

Conflict Driven Clause Learning

Mohamed Siala
<https://siala.github.io>

INSA-Toulouse & LAAS-CNRS

January 18, 2022

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] \oplus Resolution [Robinson, 1965]

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] \oplus Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] \oplus Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation
- Resolution: Learning based on the rule
$$(l \vee c_1) \wedge (\neg l \vee c_2) \Rightarrow (c_1 \vee c_2)$$

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] \oplus Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation
- Resolution: Learning based on the rule
 $(l \vee c_1) \wedge (\neg l \vee c_2) \Rightarrow (c_1 \vee c_2)$
- **Can be seen as a CP Solver (Search, propagation) augmented by clause learning**

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

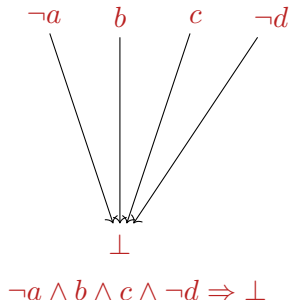
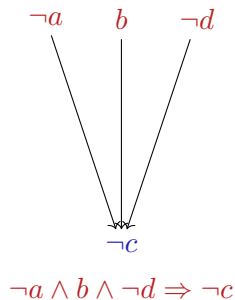
- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] \oplus Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation
- Resolution: Learning based on the rule
 $(l \vee c_1) \wedge (\neg l \vee c_2) \Rightarrow (c_1 \vee c_2)$
- **Can be seen as a CP Solver (Search, propagation) augmented by clause learning**
- But also :
 - Activity-based branching
 - Lazy data structures (2-Watched Literals)
 - Clause Database Reduction
 - Simplifications
 - Restarts
 - ...

Exercise: Propose a filtering algorithm for clauses. The algorithm takes as input a clause and has access (read and write) for the variables domains.

Unit Propagation

Given a clause C of arity n . If $n - 1$ literals are false then set the last one to be true.

Example: $(a \vee \neg b \vee \neg c \vee d)$



Two Watched Literals

- Unit propagation is implemented with an “intelligent” data structure called Two-watched literals
- Observe first that propagation happens only in two cases:
 - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
 - All literals are instantiated and none of them satisfy the clause

Two Watched Literals

- Unit propagation is implemented with an “intelligent” data structure called Two-watched literals
- Observe first that propagation happens only in two cases:
 - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
 - All literals are instantiated and none of them satisfy the clause
- Therefore for each clause C , as long as there are two literals non instantiated in C , nothing happens!

Two Watched Literals

- Unit propagation is implemented with an “intelligent” data structure called Two-watched literals
- Observe first that propagation happens only in two cases:
 - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
 - All literals are instantiated and none of them satisfy the clause
- Therefore for each clause C , as long as there are two literals non instantiated in C , nothing happens!
- The idea of the Two-watched literals is to keep 2 literals for every clause that are not instantiated. Those literals will “watch the clause” and guarantee that no propagation is needed.

Two Watched Literals

- Unit propagation is implemented with an “intelligent” data structure called Two-watched literals
- Observe first that propagation happens only in two cases:
 - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
 - All literals are instantiated and none of them satisfy the clause
- Therefore for each clause C , as long as there are two literals non instantiated in C , nothing happens!
- The idea of the Two-watched literals is to keep 2 literals for every clause that are not instantiated. Those literals will “watch the clause” and guarantee that no propagation is needed.
- If a literal watching a clause C becomes *false*, look for replacement. If no replacement found, then perform propagation

Implication Graph

	f		

$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

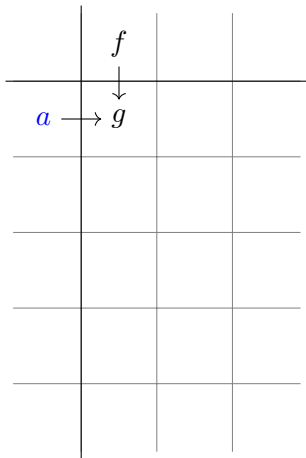
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

