# Fixed-parameter algorithms for satisfiability testing

Igor Razgon

# Content

# The SAT problem

- **Literal:** A boolean variable $x$ or its negation $\overline{x}$.

# The SAT problem

- **Literal:** A boolean variable $x$ or its negation $\overline{x}$.
- **Clause:** A disjunction of literals, e.g. $(x_1 \vee \overline{x_2} \vee x_3)$

# The SAT problem

- **Literal:** A boolean variable $x$ or its negation $\overline{x}$.
- **Clause:** A disjunction of literals, e.g. $(x_1 \vee \overline{x_2} \vee x_3)$
- **Conjuctive Normal Form (CNF):** A conjunction of clauses, e.g. $(x_1 \vee \overline{x_2} \vee x_3)(\overline{x}_1 \vee x_4)$

# The SAT problem

- **Literal:** A boolean variable $x$ or its negation $\overline{x}$.
- **Clause:** A disjunction of literals, e.g. $(x_1 \vee \overline{x_2} \vee x_3)$
- **Conjuctive Normal Form (CNF):** A conjunction of clauses, e.g. $(x_1 \vee \overline{x_2} \vee x_3)(\overline{x}_1 \vee x_4)$
- **A satisfiable CNF:** It is possible to assign variables with *true*/*false* so that the whole CNF becomes *true*

# The SAT problem

- **Literal:** A boolean variable $x$ or its negation $\overline{x}$.
- **Clause:** A disjunction of literals, e.g. $(x_1 \vee \overline{x_2} \vee x_3)$
- **Conjuctive Normal Form (CNF):** A conjunction of clauses, e.g. $(x_1 \vee \overline{x_2} \vee x_3)(\overline{x}_1 \vee x_4)$
- **A satisfiable CNF:** It is possible to assign variables with *true*/*false* so that the whole CNF becomes *true*
- **SAT problem:** Given a CNF $F$ is it satisfiable?

# The SAT problem

- **Literal:** A boolean variable $x$ or its negation $\overline{x}$.
- **Clause:** A disjunction of literals, e.g. $(x_1 \vee \overline{x_2} \vee x_3)$
- **Conjuctive Normal Form (CNF):** A conjunction of clauses, e.g. $(x_1 \vee \overline{x_2} \vee x_3)(\overline{x}_1 \vee x_4)$
- **A satisfiable CNF:** It is possible to assign variables with *true*/*false* so that the whole CNF becomes *true*
- **SAT problem:** Given a CNF $F$ is it satisfiable?
- The SAT problem is NP complete even over CNF's whose clauses are of length at most 3.

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*
  - Remove all the clauses containing $x$; remove $\overline{x}$ from all the clauses

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*
  - Remove all the clauses containing $x$; remove $\overline{x}$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (x_4)$

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*
  - Remove all the clauses containing $x$; remove $\overline{x}$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (x_4)$
- Residual formula $F|x \leftarrow false$

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*
  - Remove all the clauses containing $x$; remove $\overline{x}$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (x_4)$
- Residual formula $F|x \leftarrow false$
  - Obtained from a CNF $F$ by assigning $x$ to *false*

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*
  - Remove all the clauses containing $x$; remove $\overline{x}$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (x_4)$
- Residual formula $F|x \leftarrow false$
  - Obtained from a CNF $F$ by assigning $x$ to *false*
  - Remove all the clauses containing $\overline{x}$; remove $x$ from all the clauses

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*
  - Remove all the clauses containing $x$; remove $\overline{x}$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (x_4)$
- Residual formula $F|x \leftarrow false$
  - Obtained from a CNF $F$ by assigning $x$ to *false*
  - Remove all the clauses containing $\overline{x}$; remove $x$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (\overline{x_2} \vee x_4)$

# Additional terminology

- A CNF $F$ is *trivially satisfiable* if it has no clauses
- A CNF $F$ is *trivially unsatisfiable* if it contains an empty clause
- Residual formula $F|x \leftarrow true$
  - Obtained from a CNF $F$ by assigning $x$ to *true*
  - Remove all the clauses containing $x$; remove $\overline{x}$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (x_4)$
- Residual formula $F|x \leftarrow false$
  - Obtained from a CNF $F$ by assigning $x$ to *false*
  - Remove all the clauses containing $\overline{x}$; remove $x$ from all the clauses
  - If $F = (x_1 \vee \overline{x_2} \vee x_3) \vee (\overline{x}_1 \vee x_4)$ then $F|x_1 \leftarrow true = (\overline{x_2} \vee x_4)$
- Observation: $F$ is satisfiable if and only if either $F|x \leftarrow true$ or $F|x \leftarrow false$ is satisfiable.

