NICTA

Those who forget the past are doomed to repeat it

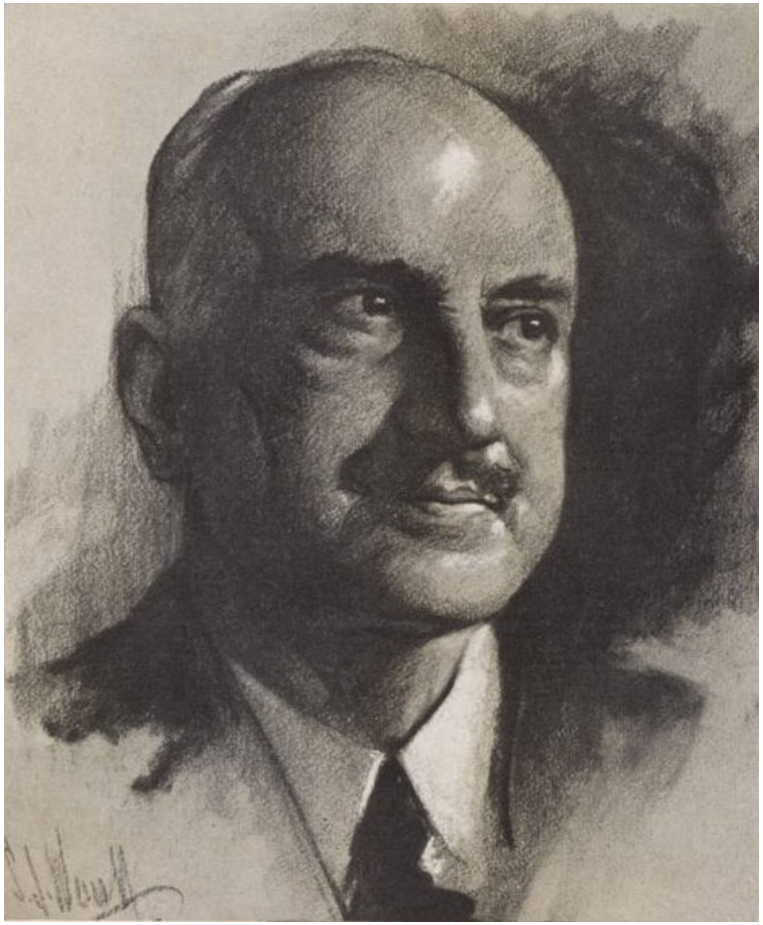
Peter J. Stuckey and countless others!

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW Government Trade & Investment
State Government Victoria
The University of Melbourne
The University of Sydney
Queensland Government
Griffith University
QUT Queensland University of Technology
The University Of Queensland Australia

Those who forget the past are doomed to repeat it



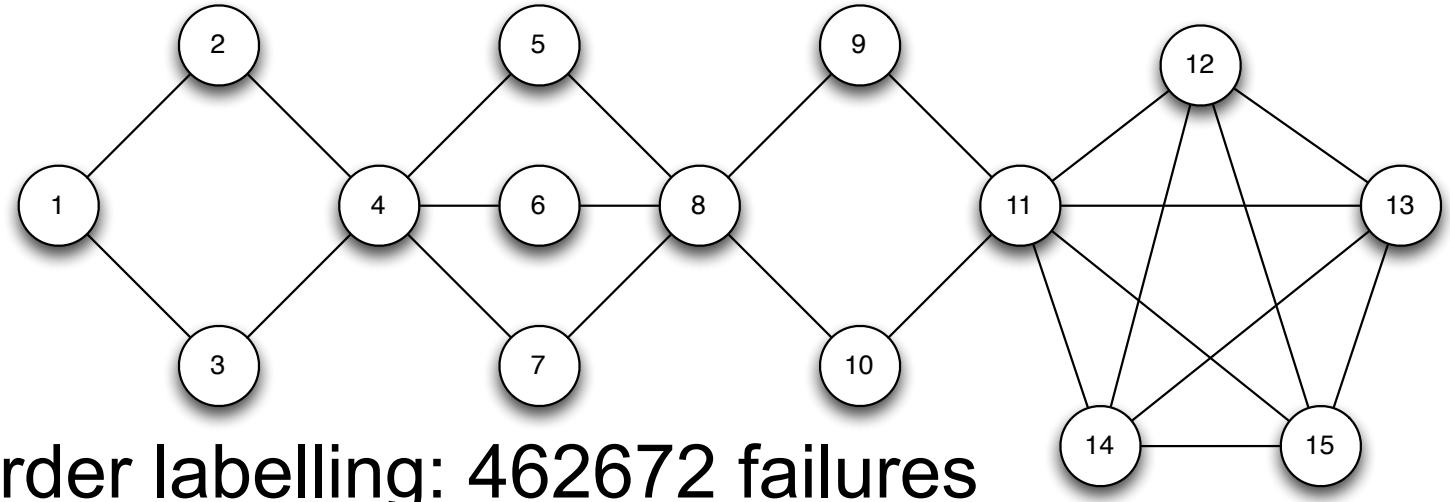
Jorge Agustín Nicolás Ruiz de Santayana y Borrás,
"Nunca me hablas de estas tema otra vez"
George Santayana, 1863-1952

Conspirators

- Ignasi Abio, Ralph Becket, Sebastian Brand, Geoffrey Chu, Michael Codish, Greg Duck, Nick Downing, Thibaut Feydy, Kathryn Francis, Graeme Gange, Vitaly Lagoon, Amit Metodi, Nick Nethercote, Roberto Nieuwenhuis, Olga Ohrimenko, Albert Oliveras, Enric Rodriguez Carbonell, Andreas Schutt, Guido Tack, Pascal Van Hentenryck, Mark Wallace
- All errors and outrageous lies are mine

How much of CP search is repeated?

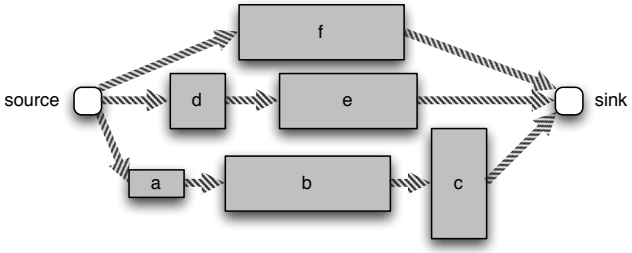
- 4 colour the graph below

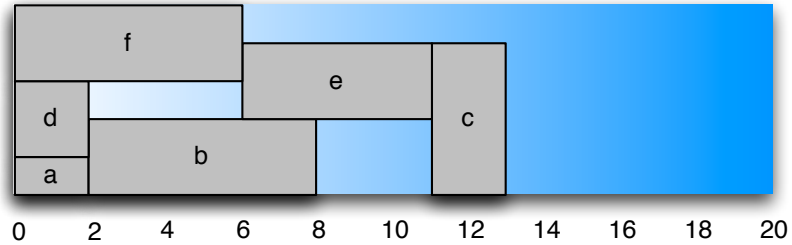


- Inorder labelling: 462672 failures
  - With learning: 18 failures
- Value symmetries removed: 19728 failures
  - With learning: 19 failures
- Reverse labelling: 24 failures
  - With learning: 18 failures

How much of CP search is repeated?

- Resource Constrained Project Scheduling
  - BL instance (20 tasks)





- Input order: 934,535 failures
  - With learning: 931 failures
- Smallest start time order: 296,567 failures
  - With learning: 551 failures
- Activity-based search: > 2,000,000 failures
  - With learning: 1144 failures

How much of CP search is repeated?

NICTA

- Short answer: a lot

- Methods to alleviate the problem
  - Symmetry/dominance handling
  - Restarts + dynamic search strategies
  - Learning/Caching

Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks

Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks

Propagation Solving (CP)

NICTA

- Complete solver for atomic constraints
  - $x = d$, $x \neq d$, $x \geq d$, $x \leq d$
  - Domain $D(x)$ records the result of solving (!)
- Propagators infer new atomic constraints from old ones
  - $x_2 \leq x_5$ infers from $x_2 \geq 2$ that $x_5 \geq 2$
  - $x_1+x_2+x_3+x_4 \leq 9$ infers from $x_1 \geq 1 \wedge x_2 \geq 2 \wedge x_3 \geq 3$ that $x_4 \leq 3$
- Inference is interleaved with search
  - Try adding c if that fails add *not* c
- Optimization is repeated solving
  - Find solution $obj = k$ resolve with $obj < k$

Finite Domain Propagation Ex.

