Lazy Clause Generation: Combining the best of SAT and CP (and MIP?) solving

Peter J. Stuckey
with help from Timo Berthold, Geoffrey Chu, Michael Codish, Thibaut Feydy, Graeme Gange, Olga Ohrimenko, Andreas Schutt, and Mark Wallace

June 2010

Propagation Based Constraint Solving

- Repeatedly run *propagators*
- Propagators change variable domains by:
  - removing values
  - changing upper and lower bounds
  - fixing to a value
- Run until fixpoint.

KEY INSIGHT:

- Changes in domains are really the fixing of Boolean variables representing domains.
- Propagation is just the generation of clauses on these variables.
- FD solving is just SAT solving: conflict analysis for FREE!

Outline

Outline

1. Finite Domain Propagation
   - FD Example
2. SAT Solving
   - SAT Example
3. Lazy Clause Generation
   - Original Lazy Clause Generation
   - Lazier Clause Generation
   - Global Constraints
   - Search
4. Related Work
5. Conclusion

Terminology

- **domain** D maps variable x to set of possible values $D(x)$
- **propagator** $f_c : D \mapsto D$ for constraint c
  - monotonic decreasing function
  - removes values from the domain which cannot be part of a solution.
- **Problem** set of propagators F and initial domain D_0
- **propagation solver** $solv(F, D) = D'$ where D' is the greatest mutual fixpoint of all $f \in F$.
- **FD solving** interleaves propagation with search: (for simplicity binary)
  - Add new search constraint c. $D' = solv(F \cup \{f_c\}, D)$
  - On failure add backtrack and add $\neg c$. $D' = solv(F \cup \{f_{\neg c}\}, D)$
  - Repeat until all variables fixed

Outline

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [1 \,..\, 4]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

x_1		
x_2		
x_3		
x_4		
x_5		

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [1..4]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

	$\mathbf{x_1 = 1}$	
x_1	1	
x_2	$[1..4]$	
x_3	$[1..4]$	
x_4	$[1..4]$	
x_5	$[1..4]$	

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1\,..\,4\,]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

	$\mathbf{x_1 = 1}$	*alldiff*	
x_1	1	1	
x_2	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	
x_3	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	
x_4	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	
x_5	$[\,1\,..\,4\,]$	$[\,1\,..\,4\,]$	

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1\,..\,4\,]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$
x_1	1	1	1
x_2	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_3	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_4	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_5	$[\,1\,..\,4\,]$	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$
			D_1

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1\,..\,4\,]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$	$\mathbf{x_5 \leq 2}$
x_1	1	1	1	1
x_2	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_3	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_4	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_5	$[\,1\,..\,4\,]$	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	2
			D_1	

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1\,..\,4\,]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$	$\mathbf{x_5 \leq 2}$	$x_2 \leq x_5$
x_1	1	1	1	1	1
x_2	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	2
x_3	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_4	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$	$[\,2\,..\,4\,]$
x_5	$[\,1\,..\,4\,]$	$[\,1\,..\,4\,]$	$[\,2\,..\,4\,]$	2	2
			D_1		

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1 \,..\, 4\,]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$	$\mathbf{x_5 \leq 2}$	$x_2 \leq x_5$	*alldiff*
x_1	1	1	1	1	1	1
x_2	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	2	2
x_3	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$
x_4	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$
x_5	$[1..4]$	$[1..4]$	$[2..4]$	2	2	2
			D_1			

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1\,..\,4\,]$$

F propagators for:

$$x_2 \le x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \le 9.$$

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \le x_5$	$\mathbf{x_5 \le 2}$	$x_2 \le x_5$	*alldiff*	$\sum \le 9$
x_1	1	1	1	1	1	1	1
x_2	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	2	2	2
x_3	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$	3
x_4	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$	3
x_5	$[1..4]$	$[1..4]$	$[2..4]$	2	2	2	2
			D_1				

Finite Domain Propagation Example

Consider the problem with:

Domain D_0:

$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1 \mathinner{..} 4\,]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$	$\mathbf{x_5 \leq 2}$	$x_2 \leq x_5$	*alldiff*	$\sum \leq 9$	*alldiff*
x_1	1	1	1	1	1	1	1	1
x_2	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	2	2	2	2
x_3	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$	3	$\emptyset$
x_4	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$	3	$\emptyset$
x_5	$[1..4]$	$[1..4]$	$[2..4]$	2	2	2	2	2
			D_1					*fail*

Backtrack

Finite Domain Propagation Example

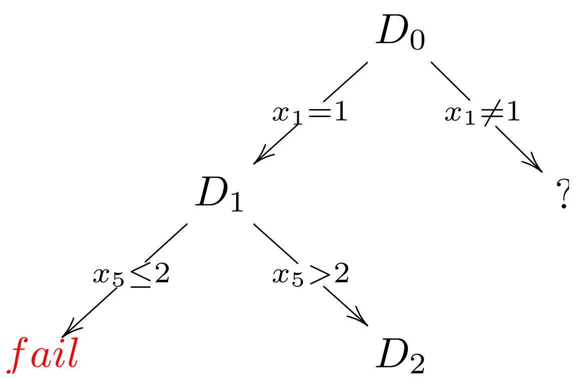
	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$
x_1	1	1	1
x_2	$[1..4]$	$[2..4]$	$[2..4]$
x_3	$[1..4]$	$[2..4]$	$[2..4]$
x_4	$[1..4]$	$[2..4]$	$[2..4]$
x_5	$[1..4]$	$[1..4]$	$[2..4]$
			D_1

Finite Domain Propagation Example

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$	$\mathbf{x_5 > 2}$
x_1	1	1	1	1
x_2	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$
x_3	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$
x_4	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$
x_5	$[1..4]$	$[1..4]$	$[2..4]$	$[3..4]$
			D_1	D_2

Finite Domain Propagation Example

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$	$\mathbf{x_5 > 2}$
x_1	1	1	1	1
x_2	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$
x_3	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$
x_4	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$
x_5	$[1..4]$	$[1..4]$	$[2..4]$	$[3..4]$
			D_1	D_2



Strengths and Weaknesses of FD solving

- Strengths
  - high level modelling
  - specialized global propagators
  - programmable search
- Weaknesses
  - Search often needs programming (weak autonomous search)
  - Optimization by repeated satisfaction search

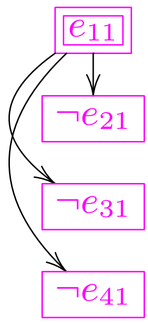
Outline

Terminology

- **literal** $l = b$ or $l = \neg b$ where b is a Boolean
- **clause** $l_1 \vee \cdots \vee l_n$ (or set of literals $\{l_1, \ldots, l_n\}$) also $\neg l_1 \wedge \cdots \wedge \neg l_{n-1} \rightarrow l_n$
- **CNF** set of clauses C
- **assignment** A is a set of literals $\{b, \neg b\} \not\subseteq A$
- **unit propagation** $up(C, A) = A'$
  - foreach clause $l_1 \vee \cdots \vee l_{n-1} \vee l_n$ where $\{\neg l_1, \ldots, \neg l_{n-1}\} \subseteq A$ add l_n to A.
  - continue to fixpoint
- **SAT solving**
  - Choose a literal l: $A' := up(C, A \cup \{l\})$
  - On failure determine a nogood $c \subseteq A$ and add it to C, backjump
  - Repeat until all variables fixed

Outline

SAT Implication Graph

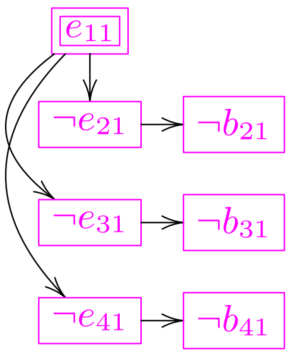


Decision e_{11}

Resolving clauses: $\neg e_{11} \vee \neg e_{21}$, $\neg e_{11} \vee \neg e_{31}$, $\neg e_{11} \vee \neg e_{41}$.

.

SAT Implication Graph

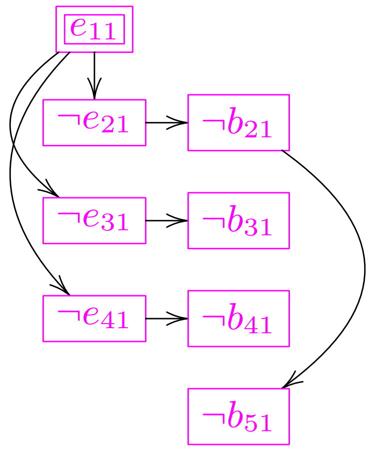


Decision e_{11}

Resolving clauses: $e_{21} \vee \neg b_{21}$, $e_{31} \vee \neg b_{31}$, $e_{41} \vee \neg b_{41}$.

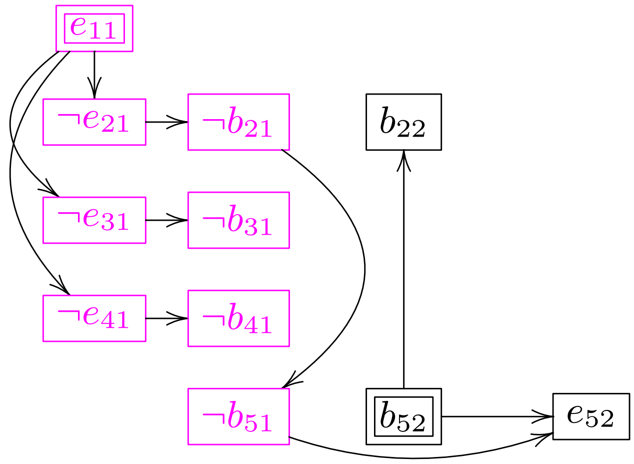
.

SAT Implication Graph



Decision e_{11}
Resolving clause: $b_{21} \vee \neg b_{51}$
Unit fixpoint

SAT Implication Graph

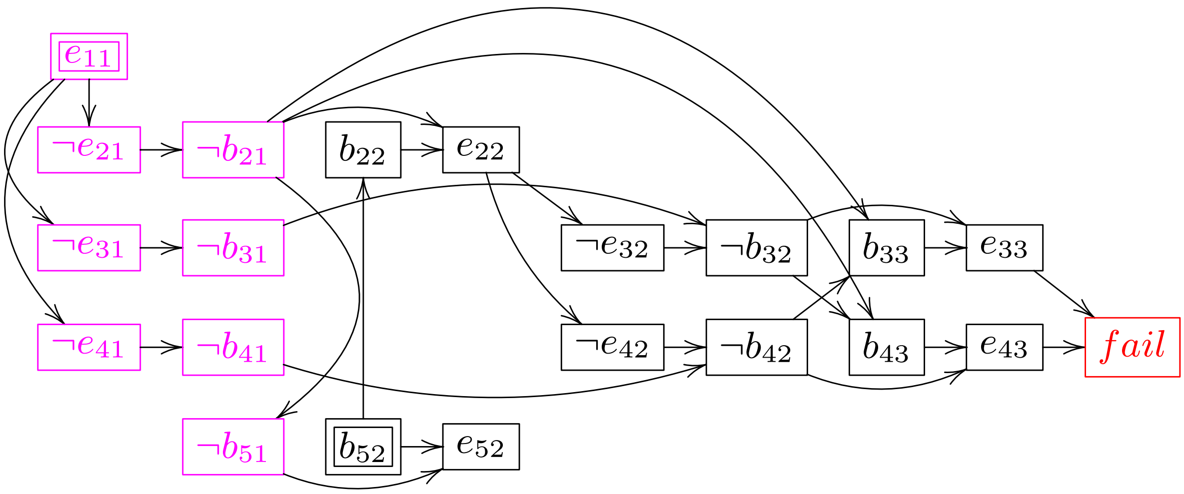


New Decision b_{52}

Resolving clauses: $b_{51} \vee \neg b_{52} \vee e_{52}$, $\neg b_{52} \vee b_{22}$

