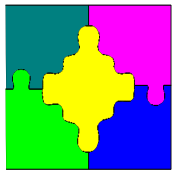
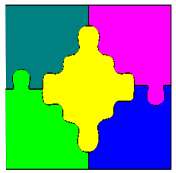


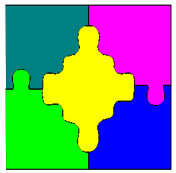
Lazy Clause Generation

A powerful hybrid solving approach combining SAT and finite domain propagation



Overview

- Original Lazy Clause Generation
  - Representing Integers
  - Explaining Propagators
  - Example
- Lazier Clause Generation
  - Lazy Variables
  - Views
  - Lazy Explanation
- Global Constraints
- Search

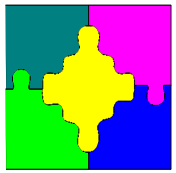


Lazy Clause Generation

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

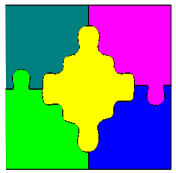
KEY INSIGHT:

- 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!



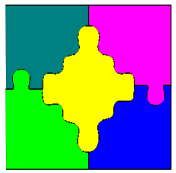
Representing Integers

- Integer x with initial domain $l..u$
  - Bounds Booleans: $[[x \leq d]]$, $l \leq d < u$
  - Equation Booleans: $[[x = d]]$, $l \leq d \leq u$
- (Efficient) Form of unary representation



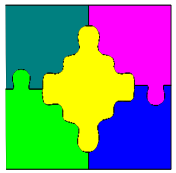
Representing Integers Exercise

- What domains are represented by
  1. $\{ [[x \leq 6]], \neg[[x \leq 2]] \}$
  2. $\{ [[x \leq 9]], \neg[[x \leq 4]], \neg[[x = 6]], \neg[[x = 8]] \}$
  3. $\{ [[x = 4]] \}$
  4. $\{ [[x \leq 5]], \neg[[x \leq 4]] \}$
  5. $\{ [[x \leq 7]], \neg[[x \leq 1]], \neg[[x = 8]] \}$
  6. $\{ [[x = 4]], [[x = 7]] \}$
  7. $\{\neg[[x \leq 7]], [[x \leq 1]] \}$



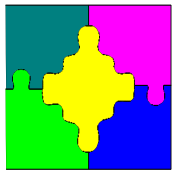
Domain Representation

- Need constraints to represent relationship amongst variables (DOM(x)):
  - $[[\, x \leq d \,]] \rightarrow [[\, x \leq d{+}1 \,]], \qquad l \leq d < u{-}1$
  - $[[\, x = d \,]] \Leftrightarrow [[\, x \leq d \,]] \wedge \neg\, [[\, x \leq d{-}1 \,]]$
- Ensures one to one correspondence between domains and assignments
- Note linear in size of domain



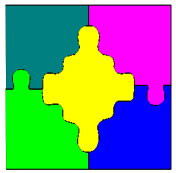
Atomic Constraints

- Atomic constraints define changes in domain
  - Fixing variable: $x = d$
  - Changing bound: $x \leq d$, $x \geq d$
  - Removing value: $x \neq d$
- Atomic constraints are just Boolean literals
  - $x = d \Leftrightarrow [\![x = d]\!]$
  - $x \leq d \Leftrightarrow [\![x \leq d]\!]$, $x \geq d \Leftrightarrow \neg [\![x \leq d\text{-}1]\!]$
  - $x \neq d \Leftrightarrow \neg [\![x = d]\!]$



Explaining Propagation

- For lazy clause generation: a propagator
  - must explain the domain changes it makes
- If $f(D) \neq D$ then propagator f returns an explanation for the atomic constraint changes
  - what parts of domain D forced the change
- Assume $D(x_1) = D(x_2) = D(x_3) = D(x_4) = D(x_5) = \{1..4\}$
- Example: alldifferent($[x_1, x_2, x_3, x_4]$)
  - $D(x_1) = \{1\}$ makes $D(x_2) = \{2..4\}$
  - Explanation: $x_1 = 1 \rightarrow x_2 \neq 1$



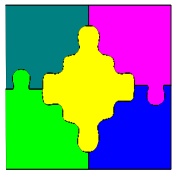
Explaining Propagation

- Explanations:

  implications of atomic constraints

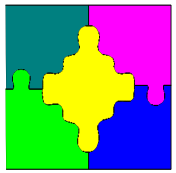
  = clauses on the Boolean literals

- $x_1 = 1 \rightarrow x_2 \neq 1$
- $[[\, x_1 = 1 \,]] \rightarrow \neg\, [[\, x_2 = 1 \,]]$
- $\neg [[\, x_1 = 1 \,]] \vee \neg [[\, x_2 = 1 \,]]$
- Unit propagation on the clause will cause the change in domain!



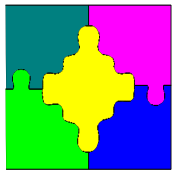
Explaining Propagation

- $x_2 \leq x_5$
  - $D(x_2) = \{2..4\}$ enforces $D(x_5) = \{2..4\}$
  - Explanation: $2 \leq x_2 \rightarrow 2 \leq x_5$
- $x_1+x_2+x_3+x_4 \leq 9$
  - $D(x_1) = \{1..4\}, D(x_2) = \{2..4\}, D(x_3) = \{3..4\}, D(x_4) = \{1..4\}$ enforces $D(x_4) = \{1..3\}$
  - Explanation: $2 \leq x_2 \wedge 3 \leq x_3 \rightarrow x_4 \leq 3$
  - Note: No $1 \leq x_1$ since this is universally true (initial domains)