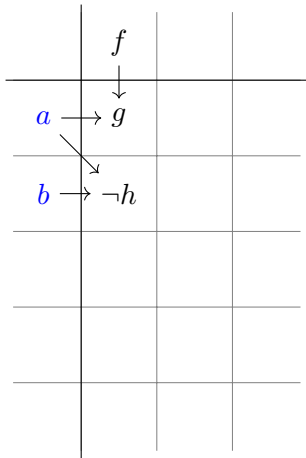
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

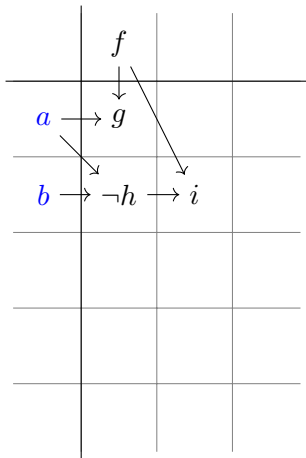
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

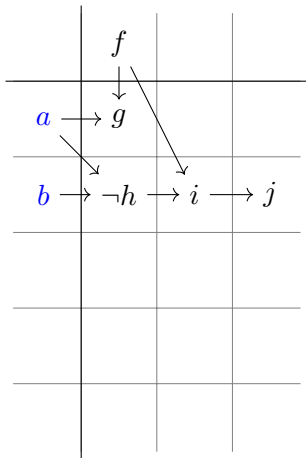
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

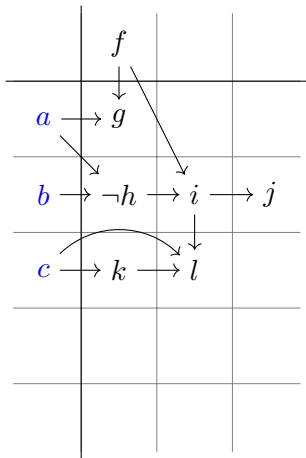
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

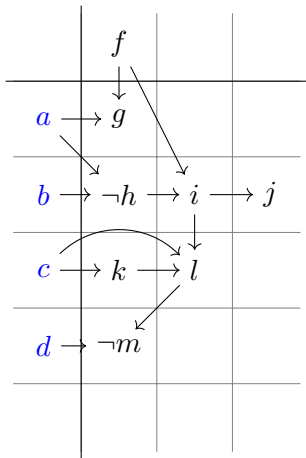
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

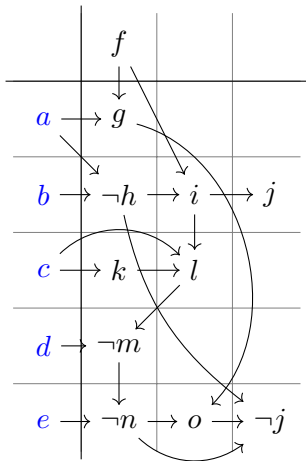
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

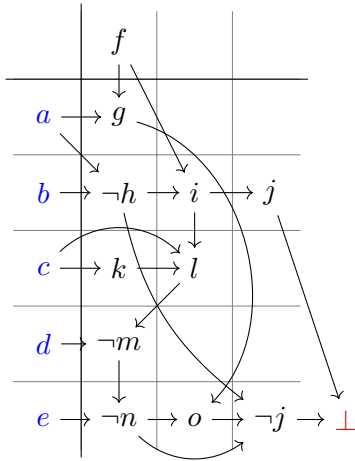
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

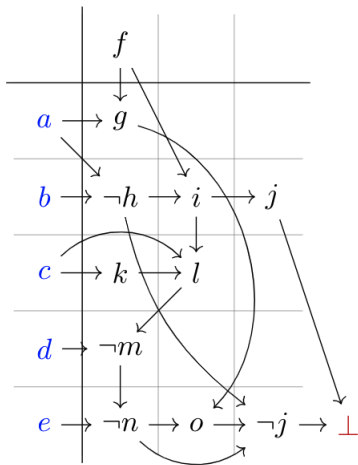
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Implication Graph



