Automated Reasoning and Program Verification

Laura Kovács
TU Vienna

Outline

Satisfiability and Randomisation

- SAT solving of randomly generated set of clauses are very hard for DPLL-based SAT solvers;
- Some small randomly generated set of clauses cannot be satisfied by DPLL-based SAT solvers, whereas random SAT algorithms can satisfy them;
- Small randomly generated set of clauses are useful for debugging SAT solvers;
- Experiments show that randomly generated set of clauses become from satisfiable to unsatisfiable in a very narrow region.

Random Clause Generation

How can one generate a random clause?

Random Clause Generation

How can one generate a random clause?
Let's first generate a random literal.

Random Clause Generation

How can one generate a random clause?
Let's first generate a random literal.

- Fix a number n of boolean variables;

Random Clause Generation

How can one generate a random clause?
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.

Random Clause Generation

How can one generate a random clause?
Let's first generate a random literal.
A random clause is a collection of random literals.

- 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.

Random Clause Generation

How can one generate a random clause?
Let's first generate a random literal.
A random clause is a collection of random literals.

- 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.
- Fix the length k of the clause;

Random Clause Generation

How can one generate a random clause?
Let's first generate a random literal.
A random clause is a collection of random literals.

- 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.
- Fix the length k of the clause;

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

SAT and k-SAT

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

k-SAT is the problem of satisfiability checking for sets of clauses of length k (k-clauses).

SAT and k-SAT

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

k-SAT is the problem of satisfiability checking for sets of clauses of length k (k-clauses).

- SAT is NP-complete;

SAT and k-SAT

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

k-SAT is the problem of satisfiability checking for sets of clauses of length k (k-clauses).

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

SAT and k-SAT

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

k-SAT is the problem of satisfiability checking for sets of clauses of length k (k-clauses).

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

SAT and k-SAT

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

k-SAT is the problem of satisfiability checking for sets of clauses of length k (k-clauses).

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

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:

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

where n is a new variable.

Randomly Generated Sets of k-Clauses

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 $k = 2$

$\neg p_2 \vee \neg p_3$

$\neg p_2 \vee p_1$

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
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

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	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 $k = 2$

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

p_1	p_2	p_3	p_4	p_5

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 $k = 2$

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

p_1	p_2	p_3	p_4	p_5

p_1	p_2	p_3	p_4	p_5
1	0	0	0	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 $k = 2$

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

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	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 $k = 2$

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

p_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 $k = 2$

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_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 $k = 2$

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

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

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 $k = 2$

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

p_1	p_2	p_3	p_4	p_5

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 $k = 2$

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

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

Number of models: 1

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

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

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

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.

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number k of literals per clause, so we will generate k-SAT instances;

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number k of literals per clause, so we will generate k-SAT instances;
- Number m of the clauses.

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number k of literals per clause, so we will generate k-SAT instances;
- Number m of the clauses.

Generate m clauses, each one has k literals randomly generated among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number k of literals per clause, so we will generate k-SAT instances;
- Number m of the clauses.

Generate m clauses, each one has k literals randomly generated among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.

Exercise: What is the probability that the resulting set is unsatisfiable for $m = 1$ and $m = 2$?

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number k of literals per clause, so we will generate k-SAT instances;
- Number m of the clauses.

Generate m clauses, each one has k literals randomly generated among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.

Note that the probability is a monotone function: the more clauses we generate, the higher chance we have that the set is unsatisfiable.

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number k of literals per clause, so we will generate k-SAT instances;
- Real number r: ratio of clauses per variable.

Generate $[rn]$ clauses, each one has k literals randomly generated among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.

Note that the probability is a monotone function: the more clauses we generate, the higher chance we have that the set is unsatisfiable.

Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- Number n of boolean variables;
- Number k of literals per clause, so we will generate k-SAT instances;
- Real number r: ratio of clauses per variable.

Generate $[rn]$ clauses, each one has k literals randomly generated among $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$ with an equal probability.