NICTA

```
array[1..5] of var 1..4: x;
constraint alldifferent([x[1],x[2],x[3],x[4]);
constraint x[2] <= x[5];
constraint x[1] + x[2] + x[3] + x[4] <= 9;
```

	x_1=1	*alldiff*	$x_2 \le x_5$	x_5>2	$x_2 \le x_5$	*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*

FD propagation

NICTA

- Strengths
  - High level modelling
  - Specialized global propagators capture substructure
    - and all work together
  - Programmable search
- Weaknesses
  - Weak autonomous search (improved recently)
  - Optimization by repeated satisfaction
  - Small models can be intractable

Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks

Lazy Clause Generation (LCG)

NICTA

- A hybrid SAT and CP solving approach
- Add explanation and nogood learning to a propagation based solver
- Key change
  - Modify propagators to explain their inferences as clauses
  - Propagate these clauses to build up an implication graph
  - Use SAT conflict resolution on the implication graph

LCG in a Nutshell

NICTA

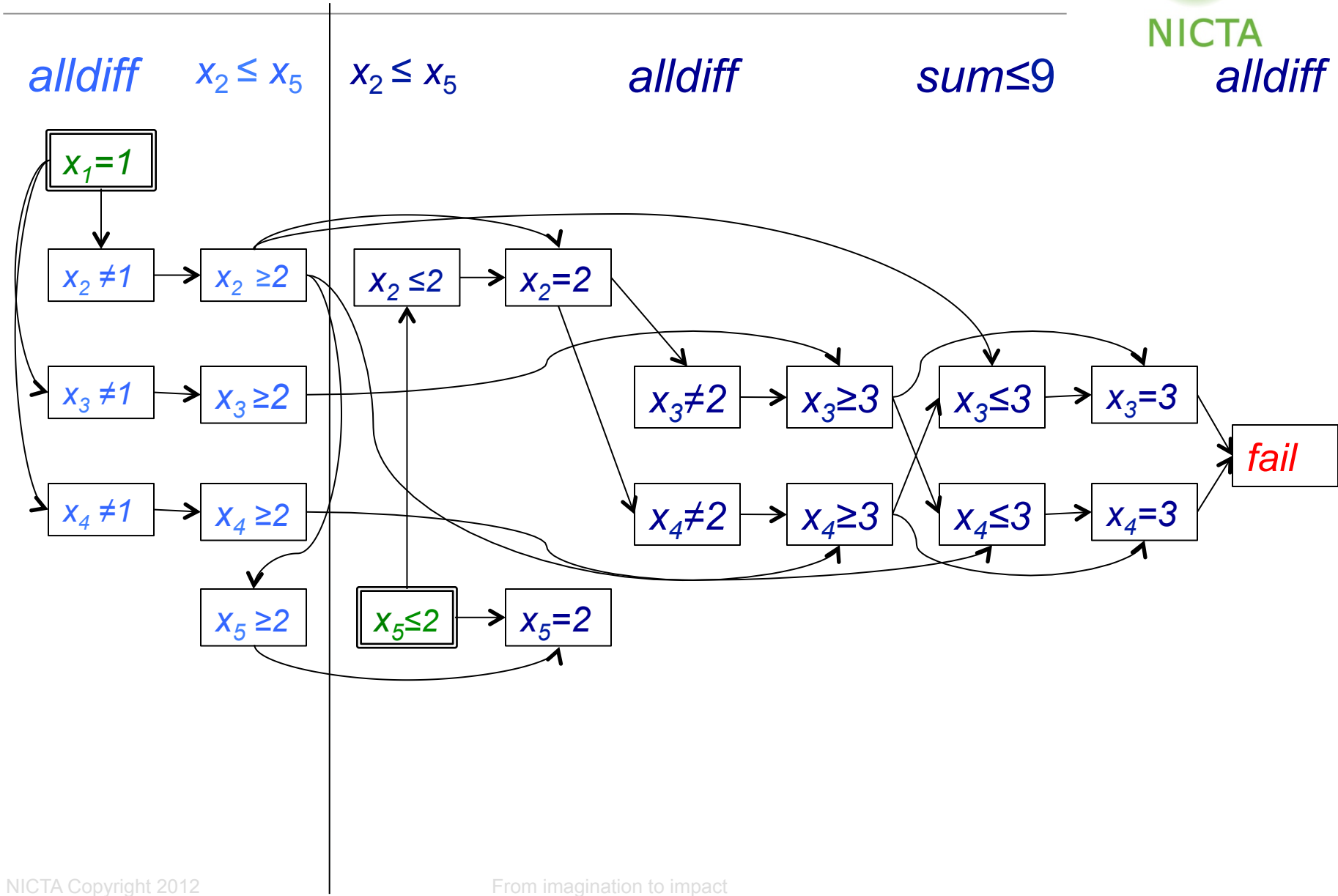
- Integer variable x in $l..u$ encoded as Booleans
  - $[x \leq d]$, d in $l..u$-1
  - $[x = d]$, d in $l..u$
- Dual representation of domain $D(x)$
- Restrict to atomic changes in domain (literals)
  - $x \leq d$ (itself)
  - $x \geq d$! $[x \leq d\text{-}1]$ use $[x \geq d]$ as shorthand
  - $x = d$ (itself)
  - $x \neq d$! $[x = d]$ use $[x \neq d]$ as shorthand
- Clauses DOM to model relationship of Booleans
  - $[x \leq d] \rightarrow [x \leq d+1]$, d in $l..u$-2
  - $[x = d] \Leftrightarrow [x \leq d] \wedge ! [x \leq d\text{-}1]$, d in l+1..u-1

