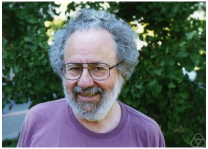
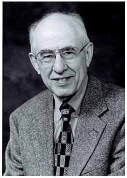
DPLL: satisfiability algorithm for formulas in clausal normal form

DPLL: satisfiability of clausal normal forms

DPLL:



Martin Davis



Hilary Putnam

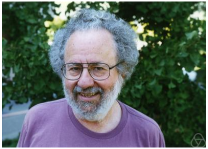
Donald Loveland

George Logemann

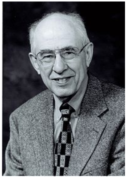
DPLL (Davis, Putnam, Loveland and Logemann)
A method for checking satisfiability of sets of cluases.

DPLL: satisfiability of clausal normal forms

DPLL:



Martin Davis



Hilary Putnam

Donald Loveland

George Logemann

DPLL (Davis, Putnam, Loveland and Logemann)
A method for checking satisfiability of sets of cluases.

DPLL: satisfiability of clausal normal forms

DPLL ingredients:

- Unit Propagation
- Splitting

Satisfiability-checking for sets of clauses

The CNF transformation of

$$\neg((p \to q) \land (p \land q \to r) \to (p \to r))$$

gives the set of four clauses:

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

Every interpretation that satisfies this set of clauses must assign 1 to p and 0 to r, so we do not have to guess values of these variables.

In fact, we can do even better and establish unsatisfiability in this case without any guessing.

Satisfiability-checking for sets of clauses

The CNF transformation of

$$\neg((p \to q) \land (p \land q \to r) \to (p \to r))$$

gives the set of four clauses:

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

Every interpretation that satisfies this set of clauses must assign 1 to p and 0 to r, so we do not have to guess values of these variables.

In fact, we can do even better and establish unsatisfiability in this case without any guessing.

Satisfiability-checking for sets of clauses

The CNF transformation of

$$\neg((p \to q) \land (p \land q \to r) \to (p \to r))$$

gives the set of four clauses:

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

Every interpretation that satisfies this set of clauses must assign 1 to p and 0 to r, so we do not have to guess values of these variables.

In fact, we can do even better and establish unsatisfiability in this case without any guessing.

Searching for satisfiability

$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$

$\neg p \vee q$
$\neg p \vee \neg q \vee r$
p
$\neg r$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$

$\neg p \vee q$
$\neg p \vee \neg q \vee r$
p
$\neg r$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$

$\neg p \vee q$
$\neg p \vee \neg q \vee r$
p
$\neg r$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \vee q \\ \neg p \vee \neg q \vee r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$\neg p \vee q$
$\neg p \vee \neg q \vee r$
p
$\neg r$

This set of clauses is unsatisfiable.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\begin{array}{l} \neg p \lor q \\ \neg p \lor \neg q \lor r \, \square \\ p \\ \neg r \end{array}$$

This set of clauses is unsatisfiable.

Unit propagation

Let S be a set of clauses.

Unit propagation. Repeatedly perform the following transformation:

If S contains a unit clause, i.e. a clause consisting of one literal L, then

1. remove from S every clause of the form $L \vee C$;
2. replace in S every clause of the form $\overline{L} \vee C$ by the clause C.

Lemma. Unit propagation is satisfiability preserving transformation.

Unit propagation

Let S be a set of clauses.

Unit propagation. Repeatedly perform the following transformation:

If S contains a unit clause, i.e. a clause consisting of one literal L, then

1. remove from S every clause of the form $L \vee C$;
2. replace in S every clause of the form $\overline{L} \vee C$ by the clause C.

Lemma. Unit propagation is satisfiability preserving transformation.

Unit Propagation, Example

n_1
$\neg n_1 \vee \neg n_2$
$n_1 \vee n_2$
$\neg n_2 \vee \neg n_3 \vee n_7$
$n_3 \vee n_2$
$\neg n_7 \vee n_2$
$\neg n_3 \vee n_4$
$\neg n_3 \vee n_5$
$\neg n_4 \vee \neg n_5 \vee n_3$
$\neg n_4 \vee \neg p \vee q$
$p \vee n_4$
$\neg q \vee n_4$
$\neg n_5 \vee \neg n_6 \vee r$
$n_6 \vee n_5$
$\neg r \vee n_5$
$\neg n_6 \vee p$
$\neg n_6 \vee q$
$\neg p \vee \neg q \vee n_6$
$\neg n_7 \vee \neg p \vee r$
$p \vee n_7$
$\neg r \vee n_7$

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

n_1	$\neg q \vee n_4$
$\neg n_1 \vee \neg n_2$	$\neg n_5 \vee \neg n_6 \vee r$
$n_1 \vee n_2$	$n_6 \vee n_5$
$\neg n_2 \vee \neg n_3 \vee n_7$	$\neg r \vee n_5$
$n_3 \vee n_2$	$\neg n_6 \vee p$
$\neg n_7 \vee n_2$	$\neg n_6 \vee q$
$\neg n_3 \vee n_4$	$\neg p \vee \neg q \vee n_6$
$\neg n_3 \vee n_5$	$\neg n_7 \vee \neg p \vee r$
$\neg n_4 \vee \neg n_5 \vee n_3$	$p \vee n_7$
$\neg n_4 \vee \neg p \vee q$	$\neg r \vee n_7$
$p \vee n_4$	

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

$\neg q \vee n_4$

$\neg n_2$

$\neg n_5 \vee \neg n_6 \vee r$

$n_6 \vee n_5$

$\neg n_2 \vee \neg n_3 \vee n_7$

$\neg r \vee n_5$

$n_3 \vee n_2$

$\neg n_6 \vee p$

$\neg n_7 \vee n_2$

$\neg n_6 \vee q$

$\neg n_3 \vee n_4$

$\neg p \vee \neg q \vee n_6$

$\neg n_3 \vee n_5$

$\neg n_7 \vee \neg p \vee r$

$\neg n_4 \vee \neg n_5 \vee n_3$

$p \vee n_7$

$\neg n_4 \vee \neg p \vee q$

$\neg r \vee n_7$

$p \vee n_4$

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

$\neg n_2$

$\neg n_2 \lor \neg n_3 \lor n_7$
$n_3 \lor n_2$
$\neg n_7 \lor n_2$
$\neg n_3 \lor n_4$
$\neg n_3 \lor n_5$
$\neg n_4 \lor \neg n_5 \lor n_3$
$\neg n_4 \lor \neg p \lor q$
$p \lor n_4$

$\neg q \lor n_4$
$\neg n_5 \lor \neg n_6 \lor r$
$n_6 \lor n_5$
$\neg r \lor n_5$
$\neg n_6 \lor p$
$\neg n_6 \lor q$
$\neg p \lor \neg q \lor n_6$
$\neg n_7 \lor \neg p \lor r$
$p \lor n_7$
$\neg r \lor n_7$