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

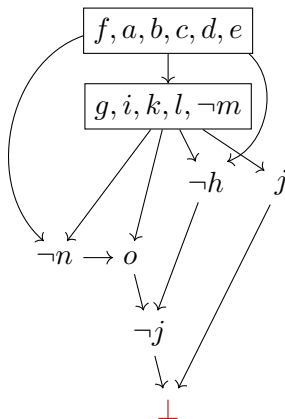
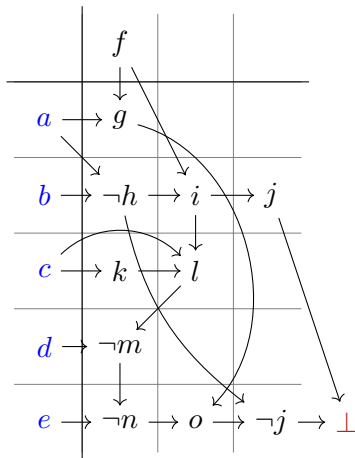
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

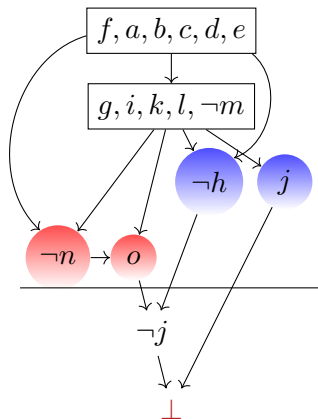
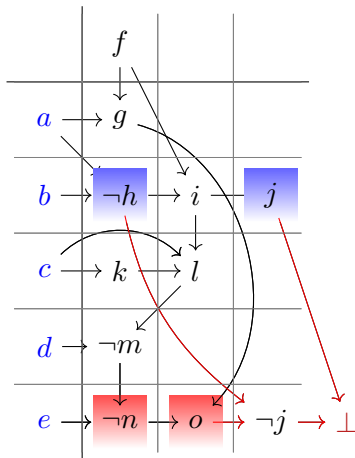
$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

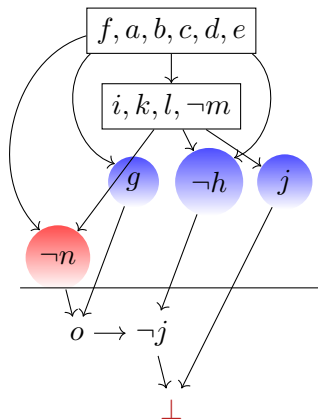
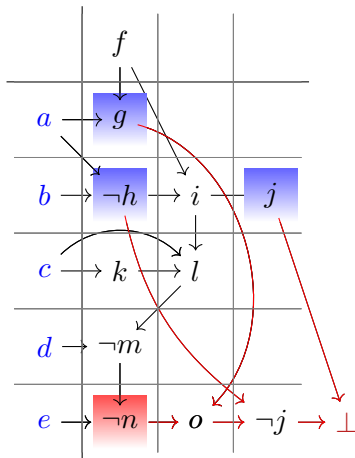
Conflict Analysis



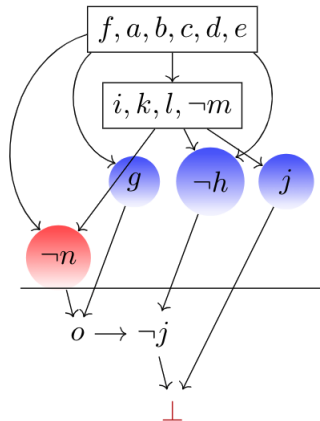
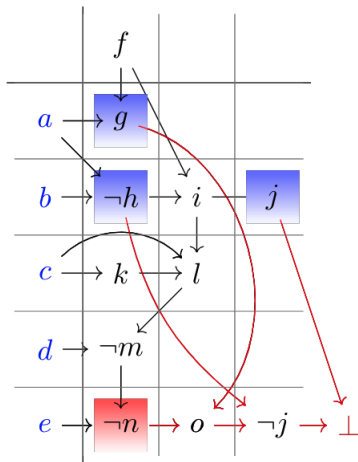
Conflict Analysis