LCG in a Nutshell

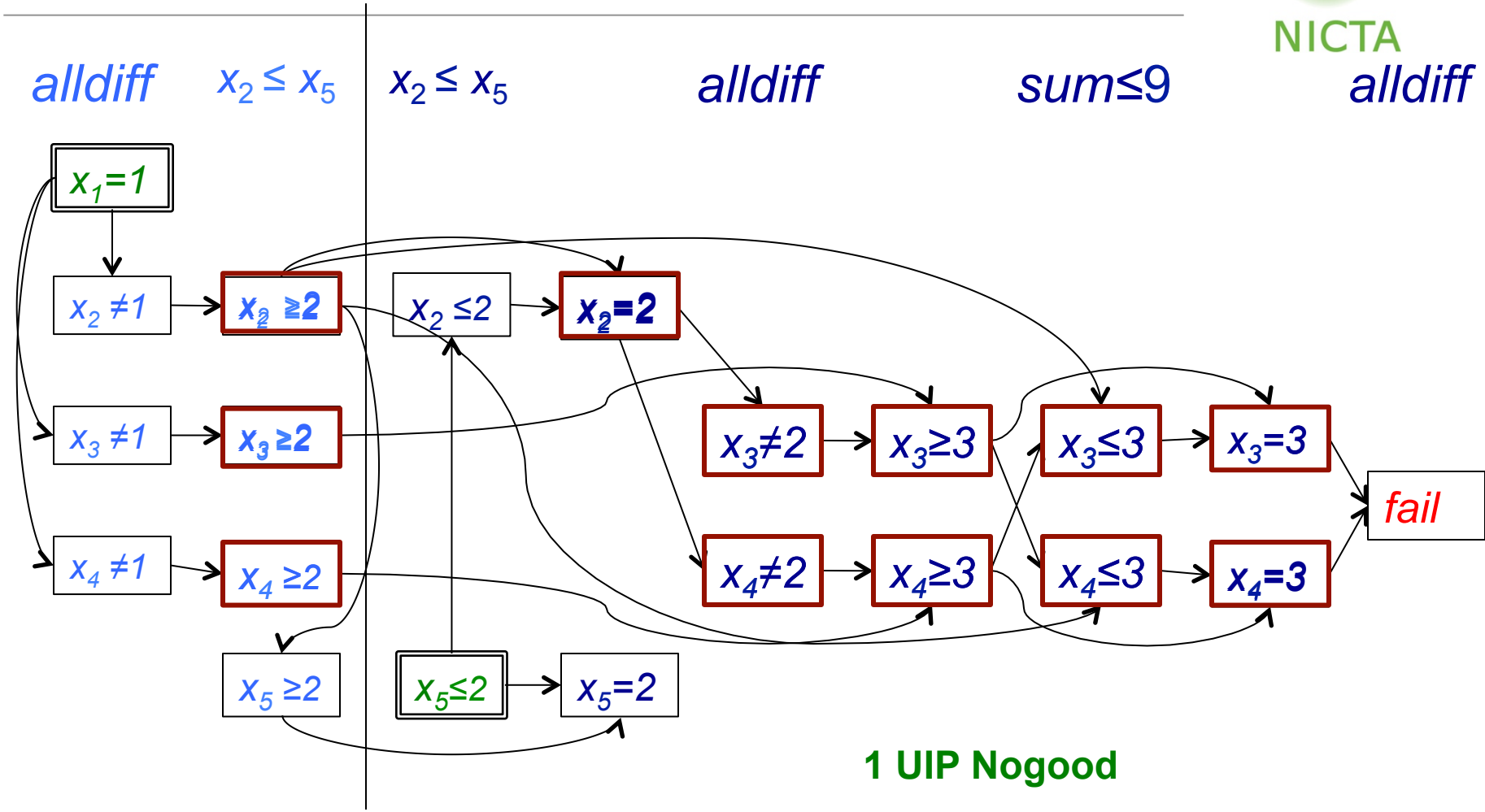
NICTA

- Propagation is clause generation
  - e.g. $[x \leq 2]$ and $x \geq y$ means that $[y \leq 2]$
  - clause $[x \leq 2] \rightarrow [y \leq 2]$
- Consider
  - `alldifferent([x[1],x[2],x[3],x[4]);`
- Setting $x_1 = 1$ we generate new inferences
  - $x_2 \neq 1$, $x_3 \neq 1$, $x_4 \neq 1$
- Add clauses
  - $[x_1 = 1] \rightarrow [x_2 \neq 1]$, $[x_1 = 1] \rightarrow [x_3 \neq 1]$, $[x_1 = 1] \rightarrow [x_4 \neq 1]$
  - i.e. $![x_1 = 1] \vee ![x_2 = 1]$, …
- Propagate these new clauses

Lazy Clause Generation Ex.



1UIP Nogood Creation

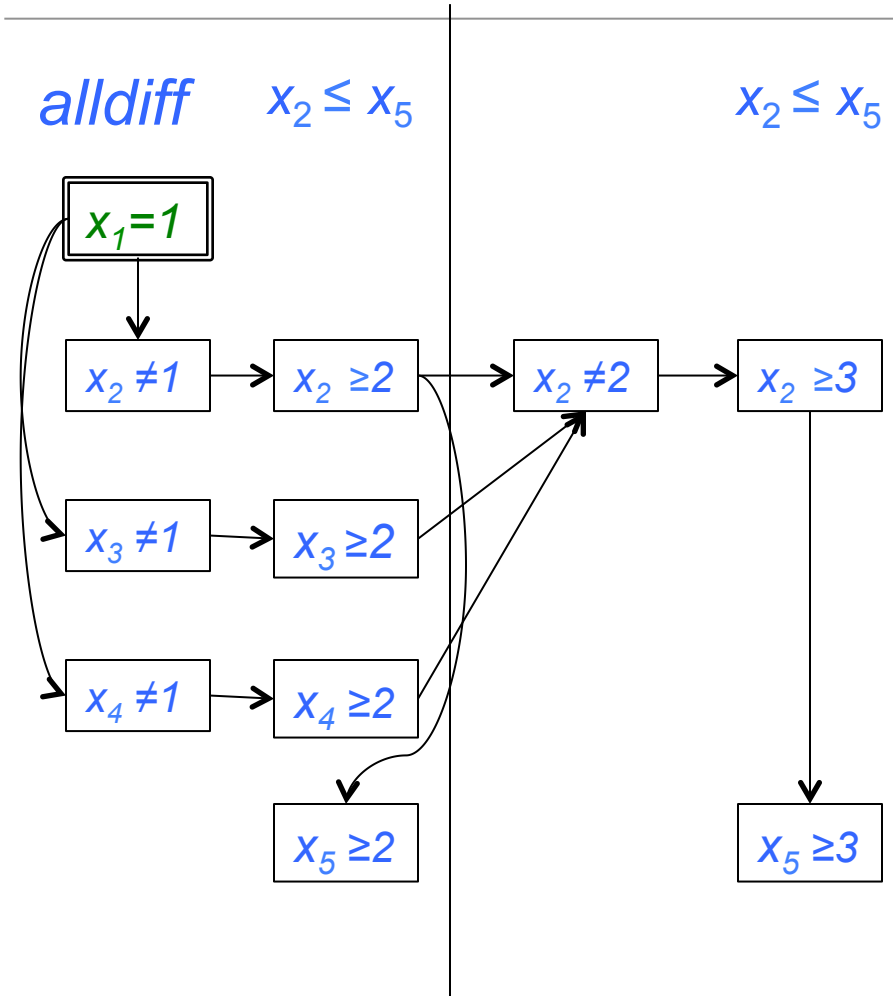


1 UIP Nogood

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

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

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*

What's Really Happening

NICTA

- CP model = high level "Boolean" model
- 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 to 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

Lazy Clause Generation

NICTA

- 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

LCG for CSPs

NICTA

- If you are solving extensional CSPs
  - LCG ≅ SAT
- Hard to beat SAT on non-numeric CSPs
- Positive table of n tuples of length k
  - $k \times n$ binary clauses
  - 1 n-ary clause
  - (for domain propagation) $k \times n$ literals in reverse clauses
  - Actually we can do better with MDDs
- Negative table of n tuples of length k
  - n k-ary clauses

Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks

LCG Successes

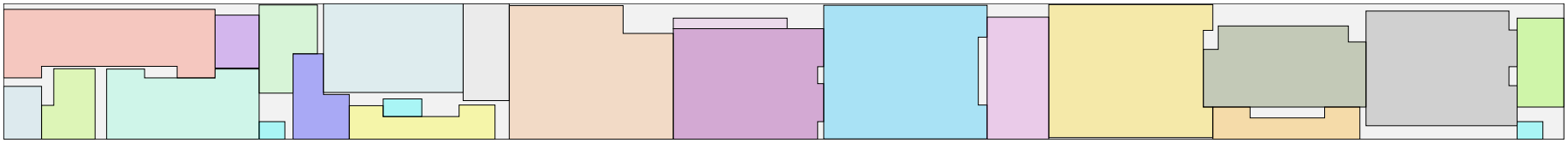
NICTA

- Scheduling
  - Resource Constrained Project Scheduling Problems (RCPSP)
    - (probably) the most studied scheduling problems
    - LCG closed 71 open problems
    - Solves more problems in 18s then previous SOTA in 1800s
  - RCPSP/Max (more complex precedence constraints)
    - LCG closed 578 open instances of 631
    - LCG recreates or betters all best known solutions by any method on 2340 instances except 3
  - RCPSP/DC (discounted cashflow)
    - Always finds solution on 19440 instances, optimal in all but 152 (versus 832 in previous SOTA)
    - LCG is the SOTA complete method for this problem

LCG Successes

NICTA

- Real World Application
  - Carpet Cutting
    - Complex packing problem
    - Cut carpet pieces from a roll to minimize length
    - Data from deployed solution



  - Lazy Clause Generation Solution
    - First approach to find and prove optimal solutions
    - Faster than the current deployed solution
    - Reduces waste by 35%

LCG Successes

NICTA

- MiniZinc Challenge
  - comparing CP solvers on a series of challenging problems
  - Competitors
    - CP solvers such as Gecode, Eclipse, SICstus Prolog
    - MIP solvers SCIP, CPLEX, Gurobi (encoding by us)
    - Decompositions to SMT and SAT solvers
  - LCG solvers (from our group) were
    - First (Chuffed) and Second (CPX) in all categories in 2011 and 2012
    - First (Chuffed) in all categories in 2010
  - Illustrates that the approach is strongly beneficial on a wide range of problems

Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks

Improving Lazy Clause Generation

NICTA

- Don’t Save Explanations
- Lazy Literal Generation
- Lazy (Backwards) Explanation
- The Globality of Explanation
- Explaining Global Constraints
- Search for LCG
- Symmetries and LCG

Don’t Save Explanations

NICTA

- Explanation clauses are only needed for conflict resolution
  - Don’t record them in the SAT solver
  - Just record them in the implication graph
  - Throw them away on backjumping
- Advantages
  - Less memory required
  - Faster
- Disadvantages
  - Memoizing complex explanations
  - Reprioritizing propagation to follow earlier paths
  - All our scheduling results save explanations

Lazy Literal Generation

NICTA

- Generate Boolean literals representing integer variables on demand
- E.g.
  - decision $x_1 = 1$ generates literal $[x_1 = 1]$
  - alldiff generates $[x_2 \geq 2]$ (equivalently $![x_2 \neq 1]$)
- Integer domain maintains relationship of literals
  - DOM clauses disappear
- A bit tricky to implement efficiently

