Fixed-parameter tractability of satisfying beyond the number of variables

Robert Crowston, Gregory Gutin, Mark Jones, Venkatesh Raman, Saket Saurabh, Anders Yeo

SAT 2012, Trento, Italy

June 20, 2012

Outline of the Talk

- Introduction
  - Fixed-parameter tractability
  - Some parameterized questions, results related to SAT
  - Above guarantee parameterizations
- The main question and the result in the paper
  - Consequences (corollaries)
  - The Algorithm outline
- Conclusions.

Fixed Parameter Tractable (FPT) Algorithms

- Input x comes with a parameter k (e.g. solution size, backdoor size to Horn)
- Solve (x, k) in $f(k) + |x|^{O(1)}$ time where f is a function of k alone.
- Parameters can be anything that makes sense in theory and practice (We will see examples shortly).
- Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
- The problem is said to be in XP if it has an $|x|^{f(k)}$ algorithm.
- There is a W-hardness theory to identify problems that are unlikely to have such algorithms

Fixed Parameter Tractable (FPT) Algorithms

- Input x comes with a parameter k (e.g. solution size, backdoor size to Horn)
- Solve (x, k) in $f(k) + |x|^{O(1)}$ time where f is a function of k alone.
- Parameters can be anything that makes sense in theory and practice (We will see examples shortly).
- Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
- The problem is said to be in XP if it has an $|x|^{f(k)}$ algorithm.
- There is a W-hardness theory to identify problems that are unlikely to have such algorithms

Fixed Parameter Tractable (FPT) Algorithms

- Input x comes with a parameter k (e.g. solution size, backdoor size to Horn)
- Solve (x, k) in $f(k) + |x|^{O(1)}$ time where f is a function of k alone.
- Parameters can be anything that makes sense in theory and practice (We will see examples shortly).
- Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
- The problem is said to be in XP if it has an $|x|^{f(k)}$ algorithm.
- There is a W-hardness theory to identify problems that are unlikely to have such algorithms

Fixed Parameter Tractable (FPT) Algorithms

- Input x comes with a parameter k (e.g. solution size, backdoor size to Horn)
- Solve (x, k) in $f(k) + |x|^{O(1)}$ time where f is a function of k alone.
- Parameters can be anything that makes sense in theory and practice (We will see examples shortly).
- Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
- The problem is said to be in XP if it has an $|x|^{f(k)}$ algorithm.
- There is a W-hardness theory to identify problems that are unlikely to have such algorithms

Fixed Parameter Tractable (FPT) Algorithms

- Input x comes with a parameter k (e.g. solution size, backdoor size to Horn)
- Solve (x, k) in $f(k) + |x|^{O(1)}$ time where f is a function of k alone.
- Parameters can be anything that makes sense in theory and practice (We will see examples shortly).
- Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
- The problem is said to be in XP if it has an $|x|^{f(k)}$ algorithm.
- There is a W-hardness theory to identify problems that are unlikely to have such algorithms

Fixed Parameter Tractable (FPT) Algorithms

- Input x comes with a parameter k (e.g. solution size, backdoor size to Horn)
- Solve (x, k) in $f(k) + |x|^{O(1)}$ time where f is a function of k alone.
- Parameters can be anything that makes sense in theory and practice (We will see examples shortly).
- Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
- The problem is said to be in XP if it has an $|x|^{f(k)}$ algorithm.
- There is a W-hardness theory to identify problems that are unlikely to have such algorithms

Fixed Parameter Tractable (FPT) Algorithms

- Input x comes with a parameter k (e.g. solution size, backdoor size to Horn)
- Solve (x, k) in $f(k) + |x|^{O(1)}$ time where f is a function of k alone.
- Parameters can be anything that makes sense in theory and practice (We will see examples shortly).
- Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
- The problem is said to be in XP if it has an $|x|^{f(k)}$ algorithm.
- There is a W-hardness theory to identify problems that are unlikely to have such algorithms

Kernelization

- *Kernelization*: Reduce (x, k) in polynomial time to an equivalent instance (x', k') such that $|x'|$ and k' are some functions of k.
- *Folklore*: Fixed-parameter tractable if and only if kernelizable.
- *Interesting question*: When is $|x'|$ a polynomial function of k?
- Recent breakthroughs in showing lower bounds for kernel sizes (under complexity theoretic assumptions).

Kernelization

- *Kernelization*: Reduce (x, k) in polynomial time to an equivalent instance (x', k') such that $|x'|$ and k' are some functions of k.
- *Folklore*: Fixed-parameter tractable if and only if kernelizable.
- *Interesting question*: When is $|x'|$ a polynomial function of k?
- Recent breakthroughs in showing lower bounds for kernel sizes (under complexity theoretic assumptions).

Kernelization

- *Kernelization*: Reduce (x, k) in polynomial time to an equivalent instance (x', k') such that $|x'|$ and k' are some functions of k.
- *Folklore*: Fixed-parameter tractable if and only if kernelizable.
- *Interesting question*: When is $|x'|$ a polynomial function of k?
- Recent breakthroughs in showing lower bounds for kernel sizes (under complexity theoretic assumptions).

Kernelization

- *Kernelization*: Reduce (x, k) in polynomial time to an equivalent instance (x', k') such that $|x'|$ and k' are some functions of k.
- *Folklore*: Fixed-parameter tractable if and only if kernelizable.
- *Interesting question*: When is $|x'|$ a polynomial function of k?
- Recent breakthroughs in showing lower bounds for kernel sizes (under complexity theoretic assumptions).