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

	$\neg q \vee n_4$
	$\neg n_5 \vee \neg n_6 \vee r$
	$n_6 \vee n_5$
	$\neg r \vee n_5$
n_3	$\neg n_6 \vee p$
$\neg n_7$	$\neg n_6 \vee q$
$\neg n_3 \vee n_4$	$\neg p \vee \neg q \vee n_6$
$\neg n_3 \vee n_5$	$\neg n_7 \vee \neg p \vee r$
$\neg n_4 \vee \neg n_5 \vee n_3$	$p \vee n_7$
$\neg n_4 \vee \neg p \vee q$	$\neg r \vee n_7$
$p \vee n_4$	

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

n_3
$\neg n_7$
$\neg n_3 \vee n_4$
$\neg n_3 \vee n_5$
$\neg n_4 \vee \neg n_5 \vee n_3$
$\neg n_4 \vee \neg p \vee q$
$p \vee n_4$

$\neg q \vee n_4$
$\neg n_5 \vee \neg n_6 \vee r$
$n_6 \vee n_5$
$\neg r \vee n_5$
$\neg n_6 \vee p$
$\neg n_6 \vee q$
$\neg p \vee \neg q \vee n_6$
$\neg n_7 \vee \neg p \vee r$
$p \vee n_7$
$\neg r \vee n_7$

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

	$\neg q \vee n_4$
	$\neg n_5 \vee \neg n_6 \vee r$
	$n_6 \vee n_5$
	$\neg r \vee n_5$
	$\neg n_6 \vee p$
	$\neg n_6 \vee q$
n_4	$\neg p \vee \neg q \vee n_6$
n_5	
	p
$\neg n_4 \vee \neg p \vee q$	$\neg r$
$p \vee n_4$	

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

$\neg q \vee n_4$
$\neg n_5 \vee \neg n_6 \vee r$
$n_6 \vee n_5$
$\neg r \vee n_5$
$\neg n_6 \vee p$
$\neg n_6 \vee q$
n_4 $\neg p \vee \neg q \vee n_6$
n_5

p
$\neg n_4 \vee \neg p \vee q$ $\neg r$
$p \vee n_4$

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

$\neg n_6$

$\neg n_6 \vee q$

$\neg q \vee n_6$

q

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

$\neg n_6$

$\neg n_6 \vee q$

$\neg q \vee n_6$

q

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

Unit Propagation, Example

$\square$

Propagating one unit

- completely removes all occurrences of the variable of this unit.
- can results in more units that in turn can be propagated!

We established unsatisfiability of this set of clauses in a completely deterministic way, by unit propagation.

DPLL $=$ splitting $+$ unit propagation

procedure $DPLL(S)$
input: set of clauses S
output: *satisfiable* or *unsatisfiable*
parameters: function $select\_literal$
begin
  $S := propagate(S)$
  if S is empty **then** **return** *satisfiable*
  if S contains $\Box$ **then** **return** *unsatisfiable*
  $L := select\_literal(S)$
  if $DPLL(S \cup \{L\}) = satisfiable$
    then **return** *satisfiable*
    else **return** $DPLL(S \cup \{\overline{L}\})$
end

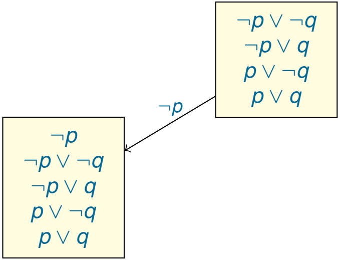
DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).

$$\begin{array}{c} \neg p \lor \neg q \\ \neg p \lor q \\ p \lor \neg q \\ p \lor q \end{array}$$

DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).



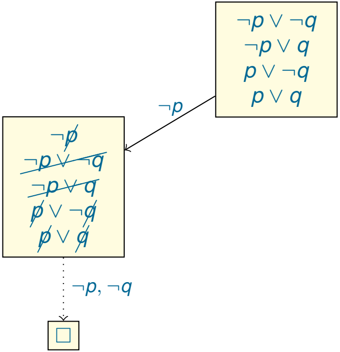
DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).

$$\begin{tikzcd}
 & \boxed{\begin{array}{c} \neg p \lor \neg q \\ \neg p \lor q \\ p \lor \neg q \\ p \lor q \end{array}} \arrow[dl, "\neg p"'] \\
\boxed{\begin{array}{c} \neg \cancel{p} \\ \cancel{\neg p \lor \neg q} \\ \cancel{\neg p \lor q} \\ \cancel{p} \lor \neg q \\ \cancel{p} \lor q \end{array}} &
\end{tikzcd}$$

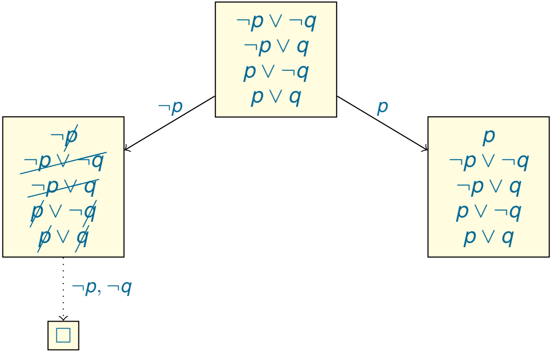
DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).



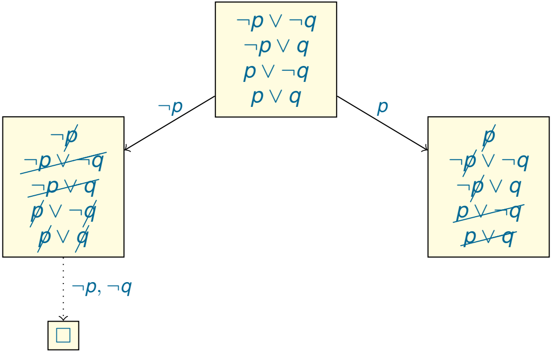
DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).



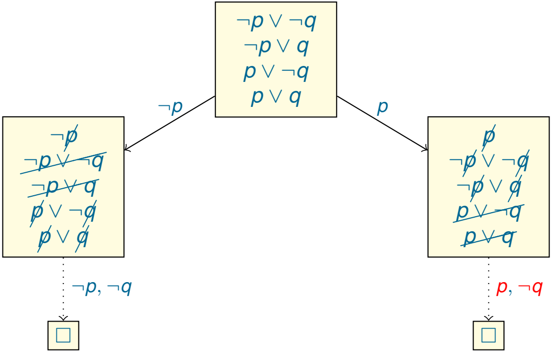
DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).



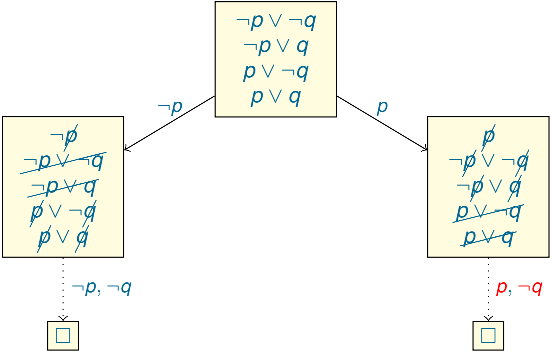
DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).



DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).



Since all branches end up in a set contaning the empty clause, the initial set of clauses is unsatisfiable.

