

Learning: from CP to SAT and back again

Ian Gent
University of St Andrews

Topics in this Series

- Why SAT & Constraints?
- SAT basics
- Constraints basics
- Encodings between SAT and Constraints
- Watched Literals in SAT and Constraints
- Learning in SAT and Constraints
- Lazy Clause Generation + SAT Modulo Theories

Learning

- *Not* talking about classical machine learning
 - though it probably falls within that definition
 - and ML has interesting applications in constraints ...
 - but still, not talking about that
- Talking about learning during search
 - parts of the search space that are no good
 - learnt at large cost
 - can be avoided in future at low cost

Obvious advantage

- Search is exponential
- subsearches are exponential
 - and tell us facts that were expensive to find out
 - so let's *learn* those facts
 - and deduce them and similar facts faster in the rest of search

Obvious problem

- How do we learn facts?
 - And reuse them in the future
- We're never going to revisit the identical search state ever again
 - so we have to *abstract* what we have learnt
 - so how do we work out something general from the specifics of this case?
 - and work out how to apply it elsewhere?
 - and with good cost-benefit ratio?

Learning in SAT & Constraints

- From Constraints ...
 - Conflict directed backjumping
- ... to SAT
 - Learning in SAT
 - VSIDS
- ... and back again
 - s-learning and g-learning in Constraints

Learning in SAT & Constraints

- From Constraints ...
 - Conflict directed backjumping
- ... to SAT
 - Learning in SAT
 - VSIDS
- ... and back again
 - s-learning and g-learning in Constraints

Learning & Backjumping in Constraints

- In this area Constraints seems to have a longer history
- Backjumping
 - Gaschnig 1977
- Learning
 - Dechter & Frost, 1990, 1994
- But I'm going to start with Conflict-directed backjumping
 - Prosser, 1993

CBJ

- Sometimes say “backjumping” as generic term
 - but dangerous as BJ is a specific algorithm (and not as good)
- Conflict directed backjumping
 - CBJ
 - I will use CBJ to include variants like FC-CBJ, MAC-CBJ
 - Patrick Prosser, 1993
 - 617 citations as I write (Google Scholar)
 - compare 215 for my most cited paper

Conflict Directed Backjumping

- Usual to distinguish between learning and backjumping based approaches
- CBJ
 - we avoid backtracking to any node
 - which is above the current node
 - and where the opposite branching choice **cannot** help
 - because we can **prove** it will not
- Learning
 - we reuse the information learnt at this node
 - at nodes which are not ancestors of the current node

Conflict set

- Key idea in CBJ
 - Same as explanation coming later
 - but I'll use the CBJ word here
- A **conflict set** at a failed node
 - is a set of assigned variables such that
 - if **any other** variable is changed there is no solution
 - equivalently:
 - every assignment with the current values in the conflict set, and arbitrary values elsewhere
 - is not a solution

Conflict Set Example

- x in $\{1,2,3\}$
- y assigned to 1
- **$x < y$**
 - conflict set is $\{y\}$
 - there is no possible value of x
 - no matter how many other variables there are in the problem

CBJ algorithm

- Whenever we get a failure, compute conflict set
- Jump back [i.e. backtrack to...] the most recently assigned variable x in conflict set
- Discard any search nodes between current node and x
- Merge current c.s. with existing c.s. of x
- *If any remaining values of x*
 - try next value
 - *else repeat this slide*

Computing Conflict Sets

- Two cases
 - backtracking
 - propagating
- Quick summary...
 - *every propagation always has a conflict set*
 - *merging is taking the union*

Merging Conflict Sets on Backtracking

- Say we have tried $x = a, b, c$
 - and we have cs_{abc} (*merged conflict set for x*)
 - and value $x = d$ has just failed
 - with cs_d which must have x in it
 - *why must x be in it?*

Merging Conflict Sets on Backtracking

- Say we have tried $x = a, b, c$
 - and we have cs_{abc} (*merged conflict set for x*)
 - and value $x = d$ has just failed
 - with cs_d which must have x in it
- How do we merge cs_{abc} and cs_d ?
 - Simply take $cs_{abc} \cup cs_d - \{x\}$
 - *why?*

$$CS_{abc} \cup CS_d - \{x\}$$

- When [if] we ever backjump from x
 - we need to know every variable which changing could lead to a solution
 - We need everything in CS_{abc}
 - otherwise $x = a, x=b$ or $x=c$ might work
 - And everything in CS_d
 - otherwise $x=d$ might work
 - But not x because we are backjumping from x

Conflict sets & Propagation

- If we are propagating (we always are)
- We can't ignore propagation for c.s's
- **$x < y, y < z$**
 - x in $\{1,2,3\}$, y in $\{1,2,3\}$, z in $\{2,3\}$
 - z assigned to 2 at search node
 - Then y assigned to 1,
 - then x fails with c.s = $\{y\}$
 - So we backjump to last var in c.s., that is y
 - *but there is no such node so we fail*
 - But we should backjump to $z=2$
 - then try $z=3$ and we can carry on

Conflict sets & Propagation

- Every time a possible value $x=a$ is deleted
 - we record a conflict set for the deletion
 - a c.s. for deletion is just like a failure c.s.
 - set so that if the variables in it take their current values, then $x=a$ is impossible
- When we propagate, merge c.s.'s which played role
 - e.g. if we are doing AC
 - relevant c.s.'s are deleted values in constraint which otherwise would form a support for $x=a$