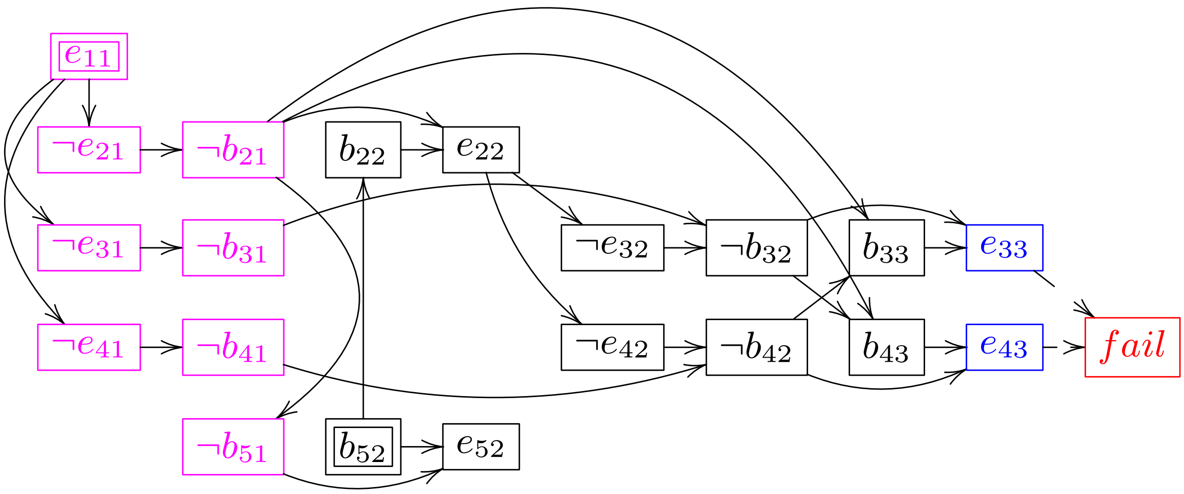
.

SAT Implication Graph



Decision b_{52}
Resolving clauses many
Conflict detected!

SAT 1UIP Conflict Resolution

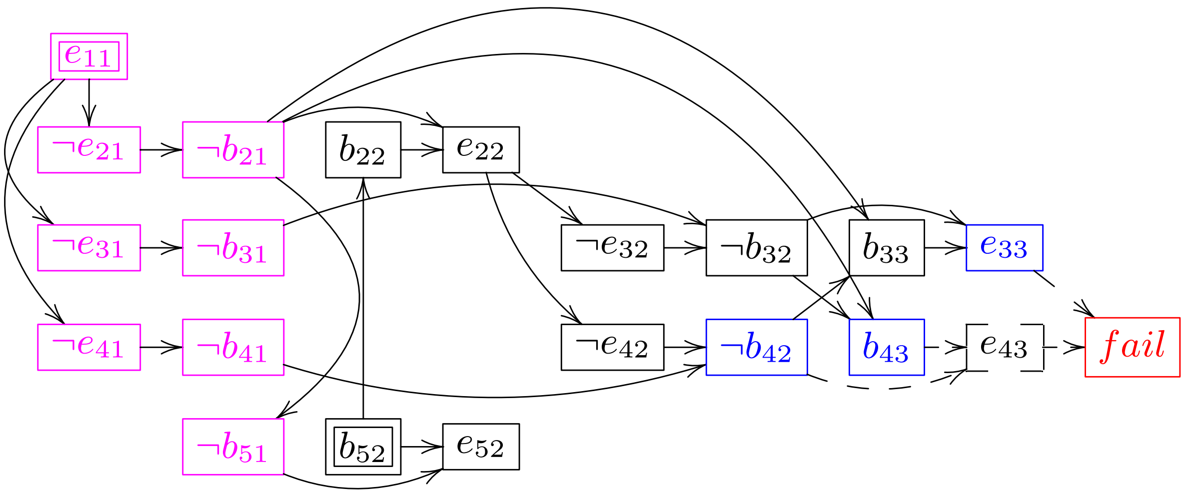


Initial nogood $(\neg e_{33} \vee \neg e_{43})$

$$e_{33} \wedge e_{43} \rightarrow false$$

.

SAT 1UIP Conflict Resolution

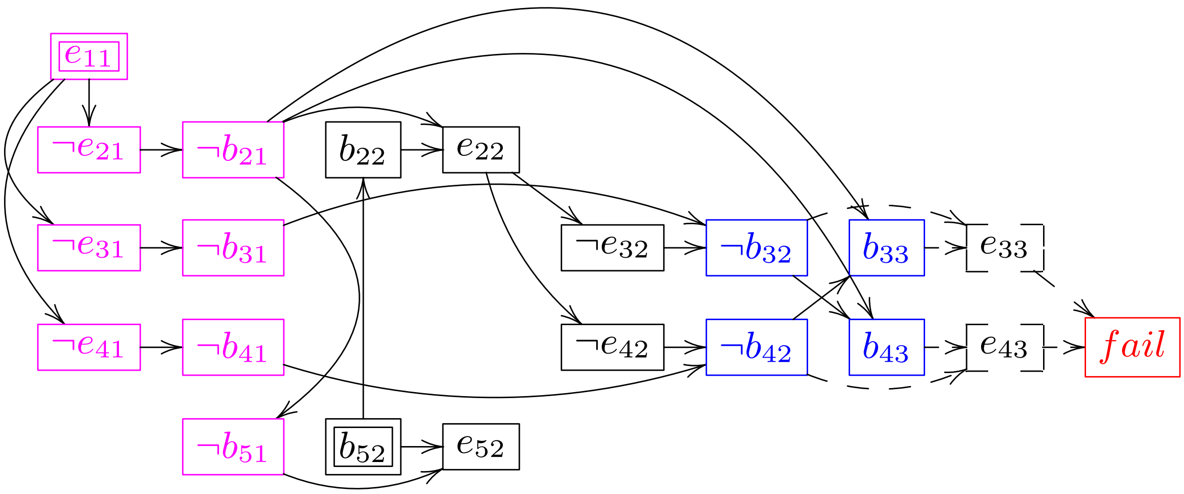


Resolving $b_{42} \vee \neg b_{43} \vee e_{43}$ gives

$$\neg b_{42} \wedge b_{43} \wedge e_{33} \rightarrow \mathit{false}$$

.

SAT 1UIP Conflict Resolution

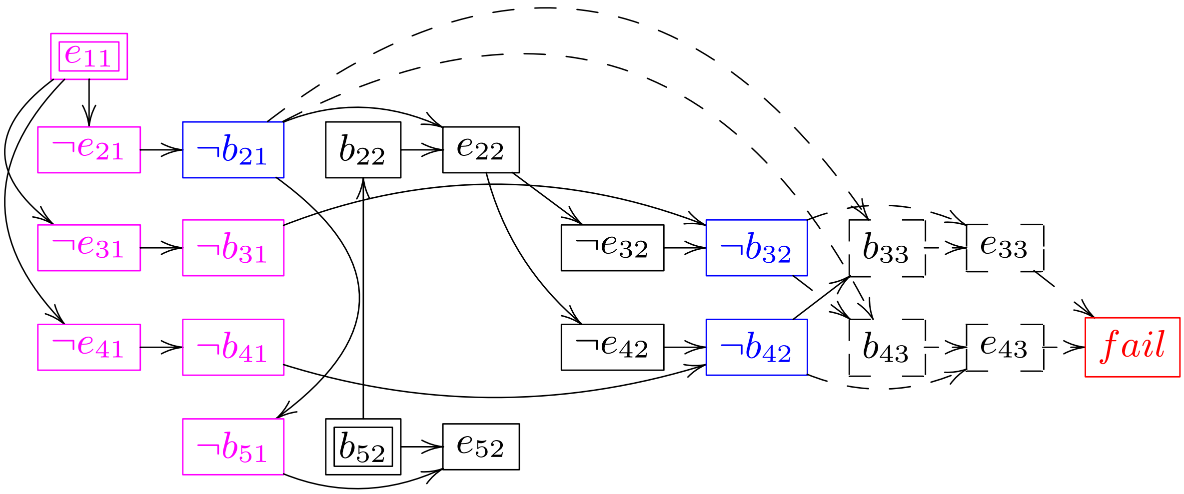


Resolving $b_{32} \vee \neg b_{33} \vee e_{33}$ gives

$$\neg b_{32} \wedge \neg b_{42} \wedge b_{33} \wedge b_{43} \rightarrow \mathit{false}$$

.

SAT 1UIP Conflict Resolution

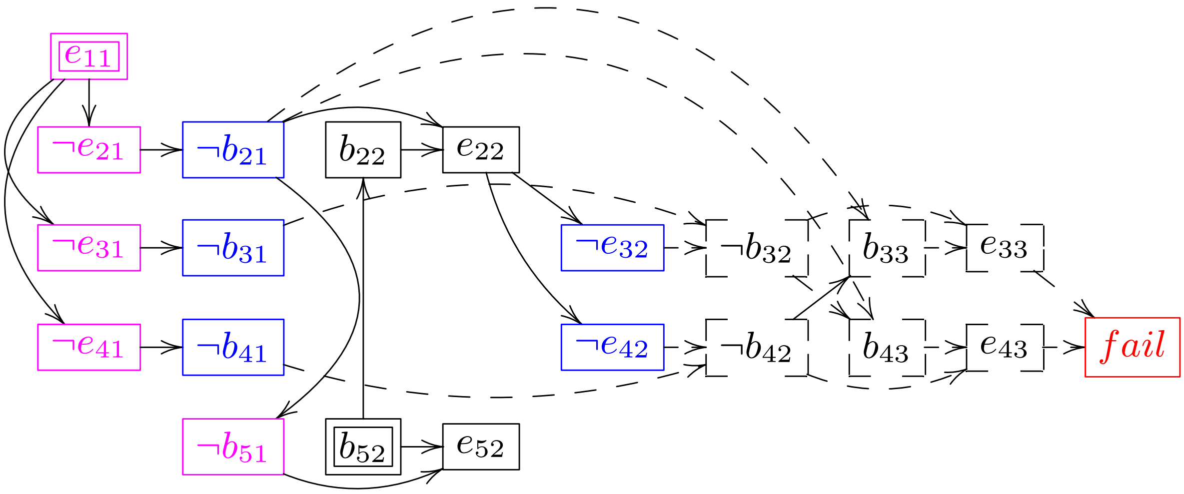


Resolving $b_{21} \vee b_{42} \vee b_{33}$ and $b_{21} \vee b_{32} \vee b_{43}$ gives

$$\neg b_{21} \wedge \neg b_{32} \wedge \neg b_{42} \rightarrow \mathit{false}$$

.