Note that the probability is a monotone function: the more clauses we generate, the higher chance we have that the set is unsatisfiable.

Denote by $\pi(r, n)$ the probability that a randomly generated set of $[rn]$ k-clauses is unsatisfiable.

Roulette

SAT Roulette

We will generate random instances of 2-SAT with 5-variables.

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

SAT Roulette

We will generate random instances of 2-SAT with 5-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?

SAT Roulette

We will generate random instances of 2-SAT with 5-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?
- What will you bet on if we generate 100 clauses?

SAT Roulette

We will generate random instances of 2-SAT with 5-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?
- What will you bet on if we generate 100 clauses?
- What will you bet on if we generate 15 clauses?

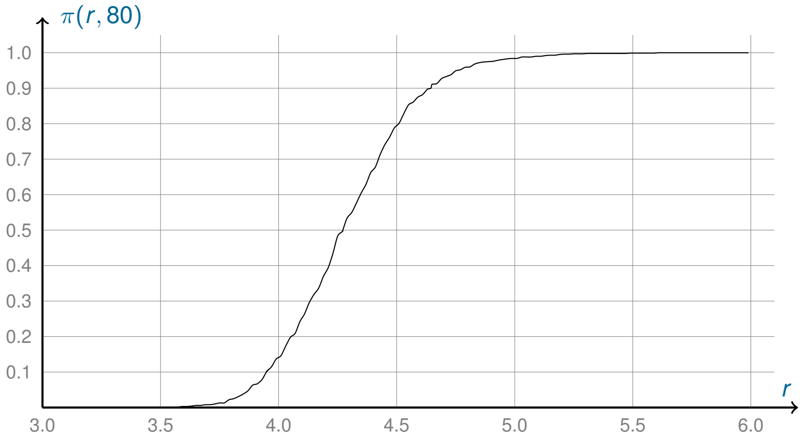
SAT Roulette

We will generate random instances of 3-SAT with 5-variables.

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

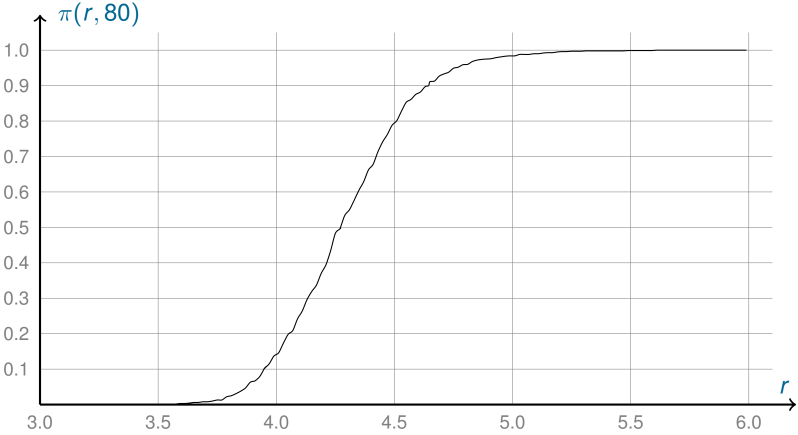
- What will you bet on if we generate 5 clauses?
- What will you bet on if we generate 100 clauses?
- What will you bet on if we generate 15 clauses?