# Backtracking algorithm

- The algorithm (the input is a CNF $F$):

# Backtracking algorithm

- The algorithm (the input is a CNF $F$):
  - If $F$ is trivially satisfiable, return 'YES'

# Backtracking algorithm

- The algorithm (the input is a CNF $F$):
  - If $F$ is trivially satisfiable, return 'YES'
  - If $F$ is trivially non-satisfiable, return 'NO'

# Backtracking algorithm

- The algorithm (the input is a CNF $F$):
  - If $F$ is trivially satisfiable, return 'YES'
  - If $F$ is trivially non-satisfiable, return 'NO'
  - Choose a variable $x$

# Backtracking algorithm

- The algorithm (the input is a CNF $F$):
  - If $F$ is trivially satisfiable, return 'YES'
  - If $F$ is trivially non-satisfiable, return 'NO'
  - Choose a variable $x$
  - Recursively apply to $F|x \leftarrow \mathit{true}$ and $F|x \leftarrow \mathit{false}$

# Backtracking algorithm

- The algorithm (the input is a CNF $F$):
  - If $F$ is trivially satisfiable, return 'YES'
  - If $F$ is trivially non-satisfiable, return 'NO'
  - Choose a variable $x$
  - Recursively apply to $F|x \leftarrow \mathit{true}$ and $F|x \leftarrow \mathit{false}$
  - If at least one of recursive applications returns 'YES' then return 'YES'; otherwise, return 'NO'.

# Complexity of SAT solving

- The runtime of backtracking is $O^*(2^n)$

# Complexity of SAT solving

- The runtime of backtracking is $O^*(2^n)$ (the star means that the polynomial factor is suppressed).

# Complexity of SAT solving

- The runtime of backtracking is $O^*(2^n)$ (the star means that the polynomial factor is suppressed).
- This is the best existing time complexity *if we do not make any apriori assumptions regarding the input*.

# Complexity of SAT solving

- The runtime of backtracking is $O^*(2^n)$ (the star means that the polynomial factor is suppressed).
- This is the best existing time complexity *if we do not make any apriori assumptions regarding the input*.
- Great open problem: can we solve the unrestricted SAT in time $O^*(c^n)$ for some $c < 2$. Many people believe it is impossible.
- Conclusion: efficient SAT solving requires making assumptions regarding the input.

# Small parameter assumption

- Besides the input size, the user normally knows a lot of additional measures (*parameters*) of the considered problem such as:

# Small parameter assumption

- Besides the input size, the user normally knows a lot of additional measures (*parameters*) of the considered problem such as:
  - Maximum allowed size of the output.

# Small parameter assumption

- Besides the input size, the user normally knows a lot of additional measures (*parameters*) of the considered problem such as:
  - Maximum allowed size of the output.
  - Structural parameters e.g. treewidth of the underlying graph.

# Small parameter assumption

- Besides the input size, the user normally knows a lot of additional measures (*parameters*) of the considered problem such as:
  - Maximum allowed size of the output.
  - Structural parameters e.g. treewidth of the underlying graph.
- Assume that some parameter $k$ is *very small* compared to the input size.

# Small parameter assumption

- Besides the input size, the user normally knows a lot of additional measures (*parameters*) of the considered problem such as:
  - Maximum allowed size of the output.
  - Structural parameters e.g. treewidth of the underlying graph.
- Assume that some parameter $k$ is *very small* compared to the input size.
- Under this small parameter assumption, we can do much better than the prute-force.

# Clique or Independent Set Problem

- Given a graph, return $max(t_1, t_2)$ where:

# Clique or Independent Set Problem

- Given a graph, return $max(t_1, t_2)$ where:
  - $t_1$ is the size of the maximum independent set
  - $t_2$ is the size of the maximum clique
- This problem is NP-hard:

# Clique or Independent Set Problem

- Given a graph, return $max(t_1, t_2)$ where:
  - $t_1$ is the size of the maximum independent set
  - $t_2$ is the size of the maximum clique
- This problem is NP-hard:
  - Consider a planar graph.
  - The maximum size of a clique is at most 4.
  - The problem is effectively equivalent to an NP-hard problem of computing the maximum independent set of a planar graph.

# Parameterized Clique or Independent Set Problem

- Problem specification:

# Parameterized Clique or Independent Set Problem

- Problem specification:
  - **Input:** graph $G$
  - **Parameter:** $k$
  - **Question** does $G$ have an independent set or clique of size at least $k$?

# Parameterized Clique or Independent Set Problem

- Problem specification:
  - **Input:** graph $G$
  - **Parameter:** $k$
  - **Question** does $G$ have an independent set or clique of size at least $k$?
- **Ramsey theorem**: there is number $R(k)$ (roughly equal $2^{k/2}$) such that any graph with at least $R(k)$ vertices has either an independent set of size at least $k$ or a clique of size at least $k$.

# Parameterized Clique or Independent Set Problem

- Problem specification:
  - **Input:** graph $G$
  - **Parameter:** $k$
  - **Question** does $G$ have an independent set or clique of size at least $k$?
- **Ramsey theorem**: there is number $R(k)$ (roughly equal $2^{k/2}$) such that any graph with at least $R(k)$ vertices has either an independent set of size at least $k$ or a clique of size at least $k$.
- **Algorithm:** If the number of vertices of $G$ is $R(k)$ or larger, return 'YES', otherwise perform brute-force search.

# Parameterized Clique or Independent Set Problem

- Problem specification:
  - **Input:** graph $G$
  - **Parameter:** $k$
  - **Question** does $G$ have an independent set or clique of size at least $k$?
- **Ramsey theorem**: there is number $R(k)$ (roughly equal $2^{k/2}$) such that any graph with at least $R(k)$ vertices has either an independent set of size at least $k$ or a clique of size at least $k$.
- **Algorithm:** If the number of vertices of $G$ is $R(k)$ or larger, return 'YES', otherwise perform brute-force search.
- The complexity of this algorithm is $O(2^{R(k)})$, it does not depend on $n$ at all!

