Fixed Parameter Tractability Above Lower Bounds

Anders Yeo

AndersYeo@gmail.com
Department of Mathematics
University of Johannesburg, S.A.

December 14, 2012

This talk

I will cover the following topics in this talk.

- Motivation for FPT, using an example from Information Security.
- Fixed Parameter Tractability (FPT) and Kernels.
- The problem Satisfying Linear Equations Over $\mathbb{F}_2$.
- The problem MAX-r-SAT.
- Conclusion.

This talk

I will cover the following topics in this talk.

- Motivation for FPT, using an example from Information Security.
- Fixed Parameter Tractability (FPT) and Kernels.
- The problem Satisfying Linear Equations Over $\mathbb{F}_2$.
- The problem MAX-r-SAT.
- Conclusion.

This talk

I will cover the following topics in this talk.

- Motivation for FPT, using an example from Information Security.
- Fixed Parameter Tractability (FPT) and Kernels.
- The problem Satisfying Linear Equations Over $\mathbb{F}_2$.
- The problem MAX-r-SAT.
- Conclusion.

This talk

I will cover the following topics in this talk.

- Motivation for FPT, using an example from Information Security.
- Fixed Parameter Tractability (FPT) and Kernels.
- The problem Satisfying Linear Equations Over $\mathbb{F}_2$.
- The problem MAX-r-SAT.
- Conclusion.

This talk

I will cover the following topics in this talk.

- Motivation for FPT, using an example from Information Security.
- Fixed Parameter Tractability (FPT) and Kernels.
- The problem Satisfying Linear Equations Over $\mathbb{F}_2$.
- The problem MAX-r-SAT.
- Conclusion.

This talk

I will cover the following topics in this talk.

- Motivation for FPT, using an example from Information Security.
- Fixed Parameter Tractability (FPT) and Kernels.
- The problem Satisfying Linear Equations Over $\mathbb{F}_2$.
- The problem MAX-r-SAT.
- Conclusion.

Motivation for FPT from IS

We consider the following problem from Information Security, which we assume your boss wants you to solve!

We are given a set of users, U, say u_1, u_2, u_3, u_4 and a set of steps, S, such as the following.

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Motivation for FPT from IS

We consider the following problem from Information Security, which we assume your boss wants you to solve!

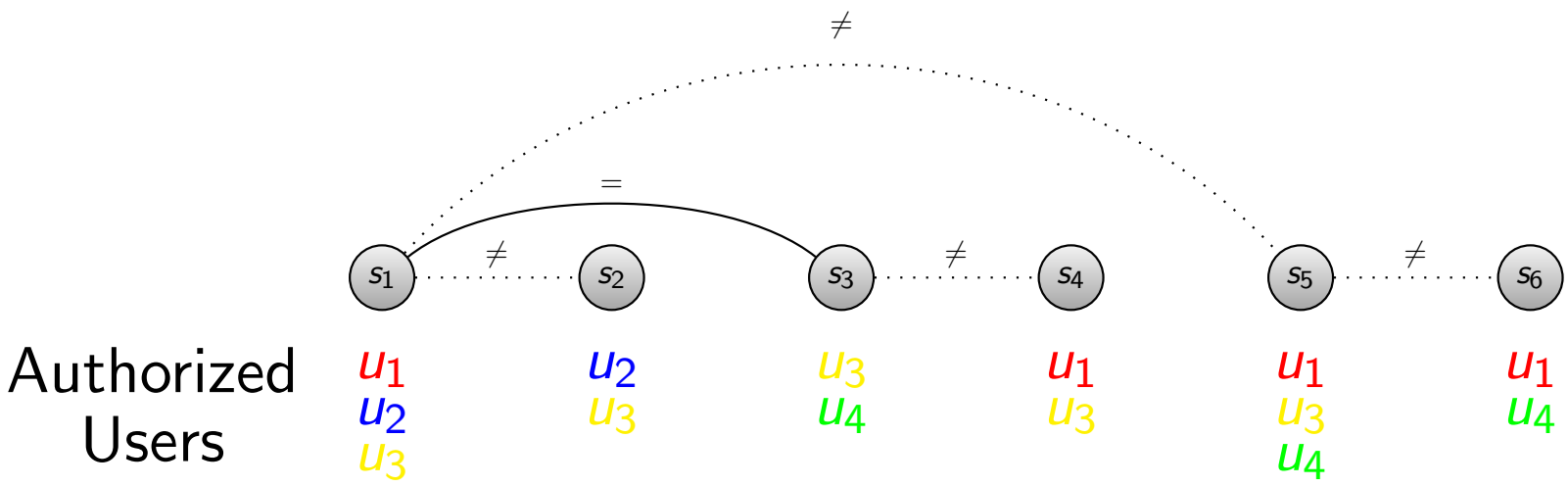
We are given a set of users, U, say u_1, u_2, u_3, u_4 and a set of steps, S, such as the following.