Kernelization

- *Kernelization*: Reduce (x, k) in polynomial time to an equivalent instance (x', k') such that $|x'|$ and k' are some functions of k.
- *Folklore*: Fixed-parameter tractable if and only if kernelizable.
- *Interesting question*: When is $|x'|$ a polynomial function of k?
- Recent breakthroughs in showing lower bounds for kernel sizes (under complexity theoretic assumptions).

Parameterized questions/answers related to SAT

Minones SAT

- Given a CNF formula, is there a satisfying assignment with at most k ones (weight at most k)?
  - $W[2]$-hard for general SAT (Dominating Set)
  - Fixed-parameter tractable (r^k) for bounded (r) CNF.
  - For 2CNF, equivalent to VERTEX COVER
- Given a CNF formula, is there a satisfying assignment with weight at least (or equal to) k?
  - $W[1]$-hard (Independent Set) even for 2SAT.

- Given a CNF formula, is there a satisfying assignment with at most k ones (weight at most k)?
  - $W[2]$-hard for general SAT (Dominating Set)
  - Fixed-parameter tractable (r^k) for bounded (r) CNF.
  - For 2CNF, equivalent to VERTEX COVER
- Given a CNF formula, is there a satisfying assignment with weight at least (or equal to) k?
  - $W[1]$-hard (Independent Set) even for 2SAT.

- Given a CNF formula, is there a satisfying assignment with at most k ones (weight at most k)?
  - $W[2]$-hard for general SAT (Dominating Set)
  - Fixed-parameter tractable (r^k) for bounded (r) CNF.
  - For 2CNF, equivalent to VERTEX COVER
- Given a CNF formula, is there a satisfying assignment with weight at least (or equal to) k?
  - $W[1]$-hard (Independent Set) even for 2SAT.

- Given a CNF formula, is there a satisfying assignment with at most k ones (weight at most k)?
  - $W[2]$-hard for general SAT (Dominating Set)
  - Fixed-parameter tractable (r^k) for bounded (r) CNF.
  - For 2CNF, equivalent to VERTEX COVER
- Given a CNF formula, is there a satisfying assignment with weight at least (or equal to) k?
  - $W[1]$-hard (Independent Set) even for 2SAT.

- Given a CNF formula, is there a satisfying assignment with at most k ones (weight at most k)?
  - $W[2]$-hard for general SAT (Dominating Set)
  - Fixed-parameter tractable (r^k) for bounded (r) CNF.
  - For 2CNF, equivalent to VERTEX COVER
- Given a CNF formula, is there a satisfying assignment with weight at least (or equal to) k?
  - $W[1]$-hard (Independent Set) even for 2SAT.

- Given a CNF formula, is there an assignment satisfying at least k clauses?
  - (Trivially) Fixed-parameter tractable (with $f(k)$ being c^k where $c < 1.4$)
- Given a 2CNF formula, is there an assignment that satisfies all but at most k clauses? (Hard for 3CNF and above).
  - Fixed-parameter tractable ($15^k --> 9^k --> 4^k --> 2.32^k$)

- Given a CNF formula, is there an assignment satisfying at least k clauses?
  - (Trivially) Fixed-parameter tractable (with $f(k)$ being c^k where $c < 1.4$)
- Given a 2CNF formula, is there an assignment that satisfies all but at most k clauses? (Hard for 3CNF and above).
  - Fixed-parameter tractable ($15^k --> 9^k --> 4^k --> 2.32^k$)

Parameterized questions/answers related to SAT

MaxSAT variations

- Given a CNF formula, is there an assignment satisfying at least k clauses?
  - (Trivially) Fixed-parameter tractable (with $f(k)$ being c^k where $c < 1.4$)
- Given a 2CNF formula, is there an assignment that satisfies all but at most k clauses? (Hard for 3CNF and above).
  - Fixed-parameter tractable ($15^k - - > 9^k - - > 4^k - - > 2.32^k$)

- Given a CNF formula, is there an assignment satisfying at least k clauses?
  - (Trivially) Fixed-parameter tractable (with $f(k)$ being c^k where $c < 1.4$)
- Given a 2CNF formula, is there an assignment that satisfies all but at most k clauses? (Hard for 3CNF and above).
  - Fixed-parameter tractable ($15^k --> 9^k --> 4^k --> 2.32^k$)

Parameterized questions/answers related to SAT

MaxSAT variations

- Given a CNF formula, is there an assignment satisfying at least k clauses?
  - (Trivially) Fixed-parameter tractable (with $f(k)$ being c^k where $c < 1.4$)
- Given a 2CNF formula, is there an assignment that satisfies all but at most k clauses? (Hard for 3CNF and above).
  - Fixed-parameter tractable ($15^k --> 9^k --> 4^k --> 2.32^k$)

Above guarantee parameterizations

- Is there an assignment satisfying at least k clauses (of the formula wth m clauses)?

  Trivially FPT, as if $k \leq m/2$, the answer is YES, and otherwise, $m \leq 2k$, and we have a kernel for the number of clauses (and one can obtain a kernel for the number of variables with some effort).

- Hence trivial if k is small, and so FPT algorithm applies only when k is large, and the c^k algorithm may not be practical.

- *More appropriate question/parameterization*: Is there an assignment satisfying at least $m/2 + k$ clauses? FPT (Mahajan and Raman, 1999).

- *Other 'above guarantee parameterizations' of SAT*: Is there an assignment satisfying at least 'Expected number by a random assignment' + k clauses?

  NOT even in XP (under ETH) unless the number of variables in each clause is bounded (or up to $\log \log n$)) (LATIN 2012).

Above guarantee parameterizations

- Is there an assignment satisfying at least k clauses (of the formula wth m clauses)?

