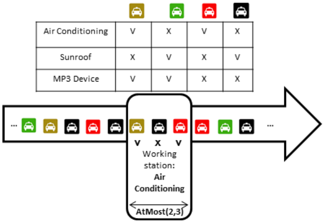
- A class of vehicles is defined by a set of options
- Each class is associated to a demand

Problem Definition



- A class of vehicles is defined by a set of options
- Each class is associated to a demand
- `Capacity` constraints: no subsequence of size q may contain more than p vehicles requiring a given option

Problem Definition



- A class of vehicles is defined by a set of options
- Each class is associated to a demand
- `Capacity` constraints: no subsequence of size q may contain more than p vehicles requiring a given option
- Is there a sequence of cars satisfying both demand and capacity constraints?

Outline

Propagation via AtMostSeqCard

Propagation via AtMostSeqCard

Definition

$\textsc{AtMostSeqCard}(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} (\sum_{l=1}^{q} x_{i+l} \leq p) \wedge (\sum_{i=1}^{n} x_i = d)$$

Propagation via AtMostSeqCard

Definition

$\textsc{AtMostSeqCard}(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq p\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example $\textsc{AtMostSeqCard}(2, 5, 4, [x_1, \ldots, x_9])$

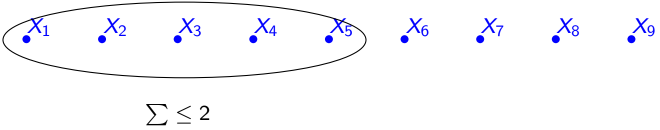
Propagation via AtMostSeqCard

Definition

AtMostSeqCard$(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq p\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example AtMostSeqCard$(2, 5, 4, [x_1, \ldots, x_9])$



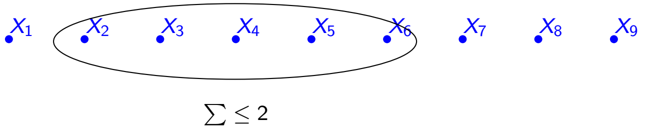
Propagation via AtMostSeqCard

Definition

$\text{AtMostSeqCard}(p, q, d, [x_1, \dots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq p\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example $\text{AtMostSeqCard}(2, 5, 4, [x_1, \dots, x_9])$



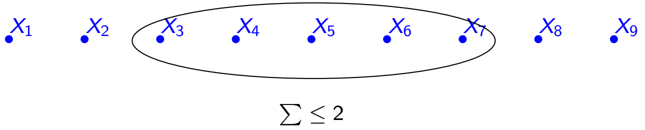
Propagation via AtMostSeqCard

Definition

AtMostSeqCard$(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq p\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example AtMostSeqCard$(2, 5, 4, [x_1, \ldots, x_9])$



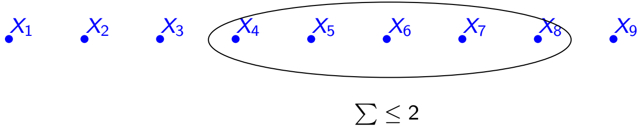
Propagation via AtMostSeqCard

Definition

$\textsc{AtMostSeqCard}(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq p\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example $\textsc{AtMostSeqCard}(2, 5, 4, [x_1, \ldots, x_9])$



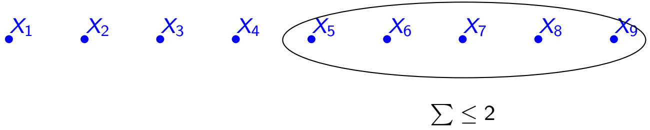
Propagation via AtMostSeqCard

Definition

AtMostSeqCard$(p, q, d, [x_1, \dots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \le p\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example AtMostSeqCard$(2, 5, 4, [x_1, \dots, x_9])$



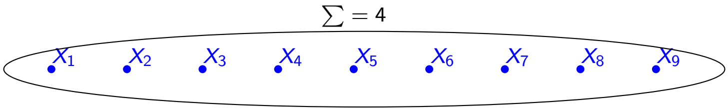
Propagation via AtMostSeqCard

Definition

AtMostSeqCard$(p, q, d, [x_1, \dots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \le p \right) \wedge \left(\sum_{i=1}^{n} x_i = d \right)$$

Example AtMostSeqCard$(2, 5, 4, [x_1, \dots, x_9])$



Propagation via AtMostSeqCard

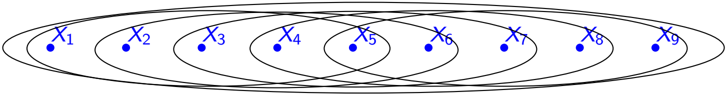
Definition

AtMostSeqCard$(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq p\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example AtMostSeqCard$(2, 5, 4, [x_1, \ldots, x_9])$

$\sum = 4$



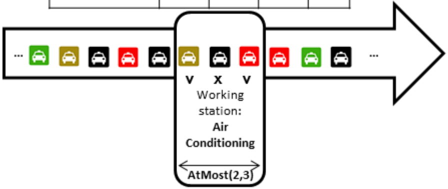
$\sum \leq 2$ $\sum \leq 2$ $\sum \leq 2$ $\sum \leq 2$ $\sum \leq 2$

AtMostSeqCard as a global constraint?

AtMostSeqCard as a global constraint?

1. Car sequencing

Air Conditioning	V	X	V	X
Sunroof	X	V	X	V
MP3 Device	V	V	X	X

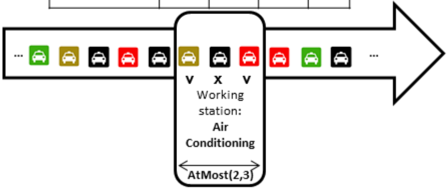


- One AtMostSeqCard per option
- Capacity constraints ⊕ demand constraints

ATMOSTSEQCARD as a global constraint?