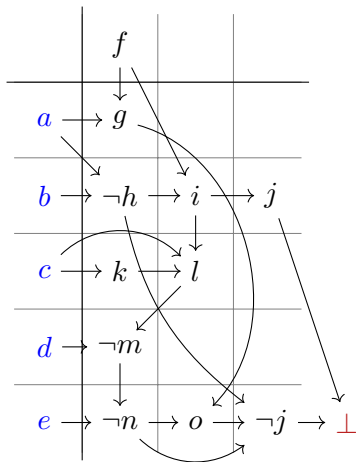
Conflict Analysis



Conflict Analysis



Conflict analysis



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

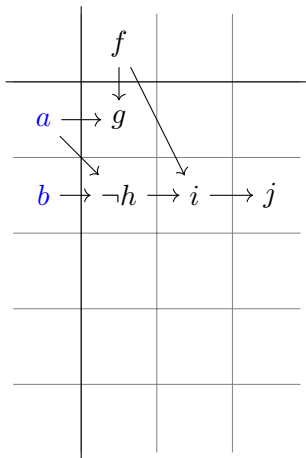
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Conflict analysis



$$\neg a \vee \neg f \vee g$$

$$\neg a \vee \neg b \vee \neg h$$

$$a \vee c$$

$$a \vee \neg i \vee \neg l$$

$$a \vee \neg k \vee \neg j$$

$$b \vee d$$

$$b \vee g \vee \neg n$$

$$b \vee \neg f \vee n \vee k$$

$$\neg c \vee k$$

$$\neg c \vee \neg k \vee \neg i \vee l$$

$$c \vee h \vee n \vee \neg m$$

$$c \vee l$$

$$d \vee \neg k \vee l$$

$$d \vee \neg g \vee l$$

$$\neg g \vee n \vee o$$

$$h \vee \neg o \vee \neg j \vee n$$

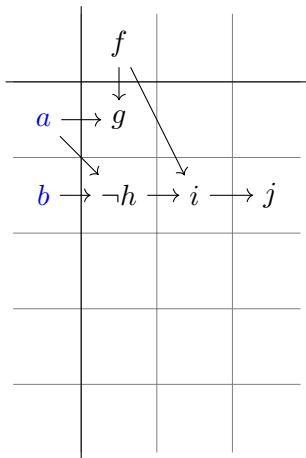
$$\neg i \vee j$$

$$\neg d \vee \neg l \vee \neg m$$

$$\neg e \vee m \vee \neg n$$

$$\neg f \vee h \vee i$$

Conflict analysis

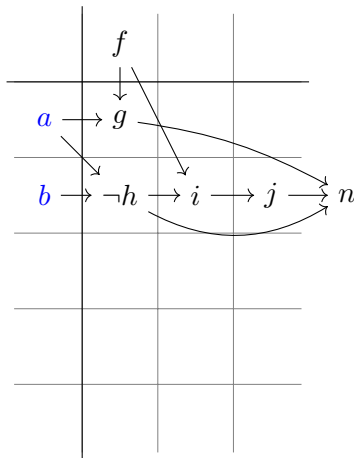


$\neg a \vee \neg f \vee g$
 $\neg a \vee \neg b \vee \neg h$
 $a \vee c$
 $a \vee \neg i \vee \neg l$
 $a \vee \neg k \vee \neg j$
 $b \vee d$
 $b \vee g \vee \neg n$
 $b \vee \neg f \vee n \vee k$
 $\neg c \vee k$
 $\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$
 $c \vee l$
 $d \vee \neg k \vee l$
 $d \vee \neg g \vee l$
 $\neg g \vee n \vee o$
 $h \vee \neg o \vee \neg j \vee n$
 $\neg i \vee j$
 $\neg d \vee \neg l \vee \neg m$
 $\neg e \vee m \vee \neg n$
 $\neg f \vee h \vee i$