Conflict sets & Propagation

- **$x < y, y < z$**
 - x in $\{1,2,3\}$, y in $\{1,2,3\}$, z in $\{2,3\}$
 - z assigned to 2 at search node
 - Then $y=2$ is deleted, conflict set $\{z\}$
 - And $y=3$ is deleted, conflict set $\{z\}$
 - then x fails with c.s = $\text{merge}(y=1/\{z\}, y=2/\{z\}) = \{z\}$
 - So we backjump to last var in c.s., i.e. z
 - then try $z=3$ and we can carry on

Conflict sets & explanations

- Going to return in a bit to key questions:
 - how do we compute explanations (conflict sets) from propagators?
 - and then handle the merging of them from propagators?

Learning in SAT & Constraints

- From Constraints ...
 - Conflict directed backjumping
- ... to SAT
 - Learning in SAT
 - VSIDS
- ... and back again
 - s-learning and g-learning in Constraints

CBJ in SAT

- Very natural view of CBJ in SAT
- conflict sets are *clauses*
 - e.g. conflict set after failure of $x=a$ is $\{y,z\}$, with $y=b, z=c$
 - $-xa$ OR $-yb$ OR $-zc$
 - e.g. conflict set for $x=b$ is $\{y,w\}$ with $w=d$
 - $-xb$ OR $-yb$ OR $-wd$
- conflict set merging is *resolution*
 - We have At-Least-One clause
 - xa OR xb
 - resolve to get xb OR $-yb$ OR $-zc$
 - resolve to get $-yb$ OR $-zc$ OR $-wd$

CBJ in SAT

- *Very easy indeed* to work out conflict set
- If failure (i.e. empty clause) arises in clause C ...
 - ... conflict set is C
- If clause C becomes unit setting $x=0$
 - ... conflict set explaining $x \neq 1$ is C

CBJ in SAT

- CBJ was brought across to SAT in 96, 97
- “Using CSP Look-Back Techniques to Solve Real-World SAT Instances”, 1997
 - Bayardo & Schrag
 - 593 citations (Google Scholar)
- All SAT solvers do backjumping/learning
 - Much research on SAT solvers in mid-late 90s
 - Bayardo & Schrag porting of CBJ one key piece of research

CBJ in SAT

- Another key piece of work was GRASP
 - Marques-Silva & Sakallah, 97, 99
 - 693 and 811 citations for the two papers
 - Doesn't cite Prosser this time
 - Ginsberg 93 (529 citations)
 - Sussmann/Stallmann 77 (677 citations)
 - i.e. still brought backjumping to SAT from Constraints

Citation numbers

- 617 (Prosser),
- 677 (Sussmann & Stallmann)
- 529 (Ginsberg)
- 593 (Bayardo & Schrag)
- 693 (Marques-Silva & Sakallah)
- 811 (Marques-Silva & Sakallah)
- To get an idea of scale
 - 656 citations
 - Jean-Charles Régim 1994, All-different GAC propagator

Learning in SAT & Constraints

- From Constraints ...
 - Conflict directed backjumping
- ... to SAT
 - Learning in SAT
 - VSIDS
- ... and back again
 - s-learning and g-learning in Constraints

Learning in SAT

- Going to present this with constraints in mind
 - so sometimes slightly more general than what SAT does
- But still will start with SAT
 - and move on to constraints later

Explanation (general)

- An *explanation* for a assignment ($x = a$) or disassignment ($x \neq a$) is
 - a set of assignments or disassignments
 - such that if this set is all (dis-)assigned
 - appropriate propagation level
 - will force $x = a$ (or $x \neq a$)

Explanation (Unit propagation)

- An *explanation* for a literal ($x = 0$ or $x = 1$) is
 - a set of literals
 - such that if this set is all assigned
 - unit propagation
 - will force the literal to be true
 - i.e. the negation of remaining literals in clause which caused the unit propagation to happen
- Also *explanation* for *failure*
 - exactly analogous
 - i.e. negation of all literals in failed clause

SAT Example

- You must invite somebody:
 - The ambassador asks you to invite a Francophone ambassador so his daughter can practice her French:
 - The Belgian, German and Dutch ambassadors are badly behaved when they get together, so they mustn't all be invited:
 - If you invite the Dutch ambassador, you must also invite the Belgian ambassador:
- $B \vee N \vee F \vee G$
 - $B \vee F$
 - $\neg B \vee \neg G \vee \neg N.$
 - $N \Rightarrow B$
 - $\neg N \vee B$

SAT Example

- You must invite somebody:
- The ambassador asks you to invite a Francophone ambassador so his daughter can practice her French:
 - $B \vee N \vee F \vee G$
 - $B \vee F$
- The Belgian, German and Dutch ambassadors are badly behaved when they get together, so they mustn't all be invited:
 - $\neg B \vee \neg G \vee \neg N.$
 - $\neg N \vee B$
- If you invite the Dutch ambassador, you must also invite the Belgian ambassador:

SAT Example

- You must invite somebody:
- The ambassador asks you to invite a Francophone ambassador so his daughter can practice her French:
- The Belgian, German and Dutch ambassadors are badly behaved when they get together, so they mustn't all be invited:
- If you invite the Dutch ambassador, you must also invite the Belgian ambassador:

1. $B \vee N \vee F \vee G$

2. $B \vee F$

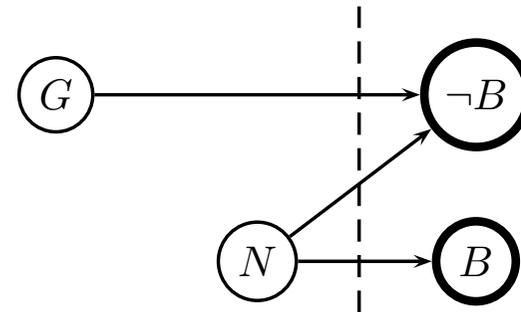
3. $\neg B \vee \neg G \vee \neg N.$

4. $\neg N \vee B$

- Suppose we have N, G
 - (4) explanation of B is N
 - (3) explanation of $\neg B$ is G, N
 - (2) explanation of F is $\neg B$

SAT Implication Graph

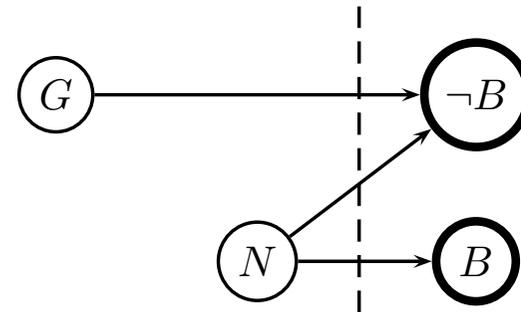
- An implication graph (IG) for the current state of variables is a directed acyclic graph where
 - each node is a currently true literal, e.g. v when variable $v \leftarrow 1$, and
 - there is an edge from u to v iff u appears in the explanation for v .



- $B \vee N \vee F \vee G$
- $B \vee F$
- $\neg B \vee \neg G \vee \neg N$.
- $\neg N \vee B$
- set **G** and **N**

Implication Graph Cut

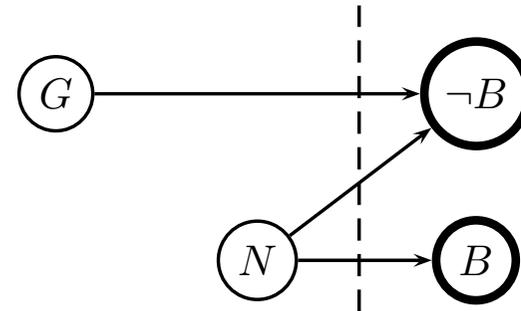
- A **cut** of an IG containing mutually inconsistent nodes is a partition (S, T) of vertices such that
 - all nodes corresponding to decision assignments belong to S ,
 - the mutually inconsistent nodes are in T
 - if a node $x \in T$,
 - **either** all its direct predecessors in T
 - **or** all its direct predecessors are in S .
- Asserting all events of a cut and enforcing the same consistency level as the explanations were built with will lead to the same failure.
- Cuts often written by literals immediately to left of cut
 - e.g. $\{G, N\}$



- $B \vee N \vee \neg G$
- $B \vee F$
- $\neg B \vee \neg G \vee \neg N$.
- $\neg N \vee B$
- set **G** and **N**

Cuts in graphs

- A cut in the graph gives us something we can learn
- We can add a clause from the cut
 - e.g. cut $\{G, N\}$
 - learn $\neg G \vee \neg N$
 - Should avoid the same mistake in the future
- But how do we find a good cut?



- $B \vee N \vee F \vee G$
- $B \vee F$
- $\neg B \vee \neg G \vee \neg N.$
- $\neg N \vee B$
- set **G** and **N**

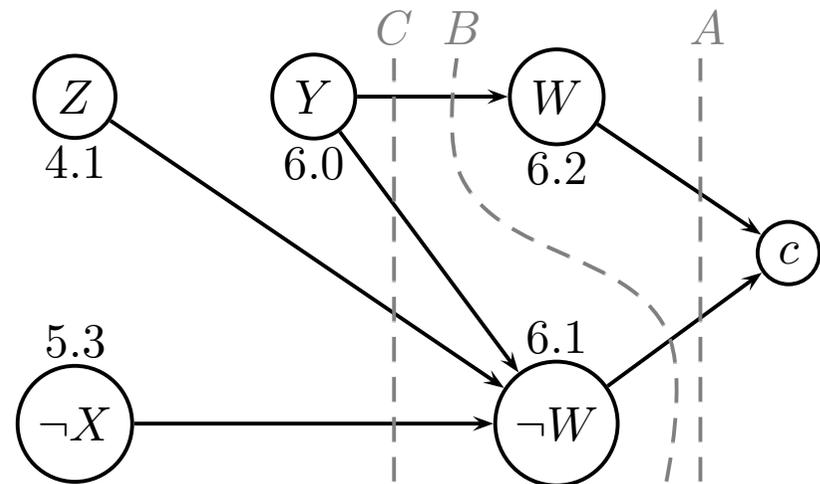
First UIP Cut

- If cut is too specific...
 - doesn't actually avoid the work of branching
- If cut is too general...
 - it may not save work
 - e.g. just the set of branching decisions
- Want a compromise
 - First Unique Implication Point
 - Find cut such that both contradictory literals forced by branching decision and nothing else at this depth