SAT 1UIP Conflict Resolution

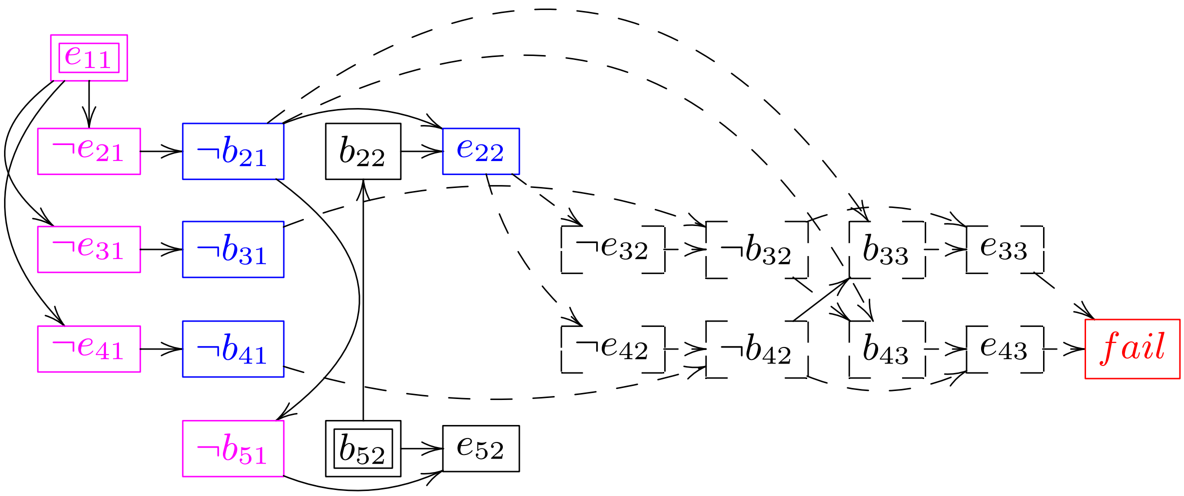


Resolving $b_{31} \vee e_{32} \vee \neg b_{32}$ and $b_{41} \vee e_{42} \vee \neg b_{42}$ gives

$$\neg b_{21} \wedge \neg b_{31} \wedge \neg b_{41} \wedge \neg e_{32} \wedge \neg e_{42} \rightarrow \mathit{false}$$

.

SAT 1UIP Conflict Resolution

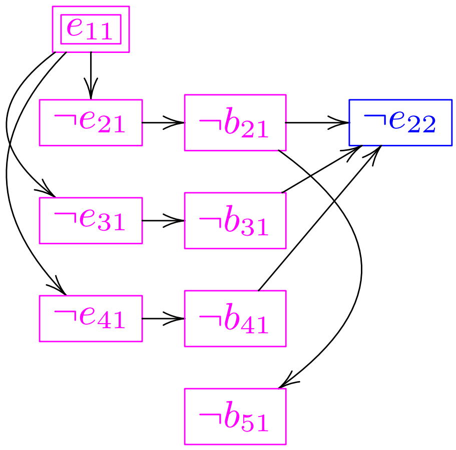


Resolving $\neg e_{22} \vee \neg e_{32}$ and $\neg e_{22} \vee \neg e_{42}$ gives

$$\neg b_{21} \wedge \neg b_{31} \wedge \neg b_{41} \wedge e_{22} \rightarrow \mathit{false}$$

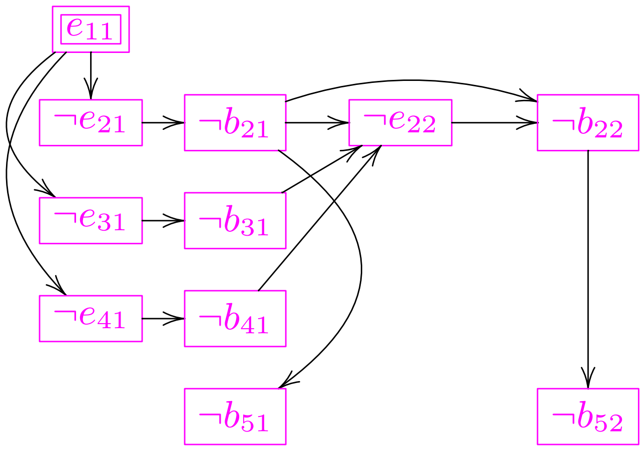
The 1UIP nogood! $b_{21} \vee b_{31} \vee b_{41} \vee \neg e_{22}$

SAT Backjumping



Backjump
Apply nogood: $b_{21} \vee b_{31} \vee b_{41} \vee \neg e_{22}$
Continue to unit fixpoint

SAT Implication Graph



Continue to unit fixpoint

Resolving clauses $b_{21} \vee \neg b_{22} \vee e_{22}$, $\neg b_{52} \vee b_{22}$

Unit fixpoint

SAT engineering

- Cornerstones of modern SAT solvers
  - Watched literals: efficient implementation of unit propagation
  - 1UIP nogoods: record effective nogoods (efficiently)
  - Activity-based search: concentrate on variables involved in recent failures
  - Restarts
- Other features
  - Deep backjumping
  - Activity based forgetting of nogoods
  - Retry last used value for a variable

Strengths and Weaknesses of SAT solving

- Strengths
  - Learning avoids repeating the same subsearch
  - Can deal with (low) millions of variables and clauses
  - Strong autonomous search
- Weaknesses
  - Optimization by repeated satisfaction search
  - Have to model entirely in clauses/Booleans (can definitely blow the limits above)

Outline

Outline

Representing Integer and Set Variables

- Integer variable x: represented using Booleans
  - $[\![x = d]\!], d \in [\, l \,..\, u\,] = D_0(x)$,
  - $[\![x \leq d]\!], l \leq d < u$.
- Clauses to maintain consistency: DOM

$$\begin{aligned} [\![x \leq d]\!] &\rightarrow [\![x \leq d+1]\!] \quad l \leq d < u-1 \\ [\![x = d]\!] \leftrightarrow [\![x \leq d]\!] &\wedge \neg[\![x \leq d-1]\!] \quad l < d \leq u \end{aligned}$$

- Unary arithmetic representation (linear in size)
- One to one correspondence domains D and assignment A unit fixpoints of DOM $A = up(DOM, A)$

Atomic Constraints

- **atomic constraints** define changes in domains
  - Fixing variable: $x_i = d$
  - Removing value: $x_i \neq d$
  - Bounding variable: $x_i \leq d$, $x_i \geq d$
- Atomic constraints are just Boolean literals!

$$
\begin{aligned}
x_i = d &\equiv [\![x_i = d]\!] \\
x_i \neq d &\equiv \neg[\![x_i = d]\!] \\
x_i \leq d &\equiv [\![x_i \leq d]\!] \\
x_i \geq d &\equiv \neg[\![x_i \leq d - 1]\!]
\end{aligned}
$$

Lazy Clause Generation Propagators

- When $f(D) \neq D$ (new information)
- Propagator explains each atomic constraint change
- What part of the current domain D created the new inference!
  - $D(x_1) = \{1\}$, $D(x_2) = D(x_3) = D(x_4) = [1..4]$,
    $\texttt{alldifferent}([x_1, x_2, x_3, x_4])$
  - $f_{alldiff}(D)$ implies $x_2 \neq 1$, $x_3 \neq 1$, $x_4 \neq 1$
  - explanations $x_1 = 1 \rightarrow x_2 \neq 1$, $x_1 = 1 \rightarrow x_3 \neq 1$, $x_1 = 1 \rightarrow x_4 \neq 1$,
- Adds explanation as clauses, unit propagate on Booleans
- Propagator similarly explains failure.
  - $D(x_3) = \{3\}$, $D(x_4) = \{3\}$, $\texttt{alldifferent}([x_1, x_2, x_3, x_4])$
  - $f_{alldiff}(D)$ gives a false domain
  - explanation $x_3 = 3 \wedge x_4 = 3 \rightarrow \mathit{fail}$

Finite Domain Propagation Example Redux

Consider the problem with:

Domain D_0:

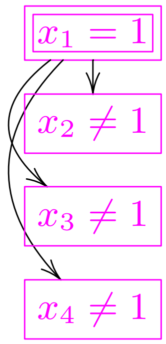
$$D_0(x_1) = D_0(x_2) = D_0(x_3) = D_0(x_4) = D_0(x_5) = [\,1\,..\,4\,]$$

F propagators for:

$$x_2 \leq x_5,\ \texttt{alldifferent}([x_1, x_2, x_3, x_4]),\ x_1 + x_2 + x_3 + x_4 \leq 9.$$

Lazy Clause Generation example

alldiff



Search: $x_1 = 1$

$D(x_1) = \{1\}$, $D(x_2) = D(x_3) = D(x_4) = D(x_5) = [1\,..\,4]$,

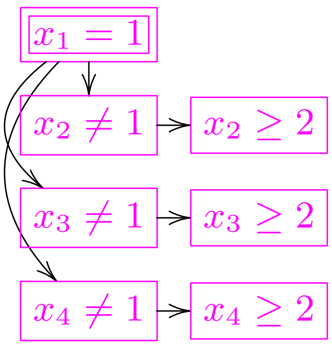
Propagate `alldifferent`$([x_1, x_2, x_3, x_4])$ on D

Determines $x_2 \neq 1$, $x_3 \neq 1$, $x_4 \neq 1$

Explanations $x_1 = 1 \rightarrow x_2 \neq 1$, $x_1 = 1 \rightarrow x_3 \neq 1$, $x_1 = 1 \rightarrow x_4 \neq 1$

Lazy Clause Generation example

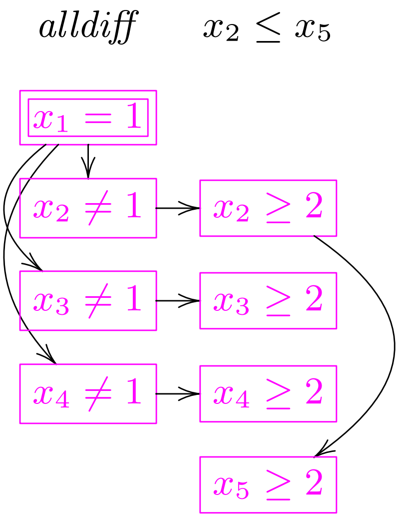
alldiff



Propagate DOM clauses: $x_2 \neq 1 \rightarrow x_2 \geq 2$, ...
Ignoring DOM clauses: $x_1 = 1 \rightarrow x_1 \neq 2$, $x_1 = 1 \rightarrow x_1 \leq 3$, ...
Domain

$$D(x_1) = \{1\},\ D(x_2) = D(x_3) = D(x_4) = [\,2\,..\,4\,],\ D(x_5) = [\,1\,..\,4\,]$$

Lazy Clause Generation example



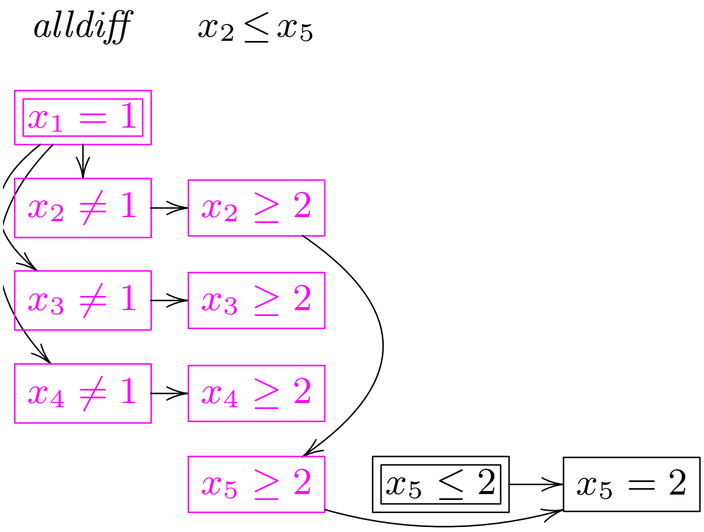
Propagate $x_2 \leq x_5$

Determines $x_5 \geq 2$ with explanation $x_2 \geq 2 \rightarrow x_5 \geq 2$

FIXPOINT:

$D_1(x_1) = \{1\}, \ D_1(x_2) = D_1(x_3) = D_1(x_4) = D_1(x_5) = [2 \,..\, 4]$

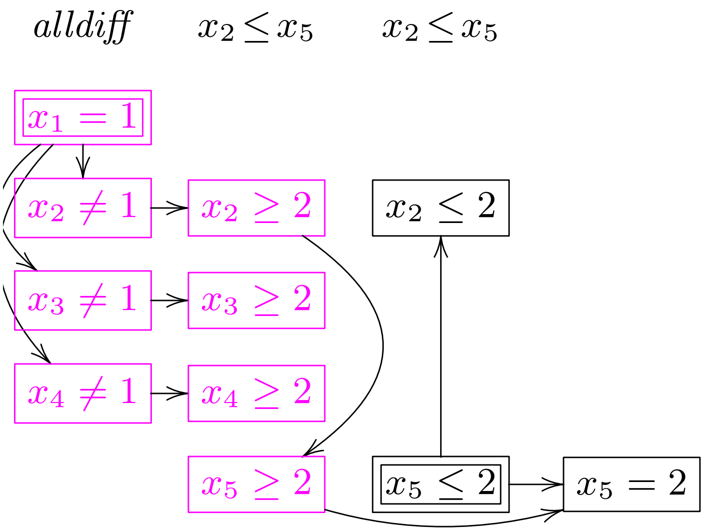
Lazy Clause Generation example



Search $x_5 \leq 2$

Domain constraints determine $x_5 = 2$ with explanation $x_5 \geq 2 \wedge x_5 \leq 2 \rightarrow x_5 = 2$

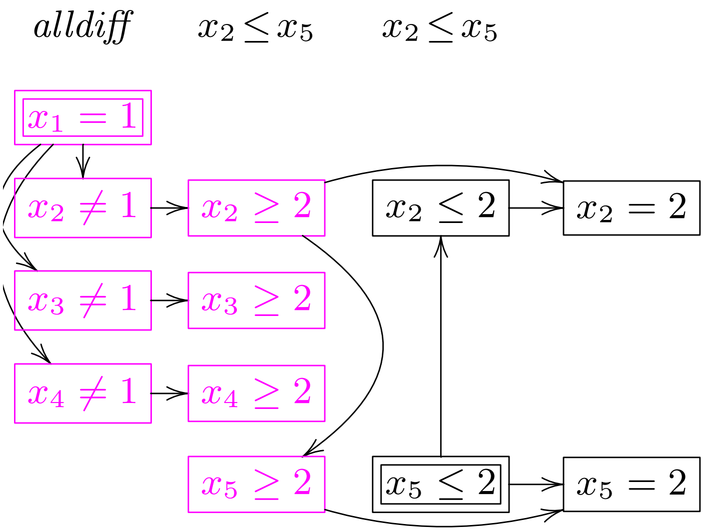
Lazy Clause Generation example



Propagate $x_2 \leq x_5$

Determine $x_2 \leq 2$ with explanation $x_5 \leq 2 \rightarrow x_2 \leq 2$

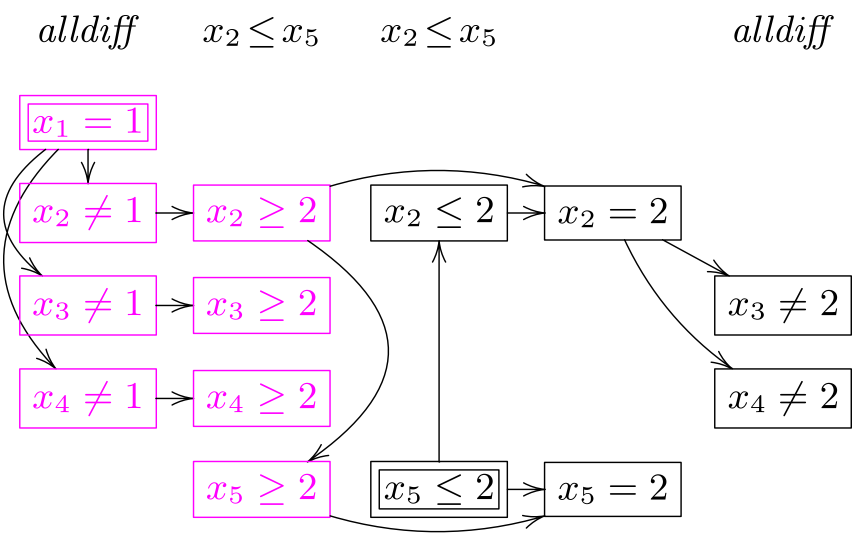
Lazy Clause Generation example



Domain constraints determine $x_2 = 2$ with explanation
$x_2 \geq 2 \wedge x_2 \leq 2 \rightarrow x_2 = 2$
Domain:

$D(x_1) = \{1\}$, $D(x_2) = \{2\}$, $D(x_3) = D(x_4) = [2..4]$, $D(x_5) = \{2\}$

Lazy Clause Generation example

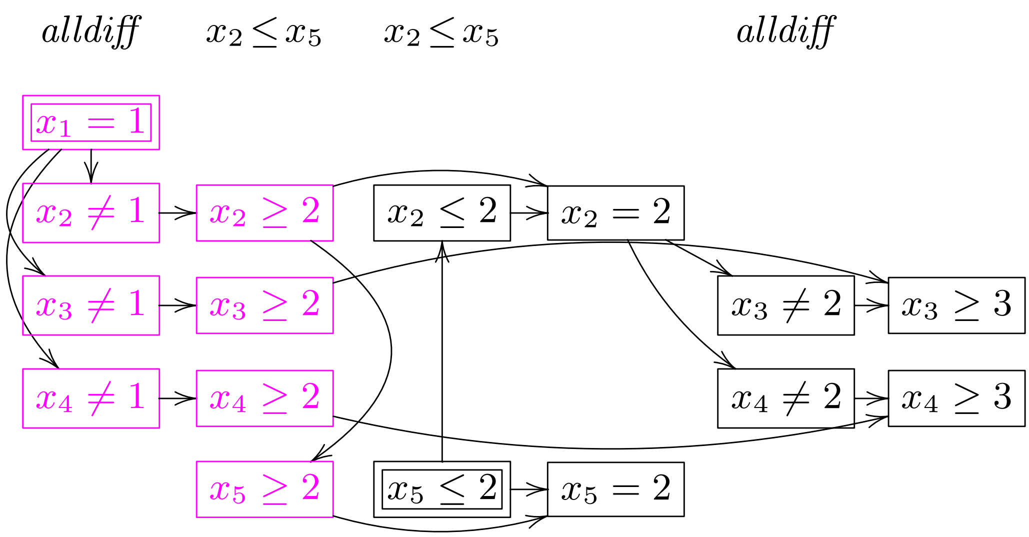


Propagate `alldifferent`$([x_1, x_2, x_3, x_4])$

Determines $x_3 \neq 2$ and $x_4 \neq 2$

with explanations $x_2 = 2 \rightarrow x_3 \neq 2$, $x_2 = 2 \rightarrow x_4 \neq 2$,

Lazy Clause Generation example

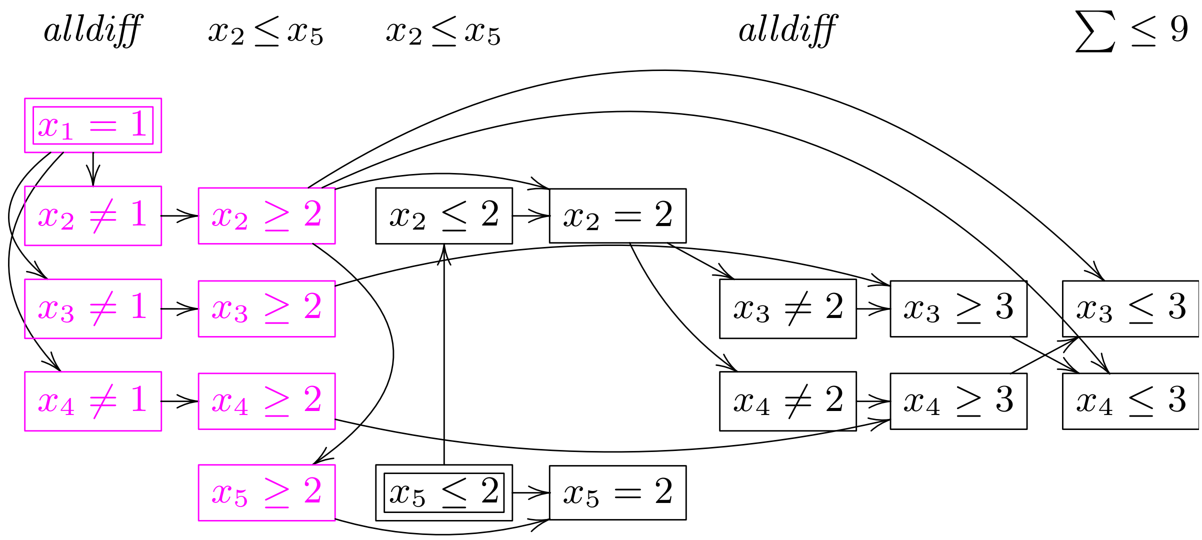


Domain constraints determine $x_3 \geq 3$ and $x_4 \geq 3$

Domain

$$D(x_1) = \{1\},\ D(x_2) = \{2\},\ D(x_3) = D(x_4) = [3\,..\,4],\ D(x_5) = \{2\}$$

Lazy Clause Generation example



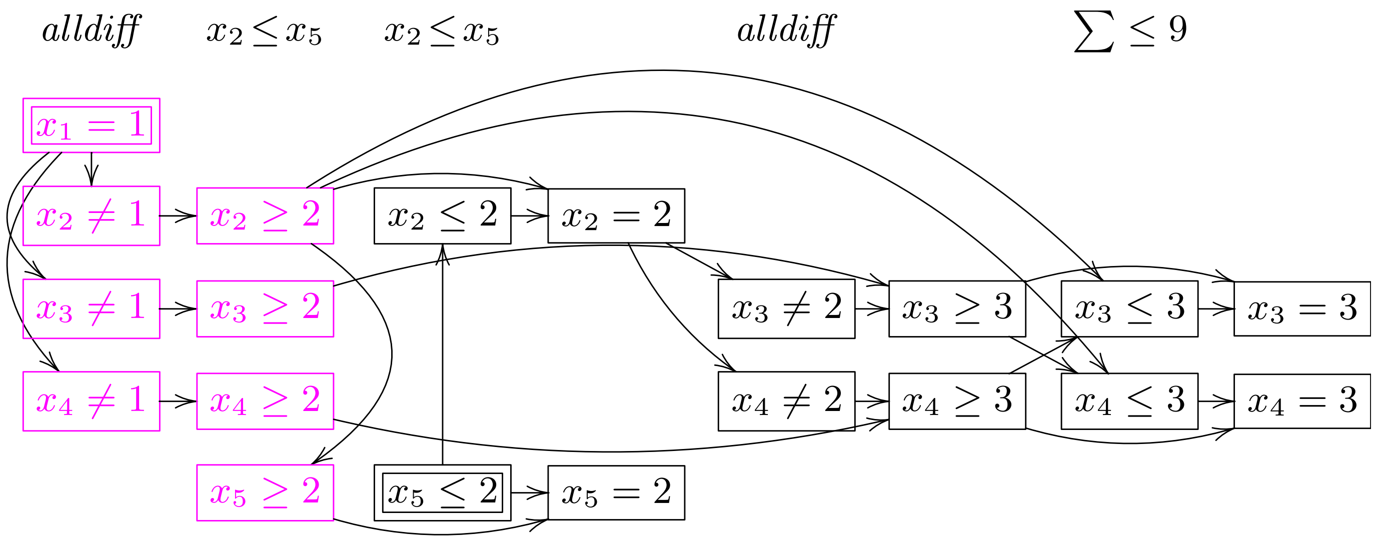
Propagate $x_1 + x_2 + x_3 + x_4 \leq 9$

Determines $x_3 \leq 3$ and $x_4 \leq 3$

with explanations $x_2 \geq 2 \wedge x_4 \geq 3 \rightarrow x_3 \leq 3$ and similar

.