# Fixed-parameter computation

- Given a computational problem with input size $n$ and parameter $k$.

# Fixed-parameter computation

- Given a computational problem with input size $n$ and parameter $k$.
- A *fixed-parameter algorithm* solves this problem in time $O(f(k) * n^c)$ where $c$ is a constant (usually $c \leq 3$).

# Fixed-parameter computation

- Given a computational problem with input size $n$ and parameter $k$.
- A *fixed-parameter algorithm* solves this problem in time $O(f(k) * n^c)$ where $c$ is a constant (usually $c \leq 3$).
- Prolbems that can be solved this way are fixed-parameter tractable (FPT).

# Fixed-parameter computation

- Given a computational problem with input size $n$ and parameter $k$.
- A *fixed-parameter algorithm* solves this problem in time $O(f(k) * n^c)$ where $c$ is a constant (usually $c \leq 3$).
- Prolbems that can be solved this way are fixed-parameter tractable (FPT).
- The area that studies fixed-parameter tractability phenomena is called *parameterized complexity*.

# Fixed-parameter computation

- Given a computational problem with input size $n$ and parameter $k$.
- A *fixed-parameter algorithm* solves this problem in time $O(f(k) * n^c)$ where $c$ is a constant (usually $c \leq 3$).
- Prolbems that can be solved this way are fixed-parameter tractable (FPT).
- The area that studies fixed-parameter tractability phenomena is called *parameterized complexity*.
- We will see how the methodology is applied in the area of SAT solving.
- The considered parameters will measure 'closeness' of the given instance to a polynomially solvable class.

# Backdoor set w.r.t. to tjhe given subclass of SAT

- There are quite a few polynomially solvable classes of SAT (e.g. 2SAT, Horn, Renamable Horn, etc.).

# Backdoor set w.r.t. to tjhe given subclass of SAT

- There are quite a few polynomially solvable classes of SAT (e.g. 2SAT, Horn, Renamable Horn, etc.).
- A **backdoor set** of a CNF $F$ w.r.t. to the given polynomially solvable class $A$: a subset $S$ of the variables of $F$ such that any assignment of $S$ produces a residual formula that belongs to $A$

# Backdoor set w.r.t. to tjhe given subclass of SAT

- There are quite a few polynomially solvable classes of SAT (e.g. 2SAT, Horn, Renamable Horn, etc.).
- A **backdoor set** of a CNF $F$ w.r.t. to the given polynomially solvable class $A$: a subset $S$ of the variables of $F$ such that any assignment of $S$ produces a residual formula that belongs to $A$
- Correspondng parameterized problem: given a CNF $F$ is there a backdoor set of size at most $k$ w.r.t. to the given class $A$?

# Backdoor set w.r.t. to tjhe given subclass of SAT

- There are quite a few polynomially solvable classes of SAT (e.g. 2SAT, Horn, Renamable Horn, etc.).
- A **backdoor set** of a CNF $F$ w.r.t. to the given polynomially solvable class $A$: a subset $S$ of the variables of $F$ such that any assignment of $S$ produces a residual formula that belongs to $A$
- Correspondng parameterized problem: given a CNF $F$ is there a backdoor set of size at most $k$ w.r.t. to the given class $A$?
- Motivation to consider such problems:

# Backdoor set w.r.t. to tjhe given subclass of SAT

- There are quite a few polynomially solvable classes of SAT (e.g. 2SAT, Horn, Renamable Horn, etc.).
- A **backdoor set** of a CNF $F$ w.r.t. to the given polynomially solvable class $A$: a subset $S$ of the variables of $F$ such that any assignment of $S$ produces a residual formula that belongs to $A$
- Correspondng parameterized problem: given a CNF $F$ is there a backdoor set of size at most $k$ w.r.t. to the given class $A$?
- Motivation to consider such problems:
  - Assume that some real-world class of instances has a small backdoor w.r.t. to some class.

# Backdoor set w.r.t. to tjhe given subclass of SAT

- There are quite a few polynomially solvable classes of SAT (e.g. 2SAT, Horn, Renamable Horn, etc.).
- A **backdoor set** of a CNF $F$ w.r.t. to the given polynomially solvable class $A$: a subset $S$ of the variables of $F$ such that any assignment of $S$ produces a residual formula that belongs to $A$
- Correspondng parameterized problem: given a CNF $F$ is there a backdoor set of size at most $k$ w.r.t. to the given class $A$?
- Motivation to consider such problems:
  - Assume that some real-world class of instances has a small backdoor w.r.t. to some class.
  - We apply the fixed-parameter algorithm at the preprocessing to find such backdoor set.