  Trivially FPT, as if $k \leq m/2$, the answer is YES, and otherwise, $m \leq 2k$, and we have a kernel for the number of clauses (and one can obtain a kernel for the number of variables with some effort).

- Hence trivial if k is small, and so FPT algorithm applies only when k is large, and the c^k algorithm may not be practical.

- *More appropriate question/parameterization*: Is there an assignment satisfying at least $m/2 + k$ clauses? FPT (Mahajan and Raman, 1999).

- *Other 'above guarantee parameterizations' of SAT*: Is there an assignment satisfying at least 'Expected number by a random assignment' + k clauses?

  NOT even in *XP* (under ETH) unless the number of variables in each clause is bounded (or up to $\log \log n$)) (LATIN 2012).

Above guarantee parameterizations

- Is there an assignment satisfying at least k clauses (of the formula wth m clauses)?

  Trivially FPT, as if $k \leq m/2$, the answer is YES, and otherwise, $m \leq 2k$, and we have a kernel for the number of clauses (and one can obtain a kernel for the number of variables with some effort).

- Hence trivial if k is small, and so FPT algorithm applies only when k is large, and the c^k algorithm may not be practical.

- *More appropriate question/parameterization*: Is there an assignment satisfying at least $m/2 + k$ clauses? FPT (Mahajan and Raman, 1999).

- *Other 'above guarantee parameterizations' of SAT*: Is there an assignment satisfying at least 'Expected number by a random assignment' + k clauses?

  NOT even in *XP* (under ETH) unless the number of variables in each clause is bounded (or up to log log n)) (LATIN 2012).

Above guarantee parameterizations

- Is there an assignment satisfying at least k clauses (of the formula wth m clauses)?

  Trivially FPT, as if $k \leq m/2$, the answer is YES, and otherwise, $m \leq 2k$, and we have a kernel for the number of clauses (and one can obtain a kernel for the number of variables with some effort).

- Hence trivial if k is small, and so FPT algorithm applies only when k is large, and the c^k algorithm may not be practical.

- *More appropriate question/parameterization*: Is there an assignment satisfying at least $m/2 + k$ clauses? FPT (Mahajan and Raman, 1999).

- *Other 'above guarantee parameterizations' of SAT*: Is there an assignment satisfying at least 'Expected number by a random assignment' + k clauses?

  NOT even in *XP* (under ETH) unless the number of variables in each clause is bounded (or up to $\log \log n$)) (LATIN 2012).

Above guarantee parameterizations

- Is there an assignment satisfying at least k clauses (of the formula wth m clauses)?

  Trivially FPT, as if $k \leq m/2$, the answer is YES, and otherwise, $m \leq 2k$, and we have a kernel for the number of clauses (and one can obtain a kernel for the number of variables with some effort).

- Hence trivial if k is small, and so FPT algorithm applies only when k is large, and the c^k algorithm may not be practical.

- *More appropriate question/parameterization*: Is there an assignment satisfying at least $m/2 + k$ clauses? FPT (Mahajan and Raman, 1999).

- *Other 'above guarantee parameterizations' of SAT*: Is there an assignment satisfying at least 'Expected number by a random assignment' $+ k$ clauses?

  NOT even in *XP* (under ETH) unless the number of variables in each clause is bounded (or up to $\log \log n$)) (LATIN 2012).

Above guarantee parameterizations

- Is there an assignment satisfying at least k clauses (of the formula wth m clauses)?

  Trivially FPT, as if $k \leq m/2$, the answer is YES, and otherwise, $m \leq 2k$, and we have a kernel for the number of clauses (and one can obtain a kernel for the number of variables with some effort).

- Hence trivial if k is small, and so FPT algorithm applies only when k is large, and the c^k algorithm may not be practical.

- *More appropriate question/parameterization*: Is there an assignment satisfying at least $m/2 + k$ clauses? FPT (Mahajan and Raman, 1999).

- *Other 'above guarantee parameterizations' of SAT*: Is there an assignment satisfying at least 'Expected number by a random assignment' $+ k$ clauses?

  NOT even in *XP* (under ETH) unless the number of variables in each clause is bounded (or up to $\log \log n$)) (LATIN 2012).

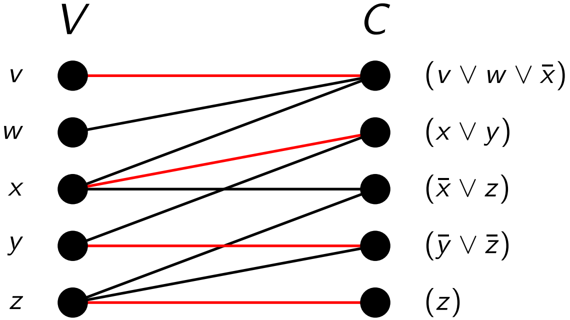
Above guarantee parameterizations

- Is there an assignment satisfying at least k clauses (of the formula wth m clauses)?

  Trivially FPT, as if $k \leq m/2$, the answer is YES, and otherwise, $m \leq 2k$, and we have a kernel for the number of clauses (and one can obtain a kernel for the number of variables with some effort).

- Hence trivial if k is small, and so FPT algorithm applies only when k is large, and the c^k algorithm may not be practical.
- *More appropriate question/parameterization*: Is there an assignment satisfying at least $m/2 + k$ clauses? FPT (Mahajan and Raman, 1999).
- *Other 'above guarantee parameterizations' of SAT*: Is there an assignment satisfying at least 'Expected number by a random assignment' $+ k$ clauses?
  NOT even in *XP* (under ETH) unless the number of variables in each clause is bounded (or up to $\log \log n$)) (LATIN 2012).