1. Car sequencing

Air Conditioning	V	X	V	X
Sunroof	X	V	X	V
MP3 Device	V	V	X	X



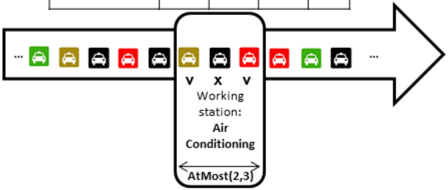
- One ATMOSTSEQCARD per option
- Capacity constraints $\oplus$ demand constraints

2. But also useful in crew-rostering

AtMostSeqCard as a global constraint?

1. Car sequencing

Air Conditioning	V	X	V	X
Sunroof	X	V	X	V
MP3 Device	V	V	X	X



- One AtMostSeqCard per option
- Capacity constraints $\oplus$ demand constraints

2. But also useful in crew-rostering

Arc Consistency

$$\textsc{AtMostSeqCard}(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$$

$$\textsc{AtMostSeq}(p, q, [x_1, \ldots, x_n]) \wedge \textsc{Cardinality}(d, [x_1, \ldots, x_n])$$

Arc Consistency

$\textsc{AtMostSeqCard}(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$\textsc{AtMostSeq}(p, q, [x_1, \ldots, x_n]) \wedge \textsc{Cardinality}(d, [x_1, \ldots, x_n])$

- $\textsc{AtMostSeq} \oplus \textsc{Cardinality}$ is not enough

Arc Consistency

$\textsc{AtMostSeqCard}(p, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$\textsc{AtMostSeq}(p, q, [x_1, \ldots, x_n]) \wedge \textsc{Cardinality}(d, [x_1, \ldots, x_n])$

- $\textsc{AtMostSeq} \oplus \textsc{Cardinality}$ is not enough

$\textsc{AtMostSeqCard}$ as a particular case?

- $\textsc{cost-Regular}$: $O(2^q n)$ [van Hoeve et al., 2009]
- $\textsc{Gen-Sequence}$: $O(n^3)$ [van Hoeve et al., 2009]
- $\textsc{Gen-Sequence}$: $O(n^2.log(n))$ down a branch $\oplus$ initial compilation of $O(q.n^2)$. [Maher et al., 2008].

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \ldots, x_{22}])$

. 0 . . . . 0 1 0 1

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \ldots, x_{22}])$

. 0 . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \dots, x_{22}])$

. 0 . . . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

. 0 . . . . . . 0 1 0 1

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \ldots, x_{22}])$

. 0 . . . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

. 0 . . . . . . 0 1 0 1

1 0 **1** **1** **1** 0 0 0 0 1 0

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \dots, x_{22}])$

. 0 . . . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

. 0 . . . . . . 0 1 0 ■ 1

1 0 **1** **1** **1** 0 0 0 0 1 0 max added= 4

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \dots, x_{22}])$

. 0 . . . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

. 0 . . . . . . 0 1 0 ▮ 1

1 0 **1** **1** **1** 0 0 0 0 1 0 max added= 4

1 **1** 0 0 0 0 **1** **1** **1** 1

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \ldots, x_{22}])$

. 0 . . . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

. 0 . . . . . . 0 1 0 █ 1

1 0 **1** **1** **1** 0 0 0 0 1 0 max added= 4

max added 5 **1** **1** 0 0 0 0 **1** **1** **1** 1

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \ldots, x_{22}])$

. 0 . . . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

. 0 . . . . . . 0 1 0 █ 1

1 0 **1** **1** **1** 0 0 0 0 1 0 max added= 4

max added 5 **1** **1** 0 0 0 0 **1** **1** **1** 1

Maximum possible 9 < residual demand

AC on ATMOSTSEQCARD

Key idea

- Enforce AC on ATMOSTSEQ and CARDINALITY
- Complete the filtering based on a greedy rule

An example with ATMOSTSEQCARD$(4, 8, 12, [x_1, \ldots, x_{22}])$

. 0 . . . . . . 0 1 0 1

ATMOSTSEQ and CARDINALITY are AC

. 0 . . . . . . 0 1 0 ■ 1

1 0 **1** **1** **1** 0 0 0 0 1 0 max added= 4

max added 5 **1** **1** 0 0 0 0 **1** **1** **1** 1

Maximum possible 9 < residual demand

. 0 . . . . . . 0 1 0 1 1

Achieving Arc consistency

- `leftmost`: a greedy rule computing an assignment w of maximum cardinality with respect to AtMostSeq.
- `leftmost_count`: a linear implementation returning for each i the maximum cardinality that can be added until i
- L: `leftmost_count` from left to right
- R: `leftmost_count` from right to left

Achieving Arc consistency

- `leftmost`: a greedy rule computing an assignment w of maximum cardinality with respect to ATMOSTSEQ.
- `leftmost_count`: a linear implementation returning for each i the maximum cardinality that can be added until i
- L: `leftmost_count` from left to right
- R: `leftmost_count` from right to left

Achieving AC in linear time

1. AC on ATMOSTSEQ and CARDINALITY
2. - If $L[n] < d_{res}$: failure
   - If $L[n] = d_{res}$, then $\forall i$:
     - If $L[i] + R[n-i+1] \leq d_{res}$, then x_i is assigned to 0.
     - If $L[i-1] + R[n-i] < d_{res}$, then x_i is assigned to 1.