# Backdoor set w.r.t. to tjhe given subclass of SAT

- There are quite a few polynomially solvable classes of SAT (e.g. 2SAT, Horn, Renamable Horn, etc.).
- A **backdoor set** of a CNF $F$ w.r.t. to the given polynomially solvable class $A$: a subset $S$ of the variables of $F$ such that any assignment of $S$ produces a residual formula that belongs to $A$
- Correspondng parameterized problem: given a CNF $F$ is there a backdoor set of size at most $k$ w.r.t. to the given class $A$?
- Motivation to consider such problems:
  - Assume that some real-world class of instances has a small backdoor w.r.t. to some class.
  - We apply the fixed-parameter algorithm at the preprocessing to find such backdoor set.
  - Then solve the instance, branching only on the variables of this set.

# State of the art

- Fixed-parameter tractability of computing backdoors is now well understood for most polytime solvable classes.
- See the recent review "Backdoors for Satisfaction" of Gaspers and Szeider (available on arxiv).
- I will concentrate on one result related to my own research.

# Renamable Horn formulas

- Horn formulas: each clause contains at most one positive literal.
  Example: $(X_1 \vee \overline{X_2} \vee \overline{X_3})(X_2 \vee \overline{X_1} \vee \overline{X_4})(X_3 \vee \overline{X_2} \vee \overline{X_4})$

# Renamable Horn formulas

- Horn formulas: each clause contains at most one positive literal.
  Example: $(X_1 \vee \overline{X_2} \vee \overline{X_3})(X_2 \vee \overline{X_1} \vee \overline{X_4})(X_3 \vee \overline{X_2} \vee \overline{X_4})$
- Renamable Horn (RHorn) formulas: can be transformed into Horn fomulas by renmaing of a subset of variables (replacing positive occurrences by negative ones and vice versa).
  Example: $(X_1 \vee X_2 \vee \overline{X_3})(\overline{X_2} \vee \overline{X_1} \vee \overline{X_4})(X_3 \vee X_2 \vee \overline{X_4})$ is not Horn but RHorn that can be transformed to Horn by renaming of $X_2$.

# Renamable Horn formulas

- Horn formulas: each clause contains at most one positive literal.
  Example: $(X_1 \vee \overline{X_2} \vee \overline{X_3})(X_2 \vee \overline{X_1} \vee \overline{X_4})(X_3 \vee \overline{X_2} \vee \overline{X_4})$
- Renamable Horn (RHorn) formulas: can be transformed into Horn fomulas by renmaing of a subset of variables (replacing positive occurrences by negative ones and vice versa).
  Example: $(X_1 \vee X_2 \vee \overline{X_3})(\overline{X_2} \vee \overline{X_1} \vee \overline{X_4})(X_3 \vee X_2 \vee \overline{X_4})$ is not Horn but RHorn that can be transformed to Horn by renaming of $X_2$.
- The SAT problem for RHorn CNF can be solved in a linear time.
- Many real-wolrd instances are close to being RHorn.
- So, it is good to be able to efficiently compute small RHorn backdoor sets

# RHorn deletion backdoor sets

- Computing RHorn backdoor is W[2]-hard, i.e. very unlikely to be FPT (Proposition 6. in the above survey of Gaspers and Szeider)

# RHorn deletion backdoor sets

- Computing RHorn backdoor is W[2]-hard, i.e. very unlikely to be FPT (Proposition 6. in the above survey of Gaspers and Szeider)
- RHorn deletion backdoor: a subset of variables whose *removal* makes the formula belong to clas RHorn.
- The deletion backdoor is generally larger than the ordinary backdoor yet quite small for practical instances.

# RHorn deletion backdoor sets

- Computing RHorn backdoor is W[2]-hard, i.e. very unlikely to be FPT (Proposition 6. in the above survey of Gaspers and Szeider)
- RHorn deletion backdoor: a subset of variables whose *removal* makes the formula belong to clas RHorn.
- The deletion backdoor is generally larger than the ordinary backdoor yet quite small for practical instances.
- Example: $(X_1 \vee X_2 \vee X_3 \vee X_4)(\overline{X_1} \vee \overline{X_2} \vee \overline{X_3} \vee \overline{X_4})$
- $\{X_1\}$ is a RHorn backdoor of the above formula but not deletion backdoor. A RHorn deletion backdoor set is $\{X_1, X_2\}$.

# RHorn deletion backdoor sets

- RHorn CNFs can be easily recognized by solving a 2SAT problem.
  - Each variable $X$ is associated with varable $RX$ whose truth value determines whether $X$ is to be renamed.
  - The forbidden combinations of renamings for all pairs of variables occurring in the same clause can be expressed as a 2SAT instance.
  - The formula is RHorn if and only if this 2SAT is satisfiable.

# RHorn deletion backdoor sets

- RHorn CNFs can be easily recognized by solving a 2SAT problem.
  - Each variable $X$ is associated with varable $RX$ whose truth value determines whether $X$ is to be renamed.
  - The forbidden combinations of renamings for all pairs of variables occurring in the same clause can be expressed as a 2SAT instance.
  - The formula is RHorn if and only if this 2SAT is satisfiable.
- The given CNF has an RHorn deletion backdoor of size $k$ if and only if the above 2SAT instance can be made satisfiable by removal of at most $k$ variables. (Gottlob and Szeider, Computer Journal, 2008).
- Thus the fixed-parameter tractability of RHorn deletion backdoor has been reduced to fixed-parameter tractability of Min-2CNF-deletion problem.