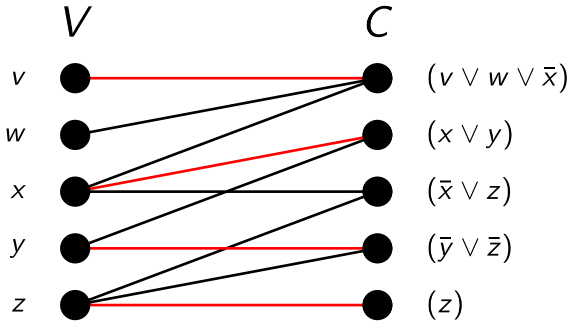
This paper



Let $B = (V, F)$ be the natural variable-clause incidence bipartite graph of the formula F, and let μ be the maximum matching in B.

- Clearly at least μ clauses can be satisfied.
- What about $\mu + k$ clauses? It is fixed-parameter tractable to decide whether $\mu + k$ clauses can be satisfied (k is the parameter).

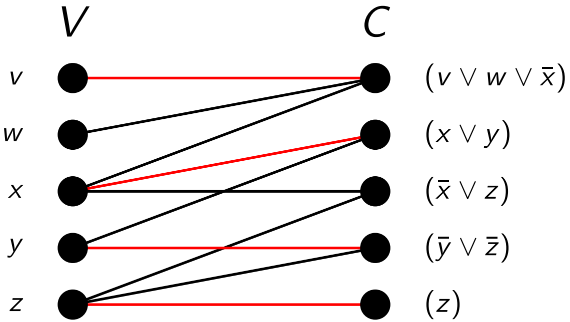
This paper



Let $B = (V, F)$ be the natural variable-clause incidence bipartite graph of the formula F, and let μ be the maximum matching in B.

- Clearly at least μ clauses can be satisfied.
- What about $\mu + k$ clauses? It is fixed-parameter tractable to decide whether $\mu + k$ clauses can be satisfied (k is the parameter).

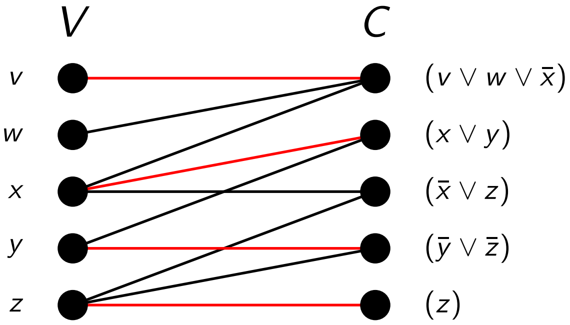
This paper



Let $B = (V, F)$ be the natural variable-clause incidence bipartite graph of the formula F, and let μ be the maximum matching in B.

- Clearly at least μ clauses can be satisfied.
- What about $\mu + k$ clauses? It is fixed-parameter tractable to decide whether $\mu + k$ clauses can be satisfied (k is the parameter).

This paper



Let $B = (V, F)$ be the natural variable-clause incidence bipartite graph of the formula F, and let μ be the maximum matching in B.

- Clearly at least μ clauses can be satisfied.
- What about $\mu + k$ clauses? It is fixed-parameter tractable to decide whether $\mu + k$ clauses can be satisfied (k is the parameter).

The consequences (Related work)

- For a *variable-matched* formula (autarky/Crown reduced $\mu = n$), it is FPT to decide whether $n + k$ clauses are satisfiable.

- *Minimal unsatisfiable formulae*
  - are in general hard to recognize (D^p-complete – Papadimitriou and Wolfe);
  - have no crowns, and hence are variable matched (and so one can satisfy at least n clauses)
  - can be recognized in polynomial time if they have $n + k$ clauses (asked by Kleine Buning, shown XP by Kullman, Fleischner and Szeider, FPT by Szeider)
  - This follows as a corollary of our result.

The consequences (Related work)

- For a *variable-matched* formula (autarky/Crown reduced $\mu = n$), it is FPT to decide whether $n + k$ clauses are satisfiable.
- *Minimal unsatisfiable formulae*
  - are in general hard to recognize (D^p-complete – Papadimitriou and Wolfe);
  - have no crowns, and hence are variable matched (and so one can satisfy at least n clauses)
  - can be recognized in polynomial time if they have $n + k$ clauses (asked by Kleine Buning, shown XP by Kullman, Fleischner and Szeider, FPT by Szeider)
  - This follows as a corollary of our result.

The consequences (Related work)

- For a *variable-matched* formula (autarky/Crown reduced $\mu = n$), it is FPT to decide whether $n + k$ clauses are satisfiable.
- *Minimal unsatisfiable formulae*
  - are in general hard to recognize (D^p-complete – Papadimitriou and Wolfe);
  - have no crowns, and hence are variable matched (and so one can satisfy at least n clauses)
  - can be recognized in polynomial time if they have $n + k$ clauses (asked by Kleine Buning, shown XP by Kullman, Fleischner and Szeider, FPT by Szeider)
  - This follows as a corollary of our result.

The consequences (Related work)

- For a *variable-matched* formula (autarky/Crown reduced $\mu = n$), it is FPT to decide whether $n + k$ clauses are satisfiable.
- *Minimal unsatisfiable formulae*
  - are in general hard to recognize (D^p-complete – Papadimitriou and Wolfe);
  - have no crowns, and hence are variable matched (and so one can satisfy at least n clauses)
  - can be recognized in polynomial time if they have $n + k$ clauses (asked by Kleine Buning, shown XP by Kullman, Fleischner and Szeider, FPT by Szeider)
  - This follows as a corollary of our result.