Example

$\textsc{AtMostSeqCard}(4, 8, 12)$

. 0 . . . . . . 0 1 0 1

Example

AtMostSeqCard$(4, 8, 12)$

	.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
$\overrightarrow{w}[i]$	**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1

Example

$\textsc{AtMostSeqCard}(4, 8, 12)$

		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1
$L[i]$	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10

Example

AtMostSeqCard$(4, 8, 12)$

		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1
$L[i]$	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1

Example

AtMostSeqCard(4, 8, 12)

		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$R[i]$		10	9	9	9	8	7	6	6	6	6	6	6	5	4	3	3	3	3	3	2	1	0	0

Example

$\textsc{AtMostSeqCard}(4, 8, 12)$

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$R[i]$		10	9	9	9	8	7	6	6	6	6	6	6	5	4	3	3	3	3	3	2	1	0	0
$L[i] + R[n-i+1]$		11	10	11	12	12	11	10	10	10	10	10	11	11	11	10	10	10	11	11	11	11	10	
$L[i-1] + R[n-i]$		9	10	10	10	10	10	10	10	10	10	10	9	9	9	10	10	10	10	10	9	9	10	

Example

$\textsc{AtMostSeqCard}(4, 8, 12)$

		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$R[i]$		10	9	9	9	8	7	6	6	6	6	6	6	5	4	3	3	3	3	3	2	1	0	0
$L[i] + R[n-i+1]$		11	10	11	12	12	11	10	10	10	10	10	11	11	11	10	10	10	11	11	11	11	10	
$L[i-1] + R[n-i]$		9	10	10	10	10	10	10	10	10	10	10	9	9	9	10	10	10	10	10	9	9	10	
AC		**1**	0	.	.	.	.	**0**	**0**	0	1	0	**1**	**1**	**1**	**0**	**0**	**0**	.	.	**1**	**1**	1	

Experimental Results

Experimental Results

Variables

- Class variables: n integer variables $\{x_1, \ldots, x_n\}$
- Option variables: nm Boolean variables $\{y_1^1, \ldots, y_n^m\}$

Experimental Results

Variables

- Class variables: n integer variables $\{x_1, \ldots, x_n\}$
- Option variables: nm Boolean variables $\{y_1^1, \ldots, y_n^m\}$

Constraints

1. *Demand constraints*: $\forall c \in \{1..k\}, \left|\{i \mid x_i = c\}\right| = d_c^{class}$: Global Cardinality Constraint.
2. *Capacity constraints*:
   1. A naive decomposition: DECOMPOSITION
   2. Global Sequencing Constraint: GSC [Régin and Puget, 1997]
   3. ATMOSTSEQCARD: AMSC
   4. Combine ATMOSTSEQCARD and GSC: GSC⊕AMSC
3. *Channeling*: between option and class variables

Experimental results: Car-Sequencing

Experimental results: Car-Sequencing

	set1 (70 × 42 × 5)				set2 (4 × 42 × 5)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
DECOMPOSITION	8480	231.2K	25.0M	13.93	95	1.4M	15.3M	76.60
GSC	11218	1.7K	2.3M	3.60	325	131.7K	1.5M	110.99
ATMOSTSEQCARD	10702	39.1K	13.8M	4.43	360	690.8K	10.2M	72.00
GSC⊕AMSC	11243	1.2K	1.1M	3.43	339	118.4K	1.0M	106.53

	set3 (5 × 42 × 5)				set4 (7 × 42 × 5)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
DECOMPOSITION	0	-	-	$>$ 1200	64	543.3K	13.7M	43.81
GSC	31	55.3K	218.5K	276.06	140	25.2K	321.9K	56.61
ATMOSTSEQCARD	16	40.3K	83.4K	8.62	153	201.4K	3.2M	33.56
GSC⊕AMSC	32	57.7K	244.7K	285.43	147	23.8K	371.0K	66.45

Experimental results: Car-Sequencing

	set1 (70 × 42 × 5)				set2 (4 × 42 × 5)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
DECOMPOSITION	8480	231.2K	25.0M	13.93	95	1.4M	15.3M	76.60
GSC	11218	1.7K	2.3M	3.60	325	131.7K	1.5M	110.99
ATMOSTSEQCARD	10702	39.1K	13.8M	4.43	360	690.8K	10.2M	72.00
GSC⊕AMSC	11243	1.2K	1.1M	3.43	339	118.4K	1.0M	106.53

	set3 (5 × 42 × 5)				set4 (7 × 42 × 5)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
DECOMPOSITION	0	-	-	> 1200	64	543.3K	13.7M	43.81
GSC	31	55.3K	218.5K	276.06	140	25.2K	321.9K	56.61
ATMOSTSEQCARD	16	40.3K	83.4K	8.62	153	201.4K	3.2M	33.56
GSC⊕AMSC	32	57.7K	244.7K	285.43	147	23.8K	371.0K	66.45

- Best Models: ATMOSTSEQCARD and ATMOSTSEQCARD ⊕ GSC
- GSC saves more backtracks than ATMOSTSEQCARD but extremely slow

Experimental results: Car-Sequencing

	set1 (70 × 42 × 5)				set2 (4 × 42 × 5)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
DECOMPOSITION	8480	231.2K	25.0M	13.93	95	1.4M	15.3M	76.60
GSC	11218	1.7K	2.3M	3.60	325	131.7K	1.5M	110.99
ATMOSTSEQCARD	10702	39.1K	13.8M	4.43	360	690.8K	10.2M	72.00
GSC⊕AMSC	11243	1.2K	1.1M	3.43	339	118.4K	1.0M	106.53

	set3 (5 × 42 × 5)				set4 (7 × 42 × 5)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
DECOMPOSITION	0	-	-	> 1200	64	543.3K	13.7M	43.81
GSC	31	55.3K	218.5K	276.06	140	25.2K	321.9K	56.61
ATMOSTSEQCARD	16	40.3K	83.4K	8.62	153	201.4K	3.2M	33.56
GSC⊕AMSC	32	57.7K	244.7K	285.43	147	23.8K	371.0K	66.45

- Best Models: ATMOSTSEQCARD and ATMOSTSEQCARD ⊕ GSC
- GSC saves more backtracks than ATMOSTSEQCARD but extremely slow
- [van Hoeve et al., 2009] 65.2% while GSC⊕AMSC 96.20%

Extensions for AtMostSeqCard

Extensions for AtMostSeqCard

$$\textsc{MultiAtMostSeqCard}(p_1, .., p_m, q_1, .., q_m, d, [x_1, \dots, x_n]) \Leftrightarrow$$

$$\bigwedge_{k=1}^{m} \bigwedge_{i=0}^{n-q_k} (\sum_{l=1}^{q_k} x_{i+l} \leq p_k) \wedge (\sum_{i=1}^{n} x_i = d)$$