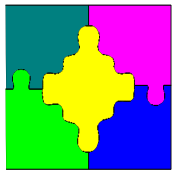
Explaining Failure

- When $f(D)(x) = \{\}$, failure detected
- The propagator must also explain failure
- alldifferent($[x_1, x_2, x_3, x_4]$)
  - $D(x_3) = \{3\}, D(x_4) = \{3\}$ gives failure
  - Explanation: $x_3 = 3 \wedge x_4 = 3 \rightarrow false$
- And
  - $D(x_1) = \{1,3\}, D(x_2) = \{1..3\}, D(x_3) = \{1,3\}, D(x_4) = \{1,3\}$!!!!
  - Explanation: $x_1 \leq 3 \wedge x_1 \neq 2 \wedge x_3 \leq 3 \wedge x_3 \neq 2 \wedge x_4 \leq 3 \wedge x_4 \neq 2 \rightarrow false$



Explanation Exercises

- Give the resulting domain and explanation for each of the following examples:
  - $D(x_1) = \{2..4\}, D(x_2) = \{1..4\}$: $x_1 + 1 \leq x_2$
  - $D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{1..2\}$, alldifferent($[x_1, x_2, x_3, x_4]$)
  - $D(x_1) = \{2,3\}, D(x_2) = \{1,4\}, D(b) = \{0,1\}, b \Leftrightarrow x_1 = x_2$
  - $D(x_1) = \{1..4\}, D(x_2) = \{1..4\}, D(x_3) = \{3\}, D(x_4) = \{1..4\}, 2x_1 + x_2 + 3x_3 + x_4 \leq 12$

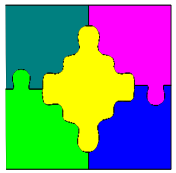


Minimal Explanations

- An explanation should be as general as possible
- **Question**: WHY?
- Sometimes there are multiple possible explanations, none better than others
- Example: $D(x_1) = \{4,6..9\}$, $D(x_2) = \{1..2\}$:

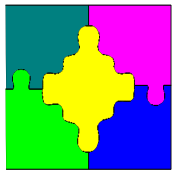
$$x_1 + 1 \leq x_2$$

  - $x_1 \geq 4 \wedge x_1 \neq 5 \wedge x_2 \leq 2 \rightarrow false$
  - $x_1 \geq 4 \wedge x_2 \leq 2 \rightarrow false$
  - $x_1 \geq 4 \wedge x_2 \leq 4 \rightarrow false$
  - $x_1 \geq 2 \wedge x_2 \leq 2 \rightarrow false$



Lazy Clause Generation

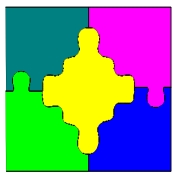
- Explanations are clauses
  - a lazy clausal representation of the propagator!
- Finite domain propagation is simply Boolean satisfaction generating the clauses defining the problem lazily
- Unit propagation on the explanations replaces finite domain propagation
- Nogood generation + activity based-search for free



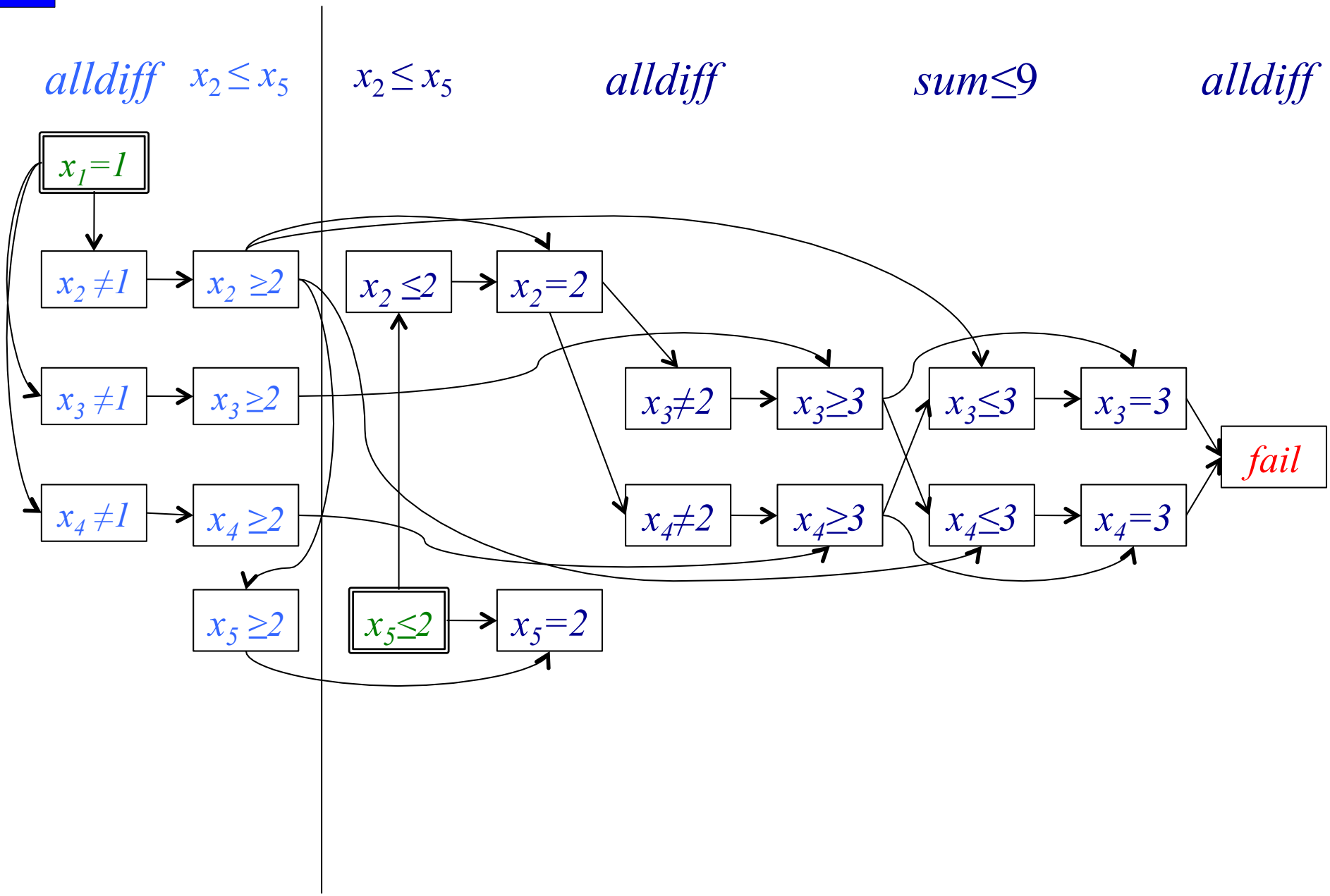
Finite Domain Propagation Ex.