Lazy Clause Generation example



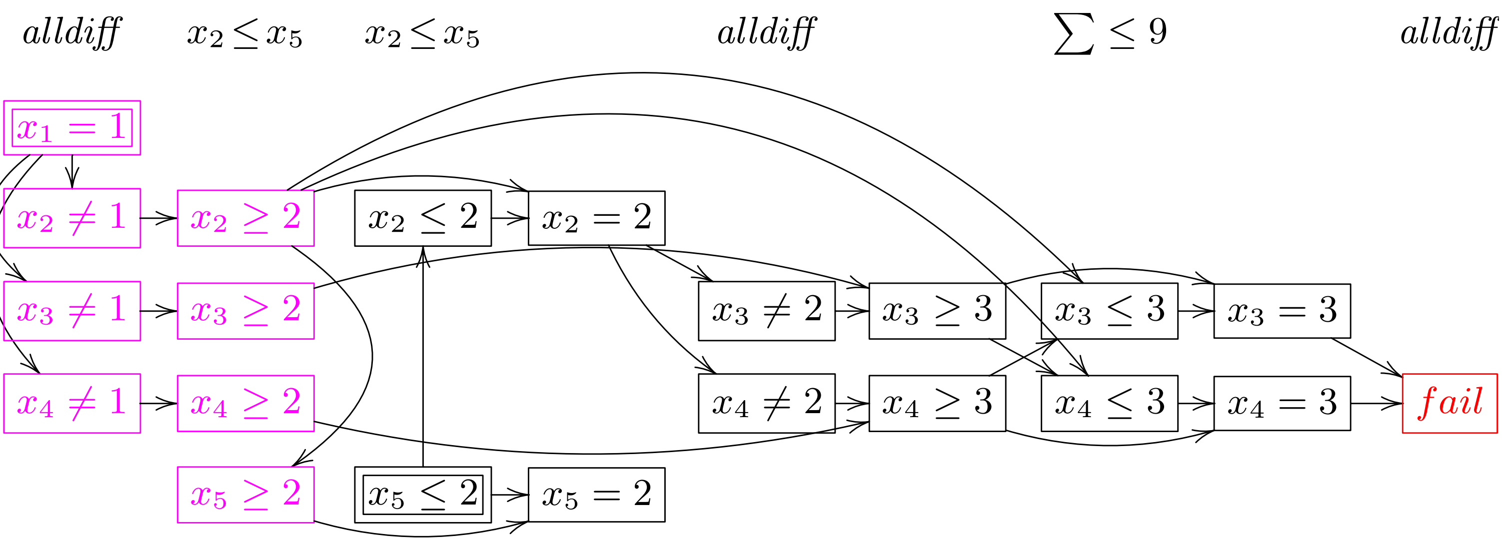
Domain constraints determine $x_3 = 3$ and $x_4 = 3$

With explanations $x_3 \geq 3 \wedge x_3 \leq 3 \rightarrow x_3 = 3$, $x_4 \geq 3 \wedge x_4 \leq 3 \rightarrow x_4 = 3$

Domain $D(x_1) = \{1\}$, $D(x_2) = \{2\}$, $D(x_3) = D(x_4) = \{3\}$, $D(x_5) = \{2\}$

.

Lazy Clause Generation example



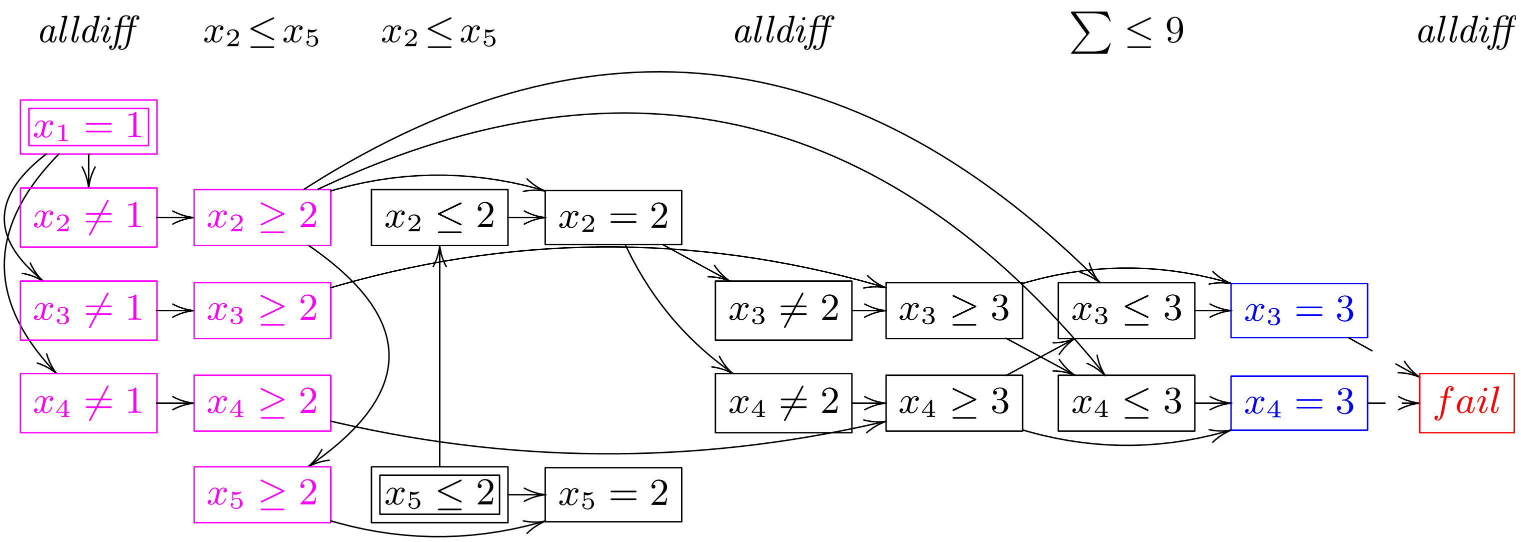
Propagate `alldifferent`$([x_1, x_2, x_3, x_4])$

Failure detected: explanation $x_3 = 3 \wedge x_4 = 3 \rightarrow \mathit{false}$

Lazy Clause Generation Example

	$\mathbf{x_1 = 1}$	*alldiff*	$x_2 \leq x_5$	$\mathbf{x_5 \leq 2}$	$x_2 \leq x_5$	*alldiff*	$\sum \leq 9$	*alldiff*
x_1	1	1	1	1	1	1	1	1
x_2	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	2	2	2	2
x_3	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$	3	$\emptyset$
x_4	$[1..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[2..4]$	$[3..4]$	3	$\emptyset$
x_5	$[1..4]$	$[1..4]$	$[2..4]$	2	2	2	2	2
			D_1					*fail*

Lazy Clause Generation Explanation

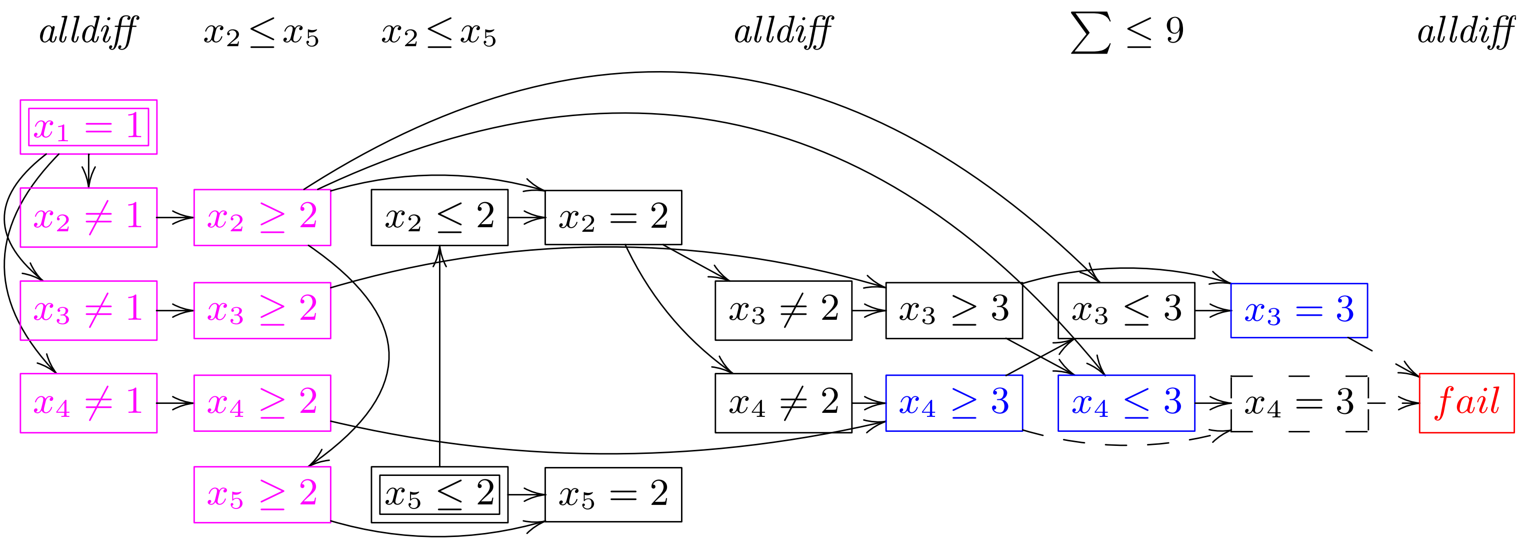


The initial nogood

$$x_3 = 3 \wedge x_4 = 3 \rightarrow \mathit{false}$$

.

Lazy Clause Generation Explanation

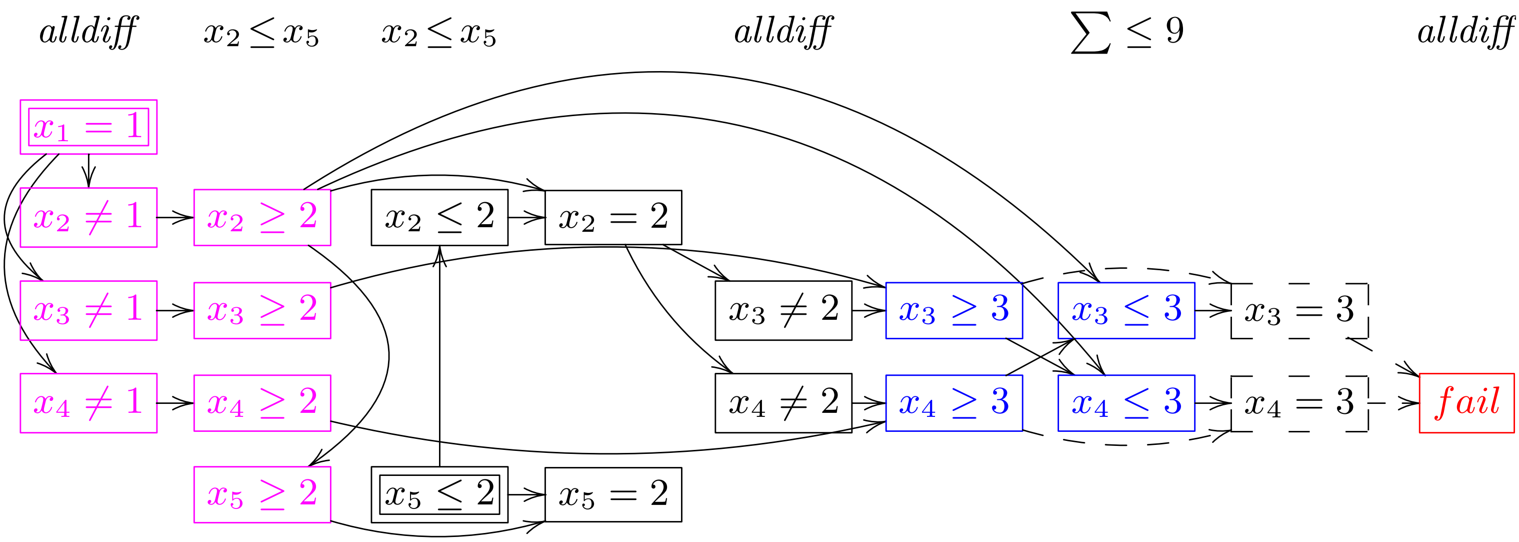


Resolving

$$x_4 \geq 3 \wedge x_4` \leq 3 \wedge x_3 = 3 \rightarrow \mathit{false}$$

.