- Clauses (fragment)

- $\neg Y \vee W$

- $X \vee \neg Y \vee \neg Z \vee \neg W$



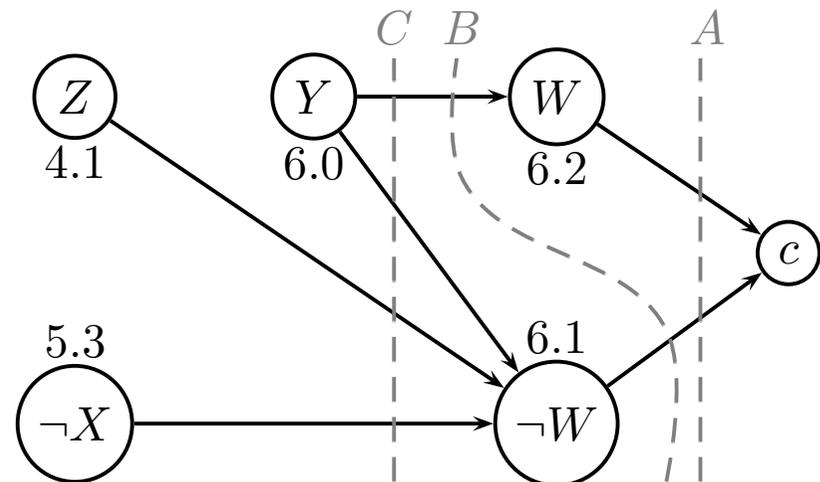
First UIP Cut

- If cut is too specific...
 - doesn't actually avoid the work of branching
- If cut is too general...
 - it may not save work
 - e.g. just the set of branching decisions
- Want a compromise
 - First Unique Implication Point
 - Find cut such that both contradictory literals forced by branching decision and nothing else at this depth

- Clauses (fragment)

- $\neg Y \vee W$

- $X \vee \neg Y \vee \neg Z \vee \neg W$



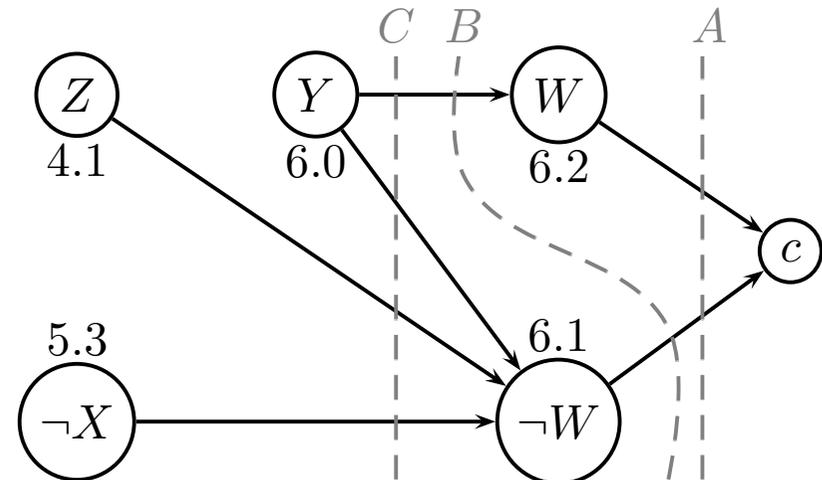
First UIP Cut

- Depth annotations
 - *6.0 = set at depth 6 by search*
 - *4.1 = set at depth 4, first propagation*
- *c* represents conflict
 - i.e. empty clause
 - i.e. $\neg Y \vee W$ with $Y=1$, $W=0$

- Clauses (fragment)

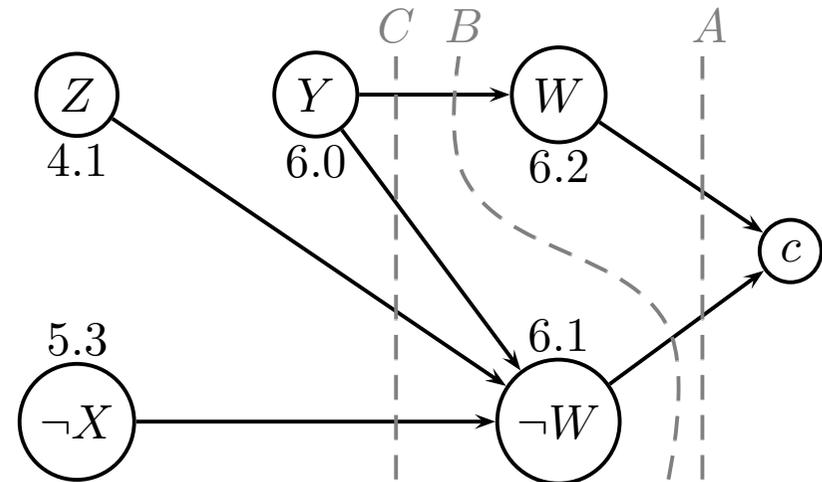
- $\neg Y \vee W$

- $X \vee \neg Y \vee \neg Z \vee \neg W$



First UIP Cut Algorithm

- Start with T = trivial cut of conflict
- Loop until we have First UIP cut
 - Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$
- choose deepest node N with predecessors in S
- add all predecessors of N to T