Lazy Literal Generation

- For constraint problems over large domains lazy literal generation is crucial (MiniZinc Chall. 2012)

	amaze	fastfood	filters	league	mspsp	nonogram	patt-set
Initial	8690	1043k	8204	341k	13534	448k	19916
Root	6409	729k	6944	211k	9779	364k	19795
Created	**2214**	**9831**	**1310**	**967**	**6832**	**262k**	**15490**
Percent	34%	1.3%	19%	0.45%	70%	72%	78%

	proj-plan	radiation	shipshed	solbat	still-life	tpp
Initial	18720	145k	2071k	12144	18947	19335
Root	18478	43144	2071k	9326	12737	18976
Created	**5489**	**1993**	**12943**	**10398**	**3666**	**9232**
Percent	30%	4.6%	0.62%	111%	29%	49%

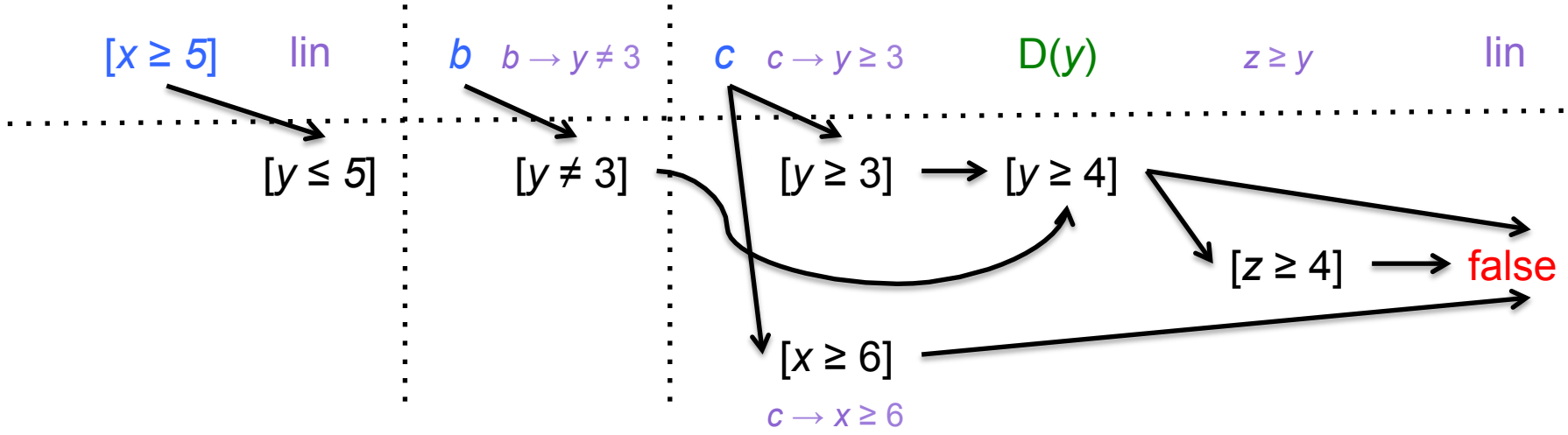
Lazy Explanation

NICTA

- Explanations only needed for nogood learning
  - Forward: record propagator causing atomic constraint
  - Backward: ask propagator to explain the constraint
- Standard for SMT and SAT extensions
- Only create needed explanations
- Scope for:
  - Explaining a more general failure than occurred
  - Making use of the current nogood in choosing an explanation
- Interacts well with lazy literal generation

(Original) LCG propagation example

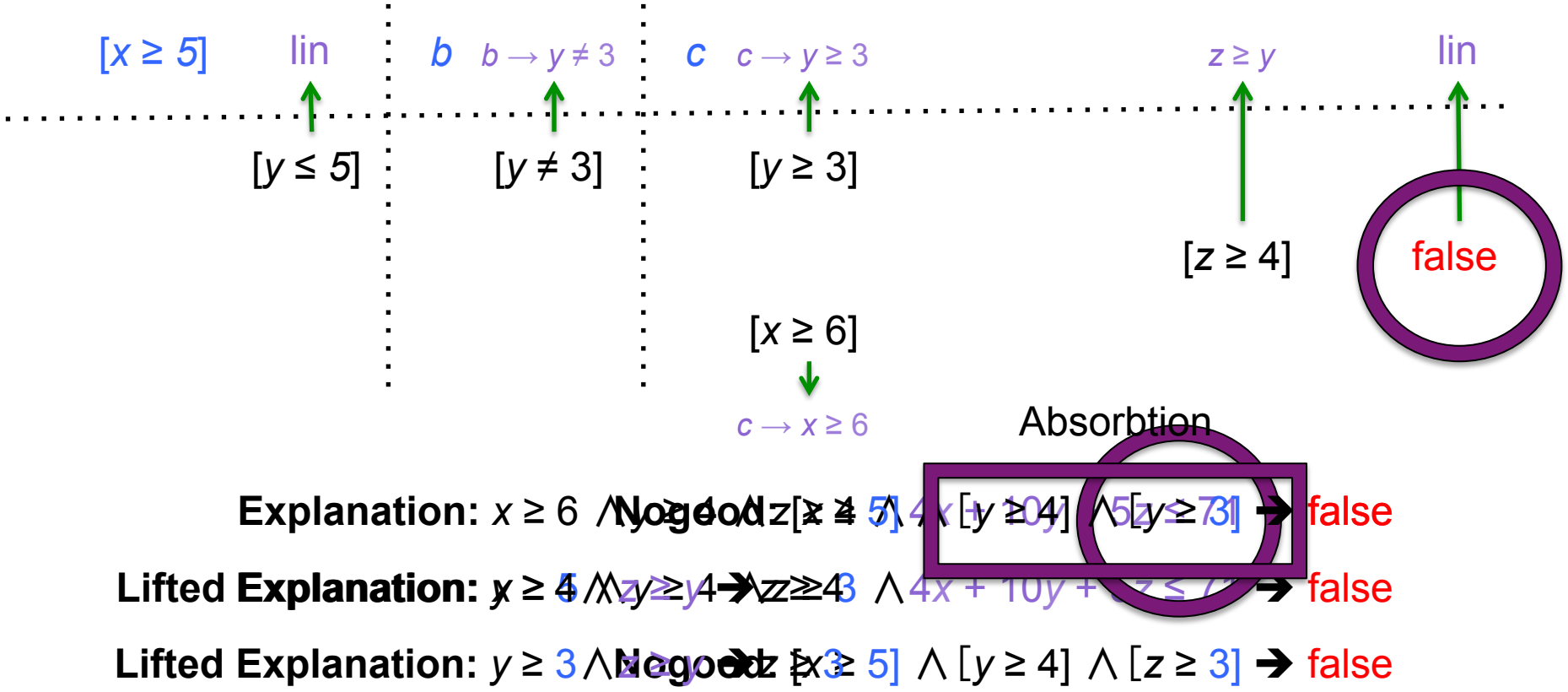
- Variables: $\{x,y,z\}$ $D(v) = [0..6]$ Booleans b,c
- Constraints:
  - $z \geq y$, $b \rightarrow y \neq 3$, $c \rightarrow y \geq 3$, $c \rightarrow x \geq 6$,
  - $4x + 10y + 5z \leq 71$ (lin)
- Execution