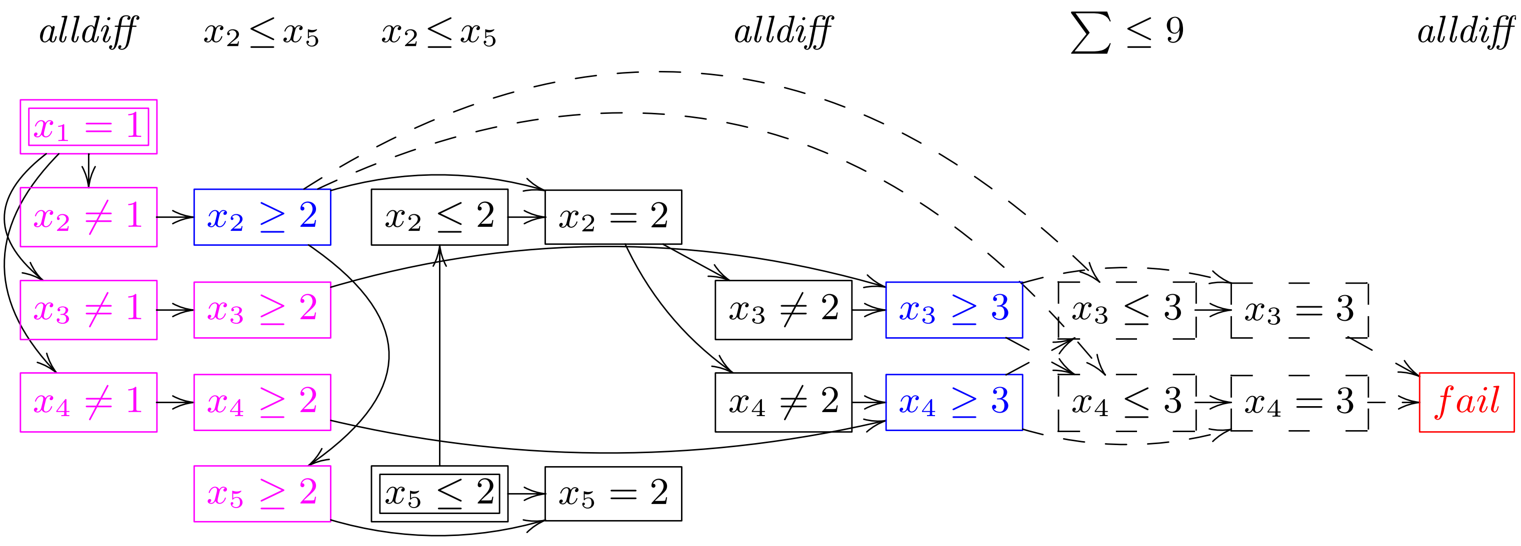
Lazy Clause Generation Explanation



Resolving

$$x_3 \geq 3 \wedge x_4 \geq 3 \wedge x_3 \leq 3 \wedge x_4 \leq 3 \rightarrow false$$

Lazy Clause Generation Explanation

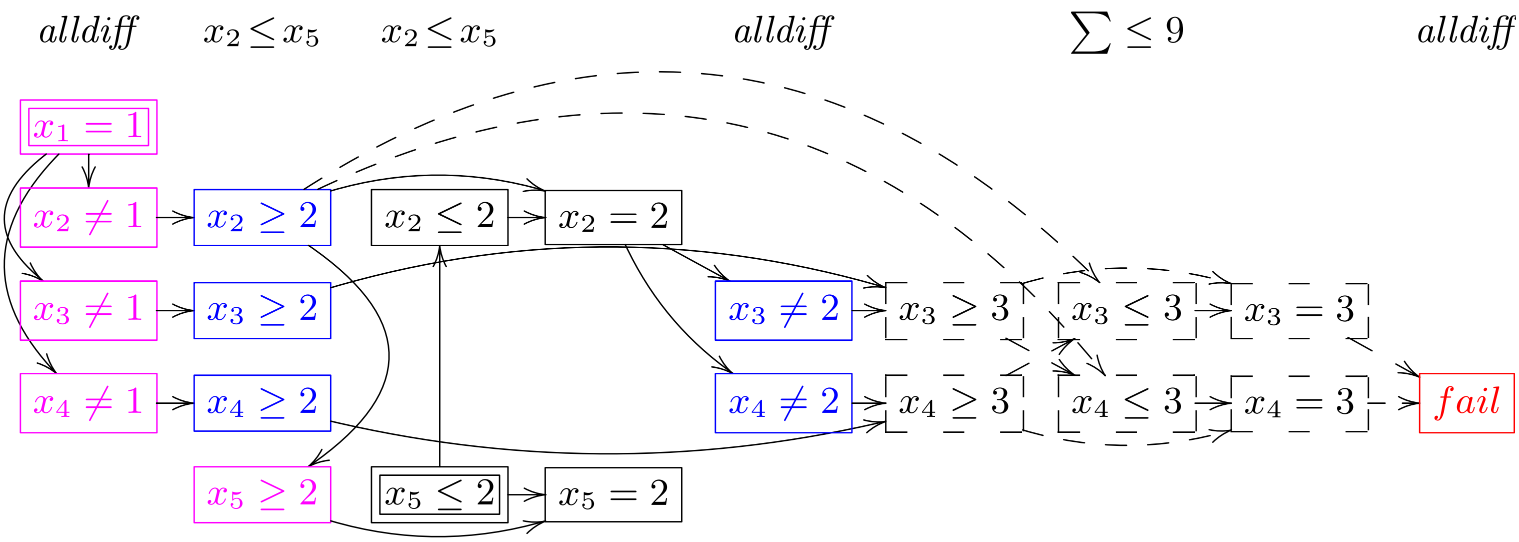


Resolving

$$x_2 \geq 2 \wedge x_3 \geq 3 \wedge x_4 \geq 3 \rightarrow \mathit{false}$$

.

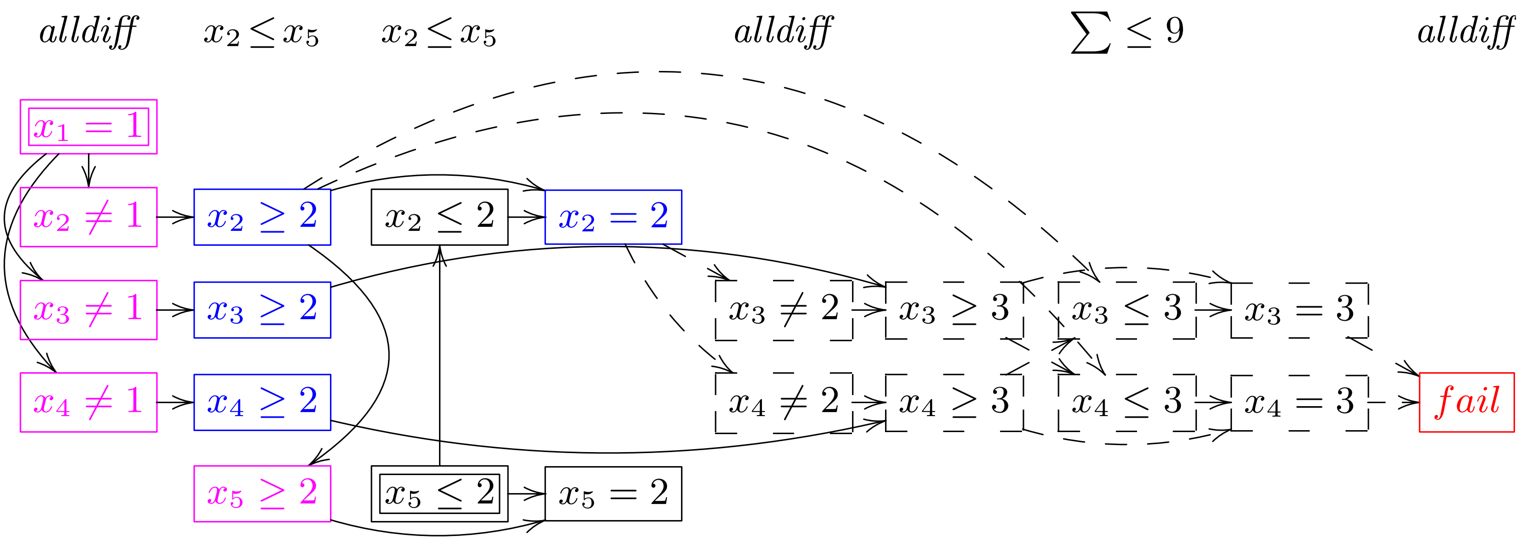
Lazy Clause Generation Explanation



Resolving

$$x_2 \geq 2 \land x_3 \geq 2 \land x_4 \geq 2 \land x_3 \neq 2 \land x_4 \neq 2 \rightarrow false$$

Lazy Clause Generation Explanation



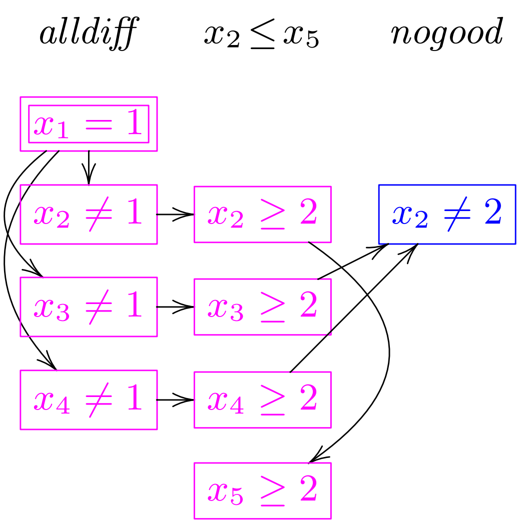
Resolving

$$x_2 \geq 2 \wedge x_3 \geq 2 \wedge x_4 \geq 2 \wedge x_2 = 2 \rightarrow false$$

Simplify! $\quad x_3 \geq 2 \wedge x_4 \geq 2 \wedge x_2 = 2 \rightarrow false$

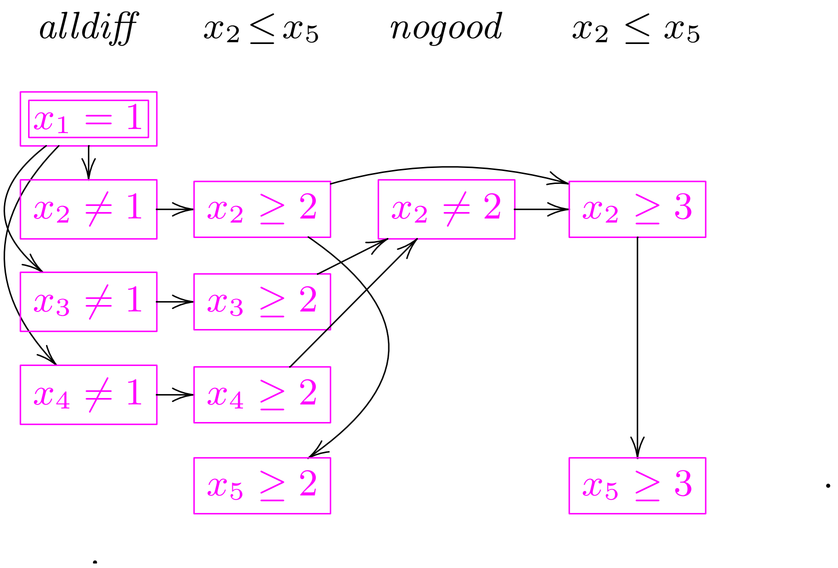
.