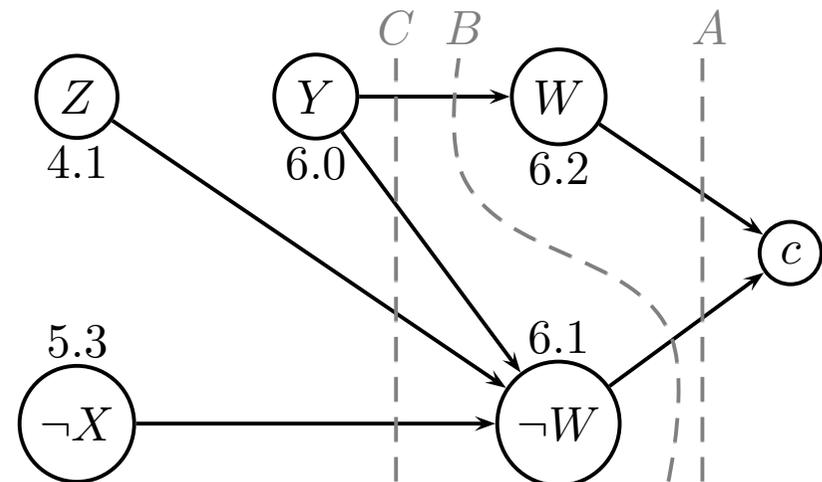
First UIP Cut Algorithm

- Start with T = trivial cut of conflict c
- Loop until we have First UIP cut
 - choose deepest node N with predecessors in S
 - add all predecessors of N to T
 - Remove N from T
- Theorem:
 - algorithm always gives us first UIP cut

- Clauses (fragment)

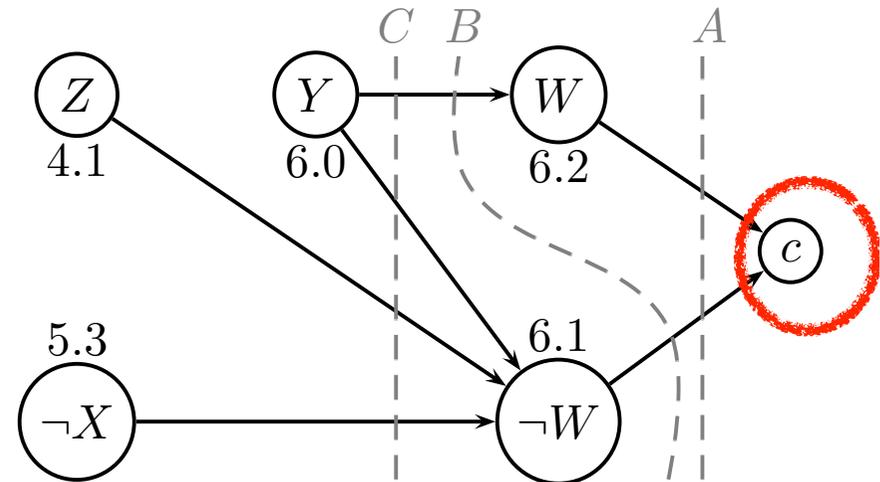
- $\neg Y \vee W$

- $X \vee \neg Y \vee \neg Z \vee \neg W$



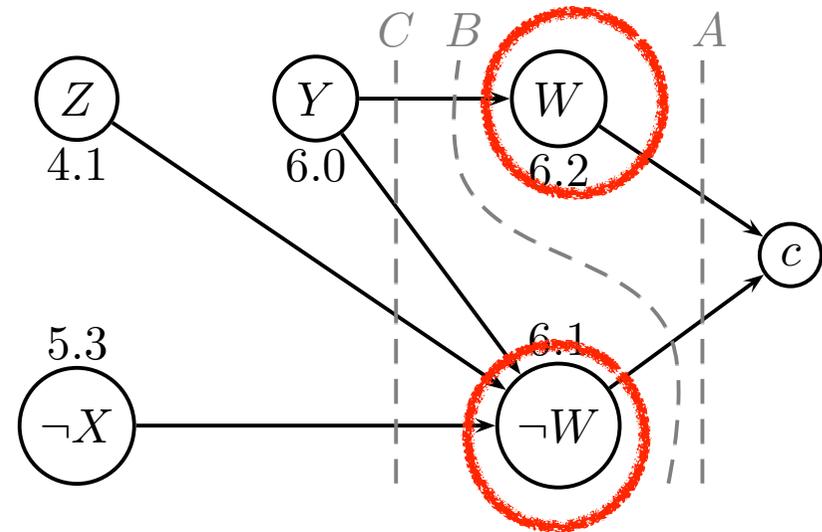
First UIP Cut Algorithm

- Start with T = trivial cut of conflict
 - $T = \{c\}$
 - line A
 - deepest node in $T = c$
- Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$