Extensions for AtMostSeqCard

$$\text{MultiAtMostSeqCard}(p_1, .., p_m, q_1, .., q_m, d, [x_1, \dots, x_n]) \Leftrightarrow$$

$$\bigwedge_{k=1}^{m} \bigwedge_{i=0}^{n-q_k} \left(\sum_{l=1}^{q_k} x_{i+l} \leq p_k\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

- The decomposition into m AtMostSeqCard is hindering propagation

Extensions for AtMostSeqCard

$\textsc{MultiAtMostSeqCard}(p_1, .., p_m, q_1, .., q_m, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{k=1}^{m} \bigwedge_{i=0}^{n-q_k} (\sum_{l=1}^{q_k} x_{i+l} \leq p_k) \wedge (\sum_{i=1}^{n} x_i = d)$$

- The decomposition into m AtMostSeqCard is hindering propagation
- The filtering for AtMostSeqCard can be adapted to achieve AC in $O(m \times n)$

Extensions for AtMostSeqCard

$$\textsc{MultiAtMostSeqCard}(p_1, .., p_m, q_1, .., q_m, d, [x_1, \ldots, x_n]) \Leftrightarrow$$

$$\bigwedge_{k=1}^{m} \bigwedge_{i=0}^{n-q_k} (\sum_{l=1}^{q_k} x_{i+l} \leq p_k) \wedge (\sum_{i=1}^{n} x_i = d)$$

- The decomposition into m AtMostSeqCard is hindering propagation
- The filtering for AtMostSeqCard can be adapted to achieve AC in $O(m \times n)$
- MultiAtMostSeqCard outperforms the other models in crew-rostering

Publications

- [Honorable mention] *An optimal arc consistency algorithm for a chain of atmost constraints with cardinality*
  Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012*
- *An optimal arc consistency algorithm for a particular case of sequence constraint*
  Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. *Constraints*, 19(1):30–56, 2014

Outline

Related work regarding the search strategy

Related work regarding the search strategy

- [Smith, 1996]: lex exploration, branching on class variables, evaluation based on: *max option*, q/p, usage rate $\frac{d.q/p}{n}$.

Related work regarding the search strategy

- [Smith, 1996]: lex exploration, branching on class variables, evaluation based on: *max option*, q/p, usage rate $\frac{d.q/p}{n}$.
- [Régin and Puget, 1997]: middle to sides exploration, branching on option variables, evaluation based on the slack.

Related work regarding the search strategy

- [Smith, 1996]: lex exploration, branching on class variables, evaluation based on: *max option*, q/p, usage rate $\frac{d.q/p}{n}$.
- [Régin and Puget, 1997]: middle to sides exploration, branching on option variables, evaluation based on the slack.
- [Gottlieb et al., 2003]: static vs. dynamic, two ways for aggregating the evaluation (lex,sum)

Related work regarding the search strategy

- [Smith, 1996]: lex exploration, branching on class variables, evaluation based on: *max option*, q/p, usage rate $\frac{d.q/p}{n}$.
- [Régin and Puget, 1997]: middle to sides exploration, branching on option variables, evaluation based on the slack.
- [Gottlieb et al., 2003]: static vs. dynamic, two ways for aggregating the evaluation (lex,sum)

Motivation

Can we combine these heuristics in one structure?

New Classification

New Classification

- `Branching`: *class*, *option*.

New Classification

- `Branching`: *class*, *option*.
- `Exploration`: *lex*, *middle*.

New Classification

- `Branching`: *class*, *option*.
- `Exploration`: *lex*, *middle*.
- `Selection`:
  1. capacity p_j/q_j
  2. demand d_j^{opt}
  3. load $\delta_j = d_j^{opt}.\frac{q_j}{p_j}$
  4. slack $\sigma_j = n - (n_j - \delta_j)$
  5. usage rate $\rho_j = \delta_j/n_j$

New Classification

- Branching: *class*, *option*.
- Exploration: *lex*, *middle*.
- Selection:
  1. capacity p_j/q_j
  2. demand d_j^{opt}
  3. load $\delta_j = d_j^{opt}.\frac{q_j}{p_j}$
  4. slack $\sigma_j = n - (n_j - \delta_j)$
  5. usage rate $\rho_j = \delta_j/n_j$
- Aggregation: $\leq_{\sum}, \leq_{Euc}, \leq_{lex}$.

New Classification

- Branching: *class*, *option*.
- Exploration: *lex*, *middle*.
- Selection:
  1. capacity p_j/q_j
  2. demand d_j^{opt}
  3. load $\delta_j = d_j^{opt} . \frac{q_j}{p_j}$
  4. slack $\sigma_j = n - (n_j - \delta_j)$
  5. usage rate $\rho_j = \delta_j / n_j$
- Aggregation: $\leq_{\sum}, \leq_{Euc}, \leq_{lex}$.

Overall 42 heuristics

$\langle\{class, option\}, \{lex, middle\}, \{q/p, d^{opt}, \delta, n{-}\sigma, \rho, 1\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$

Experiments

Experiments

- What is the best configuration?
- What are the important criteria?

Experiments

- What is the best configuration?
- What are the important criteria?

Summary

- Many good heuristics raise as untested combinations

Experiments

- What is the best configuration?
- What are the important criteria?

Summary

- Many good heuristics raise as untested combinations
- Branching and Selection are the most crucial criteria

Experiments

- What is the best configuration?
- What are the important criteria?

Summary

- Many good heuristics raise as untested combinations
- Branching and Selection are the most crucial criteria
- The most robust heuristics:
  $\langle \mathit{class}, \{\mathit{lex}, \mathit{middle}\}, \delta, \{\leq_{\sum}, \leq_{\mathit{Euc}}, \leq_{\mathit{lex}}\}\rangle$

Experiments

- What is the best configuration?
- What are the important criteria?