The consequences (Related work)

- For a *variable-matched* formula (autarky/Crown reduced $\mu = n$), it is FPT to decide whether $n + k$ clauses are satisfiable.
- *Minimal unsatisfiable formulae*
  - are in general hard to recognize (D^p-complete – Papadimitriou and Wolfe);
  - have no crowns, and hence are variable matched (and so one can satisfy at least n clauses)
  - can be recognized in polynomial time if they have $n + k$ clauses (asked by Kleine Buning, shown XP by Kullman, Fleischner and Szeider, FPT by Szeider)
  - This follows as a corollary of our result.

The consequences (Related work)

- For a *variable-matched* formula (autarky/Crown reduced $\mu = n$), it is FPT to decide whether $n + k$ clauses are satisfiable.
- *Minimal unsatisfiable formulae*
  - are in general hard to recognize (D^p-complete – Papadimitriou and Wolfe);
  - have no crowns, and hence are variable matched (and so one can satisfy at least n clauses)
  - can be recognized in polynomial time if they have $n + k$ clauses (asked by Kleine Buning, shown XP by Kullman, Fleischner and Szeider, FPT by Szeider)
  - This follows as a corollary of our result.

The Main steps of our algorithm

- Some preprocessing rules
- Branching rules
- Reduce to the $m-k$ hitting set problem (Given a family of m sets, hit all of them with a subset of at most $m-k$ elements; k is the parameter)

The Challenges/Main ideas/Contributions

- Carefully analyzing the drop in (*solnsize* − *matching size*);
- argue that they don't increase in preprocessing steps, and decrease in each branching step.
- Hence need stronger preprocessing rules.
- New (deterministic and improved randomized) algorithms for $(m-k)$ hitting set problem

The Challenges/Main ideas/Contributions

- Carefully analyzing the drop in $(solnsize - matching\ size)$;
- argue that they don't increase in preprocessing steps, and decrease in each branching step.
- Hence need stronger preprocessing rules.
- New (deterministic and improved randomized) algorithms for $(m - k)$ hitting set problem

The Challenges/Main ideas/Contributions

- Carefully analyzing the drop in ($solnsize - matching\ size$);
- argue that they don't increase in preprocessing steps, and decrease in each branching step.
- Hence need stronger preprocessing rules.
- New (deterministic and improved randomized) algorithms for $(m - k)$ hitting set problem

The Challenges/Main ideas/Contributions

- Carefully analyzing the drop in $(solnsize - matching\ size)$;
- argue that they don't increase in preprocessing steps, and decrease in each branching step.
- Hence need stronger preprocessing rules.
- New (deterministic and improved randomized) algorithms for $(m - k)$ hitting set problem

The Challenges/Main ideas/Contributions

- Carefully analyzing the drop in $(solnsize - matching\ size)$;
- argue that they don't increase in preprocessing steps, and decrease in each branching step.
- Hence need stronger preprocessing rules.
- New (deterministic and improved randomized) algorithms for $(m - k)$ hitting set problem

Simple preprocessing Rules

- P1. If a variable appears only in pure or only in negated form, set it appropriately.
- P2. (Resolution of variables with exactly two occurrences). If a variable x appears once positively and once negatively, then replace the two clauses with their union after deleting x (and $\bar{x}$).

$$(\mathbf{x} \vee y) \wedge (\bar{\mathbf{x}} \vee z)$$
$$\downarrow$$
$$(y \vee z)$$

At this point, every variable appears at least three times.

Simple preprocessing Rules

- P1. If a variable appears only in pure or only in negated form, set it appropriately.
- P2. (Resolution of variables with exactly two occurrences). If a variable x appears once positively and once negatively, then replace the two clauses with their union after deleting x (and $\bar{x}$).

$$(\mathbf{x} \vee y) \wedge (\bar{\mathbf{x}} \vee z)$$
$$\downarrow$$
$$(y \vee z)$$

At this point, every variable appears at least three times.

Simple preprocessing Rules

- P1. If a variable appears only in pure or only in negated form, set it appropriately.
- P2. (Resolution of variables with exactly two occurrences). If a variable x appears once positively and once negatively, then replace the two clauses with their union after deleting x (and $\bar{x}$).

$$(\mathbf{x} \vee y) \wedge (\bar{\mathbf{x}} \vee z)$$
$$\downarrow$$
$$(y \vee z)$$

At this point, every variable appears at least three times.

Simple preprocessing Rules

- P1. If a variable appears only in pure or only in negated form, set it appropriately.
- P2. (Resolution of variables with exactly two occurrences). If a variable x appears once positively and once negatively, then replace the two clauses with their union after deleting x (and $\bar{x}$).

$$(\mathbf{x} \vee y) \wedge (\bar{\mathbf{x}} \vee z)$$
$$\downarrow$$
$$(y \vee z)$$

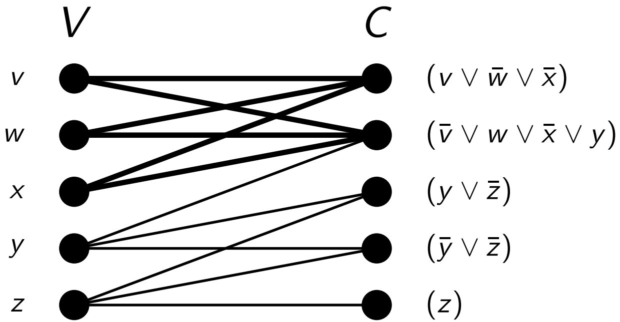
At this point, every variable appears at least three times.