DPLL. Example 2

$$\begin{array}{c} \neg p \vee \neg q \\ \neg p \vee q \\ p \vee \neg q \end{array}$$

p

$$\begin{array}{c} p \\ \neg p \vee \neg q \\ \neg p \vee q \\ p \vee \neg q \end{array}$$

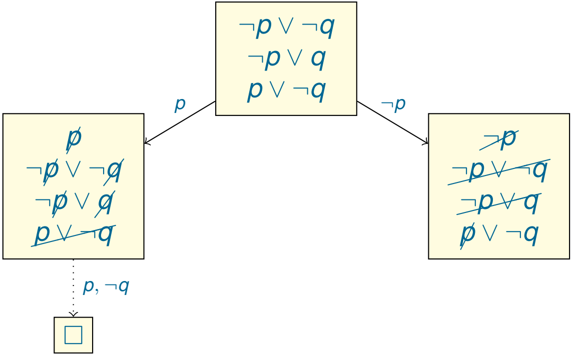
DPLL. Example 2

$$\begin{array}{c} \cancel{p} \\ \neg\cancel{p} \vee \neg q \\ \neg\cancel{p} \vee q \\ \cancel{p \vee \neg q} \end{array} \xleftarrow{p} \begin{array}{c} \neg p \vee \neg q \\ \neg p \vee q \\ p \vee \neg q \end{array}$$

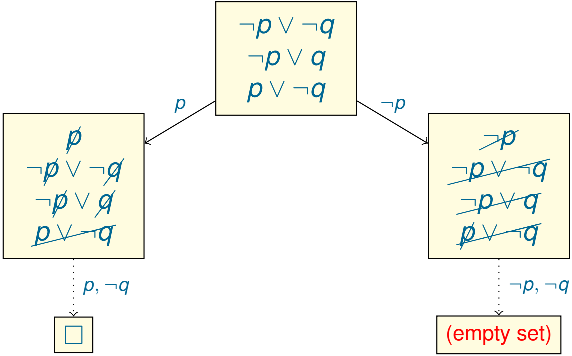
DPLL. Example 2

$$
\begin{tikzcd}
 & \begin{array}{c} \neg p \vee \neg q \\ \neg p \vee q \\ p \vee \neg q \end{array} \arrow[dl, "p"'] \\
\begin{array}{c} \cancel{p} \\ \neg \cancel{p} \vee \neg \cancel{q} \\ \neg \cancel{p} \vee \cancel{q} \\ \cancel{p \vee \neg q} \end{array} \arrow[d, dotted, "{p, \neg q}"] & \\
\square &
\end{tikzcd}
$$

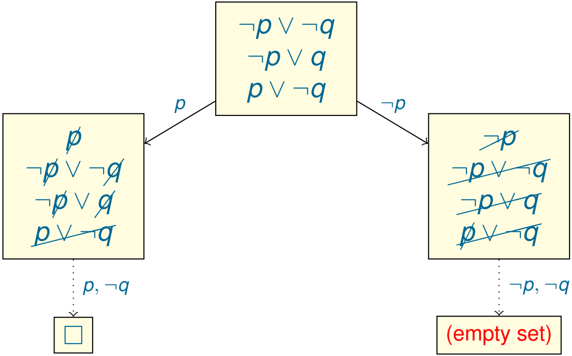
DPLL. Example 2



DPLL. Example 2

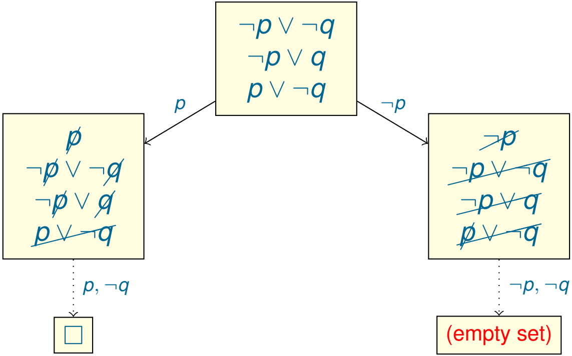


DPLL. Example 2



The set of clauses is satisfiable.

DPLL. Example 2



The set of clauses is satisfiable.

The model can be obtained by collecting all selected literals and literals used in unit propagation on the branch resulting in the empty set.

This DPLL tree gives us the model $\{p \mapsto 0, q \mapsto 0\}$.

Optimisation: tautology elimination

Tautologies are valid formulas.

Tautological clauses have the form $p \lor \neg p \lor C$:

- Every clause of the form $p \lor \neg p \lor C$ is tautology (why ?)
- There are not other tautological clauses (why ?)

Tautology elimination. We can remove tautological clauses without affecting satisfiability. (why ?)

Optimisation: tautology elimination

Tautologies are valid formulas.

Tautological clauses have the form $p \vee \neg p \vee C$:

- Every clause of the form $p \vee \neg p \vee C$ is tautology (why ?)
- There are not other tautological clauses (why ?)

Tautology elimination. We can remove tautological clauses without affecting satisfiability. (why ?)

Pure literal elimination

A literal L in S is called pure
if S contains no clauses of the form $\overline{L} \vee C$.

Optimization (pure literal elimination). We can remove clauses with pure literals without affecting satisfiability. (why ?)

Remember pure atom rule positive (negative): if an atom p occurs only positively (negatively) in a formula then we can replace it by $\top$ (respectively $\perp$).

Pure literal elimination

A literal L in S is called pure
if S contains no clauses of the form $\overline{L} \lor C$.

Optimization (pure literal elimination). We can remove clauses with pure literals without affecting satisfiability. (why ?)

Remember pure atom rule positive (negative): if an atom p occurs only positively (negatively) in a formula then we can replace it by $\top$ (respectively $\bot$).

Pure literal elimination

A literal L in S is called pure
if S contains no clauses of the form $\overline{L} \vee C$.

Optimization (pure literal elimination). We can remove clauses with pure literals without affecting satisfiability. (why ?)

Remember pure atom rule positive (negative): if an atom p occurs only positively (negatively) in a formula then we can replace it by $\top$ (respectively $\perp$).

Pure literals: example

$$\neg p_2 \vee \neg p_3$$
$$p_1 \vee \neg p_2$$
$$\neg p_1 \vee p_2 \vee \neg p_3$$
$$\neg p_1 \vee \neg p_3$$
$$p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee \neg p_3$$

Pure literals: example

$$\neg p_2 \lor \neg p_3$$
$$p_1 \lor \neg p_2$$
$$\neg p_1 \lor p_2 \lor \neg p_3$$
$$\neg p_1 \lor \neg p_3$$
$$p_1 \lor p_2$$
$$\neg p_1 \lor \neg p_2 \lor \neg p_3$$

The literal $\neg p_3$ is pure in this set. We can remove all clauses containing this literal.

Pure literals: example

$$p_1 \lor \neg p_2$$

$$p_1 \lor p_2$$

The literal $\neg p_3$ is pure in this set. We can remove all clauses containing this literal.

Pure literals: example

$$p_1 \lor \neg p_2$$

$$p_1 \lor p_2$$

The literal $\neg p_3$ is pure in this set. We can remove all clauses containing this literal.