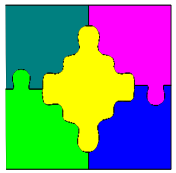
- $D(x_1) = D(x_2) = D(x_3) = D(x_4) = D(x_5) = \{1..4\}$
- $x_2 \leq x_5$, alldifferent($[x_1, x_2, x_3, x_4]$),
  $x_1+x_2+x_3+x_4 \leq 9$

	x_1=1	*alldiff*	$x_2 \leq x_5$	x_5>2	$x_2 \leq x_5$	*alldiff*	*sum*≤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*	✖
x_4	*1..4*	*2..4*	*2..4*	*2..4*	*2..4*	*3..4*	*3*	✖
x_5	*1..4*	*1..4*	*2..4*	*3..4*	*2*	*2*	*2*	*2*

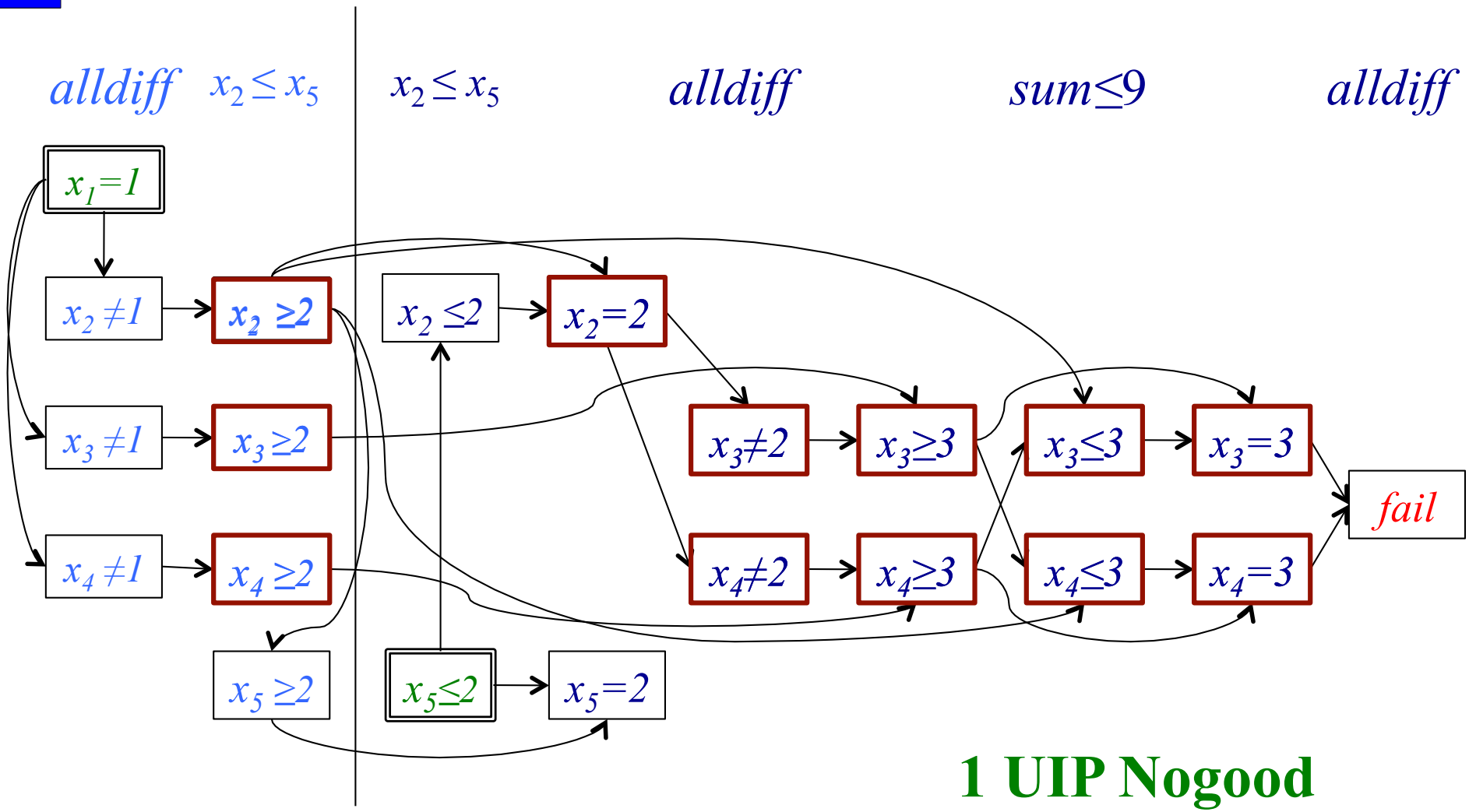


Lazy Clause Generation Ex.