Simple preprocessing Rules (Continued

- P3. If there is an autarky (crown), reduce. I.e. if there is a subset S of variables such that $|N(S)| \leq |S|$, remove those variables and the clauses containing them.

Simple preprocessing Rules (Continued

- P3. If there is an autarky (crown), reduce. I.e. if there is a subset S of variables such that $|N(S)| \leq |S|$, remove those variables and the clauses containing them.

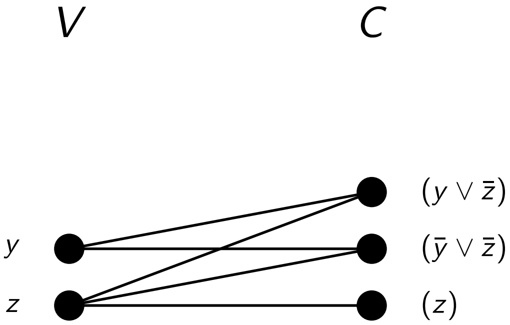


Simple preprocessing Rules

- P3. If there is an autarky (crown), reduce. I.e. if there is a subset S of variables such that $|N(S)| \leq |S|$, remove those variables and the clauses containing them.

Simple preprocessing Rules

- P3. If there is an autarky (crown), reduce. I.e. if there is a subset S of variables such that $|N(S)| \leq |S|$, remove those variables and the clauses containing them.

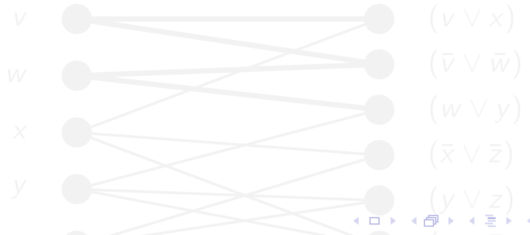


Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Test whether the clauses of $N(S)$ can be satisfied using variables in S alone.

(Such a subset can be found in polynomial time using matching theory, and sat can be checked in polynomial time using the algorithm for $n+1$ clauses satisfiability.)

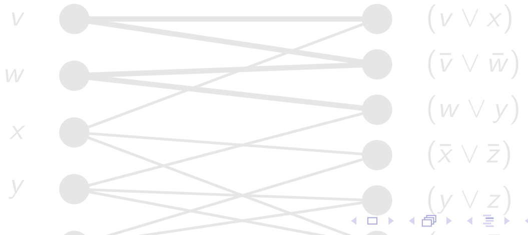
- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.



Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Test whether the clauses of $N(S)$ can be satisfied using variables in S alone. (Such a subset can be found in polynomial time using matching theory, and sat can be checked in polynomial time using the algorithm for $n + 1$ clauses satisfiability.)

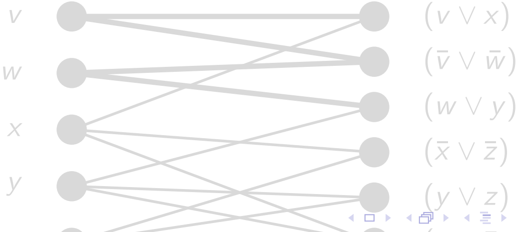
- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.



Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Test whether the clauses of $N(S)$ can be satisfied using variables in S alone. (Such a subset can be found in polynomial time using matching theory, and sat can be checked in polynomial time using the algorithm for $n+1$ clauses satisfiability.)

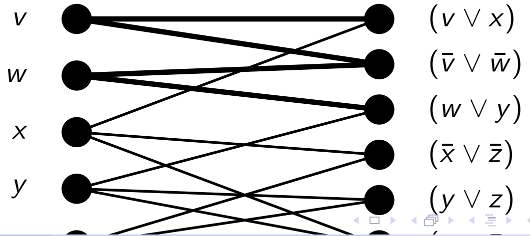
- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.



Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Test whether the clauses of $N(S)$ can be satisfied using variables in S alone. (Such a subset can be found in polynomial time using matching theory, and sat can be checked in polynomial time using the algorithm for $n+1$ clauses satisfiability.)

- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.



Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Check whether the clauses in $N(S)$ can be satisfied by variables in S alone.

- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.

At this point, for every subset S of variables $|N(S)| \geq |S| + 2$.

Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Check whether the clauses in $N(S)$ can be satisfied by variables in S alone.

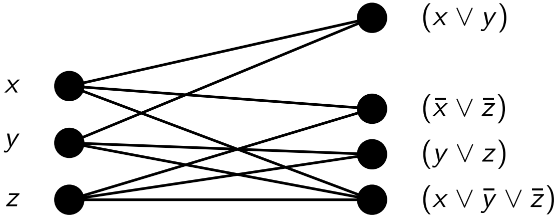
- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.

At this point, for every subset S of variables $|N(S)| \geq |S| + 2$.

Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Check whether the clauses in $N(S)$ can be satisfied by variables in S alone.

- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.

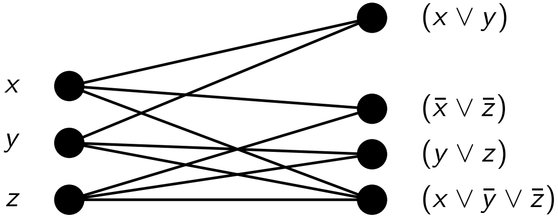


At this point, for every subset S of variables $|N(S)| \geq |S| + 2$.

Main preprocessing rule

P4. Let there be a subset S of variables such that $|N(S)| = |S| + 1$. Check whether the clauses in $N(S)$ can be satisfied by variables in S alone.