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Motivation for FPT from IS

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Now consider the following problem.

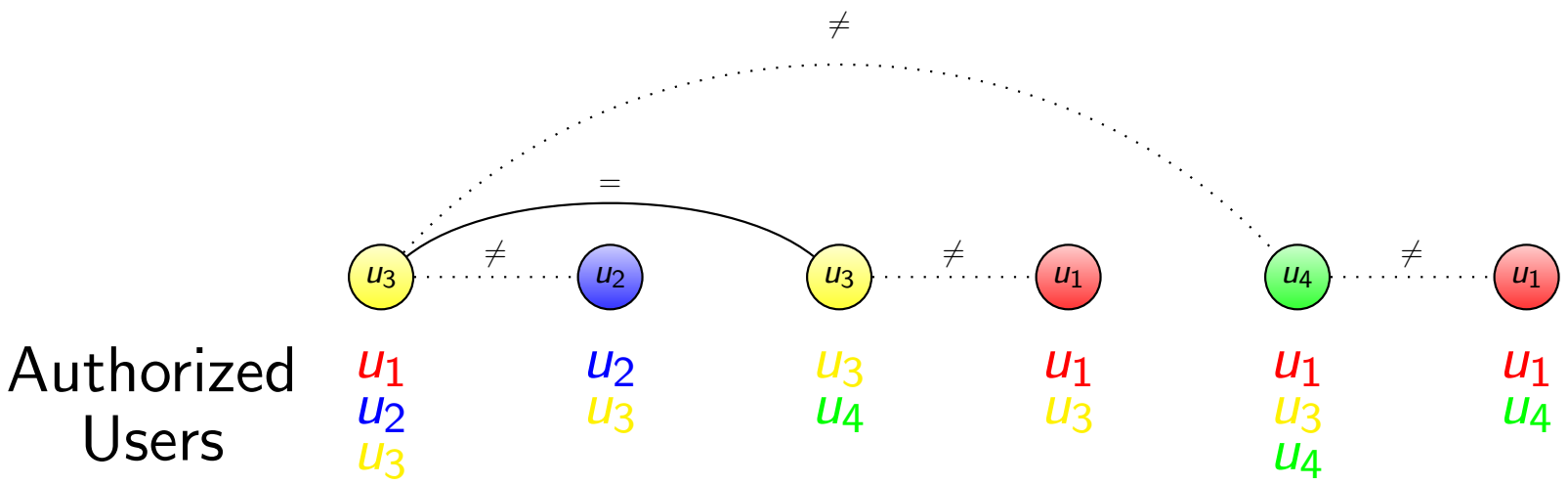


Is there a solution? Yes

Motivation for FPT from IS

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Now consider the following problem.



Is there a solution? Yes

Motivation for FPT from IS

What do you say when your boss asks you to write a program that solves this kind of problem (you may have $n = 1000$ users and $k = 6$ steps)?

Option 1: It is an NP-hard problem, so cannot be solved?
But there is only 6 steps!! You are fired!!

Option 2: No problem boss. The number of steps is small and the problem is FPT, so I can do it!?
You get to keep your job!