Now the literal p_1 is pure in this set. We can remove all clauses containing this literal.

Pure literals: example

$$\neg p_2 \lor \neg p_3$$
$$p_1 \lor \neg p_2$$
$$\neg p_1 \lor p_2 \lor \neg p_3$$
$$\neg p_1 \lor \neg p_3$$
$$p_1 \lor p_2$$
$$\neg p_1 \lor \neg p_2 \lor \neg p_3$$

The literal $\neg p_3$ is pure in this set. We can remove all clauses containing this literal.

Now the literal p_1 is pure in this set. We can remove all clauses containing this literal.

We obtained the empty set of clauses. Therefore this set is satisfiable.

Pure literals: example

$$\neg p_2 \vee \neg p_3$$
$$p_1 \vee \neg p_2$$
$$\neg p_1 \vee p_2 \vee \neg p_3$$
$$\neg p_1 \vee \neg p_3$$
$$p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee \neg p_3$$

The literal $\neg p_3$ is pure in this set. We can remove all clauses containing this literal.

Now the literal p_1 is pure in this set. We can remove all clauses containing this literal.

We obtained the empty set of clauses. Therefore this set is satisfiable.

This gives us two models:

$$\{p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0\}$$
$$\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 0\}$$

Horn clauses

A clause is called Horn if it contains at most one positive literal.

Examples: The following clauses are Horn:

$$p_1$$
$$\neg p_1 \lor p_2$$
$$\neg p_1 \lor \neg p_2 \lor p_3$$
$$\neg p_3 \lor \neg p_4$$

The following clauses are non-Horn:

$$p_1 \lor p_2$$
$$p_1 \lor \neg p_2 \lor p_3$$

Horn clauses

A clause is called Horn if it contains at most one positive literal.

Examples: The following clauses are Horn:

$$p_1$$
$$\neg p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee p_3$$
$$\neg p_3 \vee \neg p_4$$

The following clauses are non-Horn:

$$p_1 \vee p_2$$
$$p_1 \vee \neg p_2 \vee p_3$$

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$
\begin{gathered}
p_1 \\
\neg p_1 \vee p_2 \\
\neg p_1 \vee \neg p_2 \vee p_3 \\
\neg p_3 \vee \neg p_4
\end{gathered}
$$

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$
\begin{gathered}
p_1 \\
\neg p_1 \vee p_2 \\
\neg p_1 \vee \neg p_2 \vee p_3 \\
\neg p_3 \vee \neg p_4
\end{gathered}
$$

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$p_2$$
$$\neg p_2 \vee p_3$$
$$\neg p_3 \vee \neg p_4$$

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$p_3$$

$$\neg p_3 \vee \neg p_4$$

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$\neg p_4$

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$p_1$$
$$\neg p_1 \lor p_2$$
$$\neg p_1 \lor \neg p_2 \lor p_3$$
$$\neg p_3 \lor \neg p_4$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{array}{c} p_1 \\ \neg p_1 \vee p_2 \\ \neg p_1 \vee \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{array}$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{gathered} p_1 \\ \neg p_1 \vee p_2 \\ \neg p_1 \vee \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{gathered}$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.
Therefore, unit propagation applied to a set S of Horn clauses gives a set S' of Horn clauses.

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{gathered} p_1 \\ \neg p_1 \vee p_2 \\ \neg p_1 \vee \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{gathered}$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause. Therefore, unit propagation applied to a set S of Horn clauses gives a set S' of Horn clauses.

Two cases:

1. S' contains $\square$. Then, S' (and hence S) is unsatisfiable.
2. S' does not contain $\square$.
   - Each clause in S' has at least two literals.
   - Hence each clause in S' contains at least one negative literal.
   - Hence setting all variables in S' to 0 satisfies S'.

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{gathered} p_1 \\ \neg p_1 \vee p_2 \\ \neg p_1 \vee \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{gathered}$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$
Note that deleting a literal from a Horn clause gives a Horn clause. Therefore, unit propagation applied to a set S of Horn clauses gives a set S' of Horn clauses.

Two cases:

1. S' contains $\square$. Then, S' (and hence S) is unsatisfiable.
2. S' does not contain $\square$.
   - Each clause in S' has at least two literals.
   - Hence each clause in S' contains at least one negative literal;
   - Hence setting all variables in S' to 0 satisfies S'.

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{gathered} p_1 \\ \neg p_1 \vee p_2 \\ \neg p_1 \vee \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{gathered}$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$
Note that deleting a literal from a Horn clause gives a Horn clause. Therefore, unit propagation applied to a set S of Horn clauses gives a set S' of Horn clauses.

Two cases:

1. S' contains $\square$. Then, S' (and hence S) is unsatisfiable.
2. S' does not contain $\square$. Then:
   - Each clause in S' has at least two literals.
   - Hence each clause in S' contains at least one negative literal;
   - Hence setting all variables in S' to 0 satisfies S'.

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$p_1$$
$$\neg p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee p_3$$
$$\neg p_3 \vee \neg p_4$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$
Note that deleting a literal from a Horn clause gives a Horn clause.
Therefore, unit propagation applied to a set S of Horn clauses gives a set S' of Horn clauses.

Two cases:

1. S' contains $\square$. Then, S' (and hence S) is unsatisfiable.
2. S' does not contain $\square$. Then:
   - Each clause in S' has at least two literals.
   - Hence each clause in S' contains at least one negative literal;
   - Hence setting all variables in S' to 0 satisfies S'.

Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{gathered} p_1 \\ \neg p_1 \vee p_2 \\ \neg p_1 \vee \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{gathered}$$

Model: $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.
Therefore, unit propagation applied to a set S of Horn clauses gives a set S' of Horn clauses.

Two cases:

1. S' contains $\square$. Then, S' (and hence S) is unsatisfiable.
2. S' does not contain $\square$. Then:
   - Each clause in S' has at least two literals.
   - Hence each clause in S' contains at least one negative literal;
   - Hence setting all variables in S' to 0 satisfies S'.

Running a SAT solver

Small but very efficient SAT solver: MiniSat, `http://minisat.se/`.

Running a SAT solver

Small but very efficient SAT solver: MiniSat, `http://minisat.se/`.

$$p_1$$
$$\neg p_1 \lor p_2$$
$$\neg p_1 \lor \neg p_2 \lor p_3$$
$$\neg p_2 \lor \neg p_3$$

Running a SAT solver

Small but very efficient SAT solver: MiniSat, `http://minisat.se/`.

$$\begin{gathered} p_1 \\ \neg p_1 \lor p_2 \\ \neg p_1 \lor \neg p_2 \lor p_3 \\ \neg p_2 \lor \neg p_3 \end{gathered}$$

DIMACS input format:

```
p cnf 3 4
1 0
-1 2 0
-1 -2 3 0
-2 -3 0
```

Running a SAT solver

Small but very efficient SAT solver: MiniSat, `http://minisat.se/`.

$$
\begin{gathered}
p_1 \\
\neg p_1 \lor p_2 \\
\neg p_1 \lor \neg p_2 \lor p_3 \\
\neg p_2 \lor \neg p_3
\end{gathered}
$$

DIMACS input format:

```
p cnf 3 4
1 0
-1 2 0
-1 -2 3 0
-2 -3 0
```

3 variables, 4 clauses.