1UIP nogood: $c \wedge [y \neq 3]$ ➔ false or $[y \neq 3]$ ➔ !c

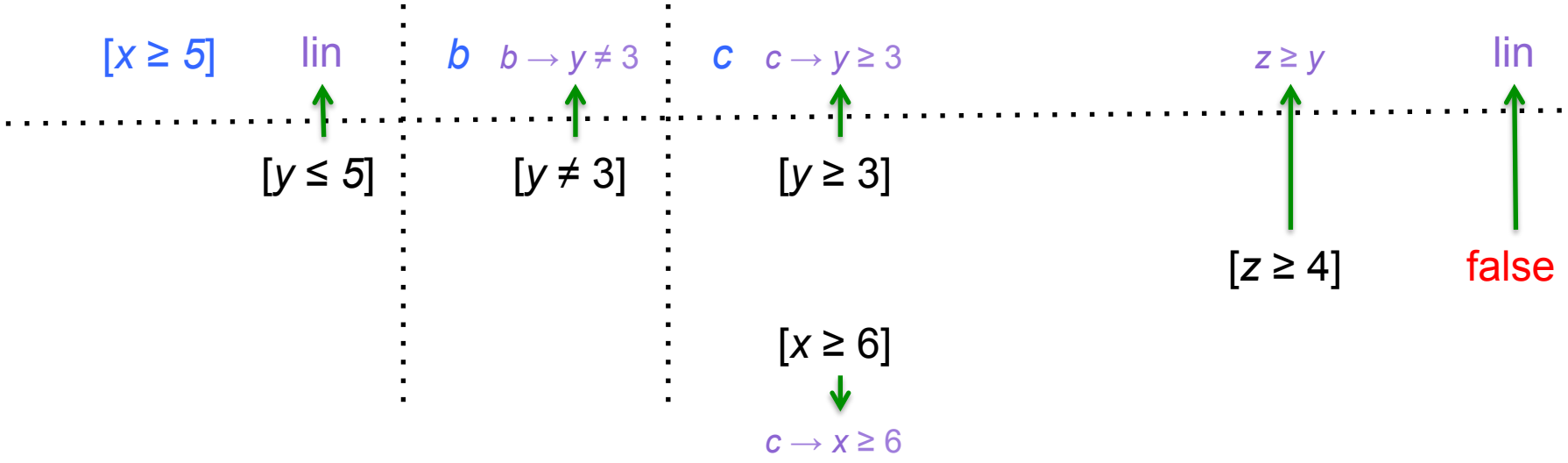
LCG propagation example

- Execution



LCG propagation example

- Execution



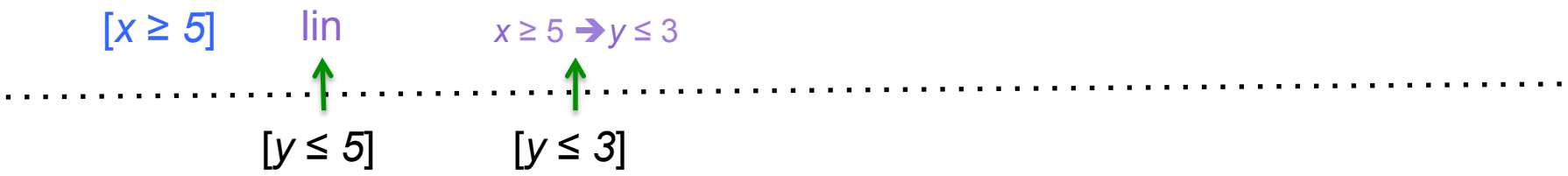
Nogood: [x ≥ 5] ∧ [y ≥ 4] ➔ false

1UIP Nogood: [x ≥ 5] ∧ [y ≥ 4] ➔ false

1UIP Nogood: [x ≥ 5] ➔ [y ≤ 3]

LCG propagation example

- Backjump



Nogood: [$x \geq 5$] $\wedge$ [$y \geq 4$] ➔ false

Backwards versus Forwards

Class	n	forward			backward			clausal		
		time	fails	len	time	fails	len	time	fails	len
amaze	(5)	113	272546	**16.8**	**96**	267012	18.2	455	**242110**	22.2
fast-food	(4)	345	241839	**44.3**	**264**	**214918**	45.6	>2617	58027[2]	159.8
filters	(4)	**613**	**883948**	**11.2**	625	906724	20.7	>901	7331[1]	10.0
league	(2)	11	74483	**28.3**	**10**	**72737**	31.1	14	81679	34.9
mspsp	(6)	**23**	**55021**	**24.3**	29	62364	53.2	44	70511	24.6
nonogram	(4)	**1965**	96461	141.5	2124	90672	168.2	>3126	32805[2]	144.8
pattern-set	(2)	451	**81397**	**180.4**	**400**	82410	180.8	>1016	3913[1]	3505.3
proj-plan	(4)	83	**74531**	42.1	**78**	82269	63.4	150	89860	46.2
radiation	(2)	1.5	**7407**	**17.3**	**1.3**	7566	22.5	1.5	7382	19.9
ship-sched	(5)	43	44897	**16.0**	**37**	**41353**	18.2	273	71120	23.2
solbat	(5)	696	**337692**	**201.2**	**679**	357009	204.0	>1528	111477[1]	239.1
still-life	(5)	735	**745949**	**21.9**	**678**	768155	30.2	>2640	269664[2]	23.7
tpp	(4)	613	8486	27.4	**126**	8490	30.1	>902	8330[1]	12.7

The Globality of Explanation

NICTA

- 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

NICTA

- 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!
  - e.g. Resource constrained project scheduling!