# Min-2CNF deletion problem

- Given a 2CNF, is it possible to remove at most $k$ clauses to make it satisfiable (FPT equivalent to the query of removal of $k$ variables).
- Was a challenging open problem for more than 10 years.
- Shown FPT in "Almost 2 SAT is Fixed-Parameter Tractable, by Razgon and O'Sullivan, Journal of Comp. and Sys. Sciences Vol 75, pp. 435-450, 2009.
- Our algorithm takes $O(15^k m^3)$ where $m$ is the number of clauses and thus not sutiable for practical applications.

# Line of improvements of the result for Min-2CNF deletion

- Runtime improvements:
  - $O^*(9^k)$ algorithm (Raman et al, ESA'11)
  - $O^*(4^k)$ algorithm (Cygan et al, IPEC'11)
  - $O^*(2.67^k)$ algorithm (Narayanaswamy et al, manuscript).

# Line of improvements of the result for Min-2CNF deletion

- Runtime improvements:
  - $O^*(9^k)$ algorithm (Raman et al, ESA'11)
  - $O^*(4^k)$ algorithm (Cygan et al, IPEC'11)
  - $O^*(2.67^k)$ algorithm (Narayanaswamy et al, manuscript).
- The most surprising development:
  - Min-2CNF deletion is *kernelizable*.
  - There is a (randomized) poly-time algorithm transforming the given instance into one whose size polynomially depends on $k$ (Kratsch and Wahlstrom, "Represenative sets and Irrelevant vertices...", available in arxiv)
  - The dependence of $k$ can be come an additive constant instead multiplicative one!
- The recent development make the parameterized approach *potentially* applicable for computing of small backdoors.

# Yes or No?

- We have seen on example of RHorn backdoors that computing small backdoors at the preprocessing stage is *potentially* practically applicable for SAT solving.

# Yes or No?

- We have seen on example of RHorn backdoors that computing small backdoors at the preprocessing stage is *potentially* practically applicable for SAT solving.
- Question: has it been applied in practice?

# Yes or No?

- We have seen on example of RHorn backdoors that computing small backdoors at the preprocessing stage is *potentially* practically applicable for SAT solving.
- Question: has it been applied in practice?
- The answer requires answering two subquestion:
  - Have small backdoors been utilized for SAT solving? YES!

# Yes or No?

- We have seen on example of RHorn backdoors that computing small backdoors at the preprocessing stage is *potentially* practically applicable for SAT solving.
- Question: has it been applied in practice?
- The answer requires answering two subquestion:
  - Have small backdoors been utilized for SAT solving? YES!
  - Have the FPT algorithms been used at the preprocessing stage? Not yet!

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking
  - The variable to branch on is chosen at random.

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking
  - The variable to branch on is chosen at random.
  - If the algorithm runs longer than some appriori specified time limit, it is terminated

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking
  - The variable to branch on is chosen at random.
  - If the algorithm runs longer than some appriori specified time limit, it is terminated
  - After the termination the algorithm starts from scratch

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking
  - The variable to branch on is chosen at random.
  - If the algorithm runs longer than some appriori specified time limit, it is terminated
  - After the termination the algorithm starts from scratch
  - If the algorithm does not return an answer after some appriori specified number of attempts, it returns 'DO NOT KNOW'

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking
  - The variable to branch on is chosen at random.
  - If the algorithm runs longer than some appriori specified time limit, it is terminated
  - After the termination the algorithm starts from scratch
  - If the algorithm does not return an answer after some appriori specified number of attempts, it returns 'DO NOT KNOW'
- Solves huge benchmark instances far out of reach of the traditional backtracking algorithms.

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking
  - The variable to branch on is chosen at random.
  - If the algorithm runs longer than some appriori specified time limit, it is terminated
  - After the termination the algorithm starts from scratch
  - If the algorithm does not return an answer after some appriori specified number of attempts, it returns 'DO NOT KNOW'
- Solves huge benchmark instances far out of reach of the traditional backtracking algorithms.
- For more details, see "Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems" by Gomes, Selman, Crato, and Kautz, Journ. of Aut. Reasoning, Vol 24, pp. 67-100 (2000).

# Randomized restarts

- Randomized restarts is a ridiculously simple modification of backtracking
  - The variable to branch on is chosen at random.
  - If the algorithm runs longer than some appriori specified time limit, it is terminated
  - After the termination the algorithm starts from scratch
  - If the algorithm does not return an answer after some appriori specified number of attempts, it returns 'DO NOT KNOW'
- Solves huge benchmark instances far out of reach of the traditional backtracking algorithms.
- For more details, see "Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems" by Gomes, Selman, Crato, and Kautz, Journ. of Aut. Reasoning, Vol 24, pp. 67-100 (2000).
- What is the reason of so good a performance?

# Oblivious computation of backdoor sets

- One possible explanationis offered in "Backdoors to typical case complexity" by Williams, Gomes, and Selman, IJCAI03:

# Oblivious computation of backdoor sets

- One possible explanationis offered in "Backdoors to typical case complexity" by Williams, Gomes, and Selman, IJCAI03:
  - A class $A$ of CNF is considered *easy* for the *given* backtracking algorithm if any $F \in A$ can be efficiently (e.g. in $O(n)$ or in $O(n^2)$) solved by *this* algorithm.