Running a SAT solver

Small but very efficient SAT solver: MiniSat, `http://minisat.se/`.

$$p_1$$
$$\neg p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee p_3$$
$$\neg p_2 \vee \neg p_3$$

DIMACS input format:

```
p cnf 3 4
1 0
-1 2 0
-1 -2 3 0
-2 -3 0
```

3 variables, 4 clauses.

Running a SAT solver

Small but very efficient SAT solver: MiniSat, `http://minisat.se/`.

$$p_1$$
$$\neg p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee p_3$$
$$\neg p_2 \vee \neg p_3$$

DIMACS input format:

```
p cnf 3 4
1 0
-1 2 0
-1 -2 3 0
-2 -3 0
```

3 variables, 4 clauses.

Advanced techniques: CDCL: Backjumping and lemma learning
Extra material/links will be added to BB (but not part of this course)
Implement DPLL/CDCL (optional)

Probabilistic analysis of satisfiability

Next:

- What is quantitative relationship between satisfiable and unsatisfiable problems? In other words if we pick a set of clauses at random with what probability it will be satisfiable?
- How can we randomly generate hard problems?
- Randomized algorithms for showing satisfiability.

SAT and k-SAT

SAT is the problem of satisfiability checking for sets of clauses.

A k-clause is a cluase with k literals. k-SAT is the problem of satisfiability checking for sets of k-clauses.

- SAT is NP-complete;
- 3-SAT is NP-complete.
- 2-SAT is decidable in linear time;

There is a simple reduction of SAT to 3-SAT based on the same ideas as used for generating short clausal forms (naming). Take a clause having more than 3 literals:

$$L_1 \vee L_2 \vee L_3 \vee L_4 \ldots$$

And replace it by two clauses (optimised definitional transformation):

$$\neg n \vee L_1 \vee L_2$$
$$n \vee L_3 \vee L_4 \ldots$$

where n is a fresh variable.

We will consider k-SAT for a fixed k.

SAT and k-SAT

SAT is the problem of satisfiability checking for sets of clauses.

A k-clause is a cluase with k literals. k-SAT is the problem of satisfiability checking for sets of k-clauses.

- SAT is NP-complete;
- 3-SAT is NP-complete.
- 2-SAT is decidable in linear time;

There is a simple reduction of SAT to 3-SAT based on the same ideas as used for generating short clausal forms (naming). Take a clause having more than 3 literals:

$$L_1 \vee L_2 \vee L_3 \vee L_4 \ldots$$

And replace it by two clauses (optimised definitional transformation):

$$\neg n \vee L_1 \vee L_2$$
$$n \vee L_3 \vee L_4 \ldots$$

where n is a fresh variable.
We will consider k-SAT for a fixed k.

SAT and k-SAT

SAT is the problem of satisfiability checking for sets of clauses.

A k-clause is a cluase with k literals. k-SAT is the problem of satisfiability checking for sets of k-clauses.

- SAT is NP-complete;
- 3-SAT is NP-complete.
- 2-SAT is decidable in linear time;

There is a simple reduction of SAT to 3-SAT based on the same ideas as used for generating short clausal forms (naming). Take a clause having more than 3 literals:

$$L_1 \vee L_2 \vee L_3 \vee L_4 \dots$$

And replace it by two clauses (optimised definitional transformation):

$$\neg n \vee L_1 \vee L_2$$
$$n \vee L_3 \vee L_4 \dots$$

where n is a fresh variable.

We will consider k-SAT for a fixed k.

SAT and k-SAT

SAT is the problem of satisfiability checking for sets of clauses.

A k-clause is a cluase with k literals. k-SAT is the problem of satisfiability checking for sets of k-clauses.

- SAT is NP-complete;
- 3-SAT is NP-complete.
- 2-SAT is decidable in linear time;

There is a simple reduction of SAT to 3-SAT based on the same ideas as used for generating short clausal forms (naming). Take a clause having more than 3 literals:

$$L_1 \vee L_2 \vee L_3 \vee L_4 \ldots$$

And replace it by two clauses (optimised definitional transformation):

$$\begin{array}{c} \neg n \vee L_1 \vee L_2 \\ n \vee L_3 \vee L_4 \ldots \end{array}$$

where n is a fresh variable.
We will consider k-SAT for a fixed k.

SAT and k-SAT

SAT is the problem of satisfiability checking for sets of clauses.

A k-clause is a cluase with k literals. k-SAT is the problem of satisfiability checking for sets of k-clauses.

- SAT is NP-complete;
- 3-SAT is NP-complete.
- 2-SAT is decidable in linear time;

There is a simple reduction of SAT to 3-SAT based on the same ideas as used for generating short clausal forms (naming). Take a clause having more than 3 literals:

$$L_1 \vee L_2 \vee L_3 \vee L_4 \ldots$$

And replace it by two clauses (optimised definitional transformation):

$$\begin{array}{c} \neg n \vee L_1 \vee L_2 \\ n \vee L_3 \vee L_4 \ldots \end{array}$$

where n is a fresh variable.

We will consider k-SAT for a fixed k.

SAT and k-SAT

SAT is the problem of satisfiability checking for sets of clauses.

A k-clause is a cluase with k literals. k-SAT is the problem of satisfiability checking for sets of k-clauses.

- SAT is NP-complete;
- 3-SAT is NP-complete.
- 2-SAT is decidable in linear time;

There is a simple reduction of SAT to 3-SAT based on the same ideas as used for generating short clausal forms (naming). Take a clause having more than 3 literals:

$$L_1 \lor L_2 \lor L_3 \lor L_4 \ldots$$

And replace it by two clauses (optimised definitional transformation):

$$\neg n \lor L_1 \lor L_2$$
$$n \lor L_3 \lor L_4 \ldots$$

where n is a fresh variable.
We will consider k-SAT for a fixed k.

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \vee \neg p_5 \vee p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \vee \neg p_5 \vee p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \vee \neg p_5 \vee p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \vee \neg p_5 \vee p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \vee \neg p_5 \vee p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \lor \neg p_5 \lor p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \vee \neg p_5 \vee p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \lor \neg p_5 \lor p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \vee \neg p_5 \vee p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Random Clause Generation

How can one generate a random k-clause?

A random clause is a collection of random literals.

Let's first generate a random literal.

- Fix a number n of boolean variables;
- Select a literal among $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$ with an equal probability.
  What is a probability of selecting $\neg p_3$?: $\frac{1}{2n}$
- random k-clause: generate k random literals What is a probability of generating $p_2 \lor \neg p_5 \lor p_3$? $\frac{1}{(2n)^3}$

Suppose we generate random clauses one after one. How does the set of models of this set change?