- If the clauses in $N(S)$ are satisfiable (with variables in S alone), then remove S and its neighbors.
- Else remove variables in S and clauses in $N(S)$ and add a clause that contains all variables in the clauses in $N(S)$ which are not in S.



At this point, for every subset S of variables $|N(S)| \geq |S| + 2$.

Branching Rules

- B1. If x and $\bar{x}$ appear at least twice in F, branch by setting $x = 1$ or $x = 0$.

  Key argument: While k, the number to be satisfied, drops by at least $n(x)$, the matching size drops by at most $n(x) - 1$ as the bipartite graph is 2-expanding (i.e. every subset has a buffer of size at least 2).

  Without loss of generality, let $n(x) = 1, n(\bar{x}) > 1$ for every variable x (Otherwise, rename by flipping).

- B2. If a clause contains two positive literals, then branch by setting each of them to 0

  As $n(x) = 1$, this clause is the only clause with positive occurrence for both the variables, so one of them can be false.

Branching Rules

- B1. If x and $\bar{x}$ appear at least twice in F, branch by setting $x = 1$ or $x = 0$.

  Key argument: While k, the number to be satisfied, drops by at least $n(x)$, the matching size drops by at most $n(x) - 1$ as the bipartite graph is 2-expanding (i.e. every subset has a buffer of size at least 2).

  Without loss of generality, let $n(x) = 1, n(\bar{x}) > 1$ for every variable x (Otherwise, rename by flipping).

- B2. If a clause contains two positive literals, then branch by setting each of them to 0

  As $n(x) = 1$, this clause is the only clause with positive occurrence for both the variables, so one of them can be false.

Branching Rules

- B1. If x and $\bar{x}$ appear at least twice in F, branch by setting $x = 1$ or $x = 0$.

  Key argument: While k, the number to be satisfied, drops by at least $n(x)$, the matching size drops by at most $n(x) - 1$ as the bipartite graph is 2-expanding (i.e. every subset has a buffer of size at least 2).

  Without loss of generality, let $n(x) = 1, n(\bar{x}) > 1$ for every variable x (Otherwise, rename by flipping).

- B2. If a clause contains two positive literals, then branch by setting each of them to 0

  As $n(x) = 1$, this clause is the only clause with positive occurrence for both the variables, so one of them can be false.

Branching Rules

- B1. If x and $\bar{x}$ appear at least twice in F, branch by setting $x = 1$ or $x = 0$.

  Key argument: While k, the number to be satisfied, drops by at least $n(x)$, the matching size drops by at most $n(x) - 1$ as the bipartite graph is 2-expanding (i.e. every subset has a buffer of size at least 2).

  Without loss of generality, let $n(x) = 1, n(\bar{x}) > 1$ for every variable x (Otherwise, rename by flipping).
- B2. If a clause contains two positive literals, then branch by setting each of them to 0

  As $n(x) = 1$, this clause is the only clause with positive occurrence for both the variables, so one of them can be false.

Branching Rules

- B1. If x and $\bar{x}$ appear at least twice in F, branch by setting $x = 1$ or $x = 0$.

  Key argument: While k, the number to be satisfied, drops by at least $n(x)$, the matching size drops by at most $n(x) - 1$ as the bipartite graph is 2-expanding (i.e. every subset has a buffer of size at least 2).

  Without loss of generality, let $n(x) = 1, n(\bar{x}) > 1$ for every variable x (Otherwise, rename by flipping).

- B2. If a clause contains two positive literals, then branch by setting each of them to 0

  As $n(x) = 1$, this clause is the only clause with positive occurrence for both the variables, so one of them can be false.

Where are we?

Every clause has at most one positive literal, and every variable has only one positive occurrence.

- If the clause containing x is (x, C), then replace it by (x) and by adding C to all clauses containing $\bar{x}$.

$$(x \vee C) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{x} \vee \bar{w})$$
$$\downarrow$$
$$(x) \wedge (\bar{x} \vee \bar{y} \vee C) \wedge (\bar{x} \vee \bar{z} \vee C) \wedge (\bar{x} \vee \bar{w} \vee C)$$

- Resulting formula has a unit clause for each variable, followed by all negated clauses.

Reduction to $m - k$ hitting set

Where are we?
Every clause has at most one positive literal, and every variable has only one positive occurrence.

- If the clause containing x is (x, C), then replace it by (x) and by adding C to all clauses containing $\bar{x}$.

$$(x \vee C) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{x} \vee \bar{w})$$
$$\downarrow$$
$$(x) \wedge (\bar{x} \vee \bar{y} \vee C) \wedge (\bar{x} \vee \bar{z} \vee C) \wedge (\bar{x} \vee \bar{w} \vee C)$$

- Resulting formula has a unit clause for each variable, followed by all negated clauses.

Reduction to $m - k$ hitting set

Where are we?
Every clause has at most one positive literal, and every variable has only one positive occurrence.

- If the clause containing x is (x, C), then replace it by (x) and by adding C to all clauses containing $\bar{x}$.

$$(x \vee C) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{x} \vee \bar{w})$$
$$\downarrow$$
$$(x) \wedge (\bar{x} \vee \bar{y} \vee C) \wedge (\bar{x} \vee \bar{z} \vee C) \wedge (\bar{x} \vee \bar{w} \vee C)$$

- Resulting formula has a unit clause for each variable, followed by all negated clauses.

Reduction to $m - k$ hitting set

Where are we?
Every clause has at most one positive literal, and every variable has only one positive occurrence.