Summary

- Many good heuristics raise as untested combinations
- Branching and Selection are the most crucial criteria
- The most robust heuristics:
  $\langle class, \{lex, middle\}, \delta, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$
- Search is as important as propagation based on two metrics *confidence* and *significance*

Publication

A study of constraint programming heuristics for the car-sequencing problem.
Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. *Engineering Applications of Artificial Intelligence*, 38(0):34 – 44, 2015

Outline

1 Context

2 Background

3 Case Study: The Car-Sequencing Problem
- Propagation
- Search
- Learning

4 Learning in Disjunctive Scheduling

5 Conclusions & Perspectives

Hybrid CP/SAT Models

- Models based on AtMostSeqCard

Hybrid CP/SAT Models

- Models based on AtMostSeqCard
- We have to explain AtMostSeqCard

Hybrid CP/SAT Models

- Models based on AtMostSeqCard
- We have to explain AtMostSeqCard

Explaining AtMostSeqCard?

- Explain AtMostSeq and Cardinality
- Explaining the extra filtering: consider the naive explanation, then try to reduce it.

Explaining failure: key idea

- `leftmost`: a greedy rule computing an assignment w of maximum cardinality with respect to AtMostSeq.
- max: a vector containing for each i the maximum cardinality in w of all subsequences involving i

Explaining failure: key idea

- `leftmost`: a greedy rule computing an assignment w of maximum cardinality with respect to AtMostSeq.
- *max*: a vector containing for each i the maximum cardinality in w of all subsequences involving i

Observations

Let S: 1 1 0 0 . subject to AtMost(2/5)

→`leftmost` on S gives 1 1 0 0 0

Consider the sequence S_0: 1 1 . 0 .

→`leftmost` on S_0 gives 1 1 0 0 0

Explaining failure: key idea

- `leftmost`: a greedy rule computing an assignment w of maximum cardinality with respect to AtMostSeq.
- max: a vector containing for each i the maximum cardinality in w of all subsequences involving i

Observations

Let S: 1 1 0 0 . subject to AtMost(2/5)

$\rightarrow$`leftmost` on S gives 1 1 0 0 0

Consider the sequence S_0: 1 1 . 0 .

$\rightarrow$`leftmost` on S_0 gives 1 1 0 0 0

Always true when $\{[\![x_i = 0]\!] \mid max(i) = p\}$

Explaining failure: key idea

- `leftmost`: a greedy rule computing an assignment w of maximum cardinality with respect to AtMostSeq.
- max: a vector containing for each i the maximum cardinality in w of all subsequences involving i

Observations

Let S: 1 1 0 0 . subject to AtMost(2/5)
→`leftmost` on S gives 1 1 0 0 0
Consider the sequence S_0: 1 1 . 0 .
→`leftmost` on S_0 gives 1 1 0 0 0

Always true when $\{[\![x_i = 0]\!] \mid max(i) = p\}$

Consider the sequence S_2: . 1 0 0 .
`leftmost` on S_2 gives 1 1 0 0 0

Explaining failure: key idea

- `leftmost`: a greedy rule computing an assignment w of maximum cardinality with respect to AtMostSeq.
- *max*: a vector containing for each i the maximum cardinality in w of all subsequences involving i

Observations

Let S: 1 1 0 0 . subject to AtMost(2/5)

→`leftmost` on S gives 1 1 0 0 0

Consider the sequence S_0: 1 1 . 0 .

→`leftmost` on S_0 gives 1 1 0 0 0

Always true when $\{[\![x_i = 0]\!] \mid max(i) = p\}$

Consider the sequence S_2: . 1 0 0 .

`leftmost` on S_2 gives 1 1 0 0 0

Always true when $\{[\![x_i = 1]\!] \mid max(i) \neq p\}$

Reduced Explanations

Reduced Explanations

A weaker domain $\widehat{\mathcal{D}}$ defined as follows:

$$
\begin{array}{ll}
\widehat{\mathcal{D}}(x_i) = \{0,1\} & \text{if } \mathcal{D}(x_i) = \{0\} \wedge max(i) = p \\
\widehat{\mathcal{D}}(x_i) = \{0,1\} & \text{if } \mathcal{D}(x_i) = \{1\} \wedge max(i) \neq p \\
\widehat{\mathcal{D}}(x_i) = \mathcal{D}(x_i) & \text{otherwise}
\end{array}
$$

Reduced Explanations

A weaker domain $\widehat{\mathcal{D}}$ defined as follows:

$$\begin{array}{ll} \widehat{\mathcal{D}}(x_i) = \{0,1\} & \text{if } \mathcal{D}(x_i) = \{0\} \wedge max(i) = p \\ \widehat{\mathcal{D}}(x_i) = \{0,1\} & \text{if } \mathcal{D}(x_i) = \{1\} \wedge max(i) \neq p \\ \widehat{\mathcal{D}}(x_i) = \mathcal{D}(x_i) & \text{otherwise} \end{array}$$

Theorem

If a failure is raised because $L[n] < d_{res}$, then the set of assignments in $\widehat{\mathcal{D}}$ is a valid nogood.

Reduced Explanations

A weaker domain $\widehat{\mathcal{D}}$ defined as follows:

$$\begin{array}{ll} \widehat{\mathcal{D}}(x_i) = \{0,1\} & \text{if } \mathcal{D}(x_i) = \{0\} \wedge max(i) = p \\ \widehat{\mathcal{D}}(x_i) = \{0,1\} & \text{if } \mathcal{D}(x_i) = \{1\} \wedge max(i) \neq p \\ \widehat{\mathcal{D}}(x_i) = \mathcal{D}(x_i) & \text{otherwise} \end{array}$$

Theorem

If a failure is raised because $L[n] < d_{res}$ *, then the set of assignments in* $\widehat{\mathcal{D}}$ *is a valid nogood.*

Time Complexity

$O(n)$ since we call `leftmost_count` once to built *max*

Example

$\mathcal{D}$: 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 . . . 1

Example

$\mathcal{D}$: 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 . . . 1

Example

$\mathcal{D}$:	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering → Failure

Example

$\mathcal{D}$:	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

max	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	2	2

Example

$\mathcal{D}$:	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

max	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	2	2
$\widehat{\mathcal{D}}$:	1	1	.	.	.	.	.	.	1	1	.	.	.	.	0	0	0	0	.	0	0	.	.	.	1