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1
1	1	1	0	0
1	1	1	0	1
1	1	1	1	0
1	1	1	1	1

Number of models: 32

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$

p_1	p_2	p_3	p_4	p_5
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1
1	1	1	0	0
1	1	1	0	1
1	1	1	1	0
1	1	1	1	1

Number of models: 32

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$

p_1	p_2	p_3	p_4	p_5
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 24

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \lor \neg p_3$
$\neg p_2 \lor p_1$

p_1	p_2	p_3	p_4	p_5
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 24

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$

p_1	p_2	p_3	p_4	p_5
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 20

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \lor \neg p_3$
$\neg p_2 \lor p_1$
$\neg p_2 \lor p_2$

p_1	p_2	p_3	p_4	p_5
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 20

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$

p_1	p_2	p_3	p_4	p_5
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 20

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 12

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 12

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5

$\neg p_2 \lor \neg p_3$
$\neg p_2 \lor p_1$
$\neg p_2 \lor p_2$
$p_1 \lor p_1$
$\neg p_5 \lor p_5$
$p_4 \lor p_5$

p_1	p_2	p_3	p_4	p_5
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 12

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 9

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$
$\neg p_5 \vee \neg p_3$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 9

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$
$\neg p_5 \vee \neg p_3$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 7

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \lor \neg p_3$
$\neg p_2 \lor p_1$
$\neg p_2 \lor p_2$
$p_1 \lor p_1$
$\neg p_5 \lor p_5$
$p_4 \lor p_5$
$\neg p_5 \lor \neg p_3$
$p_2 \lor \neg p_4$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 7

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$
$\neg p_5 \vee \neg p_3$
$p_2 \vee \neg p_4$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 4

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$
$\neg p_5 \vee \neg p_3$
$p_2 \vee \neg p_4$
$p_5 \vee \neg p_2$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 4

Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$
$\neg p_5 \vee \neg p_3$
$p_2 \vee \neg p_4$
$p_5 \vee \neg p_2$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	1	0	0	1
1	1	0	1	1

Number of models: 3

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5

$\neg p_2 \lor \neg p_3$
$\neg p_2 \lor p_1$
$\neg p_2 \lor p_2$
$p_1 \lor p_1$
$\neg p_5 \lor p_5$
$p_4 \lor p_5$
$\neg p_5 \lor \neg p_3$
$p_2 \lor \neg p_4$
$p_5 \lor \neg p_2$
$p_5 \lor p_2$

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1
1	1	0	0	1
1	1	0	1	1

Number of models: 3

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$
$\neg p_5 \vee \neg p_3$
$p_2 \vee \neg p_4$
$p_5 \vee \neg p_2$
$p_5 \vee p_2$

Number of models: 1

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5

$\neg p_2 \lor \neg p_3$
$\neg p_2 \lor p_1$
$\neg p_2 \lor p_2$
$p_1 \lor p_1$
$\neg p_5 \lor p_5$
$p_4 \lor p_5$
$\neg p_5 \lor \neg p_3$
$p_2 \lor \neg p_4$
$p_5 \lor \neg p_2$
$p_5 \lor p_2$
$\neg p_1 \lor \neg p_4$

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1

Number of models: 1

Example (obtained by a program) for $n = 5$ and $k = 2$

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1

$\neg p_2 \vee \neg p_3$
$\neg p_2 \vee p_1$
$\neg p_2 \vee p_2$
$p_1 \vee p_1$
$\neg p_5 \vee p_5$
$p_4 \vee p_5$
$\neg p_5 \vee \neg p_3$
$p_2 \vee \neg p_4$
$p_5 \vee \neg p_2$
$p_5 \vee p_2$
$\neg p_1 \vee \neg p_4$
$p_5 \vee p_2$

Number of models: 1

Example (obtained by a program) for $n = 5$ and $k = 2$

	p_1	p_2	p_3	p_4	p_5
$\neg p_2 \lor \neg p_3$					
$\neg p_2 \lor p_1$					
$\neg p_2 \lor p_2$					
$p_1 \lor p_1$					
$\neg p_5 \lor p_5$					
$p_4 \lor p_5$					
$\neg p_5 \lor \neg p_3$					
$p_2 \lor \neg p_4$					
$p_5 \lor \neg p_2$					
$p_5 \lor p_2$					
$\neg p_1 \lor \neg p_4$					
$p_5 \lor p_2$					
$\neg p_1 \lor \neg p_5$					

p_1	p_2	p_3	p_4	p_5
1	0	0	0	1

Number of models: 1

Example (obtained by a program) for $n = 5$ and $k = 2$

	p_1	p_2	p_3	p_4	p_5	p_1	p_2	p_3	p_4	p_5
$\neg p_2 \vee \neg p_3$										
$\neg p_2 \vee p_1$										
$\neg p_2 \vee p_2$										
$p_1 \vee p_1$										
$\neg p_5 \vee p_5$										
$p_4 \vee p_5$										
$\neg p_5 \vee \neg p_3$										
$p_2 \vee \neg p_4$										
$p_5 \vee \neg p_2$										
$p_5 \vee p_2$										
$\neg p_1 \vee \neg p_4$										
$p_5 \vee p_2$										
$\neg p_1 \vee \neg p_5$										

Number of models: 0

This set of 13 clauses is unsatisfiable.

Example (obtained by a program) for $n = 5$ and $k = 2$

	p_1	p_2	p_3	p_4	p_5	p_1	p_2	p_3	p_4	p_5
$\neg p_2 \lor \neg p_3$										
$\neg p_2 \lor p_1$										
$\neg p_2 \lor p_2$										
$p_1 \lor p_1$										
$\neg p_5 \lor p_5$										
$p_4 \lor p_5$										
$\neg p_5 \lor \neg p_3$										
$p_2 \lor \neg p_4$										
$p_5 \lor \neg p_2$										
$p_5 \lor p_2$										
$\neg p_1 \lor \neg p_4$										
$p_5 \lor p_2$										
$\neg p_1 \lor \neg p_5$										

Number of models: 0
This set of 13 clauses is unsatisfiable.

Increasing number of generated cluases we can observe transition from satisfiable to unsatisfiable.

Random Clause Generation

We are interested in the probability π that a set of 3-clauses is unsatisfiable.

Random Clause Generation

We are interested in the probability π that a set of 3-clauses is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number m of the clauses.
- Randomly generate m clauses with an equal probability.
- $\pi \sim \frac{\#_{unsat}}{\#_{all}}$

Random Clause Generation

We are interested in the probability π that a set of 3-clauses is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number m of the clauses.
- Randomly generate m clauses with an equal probability.
- $\pi \sim \frac{\#_{unsat}}{\#_{all}}$

Random Clause Generation

We are interested in the probability π that a set of 3-clauses is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number m of the clauses.
- Randomly generate m clauses with an equal probability.
- $\pi \sim \frac{\#_{unsat}}{\#_{all}}$

Random Clause Generation

We are interested in the probability π that a set of 3-clauses is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number m of the clauses.
- Randomly generate m clauses with an equal probability.

- $\pi \sim \frac{\#_{unsat}}{\#_{all}}$

Random Clause Generation

We are interested in the probability π that a set of 3-clauses is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number m of the clauses.
- Randomly generate m clauses with an equal probability.
- $\pi \sim \frac{\#_{unsat}}{\#_{all}}$

Important parameter: ratio of clauses per variable $r = m/n$.
We will investigate dependence of π with respect to the ratio r and the number of varibales n.