Probability of obtaining an unsatisfiable set of 3-SAT



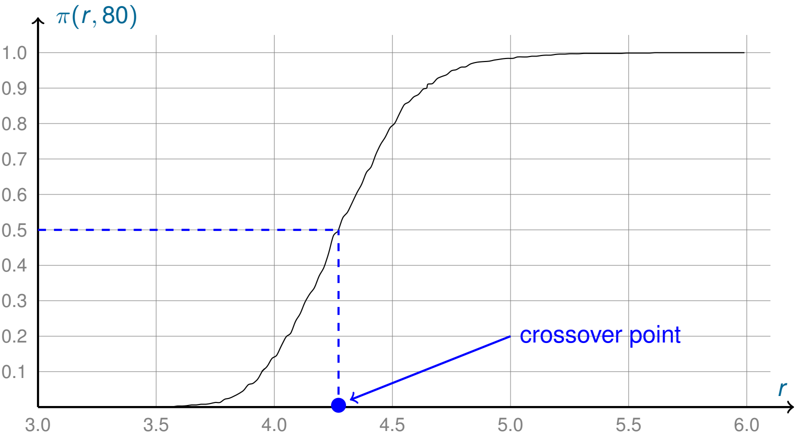
Probability of obtaining an unsatisfiable set of 3-SAT

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



Probability of obtaining an unsatisfiable set of 3-SAT

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



ϵ-window

Take a (small) number $0 < \epsilon < 0.5$. ϵ-window is the interval of values of r where the probability is between ϵ and $1 - \epsilon$.

ϵ-window

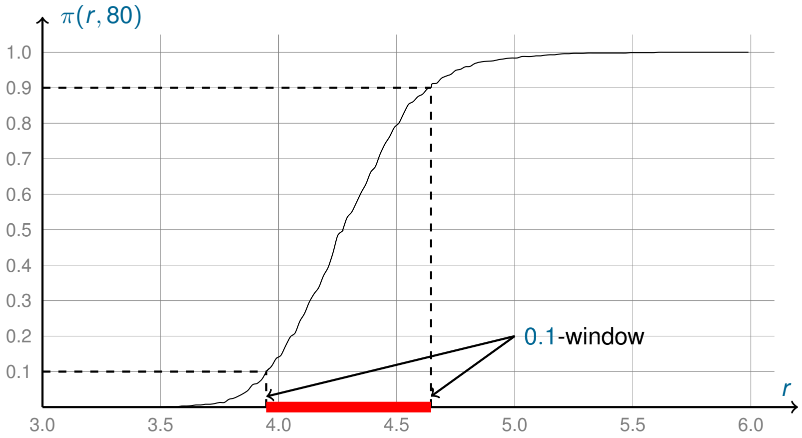
Take a (small) number $0 < \epsilon < 0.5$. ϵ-window is the interval of values of r where the probability is between ϵ and $1 - \epsilon$.

For example, take $\epsilon = 0.1$.

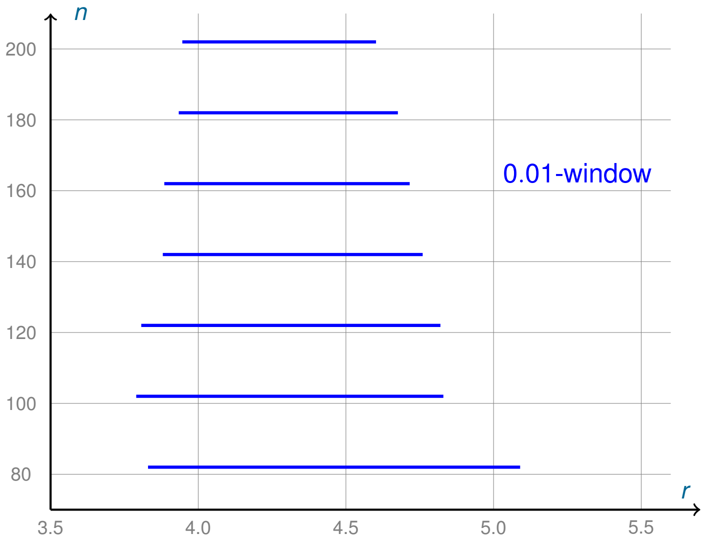
ϵ-window

Take a (small) number $0 < \epsilon < 0.5$. ϵ-window is the interval of values of r where the probability is between ϵ and $1 - \epsilon$.