Example

$\mathcal{D}$:	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering → Failure

max	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	2	2
$\widehat{\mathcal{D}}$:	1	1	.	.	.	.	.	.	1	1	.	.	.	.	0	0	0	0	.	0	0	.	.	.	1

Example

$\mathcal{D}$:	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

max	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	2	2
$\widehat{\mathcal{D}}$:	1	1	.	.	.	.	.	.	1	1	.	.	.	.	0	0	0	0	.	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

Example

$\mathcal{D}$:	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

max	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	2	2
$\widehat{\mathcal{D}}$:	1	1	.	.	.	.	.	.	1	1	.	.	.	.	0	0	0	0	.	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

Size: 22 with naive explanation and 11 with reduced explanation

Example

$\mathcal{D}$:	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

max	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	2	2
$\widehat{\mathcal{D}}$:	1	1	.	.	.	.	.	.	1	1	.	.	.	.	0	0	0	0	.	0	0	.	.	.	1
w	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1

Extra filtering $\rightarrow$ Failure

Size: 22 with naive explanation and 11 with reduced explanation

Note: not minimal

Experimental Results

Method	sat[easy] (74 × 5)			sat[hard] (7 × 5)			unsat* (28 × 5)		
	#suc	avg fails	time	#suc	avg fails	time	#suc	avg fails	time
CNF_A	370	2073	1.71	28	337194	282.35	**85**	**249301**	**105.07**
CNF_S	370	1114	0.87	31	60956	56.49	65	220658	197.03
CNF_{A+S}	370	612	0.91	34	32711	36.52	77	190915	128.09
hybrid (VSIDS)	370	903	0.23	16	207211	286.32	35	177806	224.78
hybrid (VSIDS/Slot)	370	739	0.23	35	76256	64.52	37	204858	248.24
hybrid (Slot/VSIDS)	370	132	0.04	34	4568	2.50	37	234800	287.61
hybrid (Slot)	370	132	0.04	**35**	**6304**	**3.75**	23	174097	299.24
CP	**370**	**43**	**0.03**	35	57966	16.25	0	-	-
PBO-clauses	277	538743	236.94	0	-	-	43	175990	106.92
PBO-cutting planes	272	2149	52.62	0	-	-	1	5031	53.38

Experimental Results

Method	sat [easy] (74 × 5)			sat [hard] (7 × 5)			unsat* (28 × 5)		
	#suc	avg fails	time	#suc	avg fails	time	#suc	avg fails	time
CNF_A	370	2073	1.71	28	337194	282.35	**85**	**249301**	**105.07**
CNF_S	370	1114	0.87	31	60956	56.49	65	220658	197.03
CNF_{A+S}	370	612	0.91	34	32711	36.52	77	190915	128.09
hybrid (VSIDS)	370	903	0.23	16	207211	286.32	35	177806	224.78
hybrid (VSIDS/Slot)	370	739	0.23	35	76256	64.52	37	204858	248.24
hybrid (Slot/VSIDS)	370	132	0.04	34	4568	2.50	37	234800	287.61
hybrid (Slot)	370	132	0.04	**35**	**6304**	**3.75**	23	174097	299.24
CP	**370**	**43**	**0.03**	35	57966	16.25	0	-	-
PBO-clauses	277	538743	236.94	0	-	-	43	175990	106.92
PBO-cutting planes	272	2149	52.62	0	-	-	1	5031	53.38

- Finding solutions quickly: Propagation is very important to find solutions quickly when they exist.
- For proving unsatisfiability: Clause learning is by far the most critical factor.

Publication

SAT and Hybrid Models of the Car-Sequencing problem
Christian Artigues, Emmanuel Hebrard, Valentin Mayer-Eichberger, Mohamed Siala, and Toby Walsh. In *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014*

Outline

Context

Disjunctive Scheduling

A family of scheduling problems having in common the `Unary Resource Constraint`.

Context

Disjunctive Scheduling

A family of scheduling problems having in common the `Unary Resource Constraint`.

Unary Resource Constraint [Grimes and Hebrard, 2015]

Context

Disjunctive Scheduling

A family of scheduling problems having in common the `Unary Resource Constraint`.

Unary Resource Constraint [Grimes and Hebrard, 2015]

- Decomposition using the following DISJUNCTIVE constraints:

$$\delta_{kij} = \begin{cases} 0 & \Leftrightarrow \quad t_{ik} + p_{ik} \leq t_{jk} \\ 1 & \Leftrightarrow \quad t_{jk} + p_{jk} \leq t_{ik} \end{cases} \tag{1}$$

Context

Disjunctive Scheduling

A family of scheduling problems having in common the `Unary Resource Constraint`.

Unary Resource Constraint [Grimes and Hebrard, 2015]

- Decomposition using the following DISJUNCTIVE constraints:

$$\delta_{kij} = \begin{cases} 0 & \Leftrightarrow & t_{ik} + p_{ik} \leq t_{jk} \\ 1 & \Leftrightarrow & t_{jk} + p_{jk} \leq t_{ik} \end{cases} \tag{1}$$

Our Contributions

- Alternative lazy generation approach
- Novel conflict analysis scheme

Revisiting Lazy Generation

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$
$[\![x \leq 57]\!]$	$\emptyset$

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$
$[\![x \leq 57]\!]$	$\emptyset$
$[\![x \leq 317]\!]$	$[\![x \geq 57]\!] \vee [\![x \leq 317]\!]$

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$
$[\![x \leq 57]\!]$	$\emptyset$
$[\![x \leq 317]\!]$	$[\![x \geq 57]\!] \vee [\![x \leq 317]\!]$
$[\![x \leq 203]\!]$	$([\![x \geq 203]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 203]\!])$

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$
$[\![x \leq 57]\!]$	$\emptyset$
$[\![x \leq 317]\!]$	$[\![x \geq 57]\!] \vee [\![x \leq 317]\!]$
$[\![x \leq 203]\!]$	$([\![x \geq 203]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 203]\!])$
$[\![x \leq 84]\!]$	$([\![x \geq 84]\!] \vee [\![x \leq 203]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 84]\!])$