FPT means there is an algorithm of complexity $O(f(k) \times n^c)$.

Motivation for FPT from IS

What do you say when your boss asks you to write a program that solves this kind of problem (you may have $n = 1000$ users and $k = 6$ steps)?

Option 1: It is an NP-hard problem, so cannot be solved?
But there is only 6 steps!! You are fired!!

Option 2: No problem boss. The number of steps is small and the problem is FPT, so I can do it!?
You get to keep your job!

FPT means there is an algorithm of complexity $O(f(k) \times n^c)$.

Motivation for FPT from IS

What do you say when your boss asks you to write a program that solves this kind of problem (you may have $n = 1000$ users and $k = 6$ steps)?

Option 1: It is an NP-hard problem, so cannot be solved?
But there is only 6 steps!! You are fired!!

Option 2: No problem boss. The number of steps is small and the problem is FPT, so I can do it!?
You get to keep your job!

FPT means there is an algorithm of complexity $O(f(k) \times n^c)$.

Motivation for FPT from IS

What do you say when your boss asks you to write a program that solves this kind of problem (you may have $n = 1000$ users and $k = 6$ steps)?

Option 1: It is an NP-hard problem, so cannot be solved?
But there is only 6 steps!! You are fired!!

Option 2: No problem boss. The number of steps is small and the problem is FPT, so I can do it!?
You get to keep your job!

FPT means there is an algorithm of complexity $O(f(k) \times n^c)$.

Motivation for FPT from IS

What do you say when your boss asks you to write a program that solves this kind of problem (you may have $n = 1000$ users and $k = 6$ steps)?

Option 1: It is an NP-hard problem, so cannot be solved?
But there is only 6 steps!! You are fired!!

Option 2: No problem boss. The number of steps is small and the problem is FPT, so I can do it!?
You get to keep your job!

FPT means there is an algorithm of complexity $O(f(k) \times n^c)$.

Motivation for FPT from IS

What do you say when your boss asks you to write a program that solves this kind of problem (you may have $n = 1000$ users and $k = 6$ steps)?

Option 1: It is an NP-hard problem, so cannot be solved?
But there is only 6 steps!! You are fired!!

Option 2: No problem boss. The number of steps is small and the problem is FPT, so I can do it!?
You get to keep your job!

FPT means there is an algorithm of complexity $O(f(k) \times n^c)$.

An introduction to FPT

So if somebody comes to you with a real-world NP-hard problem..... What do you do? Give up?

No!! The person coming to us would not be happy!!

In the theory of FPT we look for parameters that are small for the real world problems.

And therefore exponential growth in these parameters may be acceptable.

An introduction to FPT

So if somebody comes to you with a real-world NP-hard problem..... What do you do? Give up?

No!! The person coming to us would not be happy!!

In the theory of FPT we look for parameters that are small for the real world problems.

And therefore exponential growth in these parameters may be acceptable.

An introduction to FPT

So if somebody comes to you with a real-world NP-hard problem..... What do you do? Give up?

No!! The person coming to us would not be happy!!

In the theory of FPT we look for parameters that are small for the real world problems.

And therefore exponential growth in these parameters may be acceptable.

An introduction to FPT

So if somebody comes to you with a real-world NP-hard problem..... What do you do? Give up?

No!! The person coming to us would not be happy!!

In the theory of FPT we look for parameters that are small for the real world problems.

And therefore exponential growth in these parameters may be acceptable.

An introduction to FPT

So if somebody comes to you with a real-world NP-hard problem..... What do you do? Give up?

No!! The person coming to us would not be happy!!

In the theory of FPT we look for parameters that are small for the real world problems.

And therefore exponential growth in these parameters may be acceptable.

An introduction to FPT

Once we have identified this (hopefully) small parameter, k, we try to decide if there is an algorithm of complexity $O(f(k) \times n^c)$ (or $O(f(k) + n^c)$), where $f(k)$ is any function and c is a constant.

If there is, then we have a good algorithm as c is 'normally' small and we have limited the (inevitable) exponential growth to k, which is small.