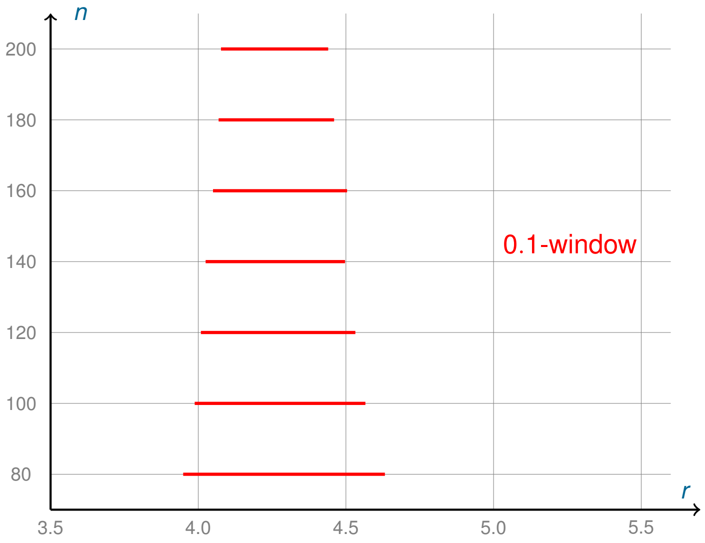
For example, take $\epsilon = 0.1$.