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$
$[\![x \leq 57]\!]$	$\emptyset$
$[\![x \leq 317]\!]$	$[\![x \geq 57]\!] \vee [\![x \leq 317]\!]$
$[\![x \leq 203]\!]$	$([\![x \geq 203]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 203]\!])$
$[\![x \leq 84]\!]$	$([\![x \geq 84]\!] \vee [\![x \leq 203]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 84]\!])$
$[\![x \leq 250]\!]$	$([\![x \geq 250]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 203]\!] \vee [\![x \leq 250]\!])$

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$
$[\![x \leq 57]\!]$	$\emptyset$
$[\![x \leq 317]\!]$	$[\![x \geq 57]\!] \vee [\![x \leq 317]\!]$
$[\![x \leq 203]\!]$	$([\![x \geq 203]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 203]\!])$
$[\![x \leq 84]\!]$	$([\![x \geq 84]\!] \vee [\![x \leq 203]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 84]\!])$
$[\![x \leq 250]\!]$	$([\![x \geq 250]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 203]\!] \vee [\![x \leq 250]\!])$

Revisiting Lazy Generation

Standard Lazy Encoding

- Generate atoms lazily when learning new clauses.
- Generate related domain clauses.
- There is a redundancy issue

Example

Atoms	clauses
$\emptyset$	$\emptyset$
$[\![x \leq 57]\!]$	$\emptyset$
$[\![x \leq 317]\!]$	$[\![x \geq 57]\!] \vee [\![x \leq 317]\!]$
$[\![x \leq 203]\!]$	$([\![x \geq 203]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 203]\!])$
$[\![x \leq 84]\!]$	$([\![x \geq 84]\!] \vee [\![x \leq 203]\!]) \wedge ([\![x \geq 57]\!] \vee [\![x \leq 84]\!])$
$[\![x \leq 250]\!]$	$([\![x \geq 250]\!] \vee [\![x \leq 317]\!]) \wedge ([\![x \geq 203]\!] \vee [\![x \leq 250]\!])$

$O(k)$ redundant clauses

Avoiding the redundancy via DomainFaithfulness

Avoiding the redundancy via DomainFaithfulness

Key Idea

- Use a single constraint responsible for the consistency of the domain.
- Whenever an atom is generated, we update the internal structure of the constraint

Avoiding the redundancy via DomainFaithfulness

Key Idea

- Use a single constraint responsible for the consistency of the domain.
- Whenever an atom is generated, we update the internal structure of the constraint

Definition

$\textsc{DomainFaithfulness}(x, [b_1 \ldots b_n], [v_1, \ldots, v_n]) : \forall i, b_i \leftrightarrow x \leq v_i$

Avoiding the redundancy via DomainFaithfulness

Key Idea

- Use a single constraint responsible for the consistency of the domain.
- Whenever an atom is generated, we update the internal structure of the constraint

Definition

$\textsc{DomainFaithfulness}(x, [b_1 \dots b_n], [v_1, \dots, v_n]) : \forall i, b_i \leftrightarrow x \leq v_i$

Avoiding the redundancy via DomainFaithfulness

Key Idea

- Use a single constraint responsible for the consistency of the domain.
- Whenever an atom is generated, we update the internal structure of the constraint

Definition

$\textsc{DomainFaithfulness}(x, [b_1 \dots b_n], [v_1, \dots, v_n]) : \forall i, b_i \leftrightarrow x \leq v_i$

Arc consistency

Can be enforced in constant amortized time complexity ($O(1)$) down a branch of the search tree

DISJUNCTIVE-based Learning

DISJUNCTIVE-based Learning

- Branch on the reified Boolean variables
- $\rightarrow$ There exists an explanation for every bound literal $[\![x \leq u]\!]$

DISJUNCTIVE-based Learning

- Branch on the reified Boolean variables
- $\rightarrow$ There exists an explanation for every bound literal $[\![x \leq u]\!]$

DISJUNCTIVE-based Learning

Two phases:

1. First UIP cut with a reified Boolean variable
2. Apply resolution for every bound literal until having a nogood with only reified Boolean variables

Disjunctive-based Learning

- Branch on the reified Boolean variables
- $\rightarrow$ There exists an explanation for every bound literal $[\![x \leq u]\!]$

Disjunctive-based Learning

Two phases:

1. First UIP cut with a reified Boolean variable
2. Apply resolution for every bound literal until having a nogood with only reified Boolean variables

- $\oplus$ No domain encoding
- $\oplus$ Scheduling horizon does not manner in size
- $\ominus$ Language of literals is restricted compared to standard approaches

Experimental results

Protocol

- **Mistral-Hybrid:** new hybrid solver with
  - backward explanation
  - semantic reductions
  - lazy generation
  - DISJUNCTIVE-based learning
- **https://github.com/siala/Hybrid-Mistral**
- Job Shop and Open Shop benchmarks

Experimental results: Job Shop

Experimental results: Job Shop

Lawrence results

Mistral(*task*)		Hybrid(*vsids*, *disj*)		Hybrid(*vsids*, *lazy*)		Hybrid(*task*, *disj*)		Hybrid(*task*, *disj*)	
T	%O	T	%O	T	%O	T	%O	T	%O
471.97	88.75	**396.20**	**92**	602.51	88	410.55	90.50	489.25	89