Random Clause Generation

We are interested in the probability π that a set of 3-clauses is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number m of the clauses.
- Randomly generate m clauses with an equal probability.
- $\pi \sim \frac{\#_{unsat}}{\#_{all}}$

Important parameter: ratio of clauses per variable $r = m/n$.
We will investigate dependence of π with respect to the ratio r and the number of varibales n.

Note that the robability $\pi(r, n)$ is a monotone function: the more clauses we generate, the higher chance we have that the set is unsatisfiable.

Roulette

We generate random instances of 3-SAT with 10-variables.

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

Roulette

We generate random instances of 3-SAT with 10-variables.

You will bet on whether the resuting set of clauses is satisfiable or not.

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

SAT Roulette

We generate random instances of 3-SAT with 10-variables.

You will bet on whether the resuting set of clauses is satisfiable or not.
What will you bet on if we generate

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

SAT Roulette

We generate random instances of 3-SAT with 10-variables.

You will bet on whether the resuting set of clauses is satisfiable or not.
What will you bet on if we generate

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

SAT Roulette

We generate random instances of 3-SAT with 10-variables.

You will bet on whether the resuting set of clauses is satisfiable or not.
What will you bet on if we generate

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

SAT Roulette

We generate random instances of 3-SAT with 10-variables.

You will bet on whether the resuting set of clauses is satisfiable or not.
What will you bet on if we generate

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

SAT Roulette

We generate random instances of 3-SAT with 10-variables.

You will bet on whether the resuting set of clauses is satisfiable or not.
What will you bet on if we generate

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

SAT Roulette

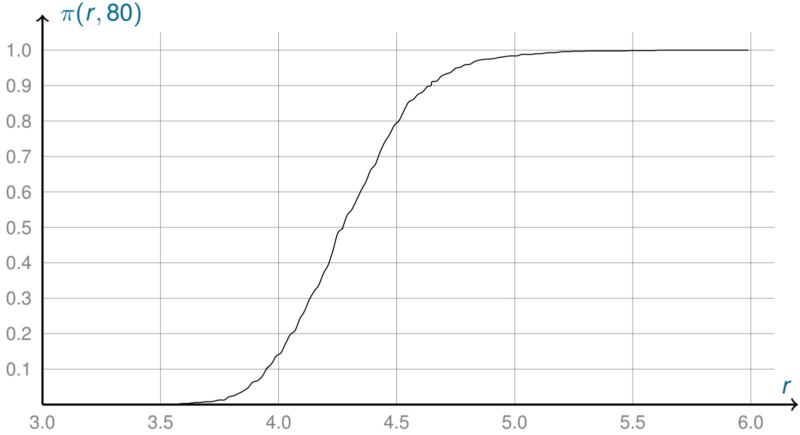
We generate random instances of 3-SAT with 10-variables.

You will bet on whether the resuting set of clauses is satisfiable or not.
What will you bet on if we generate

- 5 clauses?
- 30 clauses?
- 60 clauses?
- 100 clauses?
- 1000 clauses?

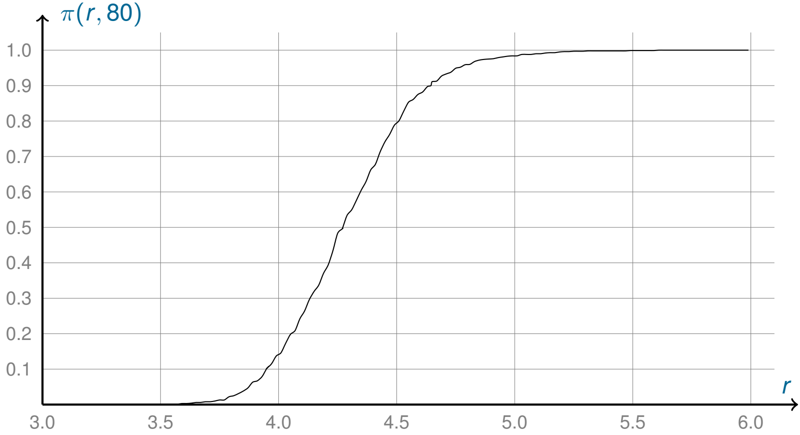
What would be your betting ratio?

Probability of obtaining an unsatisfiable set



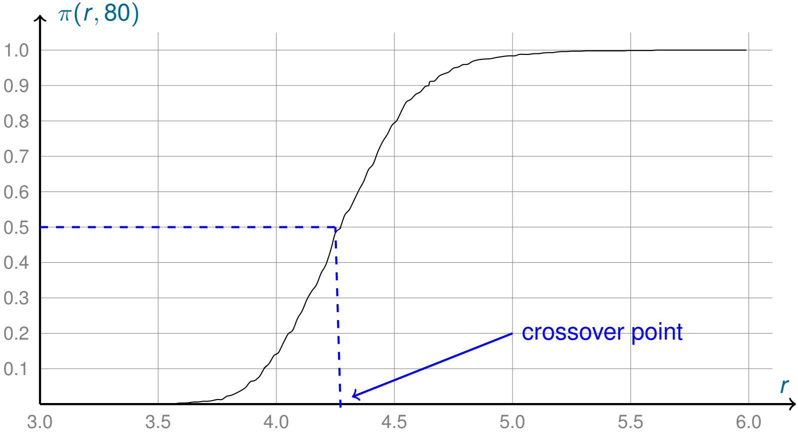
Probability of obtaining an unsatisfiable set

Crossover point: the value of r at which the probability crosses 0.5.



Probability of obtaining an unsatisfiable set

Crossover point: the value of r at which the probability crosses 0.5.



Experimentally: for large n crossover point is close to 4.25.

ϵ-window

Take a (small) number $\epsilon > 0$. ϵ-window is the interval of values of r where the probability is between ϵ and $1 - \epsilon$.

For example, take $\epsilon = 0.1$.

ϵ-window

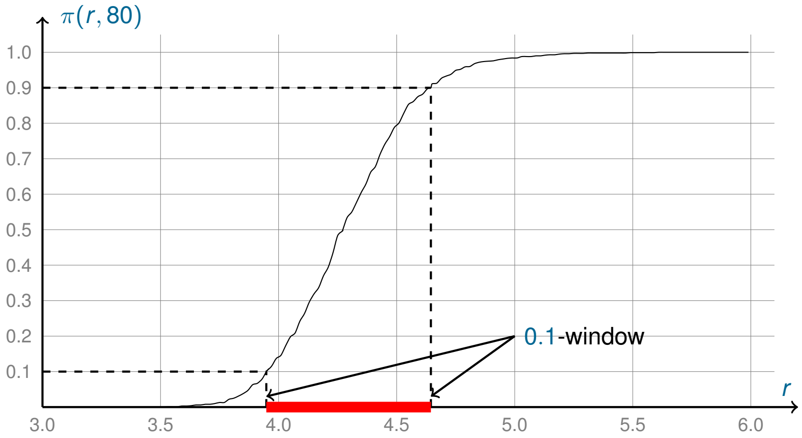
Take a (small) number $\epsilon > 0$. ϵ-window is the interval of values of r where the probability is between ϵ and $1 - \epsilon$.