First UIP Cut Algorithm

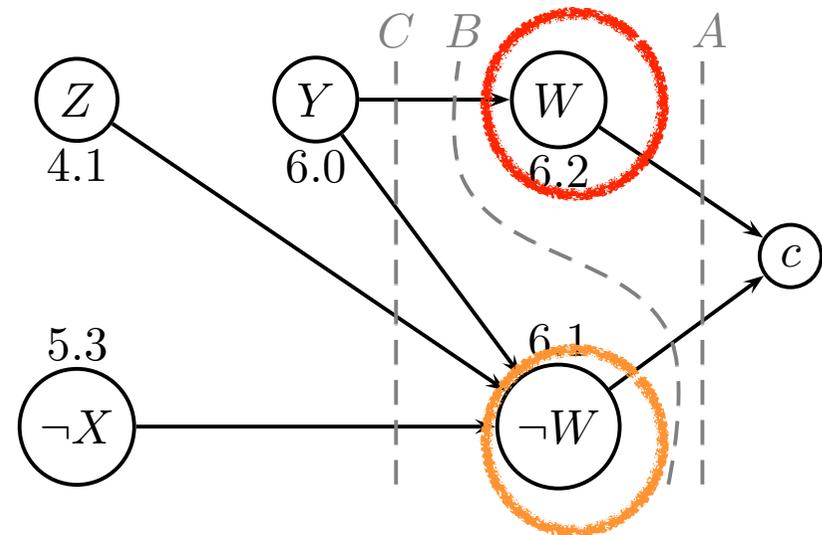
- Start with $T =$ trivial cut of conflict
 - $T = \{c\}$ (line A)
 - deepest node in $T = c$
 - Add predecessors
 - Add $W, \neg W$ to T
 - Remove c
- Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$



First UIP Cut Algorithm

- Add $W, \neg W$ to T
 - Depths 6.1, 6.2
 - $T = \{W, \neg W\}$
 - *line B*
- deepest node in $T = W$
 - Add predecessors
 - Add Y to T
 - Remove W

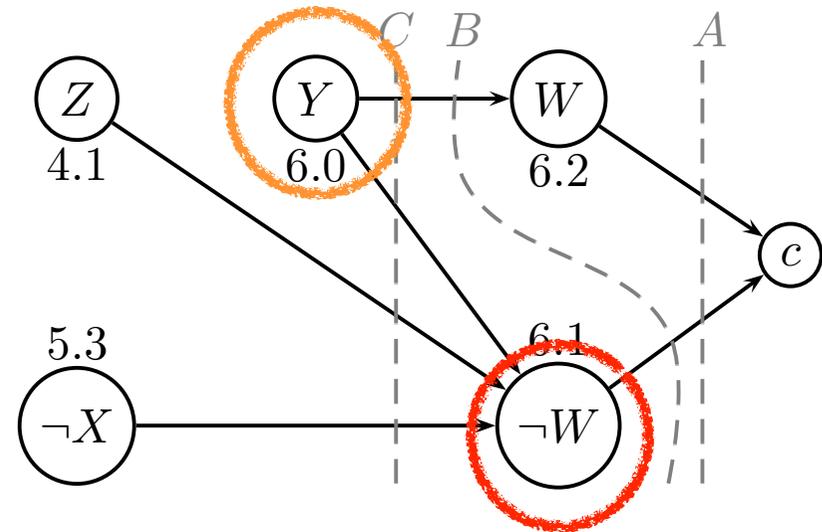
- Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$



First UIP Cut Algorithm

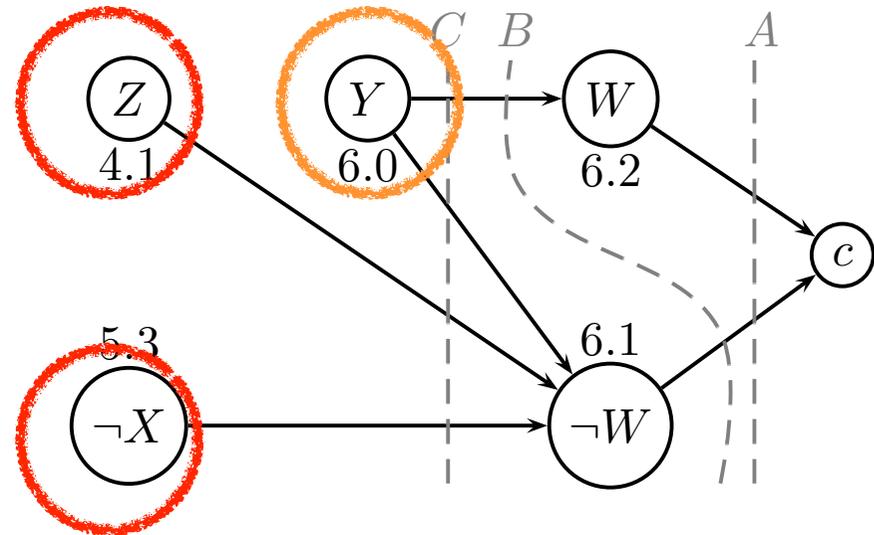
- Add Y to T
 - $T = \{Y, \neg W\}$
 - *Depths 6.0, 6.1*
- deepest node in $T = \neg W$
 - Add predecessors
 - Add $Z, \neg X, Y$ to T
 - Remove $\neg W$

- Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$



First UIP Cut Algorithm

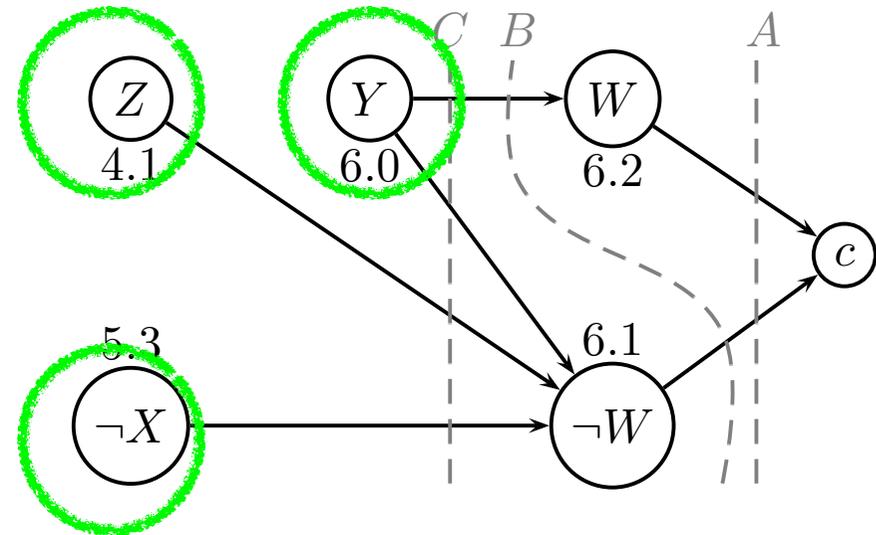
- Add $Z, \neg X, Y$ to T
 - $T = \{Y, Z, \neg X\}$
 - *line C*
 - *Depths 6.0, 4.1, 5.3*
- Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$



First UIP Cut Algorithm

- STOP
 - We have reached UIP
 - Unique depth 6 node
 - i.e. search decision
 - T is firstUIP Cut
- Cut is $T = \{Y, Z, \neg X\}$

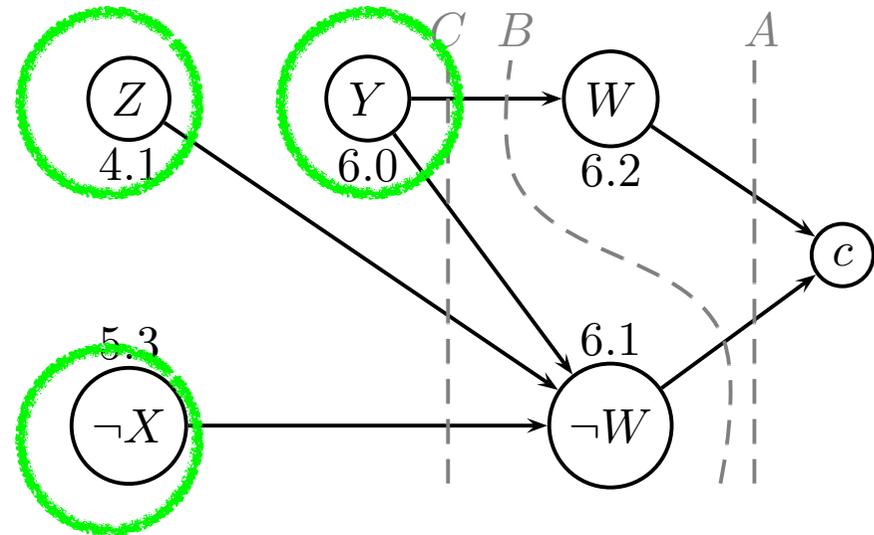
- Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$



First UIP Cut

- Now we have learnt
- $X \vee \neg Y \vee \neg Z$
- We can add this clause
- If we backtrack to the assignment $Y=1$
 - the new clause will propagate and set $Y=0$

- Clauses (fragment)
 - $\neg Y \vee W$
 - $X \vee \neg Y \vee \neg Z \vee \neg W$



Forgetting

- One problem
- If we learn a clause at every failed node
 - and search exponential nodes
 - we end up with exponentially many clauses
- Fortunately...
 - all learnt clauses are *implied*
 - i.e. does not change set of solutions
 - but may help search

Forgetting

- This means we can ...
 - delete any learnt clause ...
 - at any time ...
 - perfectly safely
- So need some kind of forgetting strategy
 - e.g. activity based
 - recently propagated clauses less likely to be forgotten

Learning in SAT & Constraints

- From Constraints ...
 - Conflict directed backjumping
- ... to SAT
 - Learning in SAT
 - VSIDS
- ... and back again
 - s-learning and g-learning in Constraints

VSIDS

- VSIDS means...
 - Variable State Independent, Decaying Sum
- Most common example of an
 - activity based heuristic
 - heuristic for choice of branching literal
- Idea is to choose the most “active” variables in some sense

VSIDS WARNING

- VSIDS comes from Chaff
 - and like watched literals is included in the Chaff patent
 - IANAL (I am not a lawyer)
 - So don't believe anything I tell you about the legal position

VSIDS

- Activity based heuristic
- Give each literal a counter.
 - Set all counters to 0
- For each new learnt clause
 - Increment counter for each literal in clause
- When we need a search decision
 - choose literal with highest counter
- Every once in a while ...
 - reduce all counters by a constant factor
 - so that inactive literals decay over time