Taillard results

	Mistral(*task*)			Hybrid(*vsids*, *disj*)			Hybrid(*vsids*, *lazy*)			Hybrid(*task*, *disj*)			Hybrid(*task*, *lazy*)		
	M		Size	M		Nodes/S	M		Nodes/S	M		Nodes/S	M		Nodes/S
	%O	T		%O	T		%O	T		%O	T		%O	T	
t01-10	90	616.22	8871.32	**90**	**477.79**	6814.73	87	999.17	1213.57	90	574.87	4869.45	85	1115.49	1261.70
	PRD			PRD			PRD			PRD			PRD		
t11-20	3.2381		6509.44	3.0350		3970.85	1.8937		520.62	0.4808		2715.29	**0.1169**		539.79
t21-30	0.7302		3935.87	0.2769		2424.16	0.4756		413.90	0.2485		1752.05	**0.1557**		437.04
t31-40	1.7227		4503.78	0.7109		2598.25	**0.3043**		555.36	0.6016		1517.04	0.4103		566.18
t41-50	2.2161		2570.10	0.4798		1530.42	**0.3036**		413.48	0.5420		994.61	0.6163		443.63
t51-60	2.0798		1952.51	2.2847		2602.31	2.7990		562.71	**0.1621**		1131	0.2419		698.37
t61-70	3.2381		1349.73	3.0350		2183.79	1.8937		522.25	0.4808		920.55	**0.1169**		584.14

- PRD: percentage relative deviation

Experimental results: Summary

- 'Light' CP models are extremely efficient with small sized instances
- These models benefit essentially from the fast exploration speed
- The impact of clause learning is more and more glaring when the size of the instance grows
- DISJUNCTIVE-based learning outperforms the other models on medium sized instances

Experimental results: lower bounds experiments

Experimental results: lower bounds experiments

Open instances from Taillard benchmark

- 7 new bounds found with DISJUNCTIVE-based and VSIDS

tai13		tai21		tai23		tai25		tai26		tai29		tai30	
new	old	new	old	new	old	new	old	new	old	new	old	new	old
1305	1282	1613	1573	1514	1474	1543	1518	1561	1558	1573	1525	1508	1485

Experimental results: lower bounds experiments

Open instances from Taillard benchmark

- 7 new bounds found with DISJUNCTIVE-based and VSIDS

tai13		tai21		tai23		tai25		tai26		tai29		tai30	
new	old	new	old	new	old	new	old	new	old	new	old	new	old
1305	1282	1613	1573	1514	1474	1543	1518	1561	1558	1573	1525	1508	1485
1342		**1642**		**1518**		**1558**		**1591**		**1573**		**1519**	

[Vilím et al., 2015]

- IBM CP-Optimizer studio
- 8h20min per instance
- Parallelization: Double threading phase
- Start search with best known bounds as an additional information.

Outline

Summary

- Contributions to each of the three aspects of constraint programming that are ‘search’, ‘propagation’ and ‘learning’ for efficiently solving sequencing and scheduling problems.

Summary

- Contributions to each of the three aspects of constraint programming that are ‘search’, ‘propagation’ and ‘learning’ for efficiently solving sequencing and scheduling problems.
- Case study: car-sequencing

Summary

- Contributions to each of the three aspects of constraint programming that are 'search', 'propagation' and 'learning' for efficiently solving sequencing and scheduling problems.
- Case study: car-sequencing
- Clause Learning in CP

Summary

- Contributions to each of the three aspects of constraint programming that are ‘search’, ‘propagation’ and ‘learning’ for efficiently solving sequencing and scheduling problems.
- Case study: car-sequencing
- Clause Learning in CP

Modern constraint programming solvers may not underestimate any of these three aspects

Future Research

- Car-Sequencing:
  - Application to 'real' industrial situations [Solnon et al., 2008].
- Propagation via AtMostSeqCard:
  - Incrementality?
  - More extensions?
- Explanation for AtMostSeqCard:
  - Minimal explanations?
  - Applications to other sequencing problems.
- Learning in Scheduling Problems:
  - Applications to other scheduling problems.
  - Learning with global constraints.
  - Hand-crafted learning.

Thank you.

References I

Beck, J. C. (2007).
Solution-guided multi-point constructive search for job shop scheduling.
Journal of Artificial Intelligence Research, 29(1):49–77.

Geelen, P. A. (1992).
Dual viewpoint heuristics for binary constraint satisfaction problems.
In *Proceedings of the 10th European Conference on Artificial Intelligence, ECAI'92, Vienna, Austria*, pages 31–35.

Gottlieb, J., Puchta, M., and Solnon, C. (2003).
A study of greedy, local search, and ant colony optimization approaches for car sequencing problems.
In *Proceedings of Applications of Evolutionary Computing, EvoWorkshop'03: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, Essex, UK*, pages 246–257.

Grimes, D. and Hebrard, E. (2015).
Solving Variants of the Job Shop Scheduling Problem through Conflict-Directed Search.
INFORMS Journal on Computing.

Haralick, R. M. and Elliott, G. L. (1980).
Increasing tree search efficiency for constraint satisfaction problems.
Artificial Intelligence, 14(3):263 – 313.

References II

Maher, M. J., Narodytska, N., Quimper, C., and Walsh, T. (2008).
Flow-based propagators for the SEQUENCE and related global constraints.
In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming, CP'08, Sydney, NSW, Australia*, pages 159–174.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001).
Chaff: Engineering an Efficient SAT Solver.
In *Proceedings of the 38th Annual Design Automation Conference, DAC'01, Las Vegas, Nevada, USA*, pages 530–535.

Régin, J. and Puget, J. (1997).
A Filtering Algorithm for Global Sequencing Constraints.
In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming, CP'97, Linz, Austria*, pages 32–46.

Siala, M., Hebrard, E., and Huguet, M. (2014).
An optimal arc consistency algorithm for a particular case of sequence constraint.
Constraints, 19(1):30–56.

Smith, B. M. (1996).
Succeed-first or Fail-first: A Case Study in Variable and Value Ordering.
Research Report 96.26 University of Leeds, School of Computer Studies.

References III

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008).
The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem.
European Journal of Operational Research, 191(3):912–927.

van Hoeve, W. J., Pesant, G., Rousseau, L., and Sabharwal, A. (2009).
New filtering algorithms for combinations of among constraints.
Constraints, 14(2):273–292.

Vilím, P., Laborie, P., and Shaw, P. (2015).
Failure-directed Search for Constraint-based Scheduling.
In *Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR'15, Barcelona, Spain*, page to appear.