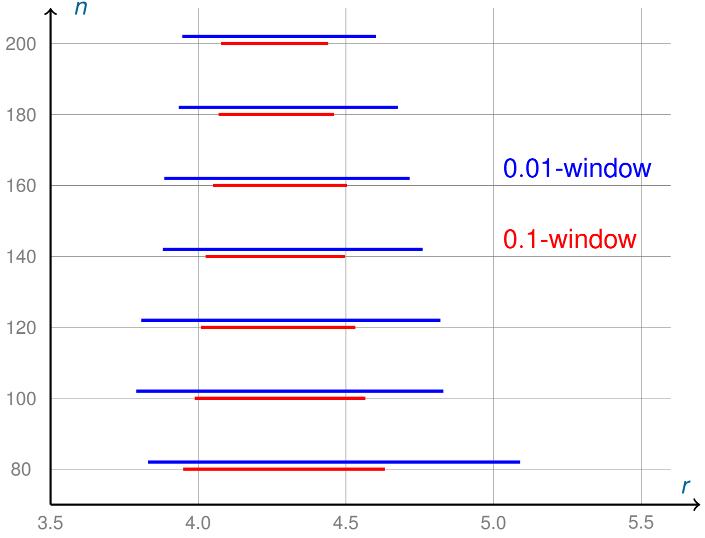
Scaling Window Effect for 3-SAT



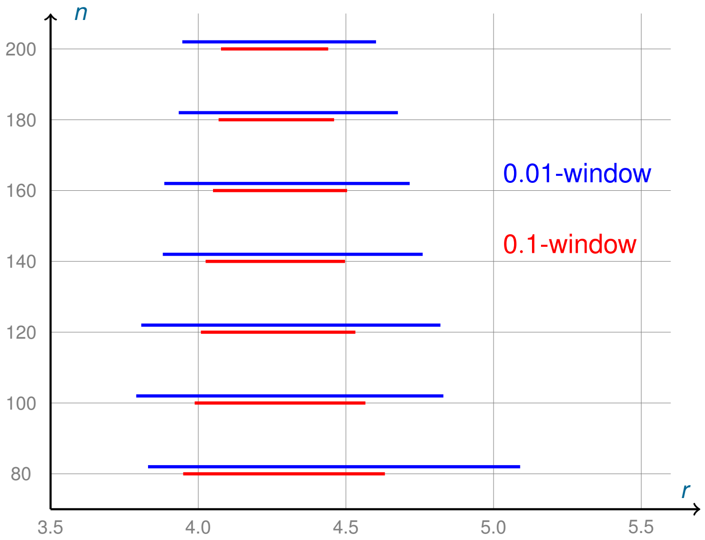
Scaling Window Effect for 3-SAT



Scaling Window Effect for 3-SAT

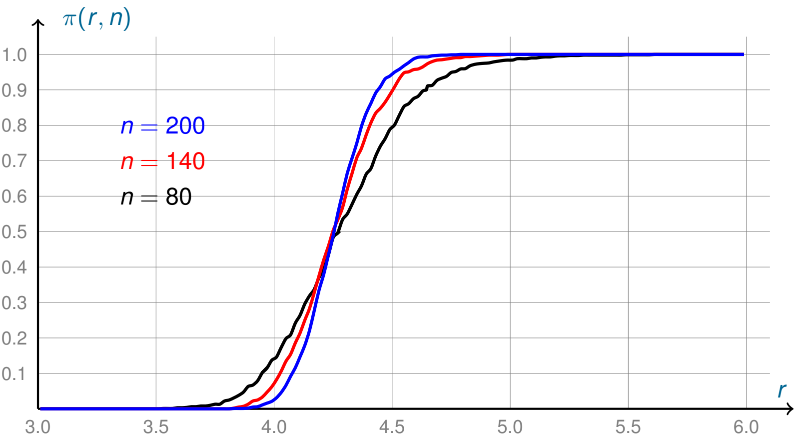


Scaling Window Effect for 3-SAT

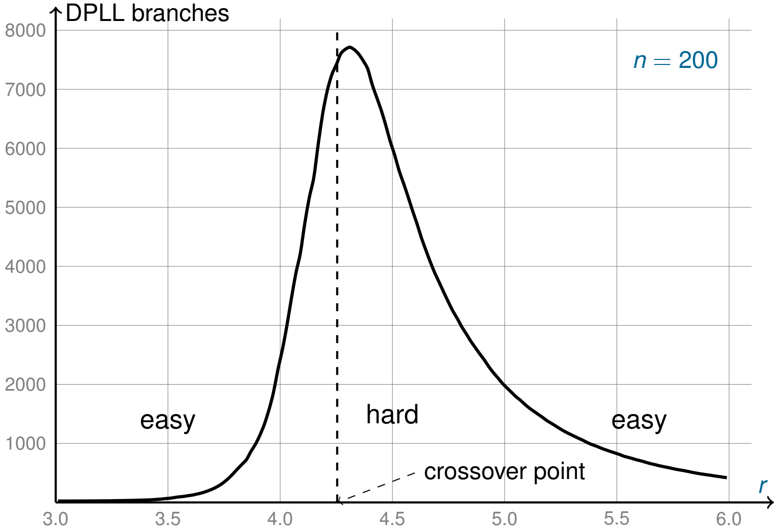


Conjecture: for $n \to \infty$ every ϵ-window "degenerates into a point".

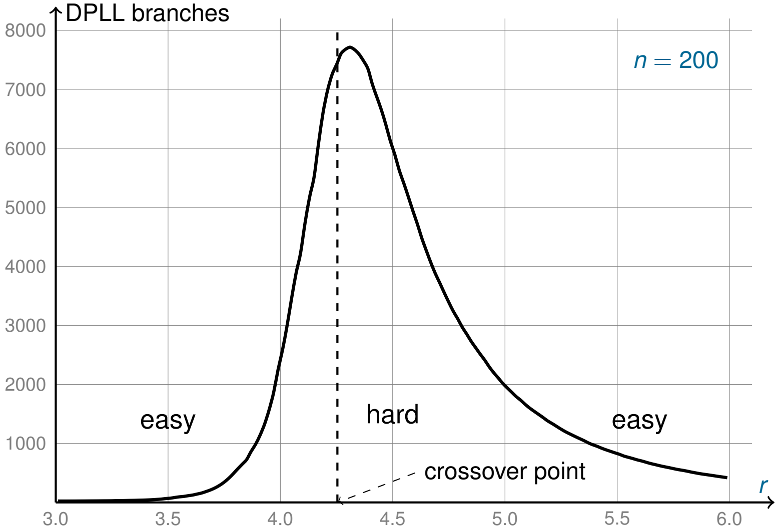
Sharp Phase Transition of $\pi(r, n)$



Easy-Hard-Easy Pattern



Easy-Hard-Easy Pattern



Experiments show that the crossover point for 3-SAT is 4.25.