Problem with understanding SAT solvers

- “Everyone knows” that
 - watched literals/learning/VSIDS/forgetting/restarts
 - is an unbeatable combination
- But nobody knows
 - if it's true
 - why
 - this is a real problem
 - seems to be little to no research evaluating these techniques scientifically

Learning in SAT & Constraints

- From Constraints ...
 - Conflict directed backjumping
- ... to SAT
 - Learning in SAT
 - VSIDS
- ... and back again
 - s-learning and g-learning in Constraints

learning in constraints

- Saw that learning in Constraints affected SAT (Frost, Dechter, Prosser, Ginsberg, Stallmann & Sussmann)
- More recently SAT work on learning come back to constraints
 - Mainly through
 - Katsirelos and Bacchus 2003, 2005
 - Katsirelos PhD thesis 2009
 - Neil Moore PhD thesis 2011

learning in constraints

- Approach has been
 - can we take ideas that work well in SAT?
 - Apply them in Constraints
 - And work out what we need to make them work?
- Answers are
 - Yes we can
 - Has been significant successes
 - But not unqualified success story

s-learning

- s-learning was first approach used
 - going back to E.g. Bayardo & Schrag
- Learn which combination of assignments cause failure/propagation
 - i.e. explanations are sets of assigned constraint variables
- seems like natural extension of SAT learning
 - since SAT based on assigned variables too

g-learning

- g-learning completely supersedes s-learning
 - major contribution of Katsirelos PhD
- A *better* natural generalisation of SAT
 - in SAT, assignment/disassignment the same thing
 - in Constraints, we need assignment/disassignment as different things

g-explanation

- A g-explanation is just a set of assignments and disassignments of variable/value pairs
 - I often call v/v pairs *literals* anyway
 - such that if all these commitments made
 - and appropriate propagators used
 - we get the failure/propagation the explanation is justifying

Two problems

- Finding good explanations from constraints
- Adapting solver to cope with explanations, backjumping, learning, etc

Finding good explanations

- For each constraint propagator
 - we need to be able to explain every propagation
 - with as good an explanation as possible
 - as efficiently as possible

All-different

- How do we explain all-different?
- Suppose it is for failure
 - Then there is a set of k variables
 - With total number of values $k-1$ or less
- An explanation is ...
 - the removals of all other values for all of the k variables (but not others)
 - Katsirelos 09
 - And we see the value of g-explanation instead of s-

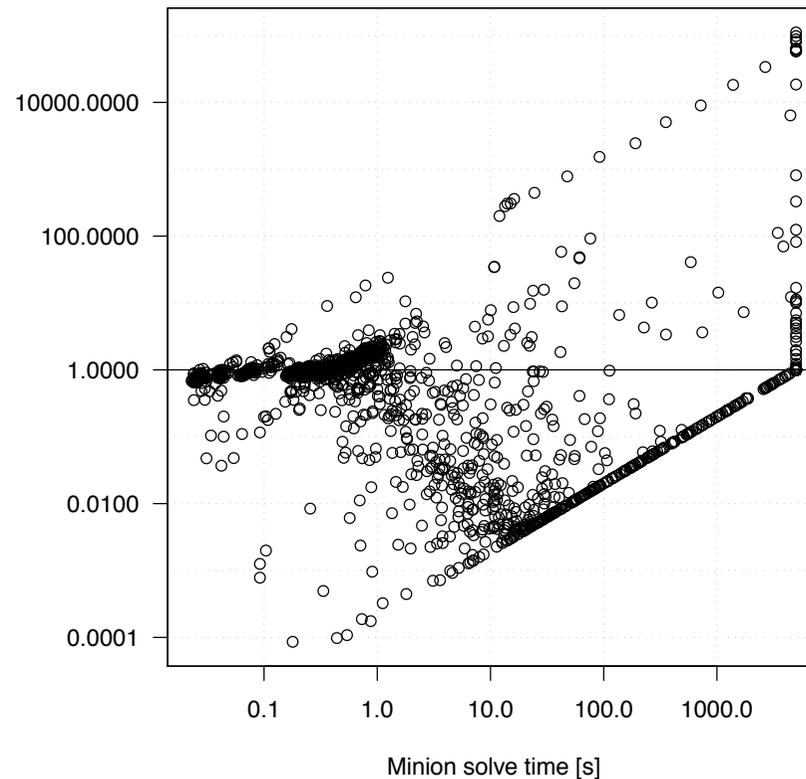
Lazy explanations

- SAT explanations are very cheap
 - based on the clause that fails/propagates
- Can be expensive to compute Constraint explanations
 - e.g. computing Hall sets after failure
 - Often the explanation is never needed
- So it's important to be *lazy* [Moore 2011]
 - When propagation happens
 - make minimal annotation of propagation
 - Only compute explanation if it is needed
 - e.g. to compute FirstUIP cut.

Win some, lose some

- Can have huge successes
 - Learning beats minion by 10,000 times
- and huge failures
 - Minion beats learning by 1000 times
 - Just because of overheads
- One workaround is to use ML
 - *learn* when to use *learning*

Minion solve time over Minion-lazy solve time



Gent, Jefferson, Kochhoff, Miguel, Moore, Nightingale, Perrie, 2010

above $y=1$ means lazy learning better