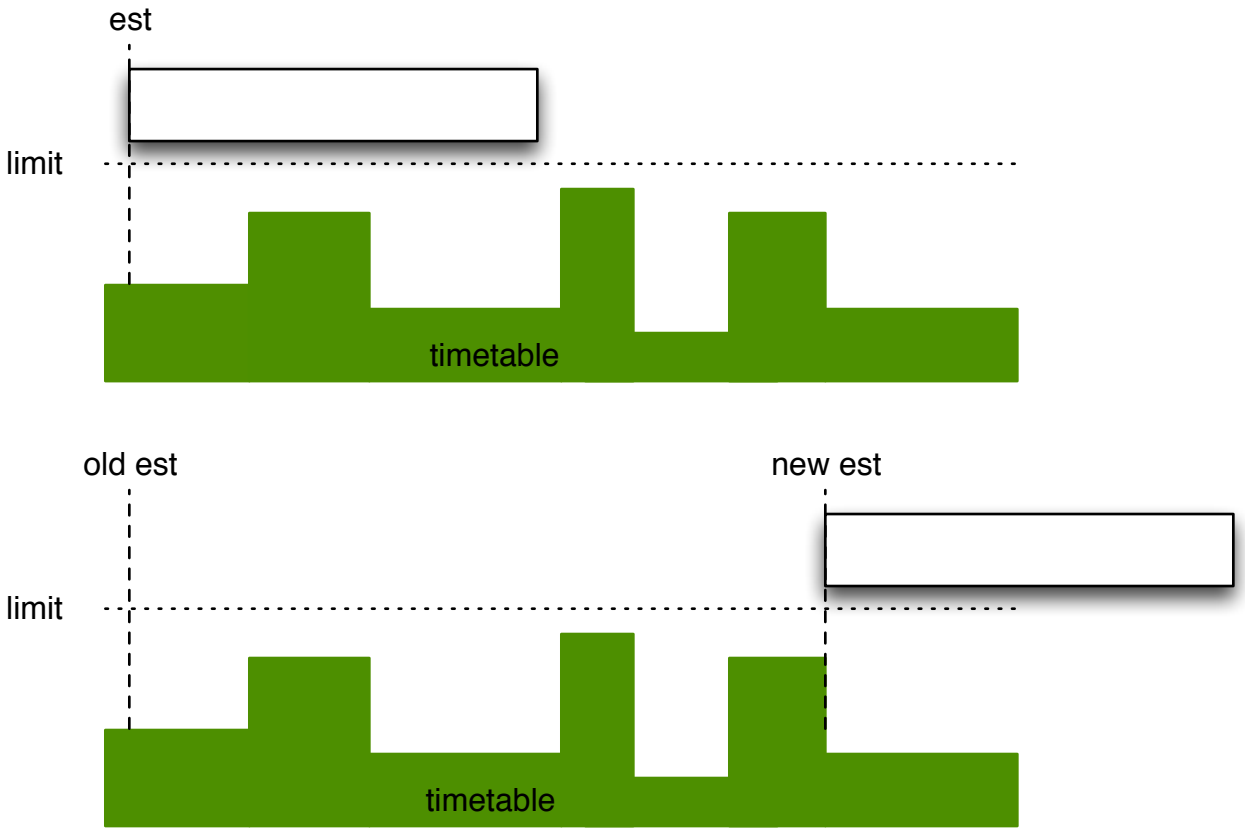
Explaining Globals

NICTA

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

Explaining Globals Piecewise

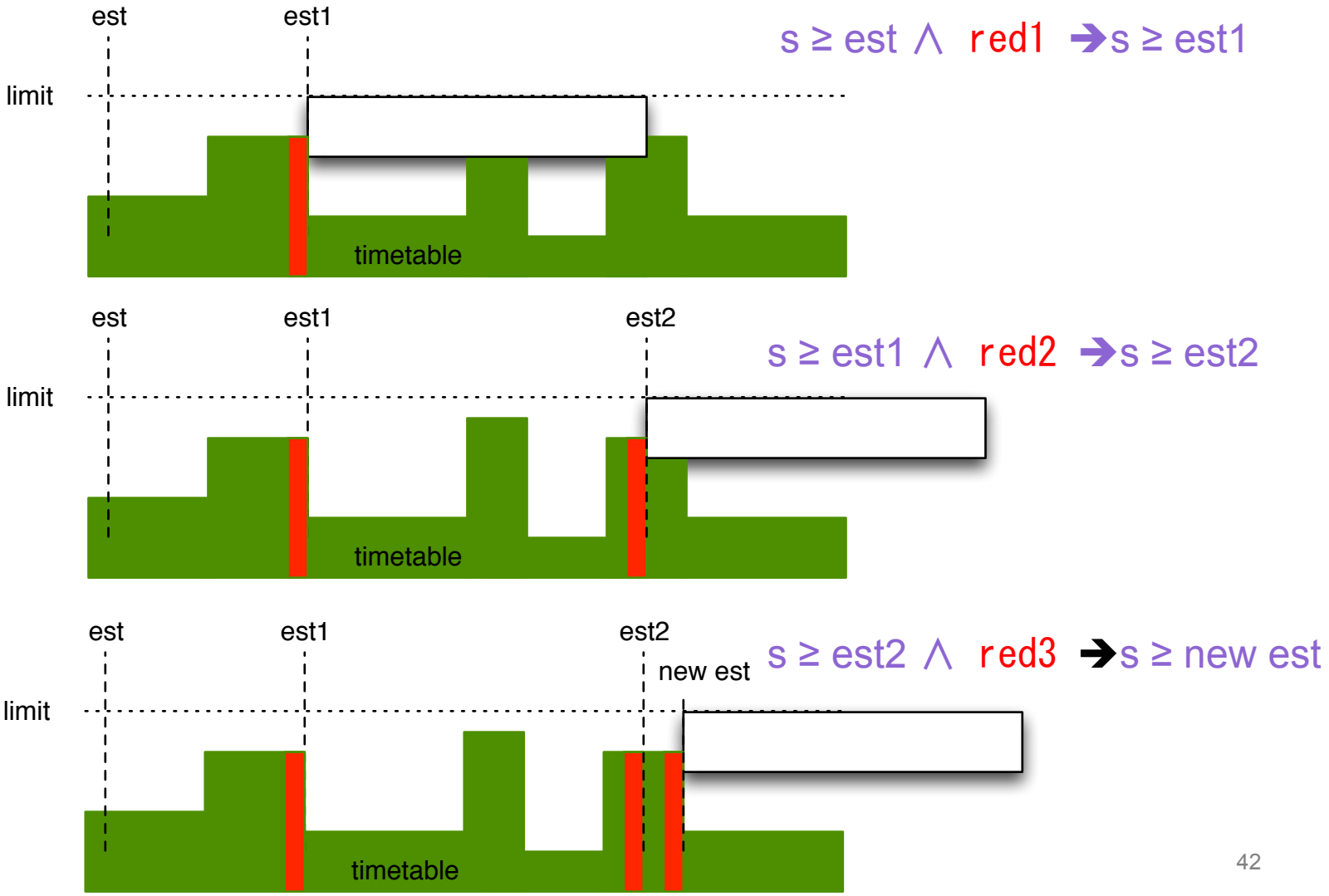
- Splitting explanations makes them more reusable: e.g. cumulative constraint propagation



Explaining Globals Piecewise

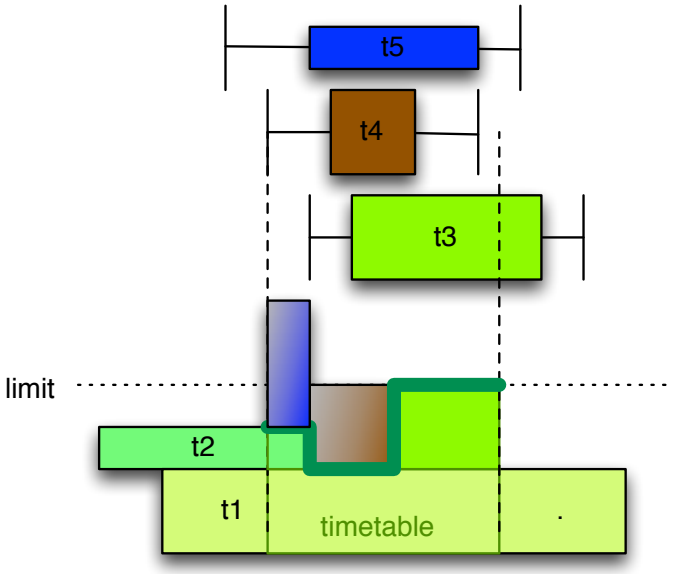
NICTA

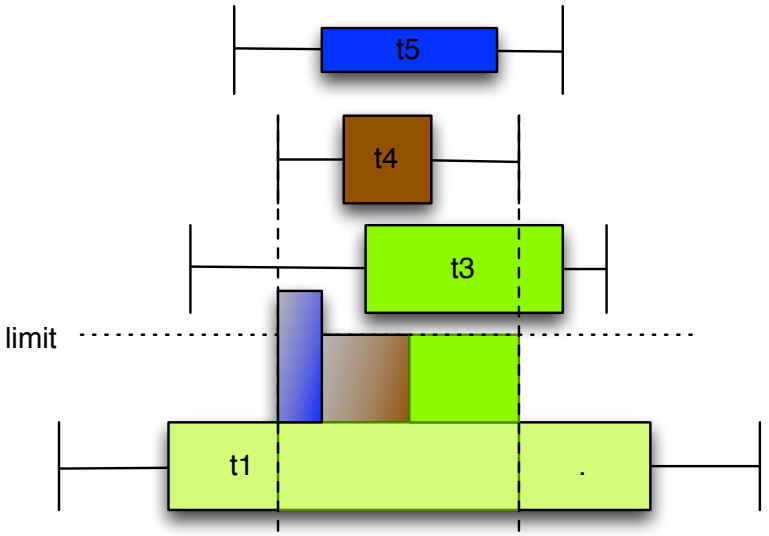
- Break it into parts $s \geq est \wedge red1 \wedge red2 \wedge red3 \rightarrow s \geq new\ est$



Weak Propagation, Strong Explanation

- Explain a weak propagator strongly
- We get strong explanations, but later!