# Oblivious computation of backdoor sets

- One possible explanationis offered in "Backdoors to typical case complexity" by Williams, Gomes, and Selman, IJCAI03:
  - A class $A$ of CNF is considered *easy* for the *given* backtracking algorithm if any $F \in A$ can be efficiently (e.g. in $O(n)$ or in $O(n^2)$) solved by *this* algorithm.
  - Given a formula $F$, a set $S$ of its variables is a *backdoor* w.r.t. $A$ if as a result of any assignment of $S$, the resulting residual formula belongs to $A$.

# Oblivious computation of backdoor sets

- One possible explanationis offered in "Backdoors to typical case complexity" by Williams, Gomes, and Selman, IJCAI03:
  - A class $A$ of CNF is considered *easy* for the *given* backtracking algorithm if any $F \in A$ can be efficiently (e.g. in $O(n)$ or in $O(n^2)$) solved by *this* algorithm.
  - Given a formula $F$, a set $S$ of its variables is a *backdoor* w.r.t. $A$ if as a result of any assignment of $S$, the resulting residual formula belongs to $A$.
  - It is empirically shown that many real-world instances have small such backdoor sets.

# Oblivious computation of backdoor sets

- One possible explanationis offered in "Backdoors to typical case complexity" by Williams, Gomes, and Selman, IJCAI03:
  - A class $A$ of CNF is considered *easy* for the *given* backtracking algorithm if any $F \in A$ can be efficiently (e.g. in $O(n)$ or in $O(n^2)$) solved by *this* algorithm.
  - Given a formula $F$, a set $S$ of its variables is a *backdoor* w.r.t. $A$ if as a result of any assignment of $S$, the resulting residual formula belongs to $A$.
  - It is empirically shown that many real-world instances have small such backdoor sets.
  - Moreover, it is argued that the restart algorithm managed to capture such sets and branch over them!

# Oblivious computation of backdoor sets

- One possible explanationis offered in "Backdoors to typical case complexity" by Williams, Gomes, and Selman, IJCAI03:
  - A class $A$ of CNF is considered *easy* for the *given* backtracking algorithm if any $F \in A$ can be efficiently (e.g. in $O(n)$ or in $O(n^2)$) solved by *this* algorithm.
  - Given a formula $F$, a set $S$ of its variables is a *backdoor* w.r.t. $A$ if as a result of any assignment of $S$, the resulting residual formula belongs to $A$.
  - It is empirically shown that many real-world instances have small such backdoor sets.
  - Moreover, it is argued that the restart algorithm managed to capture such sets and branch over them!
- The restart algorithm is *obliviously fixed-parameter*

# Oblivious computation of backdoor sets

- One possible explanationis offered in "Backdoors to typical case complexity" by Williams, Gomes, and Selman, IJCAI03:
  - A class $A$ of CNF is considered *easy* for the *given* backtracking algorithm if any $F \in A$ can be efficiently (e.g. in $O(n)$ or in $O(n^2)$) solved by *this* algorithm.
  - Given a formula $F$, a set $S$ of its variables is a *backdoor* w.r.t. $A$ if as a result of any assignment of $S$, the resulting residual formula belongs to $A$.
  - It is empirically shown that many real-world instances have small such backdoor sets.
  - Moreover, it is argued that the restart algorithm managed to capture such sets and branch over them!
- The restart algorithm is *obliviously fixed-parameter*
- Instead of branching on all $n$ variables, it branches on a hadful $k$ of them.

# Explicit preprocessing computation

- Paris et al. "Computing Horn Strong Backdoor Sets Thanks to Local Search", ICTAI 2006
  - Heuristically compute RHorn backdoors (not deletion one!) at the preprocessing stage.
  - Claim improvement of performance of ZChaff, a well known SAT solver.

# Explicit preprocessing computation

- Paris et al. "Computing Horn Strong Backdoor Sets Thanks to Local Search", ICTAI 2006
  - Heuristically compute RHorn backdoors (not deletion one!) at the preprocessing stage.
  - Claim improvement of performance of ZChaff, a well known SAT solver.
- Interesting: the backdoor set they compute is W[2]-hard: there is no hope to replace their heuristic by a fixed-parameter algorithm.

# Explicit preprocessing computation

- Paris et al. "Computing Horn Strong Backdoor Sets Thanks to Local Search", ICTAI 2006
  - Heuristically compute RHorn backdoors (not deletion one!) at the preprocessing stage.
  - Claim improvement of performance of ZChaff, a well known SAT solver.
- Interesting: the backdoor set they compute is W[2]-hard: there is no hope to replace their heuristic by a fixed-parameter algorithm.
- The perspective of applying FPT algorithms in this context is still unclear!

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments
  - Typical query: is there a satisfying assignment assigning variables of set $S_1$ with *true* and variables of set $S_2$ with *false*?

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments
  - Typical query: is there a satisfying assignment assigning variables of set $S_1$ with *true* and variables of set $S_2$ with *false*?
  - Queries have to be answered real-time

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments
  - Typical query: is there a satisfying assignment assigning variables of set $S_1$ with *true* and variables of set $S_2$ with *false*?
  - Queries have to be answered real-time
  - Queries are not known in advance.