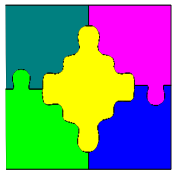
1UIP Nogood Creation



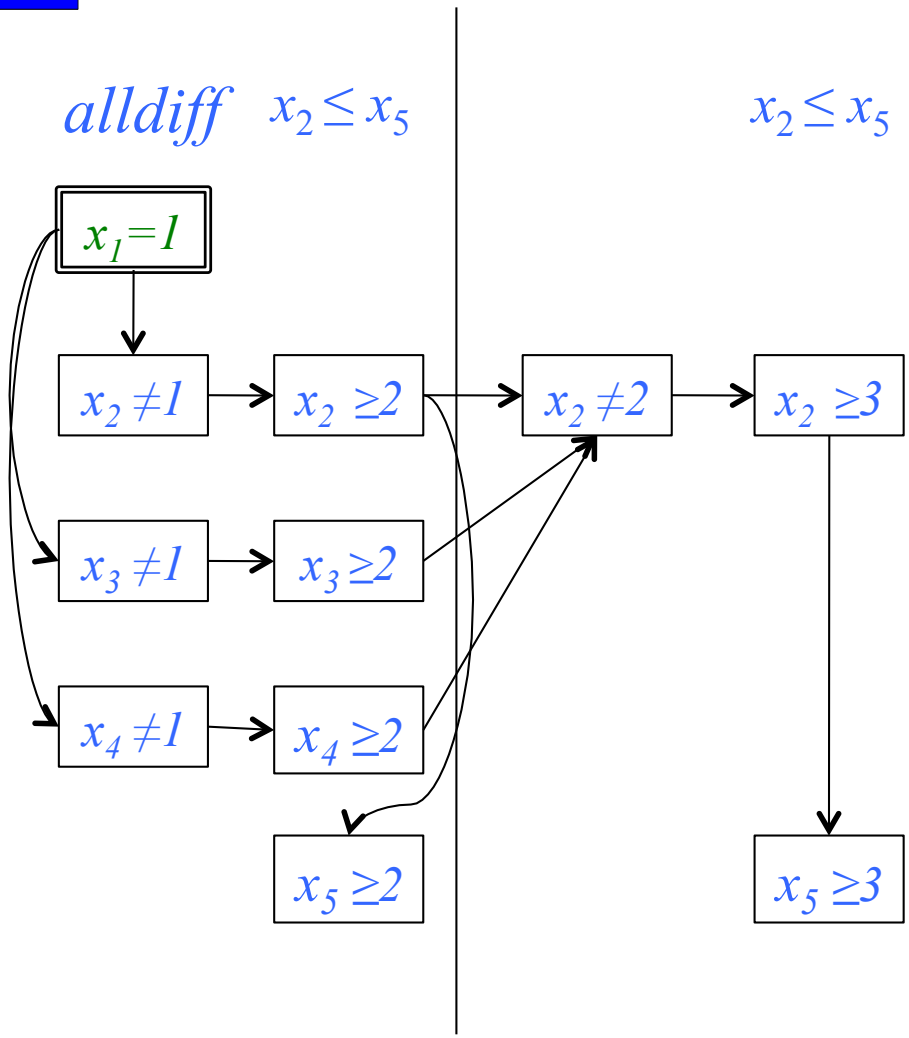
$\{x_2 \geq 2,\ x_3 \geq 2,\ x_4 \geq 2,\ x_2 = 2\}$ ➔ *false*

1 UIP Nogood

$\{[[x_2 \leq 1]], [[x_3 \leq 1]], [[x_4 \leq 1]], \neg[[x_2 = 2]]\}$

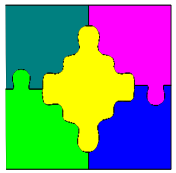


Backjumping



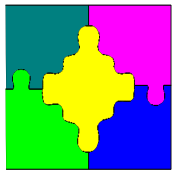
- Backtrack to second last level in nogood
- Nogood will propagate
- Note stronger domain than usual backtracking
  - $D(x_2) = \{3..4\}$

$\{x_2 \geq 2,\ x_3 \geq 2,\ x_4 \geq 2,\ x_2 = 2\}$ ➔ *false*



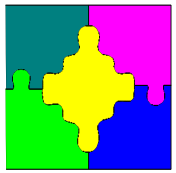
Whats Really Happening

- 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



Lazy Clause Generation Exercise

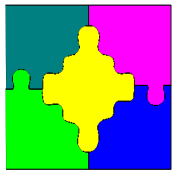
- Given constraints: $b1 \vee b2, b1 \Leftrightarrow x1 \leq x6$, b2$\Leftrightarrow x1 \geq 4, x1 + x2 + x3 + x4 \leq 11, x4 \geq x5, x3 \geq x5, x5 + x6 \leq 8$
- $D(x1) = D(x2) = D(x3) = D(x4) = D(x5) = D(x6) = \{1..5\}$
- Assume decisions in order: $x6 \geq 4, x5 \geq 2, x2 \geq 4$
- Build the implication graph, determine the 1UIP nogood. Show the result after backjumping!