Satisfiability Algorithm that Cannot Establish Unsatisfiability

procedure *CHAOS*(S)
input: set of clauses S
output: interpretation I such that $I \models S$ or *don't know*

Satisfiability Algorithm that Cannot Establish Unsatisfiability

procedure *CHAOS*(S)
input: set of clauses S
output: interpretation I such that $I \models S$ or *don't know*
parameters: positive integer *MAX-TRIES*
begin
  repeat *MAX-TRIES* times

end

Satisfiability Algorithm that Cannot Establish Unsatisfiability

```
procedure CHAOS(S)
input: set of clauses S
output: interpretation I such that I ⊨ S or don't know
parameters: positive integer MAX-TRIES
begin
  repeat MAX-TRIES times
    I := random interpretation
    if I ⊨ S then return I
  return don't know
end
```

SAT as a Decision Problem

Decision problem: any problem in any infinite domain, that has a yes-no answer. Each element of this domain is called an instance of this problem.

SAT as a Decision Problem

Decision problem: any problem in any infinite domain, that has a yes-no answer. Each element of this domain is called an instance of this problem.

Example: solvability of systems of linear inequalities over integers.

- an instance is a system of linear integer inequalities;
- an answer is yes if it the instance has a solution.

SAT as a Decision Problem

Decision problem: any problem in any infinite domain, that has a yes-no answer. Each element of this domain is called an instance of this problem.

Example: solvability of systems of linear inequalities over integers.

- an instance is a system of linear integer inequalities;
- an answer is yes if it the instance has a solution.

SAT is a decision problem:

- an instance is a finite set of clauses.
- it has a yes-no answer: yes (satisfiable) or no (unsatisfiable).

SAT as a Decision Problem

Decision problem: any problem in any infinite domain, that has a yes-no answer. Each element of this domain is called an instance of this problem.

Example: solvability of systems of linear inequalities over integers.

- an instance is a system of linear integer inequalities;
- an answer is yes if it the instance has a solution.

SAT is a decision problem:

- an instance is a finite set of clauses.
- it has a yes-no answer: yes (satisfiable) or no (unsatisfiable).

Witness for an instance $\mathcal{I}$: any data D such that, given D, one can check in polymonial time (in the size of D) that D is a yes-answer of $\mathcal{I}$.

SAT as a Decision Problem

Decision problem: any problem in any infinite domain, that has a yes-no answer. Each element of this domain is called an instance of this problem.

Example: solvability of systems of linear inequalities over integers.

- an instance is a system of linear integer inequalities;
- an answer is yes if it the instance has a solution.

SAT is a decision problem:

- an instance is a finite set of clauses.
- it has a yes-no answer: yes (satisfiable) or no (unsatisfiable).

Witness for an instance $\mathcal{I}$: any data D such that, given D, one can check in polymonial time (in the size of D) that D is a yes-answer of $\mathcal{I}$.

Satisfiability has short witnessess: interpretations.

SAT as a Decision Problem

Decision problem: any problem in any infinite domain, that has a yes-no answer. Each element of this domain is called an instance of this problem.

Example: solvability of systems of linear inequalities over integers.

- an instance is a system of linear integer inequalities;
- an answer is yes if it the instance has a solution.

SAT is a decision problem:

- an instance is a finite set of clauses.
- it has a yes-no answer: yes (satisfiable) or no (unsatisfiable).

Witness for an instance $\mathcal{I}$: any data D such that, given D, one can check in polymonial time (in the size of D) that D is a yes-answer of $\mathcal{I}$.

Satisfiability has short witnessess: interpretations.

Unsatisfiability has no polynomial-size witnessess, unless $NP = co - NP$.