Lazy Clause Generation Example



Backjump
Propagate $x_3 \geq 2 \wedge x_4 \geq 2 \rightarrow x_2 \neq 2$

Lazy Clause Generation Example



Domain constraints determine $x_2 \geq 3$

Propagate $x_2 \leq x_5$ determines $x_5 \geq 3$

Different Domain

$D'_2(x_1) = \{1\}$, $D'_2(x_2) = D'_2(x_5) = [3..4]$, $D'_2(x_3) = D'_2(x_4) = [2..4]$

What's Really Happening

- A high level "Boolean" model of the problem
- Clausal representation of the Boolean model is generated "as we go"
- All generated clauses are redundant and can be removed at any time
- We can control the size of the active "Boolean" model

Comparing with SAT on Tai open shop scheduling: (averages)
SAT generates the full Boolean model before starting solving

	Time	*solve only*	*Fails*	Max Clauses Generated
SAT	318	(89)	**3597**	13.17
LCG	**62**		6611	**1.00**

Strengths and Weaknesses of Lazy Clause Generation

- Strengths
  - High level modelling
  - Learning avoids repeating the same subsearch
  - Strong autonomous search
  - Programmable search
  - Specialized global propagators (but requires work)
- Weaknesses
  - Optimization by repeated satisfaction search
  - Overhead compared to FD when nogoods are useless

Outline

Lazy Boolean Variable Creation

- Many Boolean variables are never used
- Create them on demand
- **Array encoding**
  - Create bounds variables initially $x \leq d$
  - Only create equality variables $x = d$ on demand
    Add $x \geq d \wedge x \leq d \rightarrow x = d$
- **List encoding**
  - Create bounds variables on demand $x \leq d$
    Add $x \leq d' \rightarrow x \leq d$, $x \leq d \rightarrow x \leq d''$ where d' (d'') is next lowest (highest) existing bound
  - At most $2\times$ bounds clauses
  - Create equality variables on demand as before

Lazy Boolean Variable Creation Tradeoffs

- List versus array
- List always works! Array may require too many variables
- Implementation complexity
- List hampers learning

Tai open shop scheduling: 15x15 (average of 10 problems)

	AverageTime
array	**13.38**
list	56.66

Views (Schulte + Tack, 2005)

- **View** is a pseudo variable defined by a "bijective" function to another variable
  - $x = \alpha y + \beta$
  - $x = \texttt{bool2int}(y)$
  - $x = \neg y$
- The view variable x, does not exist, operations on it are mapped to y
- More important for lazy clause generation
  - Reduce Boolean variable representation
  - Improve nogoods (reduce search)

Constrained path covering problems: Average of 5 problems

	Time	*Fails*
views	**0.71**	**950**
no views	1.12	1231

Explanation Deletion

- Explanations only really needed for nogood learning
  - **Forward** add explanations as they are generated
  - **Backward** delete explanations as we backtrack past them
- Smaller set of clauses
- Can hamper search "Reprioritization"

Tai open shop scheduling:

	15x15	20x20
deletion	**13.38**	**39.96**
no deletion	20.58	95.88

But RCPSP worse with deletion!

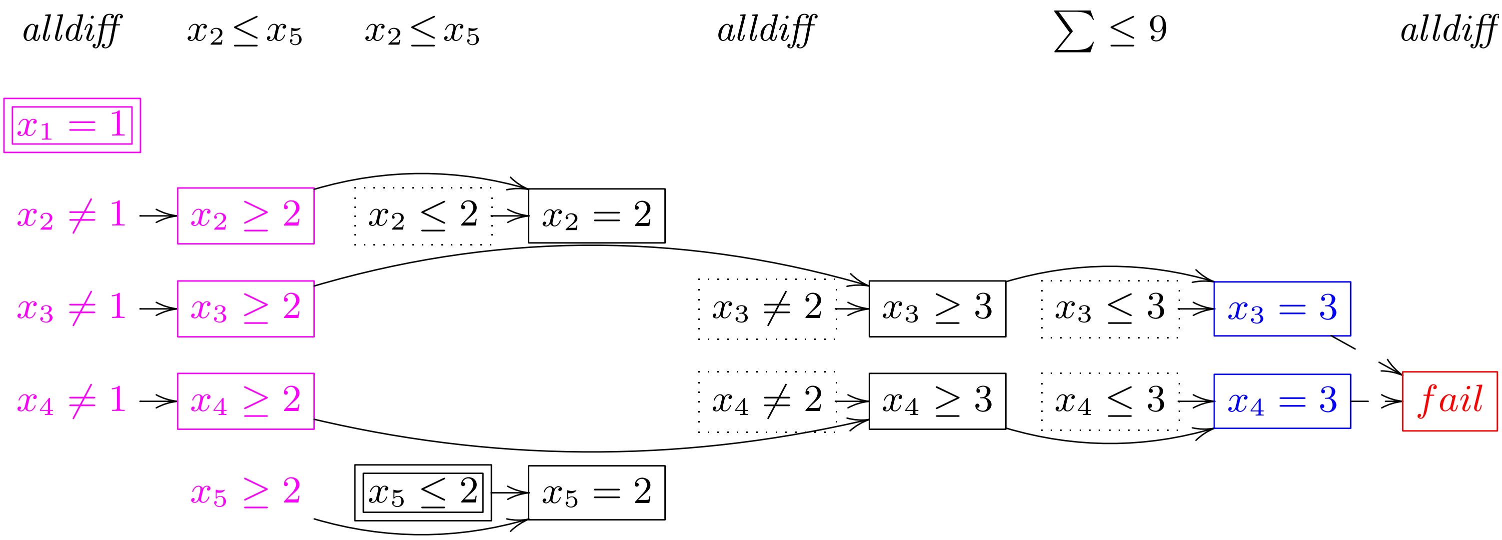
Lazy Explanation

- Explanations only needed for nogood learning
  - **Forward** record propagator causing each atomic constraint
  - **Backward** ask propagator to explain atomic constraint (if required)
- Standard for SAT extensions (MiniSAT 1.14) [See Gent et al PADL2010]
- Only create needed explanations!
- Harder implementation

Social Golfers Problems: using an MDD propagator
(each explanation as expensive as running entire propagator)

	Time	*Reasons*	*Fails*
lazy explanation	**2.38**	**14347**	**2751**
eager explanation	4.92	78177	5126

Lazy Clause Generation Explanation

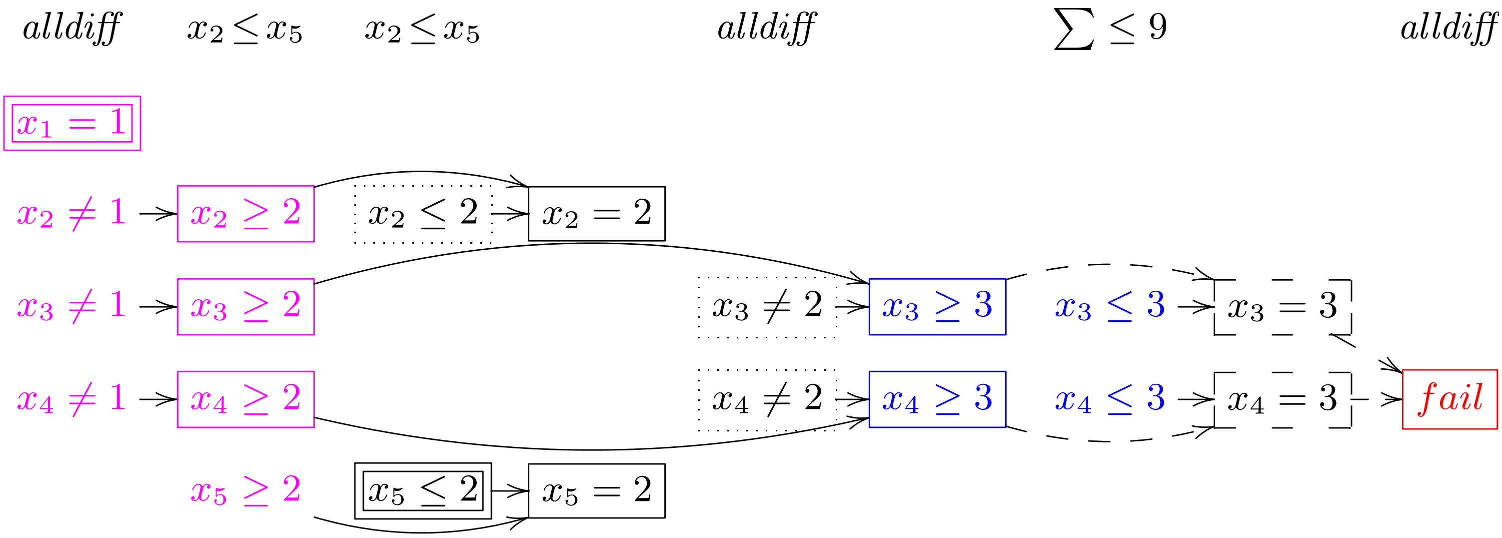


Dotted boxes explained by above propagator.
Initial nogood

$$x_3 = 3 \wedge x_4 = 3 \rightarrow fail$$

.