Comparing Versus SAT

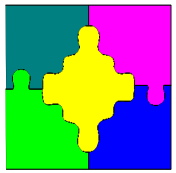
- For some models we can generate all possible explanation clauses before commencement
  - usually this is too big
- Open Shop Scheduling (tai benchmark suite)
  - averages

	Time	Solve only	Fails	Max Clauses
SAT	318	89	3597	13.17
LCG	62		6651	1.0



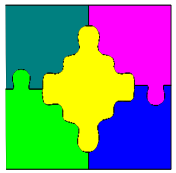
Strengths + Weaknesses

- 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



Lazy 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× bounds clauses
  - Create equality variables on demand as before

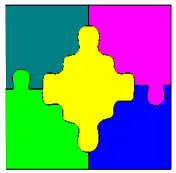


Lazy Variable Creation

List versus array

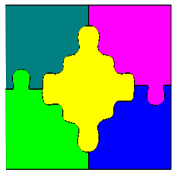
- List always works!
- Array may require too many Boolean variables
- Implementation complexity
- List hampers learning
- Tai open shop scheduling: 15x15 (average of 10 problems)

	Time
Array	13.38
List	56.66



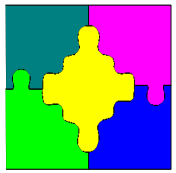
Views (Schulte + Tack 2005)

- View is a pseudo variable defined by a "bijective" function to another variable
  - $x = \alpha y + \beta$
  - $x = \text{bool2int}(y)$
  - $x = \neg y$
- 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)



Views

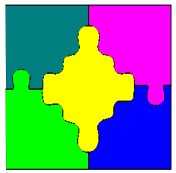
- Operations on $x = \alpha y + \beta$
- Lower bound (similar for upper bound)
  - $\mathsf{lb}(x) = \alpha\ \mathsf{lb}(y) + \beta$, when $\alpha > 0$
  - $\mathsf{lb}(x) = \alpha\ \mathsf{ub}(y) + \beta$, when $\alpha < 0$
- Set lower bound
  - $\mathsf{setlb}(x,d) = \mathsf{setlb}(y, \mathsf{ceil}((d - \beta) / \alpha))$, when $\alpha > 0$
  - $\mathsf{setlb}(x,d) = \mathsf{setub}(y, \mathsf{floor}((d - \beta) / \alpha))$, when $\alpha < 0$
- Difficulties
  - reasoning about idempotence can be tricky



Advantages of Views

- Constrained path covering problems:
- A view to implement array lookups, versus no view
- Average over 5 instances

	Time	Fails
views	0.71	950
no views	1.12	1231



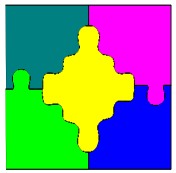
Explanation Deletion

- 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 (times):

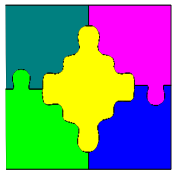
	15x15	20x20
deletion	13.38	39.96
no deletion	20.58	95.88

But worse on other benchmarks



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) and SAT Modulo Theories (SMT)
- Only create needed explanations!
- Harder implementation

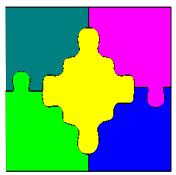


Advantages of Lazy Explanation

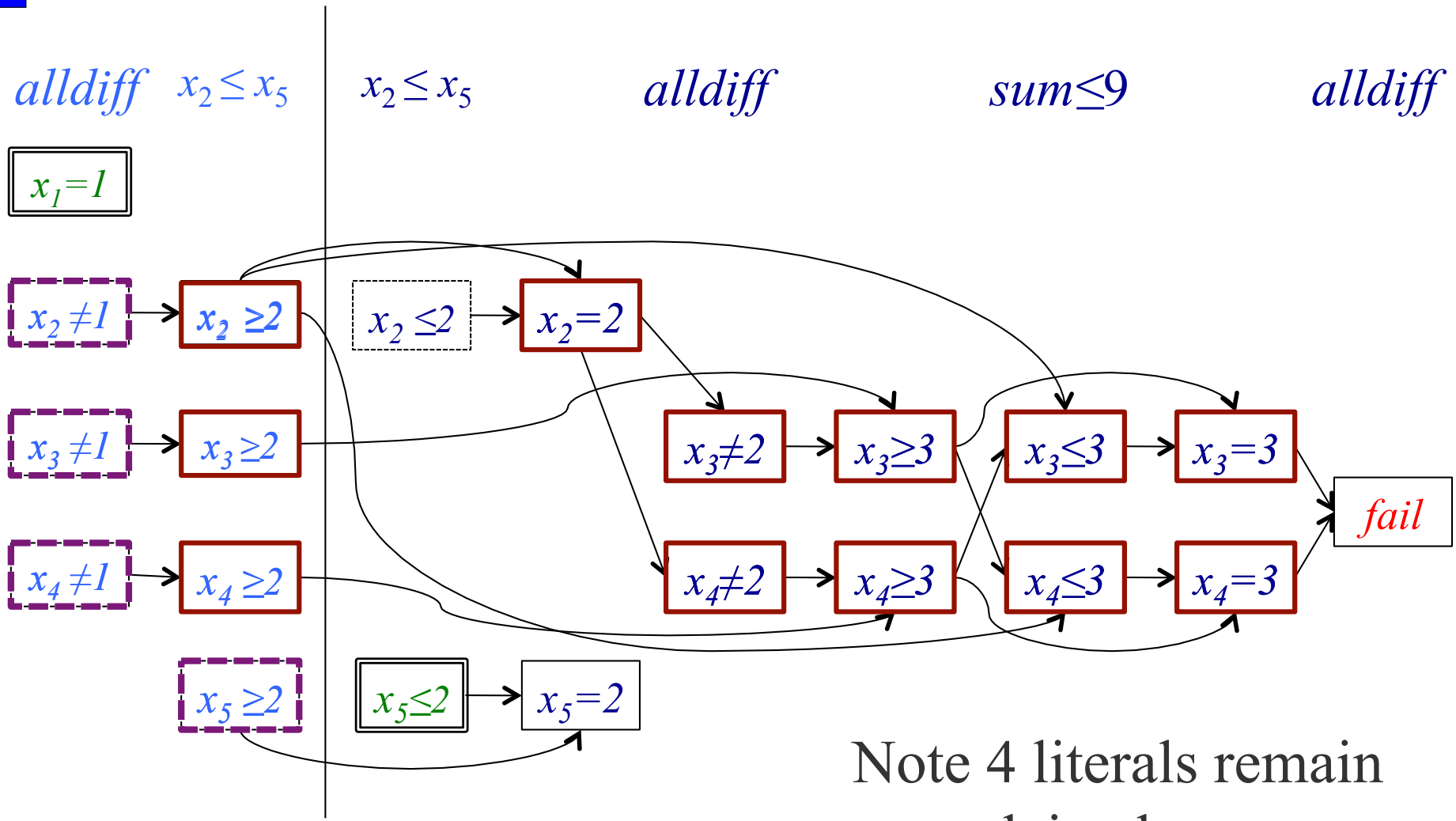
- Social Golfers Problems:
  - using a `regular` propagator
  - each explanation as expensive as running entire propagator