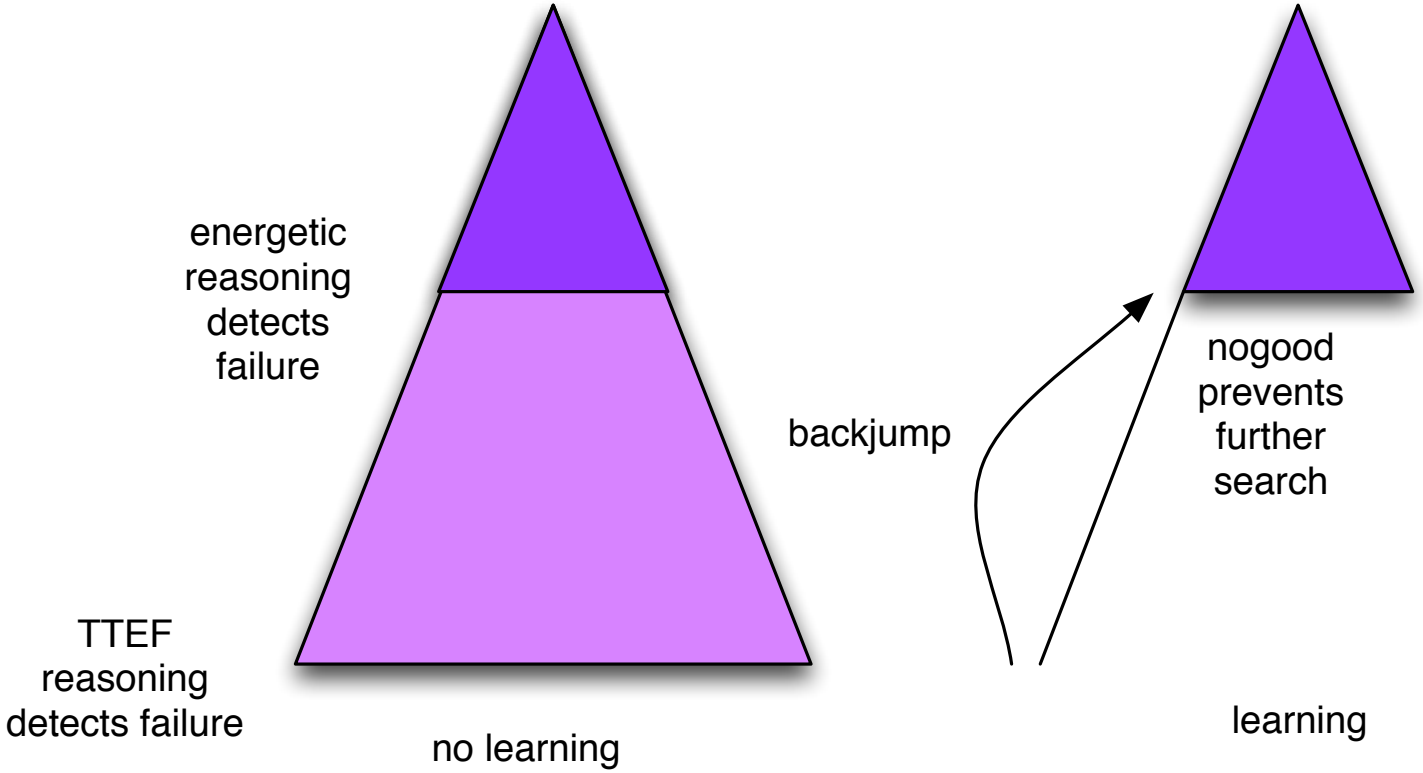




- TTEF propagation Energetic explanation
- Strong propagation algorithms less important

Weak Propagation, Strong Explanation

- Late failure discovery doesn't hurt so much



- Strong propagators are not so important!
- Strong explanations are important

Search for LCG

NICTA

- Strong Autonomous Search
- Activity based search
  - ~~Michel and Van Hentenryck CPAIOR 2012~~
  - Chaff Moskewitz et al DAC 2001
    - Bump activity of all literals seen in conflict resolution
    - Decay activity of all literals periodically
- Concentrates search on literals causing local failure
- Highly local (1000 fails ago is irrelevant)
- The ONLY SEARCH used in SAT and SMT

Search for LCG

NICTA

- Restarts are (almost) FREE
  - All failure detected in previous searches is recorded
  - Restarting never repeats work
    - Whether a fixed search
    - Or a dynamic search
- Aggressive Restarting
- Works well with activity based search
  - Concentrate on failure

Activity-based search can be BAD

NICTA

- Car sequencing problem
  - production line scheduling
- Comparing different search strategies
  - Static: selecting in order
  - DomWDeg: weight variables appearing in constraints that fail
  - Impact: prioritising decisions that reduce domains
  - Activity based

	Static	DomWDeg	Impact	Activity
Time (s)	206.3	0.8	951.3	1522.2
Solved (70)	66	70	55	47

Hybrid Searches

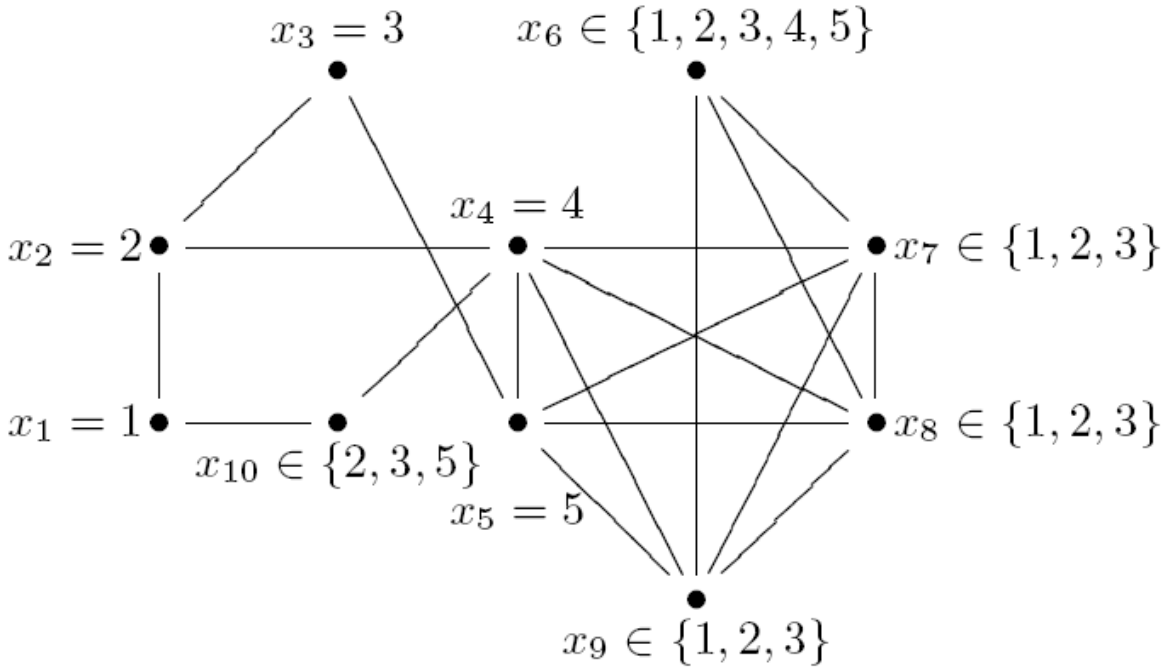
- Most of our state-of-the-art results use
- Hybrid searches
  - Problem specific objective based search
    - To find good solutions early
  - Switching to activity based search
    - To prove optimality
- Sometimes alternating the two!
- Or throwing a weighted coin to decide which
- More on why this works later

Symmetries and LCG

NICTA

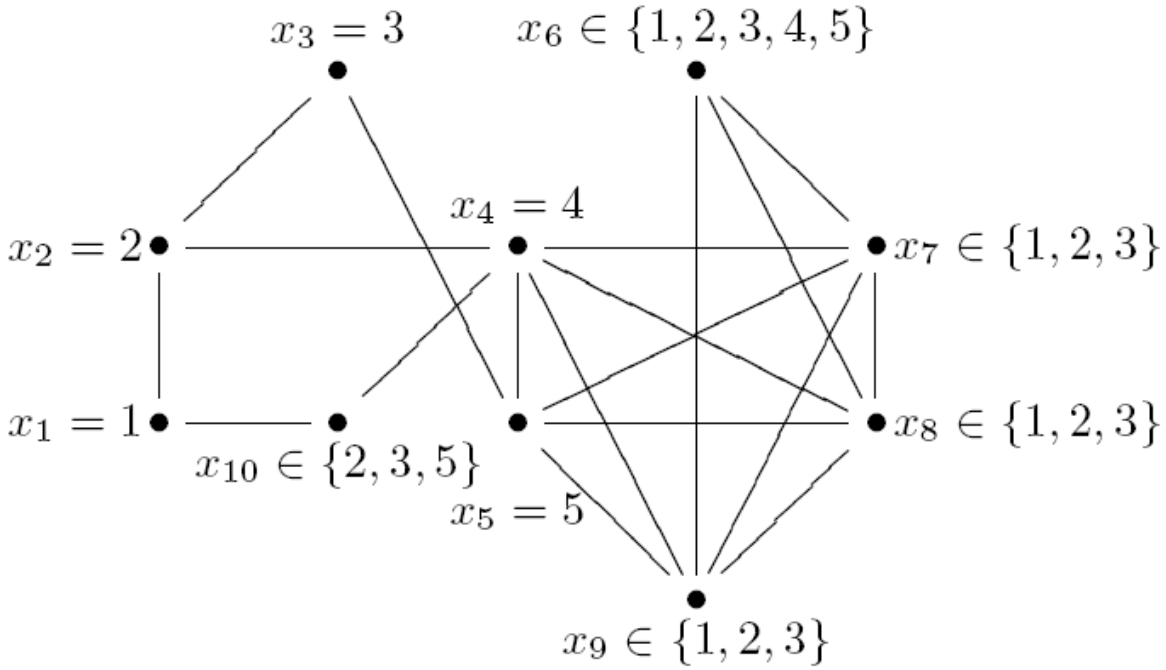
- LCG interacts well with symmetries
- Symmetry breaking constraints
  - Problem: search strategy disagrees with constraints
  - Solution: activity based search
    - Either the search agrees and constraints get no activity
    - Or the search disagrees and sym constraints get activity
- Dynamic symmetry breaking
  - SBDS is a nogood method
  - Adds symmetric versions of the decision nogood
  - LCG adds symmetric versions of the 1UIP nogood
    - Much stronger
  - No other symmetry breaking method can find these!