Lazy Clause Generation Explanation

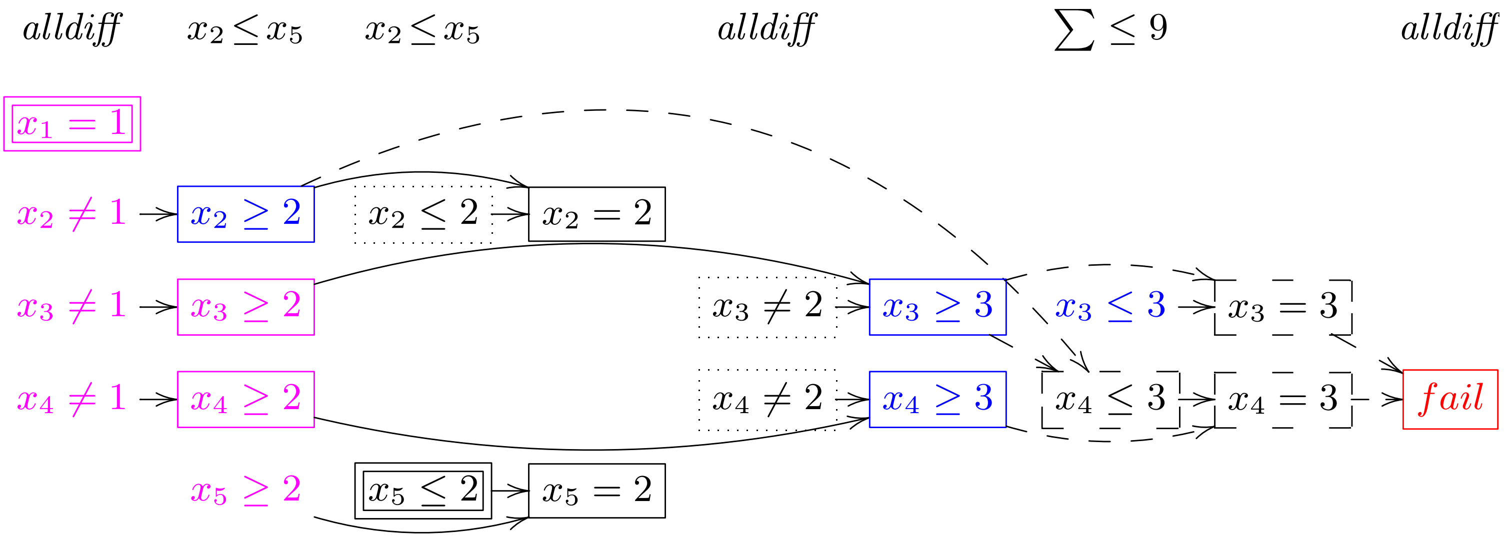


Resolving $x_3 \geq 3 \wedge x_3 \leq 3 \rightarrow x_3 = 3$ and $x_4 \geq 3 \wedge x_4 \leq 3 \rightarrow x_4 = 3$

$$x_3 \geq 3 \wedge x_4 \geq 3 \wedge x_3 \leq 3 \wedge x_4 \leq 3 \rightarrow \mathit{fail}$$

Request $x_1 + x_2 + x_3 + x_4 \leq 9$ to explain $x_4 \leq 3$

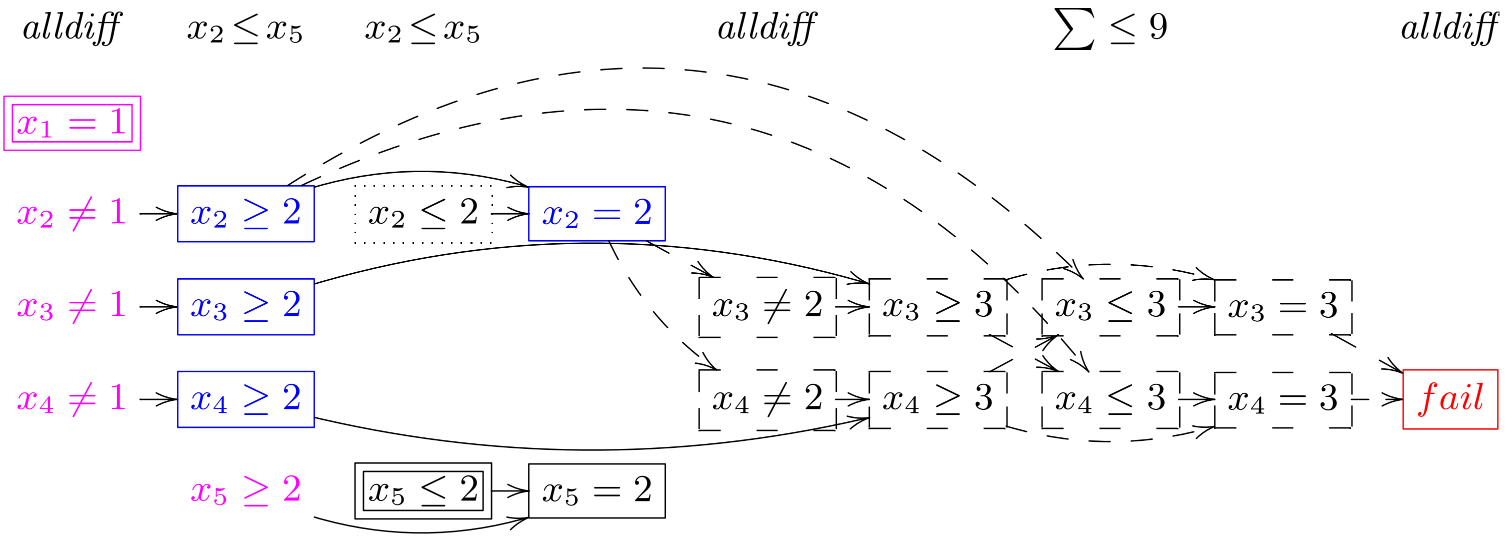
Lazy Clause Generation Explanation



Lazy Explanation $x_2 \geq 2 \wedge x_3 \geq 3 \rightarrow x_4 \leq 3$
Resolving on this gives

$$x_2 \geq 2 \wedge x_3 \geq 3 \wedge x_4 \geq 3 \wedge x_3 \leq 3 \rightarrow \mathit{fail}$$

Lazy Clause Generation Explanation



Final 1UIP nogood

$$x_2 \geq 2 \wedge x_3 \geq 2 \wedge x_4 \geq 2 \wedge x_2 = 2 \rightarrow \mathit{false}$$

Note 5 unexplained atomic constraints remain!

Outline

The Globality of Explanation

- Nogoods extract global information from the problem
- Can overcome weaknesses of local propagators
- **Example**
- $D(x_1) = D(x_2) = [\,0\,..\,100000\,]$ $x_2 \geq x_1 \wedge (b \Leftrightarrow x_1 > x_2)$
- Set $b = \mathit{true}$ and 200000 propagations later failure. YIKES
- A global difference logic propagator immediately sets $b = \mathit{false}$!
- Lazy clause generation learns $b = \mathit{false}$ after 200000 propagations
  - But never tries it again!

Globals by Decomposition

- Globals defined by decomposition
  - Don't require implementation
  - Automatically incremental
  - Allow partial state relationships to be "learned"
  - Much more attractive with lazy clause generation
- When propagation is not hampered, and size does not blowout:
  - can be good enough!

Resource constrained project scheduling problems: (`cumulative` by decomposition) closed 62 open problems % solved to optimality in time

	J60			J90			J120		
	45s	300s	1800s	45s	300s	1800s	45s	300s	1800s
Laborie	-	84.2	85.0	-	78.5	79.4	-	41.3	41.7
LCG	**85.2**	**88.1**	**89.4**	**79.8**	**81.3**	**82.5**	**42.5**	**44.8**	**45.3**

Which Decomposition?

- Different decompositions interact better or worse with lazy clause generation.
- `alldifferent`
  - **diseq**: $O(n^2)$ disequations
  - **bnd**: Bound consistent decomposition of Bessiere et al IJCAI09
  - **bnd+**: Bound consistent decomp. replacing $x \geq d \wedge x \leq d$ by $x = d$
  - **gcc**: Based on a simple global cardinality decomposition

Quasi-group completion 25x25 (average of examples solved by all)

$diseq(13)$		$bnd(11)$		$bnd+(13)$		$gcc(15)$		$CSPComp2008$ (13)	$CSPComp2008$ (12)
Time	*Fails*	*Time*	*Fails*	*Time*	*Fails*	*Time*	*Fails*	*Time*	*Time*
131	142680	757	9317	129	1144	**4.3**	**1010**	> 433	> 500

Explanations for Globals

- Globals are better than decomposition
  - More efficient
  - Stronger propagation
- Instrument global constraint to also explain its propagations
  - **mdd**: expensive each explanation as much as propagation
  - **cumulative**: choices in how to explain
- Implementation complexity, Can't learn partial state
- More efficient + stronger propagation

Resource constrained project scheduling problems:

	J60 (25% faster)			J90 (25% faster)			J120 (60% faster)		
	45s	300s	1800s	45s	300s	1800s	45s	300s	1800s
Decomp	84.8	**89.2**	89.4	79.8	81.7	82.5	42.3	45.2	45.7
Global	**85.8**	89.0	**89.6**	**80.0**	**81.9**	**82.7**	**42.7**	**45.8**	**47.0**

Outline

Nogoods and Programmed Search

- Contrary to SAT folklore
  - Activity based search can be terrible
  - Nogoods work excellently with programmed search

Constrained Path Covering Problems

	Time	*Fails*
nogoods + *VSIDS*	> 361.89	$> 30,000$
nogoods + *programmed*	**0.71**	**950**
programmed	> 240.2	$> 10,000$

Activity-based search

- An excellent default search!
- Weak at the beginning (no meaningful activities)
- Need hybrid approachs
  - Hot Restart:
    - Start with programmed search to "initialize" meaningful activities.
    - Switch to activity-based after restart
  - Use activity-based as part of a programmed search
- Much more to explore in this direction

Outline

SAT modulo theories (SMT)

- Combine a SAT solver with theory solvers to handle non Boolean constraints.
- (Original) Lazy Clause Generation is a special case
  - Each propagator is its own theory
  - Propagators do "theory propagation"
- Differences
  - LCG transmits "lower level" information
  - LCG learns "finer" nogoods
  - LCG supports programmed search
  - Global Propagators $\approx$ Theories
- Sometimes the theory view is better:
  - modulo arithmetic + Radio Link Frequency Assignment
- Sometimes finer nogoods are better
  - separation logic + Open Shop Scheduling
- Eventually the approaches will merge!

Generalized Nogoods (g-nogoods)

- Nogood learning has a long history in Constraint Programming
  - longer than in SAT?
- Traditional Nogoods: $x_1 = d_1 \wedge \cdots \wedge x_n = d_n \rightarrow \mathit{fail}$
- Generalized Nogoods: $x_1 \overset{=}{\neq} d_1 \wedge \cdots \wedge x_n \overset{=}{\neq} d_n \rightarrow \mathit{fail}$
  - Introduced by Katsirelos and Bacchus 2003
  - Used SAT technology for propagation (watched literals)
  - Equivalent to lazy clause generation without bounds constraints
  - Interesting 1UIP nogoods not effective?
  - Also defined global explanation approach for `alldifferent`
  - Didnt consider activity, forgetting and VSIDS search

Mixed Integer Programming

- Strengths
  - Can deal with 100K variables 1M linear constraints
  - Strong autonomous search
  - "Knows" where the good solutions are
- Weaknesses
  - Have to model using only linear constraints

Can we get add the optimization strength of MIP to lazy clause generation?

SCIP: Solving Constraint Integer Programs

Hybrid constraint programming and mixed integer programming (MIP)

- Linear constraints as propagators and part of global MIP
- MIP propagator explains failures (and fathoming) as nogoods

$$x_1 \begin{array}{c}\leq\\ \geq\end{array} d_1 \wedge \cdots x_n \begin{array}{c}\leq\\ \geq\end{array} d_n \rightarrow \mathit{fail}$$

- Propagates these using SAT technology
- Creates ALLUIP nogoods for MIP failures
- Very good results on some hard MIP problems

Lazy Clause Generation and MIP?

- Mixed integer programming (MIP) solvers know where the good solutions are
- Lazy clause generation and MIP are compatible
  - MIP engine explains failure and fathoming (and reduced cost bounds changes)
  - Treated like an other global propagator
  - SCIP is a lazy clause generation MIP solver!
- In order to use the MIP advantage it probably directs search
- SCIP default search:
  - pseudo costs (MIP), then activity (SAT), then impact (CP)
- Plenty more to discover on the best interaction! (see our short paper)

Outline

Conclusion

Lazy Clause Generation

- High level modelling
- Strong nogood creation
- Effective autonomous search
- Global Constraints

Defines state-of-the-art for:

- Resource constrained project scheduling (minimize makespan)
- Set constraint problems
- Nonagrams (regular constraints)

Usually 1-2 order of magnitude speedup on FD problem

Future Research

Plenty of better engineering yet to be done

Plenty of open research questions

- Best combinination with MIP solving
- Hybrid search: structured + activity based
- Parallelism
- SAT Modulo Theories and Lazy Clause Generation
- Adaptive Behaviour

Questions