	Times	Reasons	Fails
lazy explanation	2.38	14387	2751
eager explanation	4.92	78177	5126

- Surprisingly not as advantageous as it seems

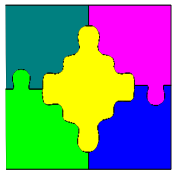


Lazy Explanation Example



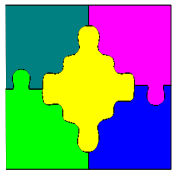
Note 4 literals remain unexplained

$\{x_2 \geq 2,\ x_3 \geq 2,\ x_4 \geq 2,\ x_2 = 2\}$ ➔ *false*



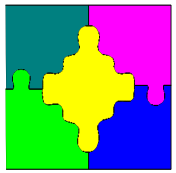
The Globality of Explanation

- 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 $b = true$ and 200000 propagations later failure.
- A global difference logic propagator immediately sets $b = false$!
- Lazy clause generation learns $b = false$ after 200000 propagations
  - But never tries it again!



Globals by Decomposition

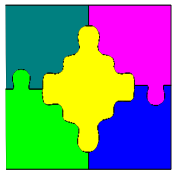
- Globals defined by decomposition
  - Don't require implementation
  - Automatically incremental
  - Allow partial state relationships to be "learned"
  - Much more attractive with lazy clause generation
- When propagation is not hampered, and size does not blowout:
  - can be good enough!



Which Decomposition?

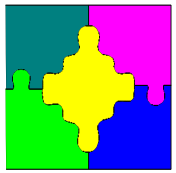
- `alldifferent` decompositions
  - **diseq:** $O(n^2)$ disequalities
  - **bnd**: bounds consistent decomposition
  - **bnd+**: Bound consistent decomposition
    - replacing $x \geq d \wedge x \leq d$ by $x = d$
  - **gcc**: based on global cardinality constraint
- Quasi-group completion 25x25 (average of examples solved by all)

diseq		bnd		bnd+		gcc	
Time	Fails	Time	Fails	Time	Fails	Time	Fails
131	1426080	757	9317	129	1144	4.3	1010