$\neg g \vee h \vee \neg j \vee n$

Conflict analysis



$\neg a \vee \neg f \vee g$
 $\neg a \vee \neg b \vee \neg h$
 $a \vee c$
 $a \vee \neg i \vee \neg l$
 $a \vee \neg k \vee \neg j$
 $b \vee d$
 $b \vee g \vee \neg n$
 $b \vee \neg f \vee n \vee k$
 $\neg c \vee k$
 $\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$
 $c \vee l$
 $d \vee \neg k \vee l$
 $d \vee \neg g \vee l$
 $\neg g \vee n \vee o$
 $h \vee \neg o \vee \neg j \vee n$
 $\neg i \vee j$
 $\neg d \vee \neg l \vee \neg m$
 $\neg e \vee m \vee \neg n$
 $\neg f \vee h \vee i$

$\neg g \vee h \vee \neg j \vee n$

Learning and Backjumping

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ and let it be the initial explanation

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than a literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than a literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal uip propagated in the last level in the current explanation, learn the associated new clause C ,

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than a literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal uip propagated in the last level in the current explanation, learn the associated new clause C , backjump (to the last level of propagated literals in C),

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than a literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal uip propagated in the last level in the current explanation, learn the associated new clause C , backjump (to the last level of propagated literals in C), propagate $\neg uip$ via the new clause,

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \dots \vee \neg l_n$ is the clause triggering failure
- Definition: Explaining a propagation of l : $l_1 \wedge \dots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \dots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \dots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than a literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal uip propagated in the last level in the current explanation, learn the associated new clause C , backjump (to the last level of propagated literals in C), propagate $\neg uip$ via the new clause, and continue the exploration

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Heavy-tail phenomena (SAT and CP)

At any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before.

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Heavy-tail phenomena (SAT and CP)

At any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before.

Hardness = Instance \oplus deterministic algorithm.

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Heavy-tail phenomena (SAT and CP)

At any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before.

Hardness = Instance \oplus deterministic algorithm.

- Randomization: breaking ties, random decision between k best choices, ...
- Restarts: Geometric/Luby

Other techniques

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity. The activity of a variable is measured by the number of times it participates in the conflict analysis.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity. The activity of a variable is measured by the number of times it participates in the conflict analysis. Each time a variable x is used during conflict analysis, its activity is incremented.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity. The activity of a variable is measured by the number of times it participates in the conflict analysis. Each time a variable x is used during conflict analysis, its activity is incremented. From time to time, the counters are divided by a constant (to diminish the effect of early conflicts).

SAT Solvers (Few examples)

- MiniSat: <http://minisat.se/>
- Glucose: <http://www.labri.fr/perso/lsimon/glucose/>
- Lingeling <http://fmv.jku.at/lingeling>
- Any Solver by Armin Biere
<http://fmv.jku.at/software/index.html>
- Any winner from past and future SAT competitions:
<https://www.satcompetition.org/>

References I



Davis, M., Logemann, G., and Loveland, D. (1962).

A Machine Program for Theorem-proving.

Communications of the ACM, 5(7):394–397.



Gomes, C. P., Selman, B., and Kautz, H. (1998).

Boosting Combinatorial Search Through Randomization.

In *Proceedings of the 15th National Conference on Artificial Intelligence, AAAI'98, and the 10th Conference on Innovative Applications of Artificial Intelligence, IAAI'98, Madison, Wisconsin*, pages 431–437.



Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001).

Chaff: Engineering an Efficient SAT Solver.

In *Proceedings of the 38th Annual Design Automation Conference, DAC'01, Las Vegas, Nevada, USA*, pages 530–535.



Robinson, J. A. (1965).

A Machine-Oriented Logic Based on the Resolution Principle.

Journal of the ACM, 12(1):23–41.



Silva, J. a. P. M. and Sakallah, K. A. (1999).

Grasp: a search algorithm for propositional satisfiability.

Computers, IEEE Transactions on, 48(5):506–521.