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments
  - Typical query: is there a satisfying assignment assigning variables of set $S_1$ with *true* and variables of set $S_2$ with *false*?
  - Queries have to be answered real-time
  - Queries are not known in advance.
- A CNF formula is transformed *offline* (in exponential time) into a representation meeting the above requirements.

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments
  - Typical query: is there a satisfying assignment assigning variables of set $S_1$ with *true* and variables of set $S_2$ with *false*?
  - Queries have to be answered real-time
  - Queries are not known in advance.
- A CNF formula is transformed *offline* (in exponential time) into a representation meeting the above requirements.
- The representation should be space-efficient.

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments
  - Typical query: is there a satisfying assignment assigning variables of set $S_1$ with *true* and variables of set $S_2$ with *false*?
  - Queries have to be answered real-time
  - Queries are not known in advance.
- A CNF formula is transformed *offline* (in exponential time) into a representation meeting the above requirements.
- The representation should be space-efficient.
- It is not easy to do since there may be exponentially many solutions.

# What is knowledge compilation?

- In some industrial application (e.g. hardware verification, car industry), CNF formulas are regarded as knowledge bases *known in advance* :
  - Queries are asked regarding the structure of satifying assignments
  - Typical query: is there a satisfying assignment assigning variables of set $S_1$ with *true* and variables of set $S_2$ with *false*?
  - Queries have to be answered real-time
  - Queries are not known in advance.
- A CNF formula is transformed *offline* (in exponential time) into a representation meeting the above requirements.
- The representation should be space-efficient.
- It is not easy to do since there may be exponentially many solutions.
- The study of various representation formalisms constitutes the field of *knowledge compilation*.

# A simple formalism: DNF

- **Elementary conjuction:** conjunction of literals (e.g. $x_1\overline{x_2}x_3$)

# A simple formalism: DNF

- **Elementary conjuction:** conjunction of literals (e.g. $x_1\overline{x_2}x_3$)
- **Disjunctive normal form (DNF):** disjunction of elementary conjunctions (e.g. $x_1\overline{x_2}x_3 \vee \overline{x_1}x_4x_5$).

# A simple formalism: DNF

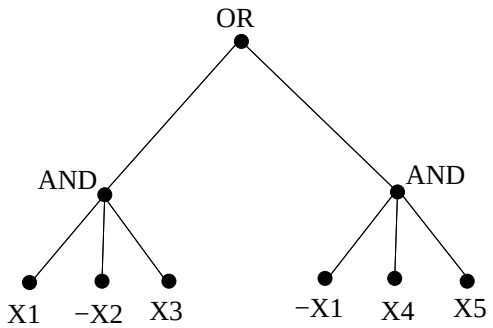
- **Elementary conjuction:** conjunction of literals (e.g. $x_1\overline{x_2}x_3$)
- **Disjunctive normal form (DNF):** disjunction of elementary conjunctions (e.g. $x_1\overline{x_2}x_3 \vee \overline{x_1}x_4x_5$).
- Given sets $S_1$ and $S_2$ of variables, it can be tested in polytime whether the given DNF has a satisfying assignment with $S_1 \leftarrow \mathit{true}$ and $S_2 \leftarrow \mathit{false}$

# A simple formalism: DNF

- **Elementary conjuction:** conjunction of literals (e.g. $x_1\overline{x_2}x_3$)
- **Disjunctive normal form (DNF):** disjunction of elementary conjunctions (e.g. $x_1\overline{x_2}x_3 \vee \overline{x_1}x_4x_5$).
- Given sets $S_1$ and $S_2$ of variables, it can be tested in polytime whether the given DNF has a satisfying assignment with $S_1 \leftarrow \mathit{true}$ and $S_2 \leftarrow \mathit{false}$
- Drawback: for many classes of simple CNFs, the corresponding DNFs are of exponential size.

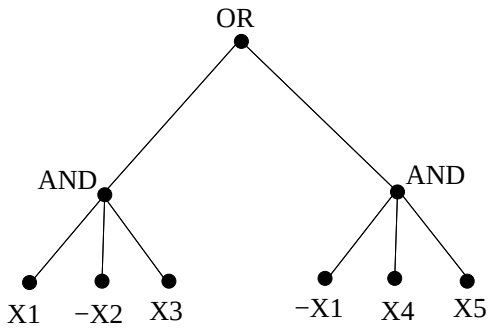
# From DNF to DNNF

- A DNF is easy to represent in the following graphical way:



# From DNF to DNNF

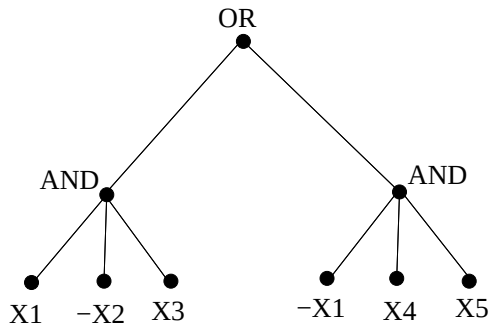
- A DNF is easy to represent in the following graphical way:




- Generalize this graph representation by allowing an arbitrary number of alternating AND-OR levels.