- If the clause containing x is (x, C), then replace it by (x) and by adding C to all clauses containing $\bar{x}$.

$$(x \vee C) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{x} \vee \bar{w})$$
$$\downarrow$$
$$(x) \wedge (\bar{x} \vee \bar{y} \vee C) \wedge (\bar{x} \vee \bar{z} \vee C) \wedge (\bar{x} \vee \bar{w} \vee C)$$

- Resulting formula has a unit clause for each variable, followed by all negated clauses.

Reduction to $m - k$ hitting set

- Without loss of generality, any assignment maximizing the number of satisfied clauses satisfies all the non unit clauses
- In particular, the maxsat assignment is a minimum hitting set for the non unit clauses.
- To satisfy $n + k$ clauses of F, we need a hitting set of size at most $N - k$ for the N non unit clauses.

Reduction to $m - k$ hitting set

- Without loss of generality, any assignment maximizing the number of satisfied clauses satisfies all the non unit clauses
- In particular, the maxsat assignment is a minimum hitting set for the non unit clauses.
- To satisfy $n + k$ clauses of F, we need a hitting set of size at most $N - k$ for the N non unit clauses.

Reduction to $m - k$ hitting set

- Without loss of generality, any assignment maximizing the number of satisfied clauses satisfies all the non unit clauses
- In particular, the maxsat assignment is a minimum hitting set for the non unit clauses.
- To satisfy $n + k$ clauses of F, we need a hitting set of size at most $N - k$ for the N non unit clauses.

Reduction to $m - k$ hitting set

- Without loss of generality, any assignment maximizing the number of satisfied clauses satisfies all the non unit clauses
- In particular, the maxsat assignment is a minimum hitting set for the non unit clauses.
- To satisfy $n + k$ clauses of F, we need a hitting set of size at most $N - k$ for the N non unit clauses.

Summary of the Algorithm

- P1. Set variables appearing in one form (pure or negated)
- P2. Resolve variables appearing once positively and once negatively.
- P3. Reduce autarkies
- P4. Reduce subsets S of variables, where $|N(S)| = |S| + 1$.
- B1. If a variable appears at least twice positively and negatively, branch.
- B2. If a clause has two positive literals, branch by setting each of them to false.
- Reduce to $m - k$ hitting set problem after some transformation.

FPT algorithm for $(m - k)$ hitting set

- (Gutin et al): The family F has a $m - k$ hitting set if and only if there is some subset S of the universe such that $|S| \leq k$ and S 'hits' at least $|S| + k$ sets.
- Find such a subset (for each size p up to k), if exists, by color coding:
  - Color the sets randomly using $p + k$ colors
  - Find a subset of the universe of size p, if exists, that 'hits' all the colors using dynamic programming
- Derandomize using hash families

FPT algorithm for $(m-k)$ hitting set

- (Gutin et al): The family F has a $m-k$ hitting set if and only if there is some subset S of the universe such that $|S| \leq k$ and S 'hits' at least $|S|+k$ sets.
- Find such a subset (for each size p up to k), if exists, by color coding:
  - Color the sets randomly using $p+k$ colors
  - Find a subset of the universe of size p, if exists, that 'hits' all the colors using dynamic programming
- Derandomize using hash families

FPT algorithm for $(m-k)$ hitting set

- (Gutin et al): The family F has a $m-k$ hitting set if and only if there is some subset S of the universe such that $|S| \leq k$ and S 'hits' at least $|S|+k$ sets.
- Find such a subset (for each size p up to k), if exists, by color coding:
  - Color the sets randomly using $p+k$ colors
  - Find a subset of the universe of size p, if exists, that 'hits' all the colors using dynamic programming
- Derandomize using hash families

FPT algorithm for $(m - k)$ hitting set

- (Gutin et al): The family F has a $m - k$ hitting set if and only if there is some subset S of the universe such that $|S| \leq k$ and S 'hits' at least $|S| + k$ sets.
- Find such a subset (for each size p up to k), if exists, by color coding:
  - Color the sets randomly using $p + k$ colors
  - Find a subset of the universe of size p, if exists, that 'hits' all the colors using dynamic programming
- Derandomize using hash families

Conclusions

- Showed FPT algorithm for deciding whether a SAT formula has an assignment satisfying at least $\mu + k$ clauses.
- We also show that the problem has no polynomial in k kernel (under complexity theoretic conditions). By a parameter preserving reduction from $m - k$-hitting set (which has been already shown to have no polynomial kernel under the same conditions)
- OPEN: Polynomial kernel for bounded CNF formulas?

Conclusions

- Showed FPT algorithm for deciding whether a SAT formula has an assignment satisfying at least $\mu + k$ clauses.
- We also show that the problem has no polynomial in k kernel (under complexity theoretic conditions). By a parameter preserving reduction from $m - k$-hitting set (which has been already shown to have no polynomial kernel under the same conditions)
- OPEN: Polynomial kernel for bounded CNF formulas?

Conclusions

- Showed FPT algorithm for deciding whether a SAT formula has an assignment satisfying at least $\mu + k$ clauses.
- We also show that the problem has no polynomial in k kernel (under complexity theoretic conditions). By a parameter preserving reduction from $m - k$-hitting set (which has been already shown to have no polynomial kernel under the same conditions)
- OPEN: Polynomial kernel for bounded CNF formulas?

Conclusions

- Showed FPT algorithm for deciding whether a SAT formula has an assignment satisfying at least $\mu + k$ clauses.
- We also show that the problem has no polynomial in k kernel (under complexity theoretic conditions). By a parameter preserving reduction from $m - k$-hitting set (which has been already shown to have no polynomial kernel under the same conditions)
- OPEN: Polynomial kernel for bounded CNF formulas?

Thank You