Symmetries and LCG



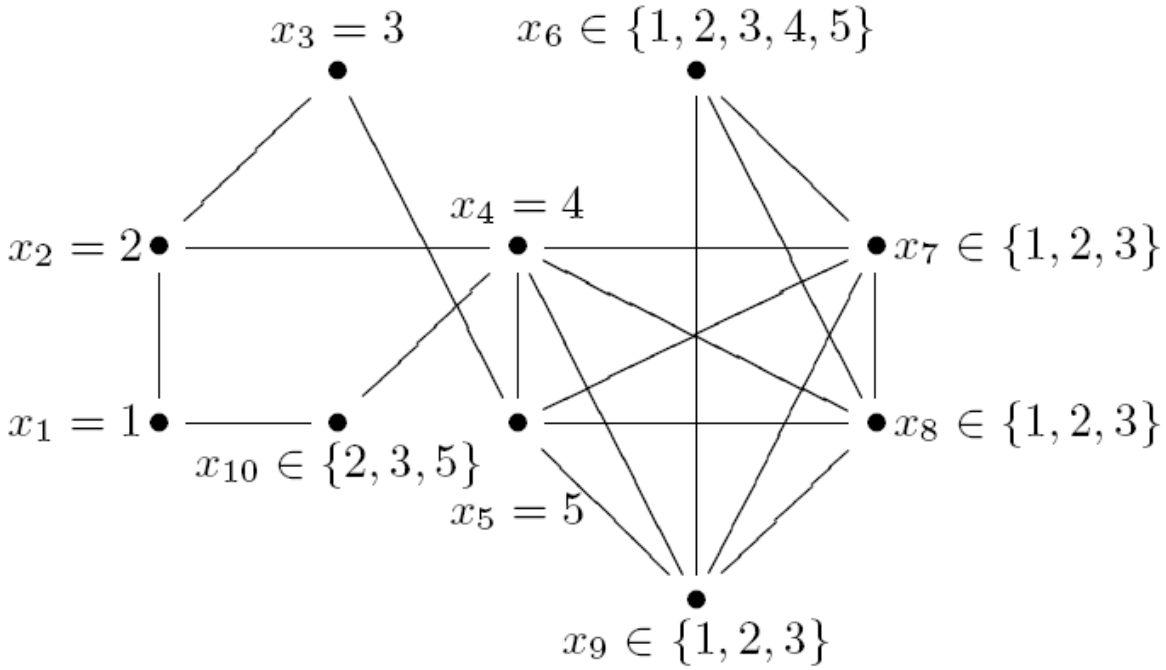
- 5-colour this graph (value symmetry)
- Already coloured x_1, x_2, x_3, x_4, x_5
- Setting $x_6 = 1$, $x_7 = 2$, causes failure
- Dec. Nogood: $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5, x_6 = 1 \rightarrow x_7 \neq 2$
- No value symmetric versions are applicable

Symmetries and LCG



- 5-colour this graph (value symmetry)
- Already coloured x_1, x_2, x_3, x_4, x_5
- Setting $x_6 = 1$, $x_7 = 2$, causes failure
- 1UIP Nogood: $x_4 = 4$, $x_5 = 5$, $x_6 = 1$ ➔ $x_7 \neq 2$
- Value Symmetric version is relevant
  - $x_4 = 4$, $x_5 = 5$, $x_6 = 1$ ➔ $x_7 \neq 3$

Symmetries and LCG



- 5-colour this graph (value symmetry)
- Already coloured x_1, x_2, x_3, x_4, x_5
- Setting $x_6 = 1$, $x_7 = 2$, causes failure
- Adding the two nogoods immediately fails with nogood
  - $x_4 = 4$, $x_5 = 5$ ➔ $x_6 \neq 1$
  - Symmetry gives: $x_4 = 4$, $x_5 = 5$ ➔ $x_6 \neq 2$ and $x_4 = 4$, $x_5 = 5$ ➔ $x_6 \neq 3$

Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks

Search is Dead, Long Live Proof

NICTA

- Search is simply a proof method
  - With learning its lemma generation
- Optimization problems
  - Require us to prove there is no better solution
  - As a side effect we find good solutions
  - Even if we cant prove optimality,
    - we should still aim to prove optimality
- Primal heuristics (good solutions fast)
  - Reduce the size of optimality proof
- Dual heuristics (good lower bounds fast)
  - Reduce the size of the optimality proof

Most controversial statement

Search is Dead, Long Live Proof

NICTA

- The role of Search
  - Find good solutions
    - Only if this helps the proof size to be reduced
  - Find powerful nogoods (lemmas)
    - That are reusable and hence reduce proof size
- Other inferences can reduce proof size
  - Symmetries
  - Dominance
  - Stronger propagators (stronger base inference)
- And a critical factor for reducing proof size
  - Stronger languages of learning

Forget Consistency

NICTA

- Domain consistency
  - Is irrelevant unless its cheap
- Bounds(R) or Bounds(Z) consistency
  - Also irrelevant
- A propagators value can be measured by
  - Strength of lemmas generated per unit time
  - This can clearly be problem dependent

The Language of Learning

- Is critical
- Consider the following MiniZinc model
  - `array[1..n] of var 1..n: x;`
  - `constraint alldifferent(x);`
  - `constraint sum(x) < n*(n+1) div 2;`
- Unsatisfiable
  - No learning

n	Failures	Time (s)
6	240	0.00
7	1680	0.01
8	13440	0.08
9	120960	0.42
10	1209600	4.47

With learning

n	Failures	Time (s)
6	270	0.00
7	1890	0.02
8	15120	0.20
9	136080	2.78
10	1360800	31.30

The Language of Learning

- Is critical
- Consider the following MiniZinc model
  - `array[1..n] of var 1..n: x;`
  - `array[1..n] of var 0..n*(n+1)div 2: s;`
  - `constraint alldifferent(x);`
  - `constraint s[1] = x[1] /\ s[n] < n*(n+1) div 2;`
  - `constraint forall(i in 2..n)(s[i]=x[i]+s[i-1]);`
- Unsatisfiable
  - No learning

n	Failures	Time (s)
6	240	0.00
7	1680	0.01
8	13440	0.08
9	120960	0.56
10	1209600	5.45

With learning

n	Failures	Time (s)
6	99	0.00
7	264	0.01
8	657	0.01
9	1567	0.04
10	3635	0.12

The Language of Lemmas

NICTA

- Critical to improving proof size
- Choose the right language for expressing lemmas
- See
  - Lazy encoding. CP2013
  - Structure based extended resolution
    - http://arxiv.org/abs/1306.4418
- Constraint Programming has a massive advantage over other complete methods since we “know” the substructures of the problem

Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks

Conclusions

- Most of CP search is repeated
- Remember the past to avoid repeating it
- Search is only a mechanism for generating good lemmas
- Consider other mechanisms for proof size reduction
  - inference, language, dominance, relaxation, decomposition, primal heuristics, CEGAR

Search is Dead

Long Live Proof

Whats left to be done?

NICTA

- Language of Learning
- Explaining propagators
  - Sometime building strong explanation is hard
- Conflict directed explanation
  - We can take into account the current conflict while explaining
- Dominances and LCG
  - Dynamic dominance breaking search with learning
- Parallelizing LCG
  - Good luck! It seems proof is essentially sequential

Whats coming

- ObjectiveCP
  - CP based on a small micro kernel
  - See Pascals talk
- ObjectiveCPExplanation
  - An LCG solver in the ObjectiveCP framework
- ObjectiveCPSchedule
  - State of the art scheduling technology
- MiniZinc 2.0

MiniZinc 2.0

NICTA

- Open LLVM architecture
- User-defined functions
  - Functional constraint modelling, functional globals
  - Better CSE
- Option types
  - Concise modelling of decisions that are only relevant dependent on other decisions
- Half reification
  - Better translation of complex logical constraints
  - Substantial efficiency improvements
  - More flexible use of globals
- Globalizer (powerful structural analysis)

Lightning Model and Solve Competition

- Be crowned the worlds best optimizer ☺
- Model and Solve Competition
  - Teams of 3
  - Solve 6 optimization problems with 5 instances each
    - Using any optimization technology you like
  - One laptop, 3 brains, 2 hours
- 16:30 Thursday 19th September
  - Lecture Hall X

- Signup now at the Registration Desk
  - Places are limited! First come first served.

Final Word

- NICTA optimization group is looking for a constraint programmer
  - Supply chains and logistics
- University of Melbourne should be advertising for a lecturer position soon in Optimization
- We are always keen to host interns in the “worlds most livable city”

- So come and join us!