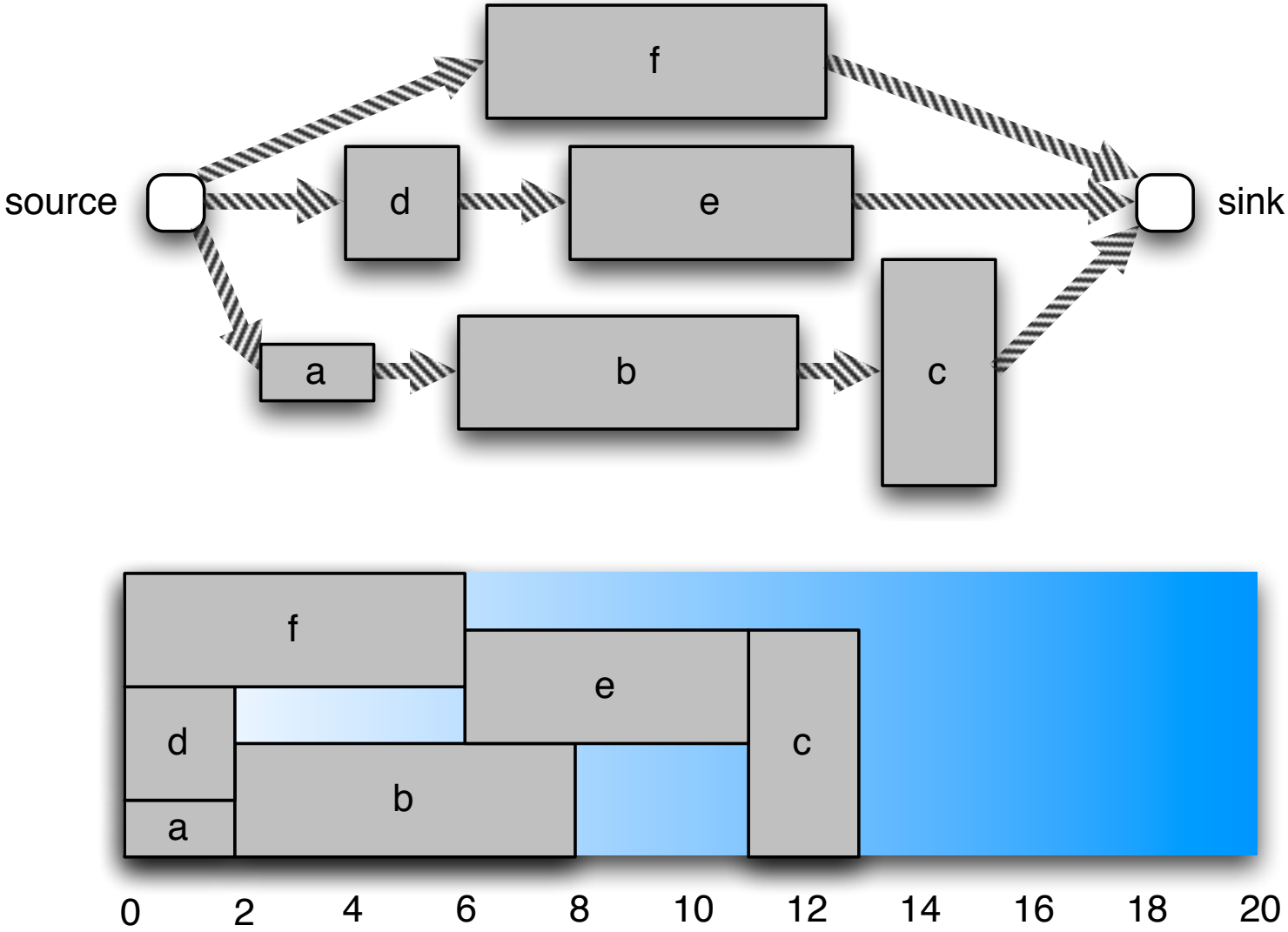


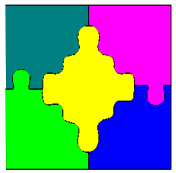
Explanation for Globals

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



Explaining cumulative





Explaining cumulative

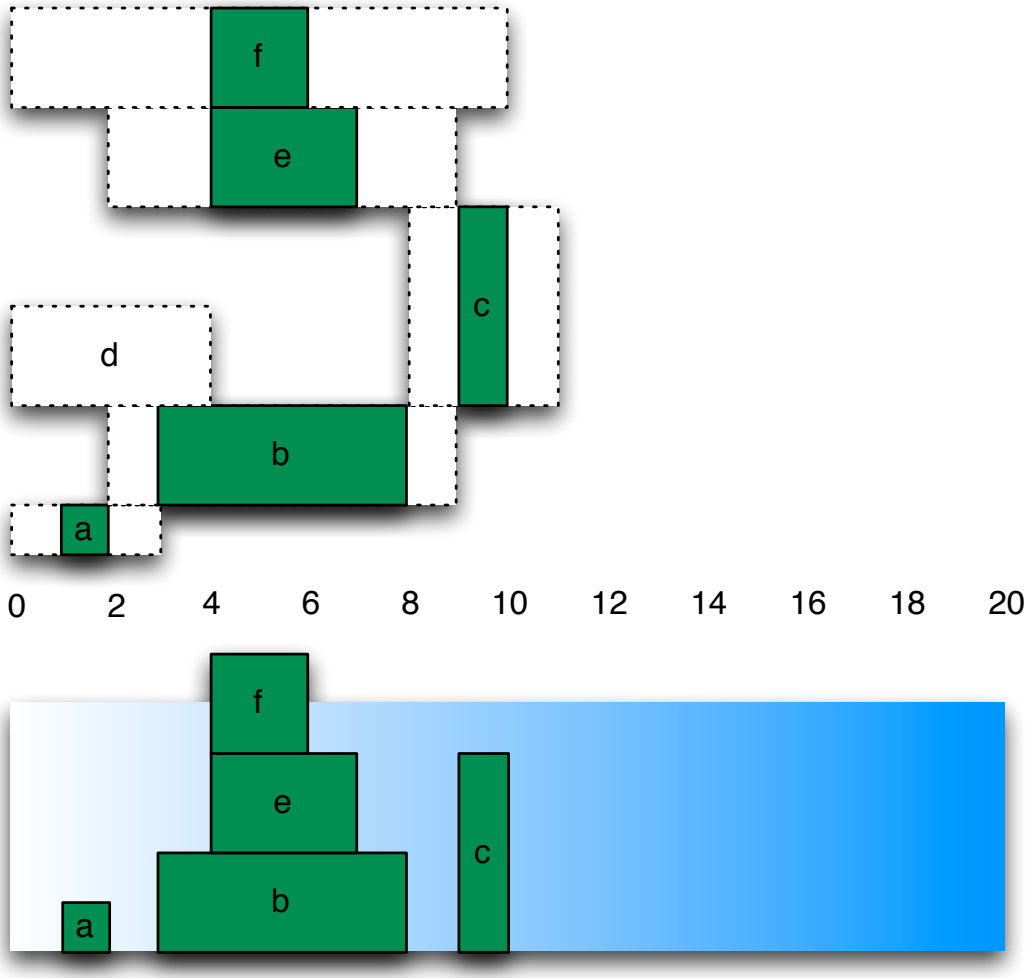
Explaining Failure

- Compulsory parts are too high:
- Naïve: just use current bounds:

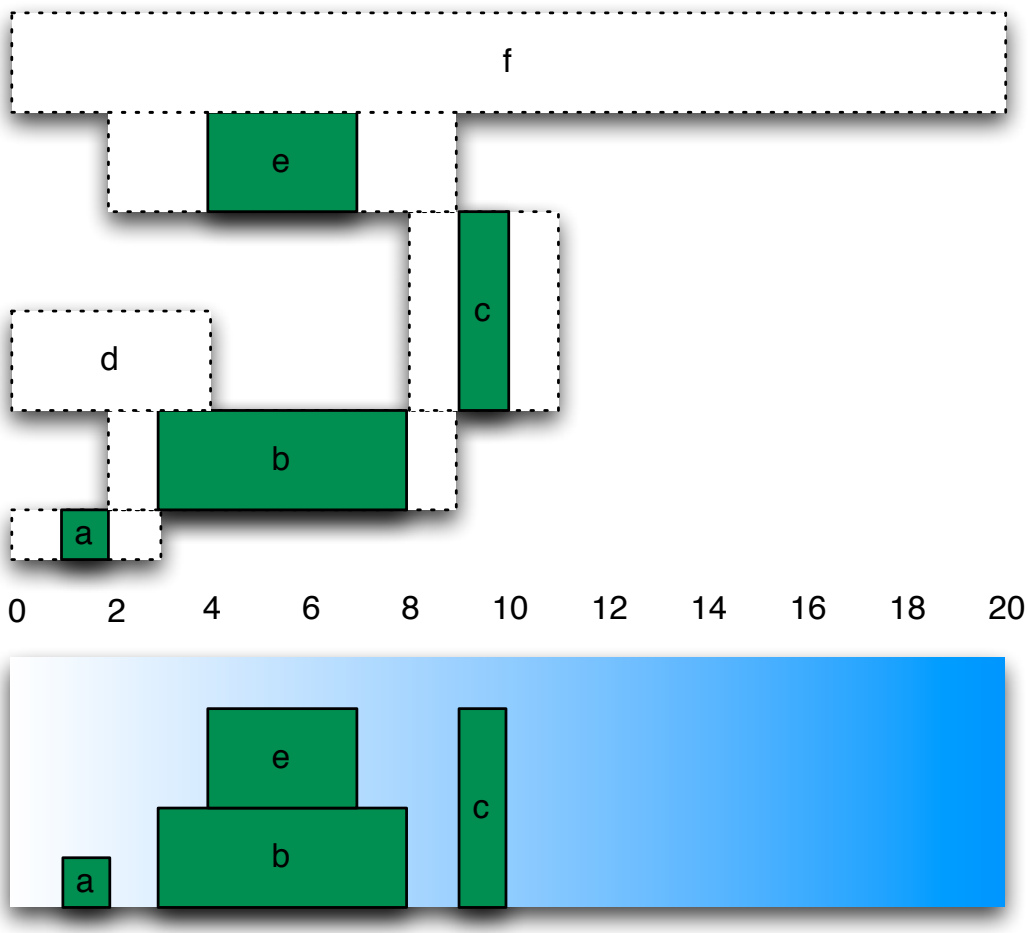
  $2 \le sb \le 3 \ \wedge \ 2 \le se \le 4 \ \wedge \ 0 \le sf \le 4 \rightarrow false$
- Pointwise: pick a point (4) and loosen bounds:

  $0 \le sb \le 4 \ \wedge \ 0 \le se \le 4 \ \wedge \ 0 \le sf \le 4 \rightarrow false$

Stronger Explanation



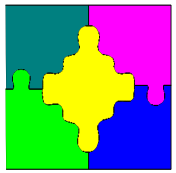
Explaining cumulative



Explaining propagation:

- *f* cant start before 10
- Naïve explanation

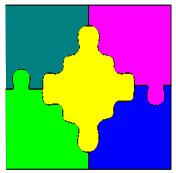
  $2 \le sb \le 3 \wedge 2 \le se \le 4 \wedge 8 \le sc \le 9 \rightarrow 10 \le sf$
- Pointwise explanation:
  - *f* cant start before 5 because of *b* and *e*
  - $0 \le sb \le 4 \wedge 0 \le se \le 4 \rightarrow 5 \le sf$
  - *f* cant start before 10 because of *c*
  - $8 \le sc \le 9 \wedge 5 \le sf \rightarrow 10 \le sf$



Search

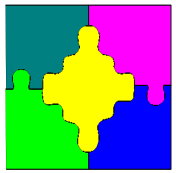
- Contrary to SAT folklore
  - Activity based search can be terrible
  - Nogoods work excellently with programmed search
- Constrained path covering problems

	Time	Fails
lcg + VSIDS	>361.89	>30,000
lcg + programmed	0.71	950
programmed	>240.2	>10,000



Activity-based search

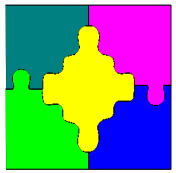
- An excellent default search!
- Weak at the beginning (no meaningful activities)
- Need hybrid approachs
  - Hot Restart:
    - Start with programmed search to “initialize” meaningful activities.
    - Switch to activity-based after restart
  - Alternating
    - Start with programmed search, switch to activity-based on restart
    - Switch search type on each restart
- Much more to explore in this direction



Lazy Clause Generation in MiniZinc

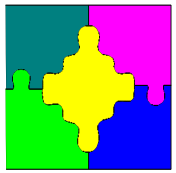
- mzn –b lazy will invoke our original lazy clause generation solver
- mzn2fzn –l <ChuffedGlobalsDir> model.mzn
  fzn_geoff model.fzn

  invokes our latest lazy clause generation solver
- Try them out on some models you have previously created in the subject!



Summary

- Lazy Clause Generation
  - High level modelling
  - Strong nogood creation
  - Effective autonomous search (but also programmed 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



Exercise 1: Explaining abs

- Write pseudo-code for the absolute value propagator: $x = \text{abs}(y)$
  - use lb(v), ub(v) and dom(v) to access the domain of variable v
  - use setlb(v,d), setub(v,d), setdom(v,S) to set the domain of v (you can assume these update monotonically e.g. dom(setdom(v,S)) = S intersect dom (v))
- Now modify the pseudo-code to explain each change of domain.