# From DNF to DNNF

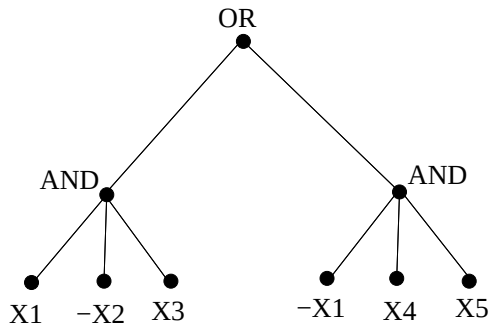
- A DNF is easy to represent in the following graphical way:




- Generalize this graph representation by allowing an arbitrary number of alternating AND-OR levels.
- Require that two subgraphs having a common AND parent do not share variables.

# From DNF to DNNF

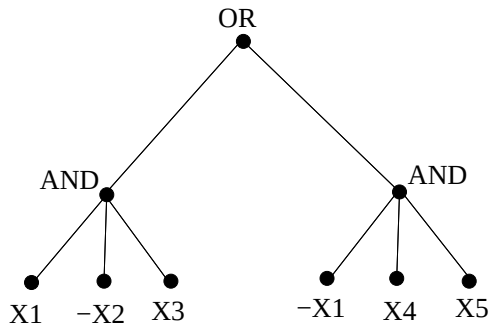
- A DNF is easy to represent in the following graphical way:




- Generalize this graph representation by allowing an arbitrary number of alternating AND-OR levels.
- Require that two subgraphs having a common AND parent do not share variables.
- We obtain a representation called Disjunctive negation normal form (DNNF)

# From DNF to DNNF

- A DNF is easy to represent in the following graphical way:




- Generalize this graph representation by allowing an arbitrary number of alternating AND-OR levels.
- Require that two subgraphs having a common AND parent do not share variables.
- We obtain a representation called Disjunctive negation normal form (DNNF)
- It is much more general than DNF, yet allows to answer a typical query in polytime.

# The complexity of DNNF

- Of course DNNF can take exponential space.

# The complexity of DNNF

- Of course DNNF can take exponential space.
- However the exponent is *not* in the number of variables, but in the *treewidth* of the target CNF.

# The complexity of DNNF

- Of course DNNF can take exponential space.
- However the exponent is *not* in the number of variables, but in the *treewidth* of the target CNF.
- The representation is generated by a fixed-parameter algorithm in terms of the treewidth.

# The complexity of DNNF

- Of course DNNF can take exponential space.
- However the exponent is *not* in the number of variables, but in the *treewidth* of the target CNF.
- The representation is generated by a fixed-parameter algorithm in terms of the treewidth.
- Successfully applied to the industrial instances of the hardware diagnosis problem

# The complexity of DNNF

- Of course DNNF can take exponential space.
- However the exponent is *not* in the number of variables, but in the *treewidth* of the target CNF.
- The representation is generated by a fixed-parameter algorithm in terms of the treewidth.
- Successfully applied to the industrial instances of the hardware diagnosis problem
- More details at: A. Darwiche "Decomposable negation normal forms", JACM, vol 48, pp. 608-647, 2001.

# Further development of the idea

- The DNNF is a fixed-parameter method in terms of *space complexity*.
- However, a fixed-parameter algorithm for knowledge compilation has been used *implicitly*.
- Since then, to the best of my knowledge, no attempt has been made to further explore the potential of fixed-parameter computation in knowledge compilation.

# Further development of the idea

- The DNNF is a fixed-parameter method in terms of *space complexity*.
- However, a fixed-parameter algorithm for knowledge compilation has been used *implicitly*.
- Since then, to the best of my knowledge, no attempt has been made to further explore the potential of fixed-parameter computation in knowledge compilation.
- Cadoli and then Hubie Chen developed a theory of parameterized compilability (classes, relationships between them etc.)
- However, this direction has not been taken any further (e.g. no concrete methods of formalisation based on new parameters).

# Possible further research

- Exploration of cliquewidth:
  - Design of representation formalism parameterised by the cliquewidth
  - Is small cliquewidth a necessary condition for succint representation by the existing formalisms.
- Exploiting sizes of various backdoor sets as possible parameters for succinct knowledge compilation.

# Summary

- There are two modes of SAT solving.

# Summary

- There are two modes of SAT solving.
- A SAT instance is the input:
  - Theory of fixed-parameter computation is more or less understood.
  - Little effect to the practical SAT solving: theory and practice go in parallel.
  - Research in Algorithms Engineering is required to close the gap, see `http://www.user.tu-berlin.de/hueffner/` for an example of successful research of this kind.

# Summary

- There are two modes of SAT solving.
- A SAT instance is the input:
  - Theory of fixed-parameter computation is more or less understood.
  - Little effect to the practical SAT solving: theory and practice go in parallel.
  - Research in Algorithms Engineering is required to close the gap, see `http://www.user.tu-berlin.de/hueffner/` for an example of successful research of this kind.
- The SAT instance is known, a query is the input.
  - Practical efficiency is well established for one particular application and one particular parameter.
  - A lot of interesting theoretical work on generalizing, extending, and better understanding of this phenomenon.