For example, take $\epsilon = 0.1$.

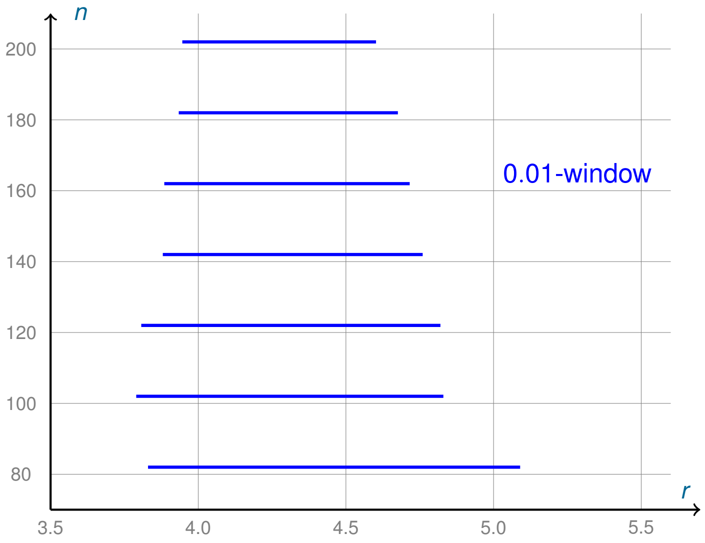
ϵ-window

Take a (small) number $\epsilon > 0$. ϵ-window is the interval of values of r where the probability is between ϵ and $1 - \epsilon$.

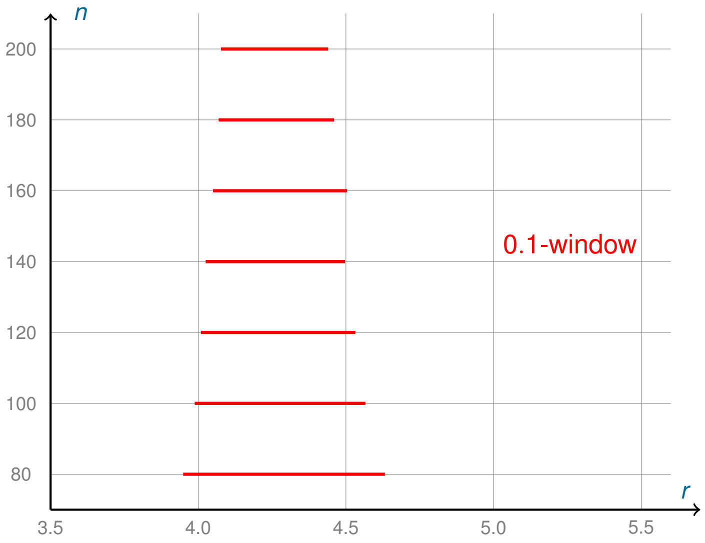
For example, take $\epsilon = 0.1$.



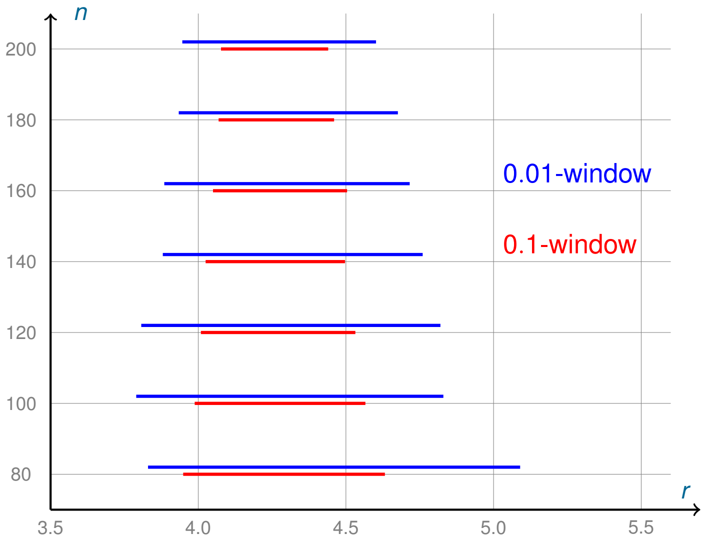
Scaling Window Effect



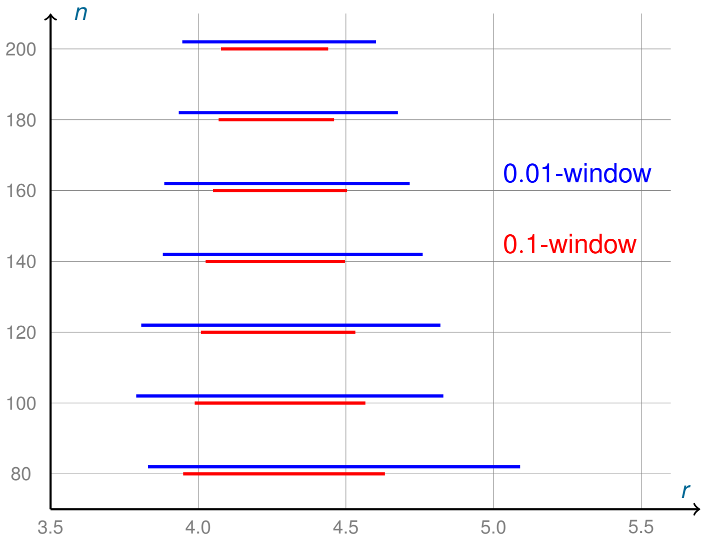
Scaling Window Effect



Scaling Window Effect

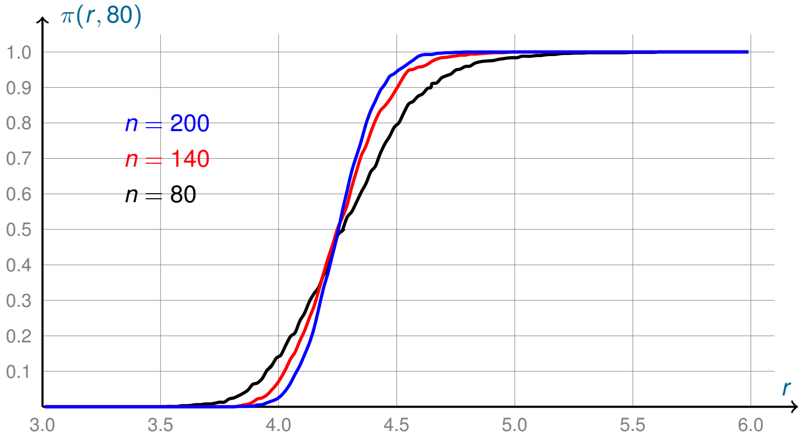


Scaling Window Effect



Conjecture: for $n \to \infty$ every ϵ-window "degenerates into a point".

Sharp Phase Transition



Easy-Hard-Easy Pattern (for SAT solvers)