For example, $n = 1000$ and $k = 10$....

- $O(n^k)$ is completely infeasible (10^{30} steps).
- $O(2^k n)$ is fine. (1.024.000 steps).

An introduction to FPT

Once we have identified this (hopefully) small parameter, k, we try to decide if there is an algorithm of complexity $O(f(k) \times n^c)$ (or $O(f(k) + n^c)$), where $f(k)$ is any function and c is a constant.

If there is, then we have a good algorithm as c is 'normally' small and we have limited the (inevitable) exponential growth to k, which is small.

For example, $n = 1000$ and $k = 10$....

- $O(n^k)$ is completely infeasible (10^{30} steps).
- $O(2^k n)$ is fine. (1.024.000 steps).

An introduction to FPT

Once we have identified this (hopefully) small parameter, k, we try to decide if there is an algorithm of complexity $O(f(k) \times n^c)$ (or $O(f(k) + n^c)$), where $f(k)$ is any function and c is a constant.

If there is, then we have a good algorithm as c is 'normally' small and we have limited the (inevitable) exponential growth to k, which is small.

For example, $n = 1000$ and $k = 10$....

- $O(n^k)$ is completely infeasible (10^{30} steps).
- $O(2^k n)$ is fine. (1.024.000 steps).

An introduction to FPT

Theorem 1 (Kernels/Preprocessing): A problem is FPT, if and only if we in polynomial time can reduce the problem to one of size $g(k)$ for some function g.

There are many problems which have been shown to be $W[1]$-hard or $W[2]$-hard which gives very strong evidence that they are not FPT.

There is also a large amount of theory about when a problem has a polynomial kernel (i.e. $g(k)$ is a polynomial in k).

An introduction to FPT

Theorem 1 (Kernels/Preprocessing): A problem is FPT, if and only if we in polynomial time can reduce the problem to one of size $g(k)$ for some function g.

There are many problems which have been shown to be $W[1]$-hard or $W[2]$-hard which gives very strong evidence that they are not FPT.

There is also a large amount of theory about when a problem has a polynomial kernel (i.e. $g(k)$ is a polynomial in k).

An introduction to FPT

Theorem 1 (Kernels/Preprocessing): A problem is FPT, if and only if we in polynomial time can reduce the problem to one of size $g(k)$ for some function g.

There are many problems which have been shown to be $W[1]$-hard or $W[2]$-hard which gives very strong evidence that they are not FPT.

There is also a large amount of theory about when a problem has a polynomial kernel (i.e. $g(k)$ is a polynomial in k).

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

- MAX-LIN problem: given a system $\mathcal{I}$ of m linear equations over n variables over $\mathbb{F}_2$.
- $\mathbb{F}_2$ is the Galois field with 2 elements ($1+1=0$).
- Each equation is assigned a positive integer weight.
- We wish to find an assignment of values to the variables in order to maximize the total weight of satisfied equations.

$z_1 = 1$	Weight 1
$z_1 + z_2 = 0$	Weight 2
$z_2 + z_3 = 1$	Weight 1
$z_1 + z_2 + z_3 = 1$	Weight 3

- **Known bound:** We can satisfy clauses of weight at least $W/2$, where W is the total weight of equations. Why? Because this is the average over all solutions

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

- MAX-LIN problem: given a system $\mathcal{I}$ of m linear equations over n variables over $\mathbb{F}_2$.
- $\mathbb{F}_2$ is the Galois field with 2 elements ($1+1=0$).
- Each equation is assigned a positive integer weight.
- We wish to find an assignment of values to the variables in order to maximize the total weight of satisfied equations.

$z_1 = 1$	Weight 1
$z_1 + z_2 = 0$	Weight 2
$z_2 + z_3 = 1$	Weight 1
$z_1 + z_2 + z_3 = 1$	Weight 3

- **Known bound:** We can satisfy clauses of weight at least $W/2$, where W is the total weight of equations. Why? Because this is the average over all solutions

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

- MAX-LIN problem: given a system $\mathcal{I}$ of m linear equations over n variables over $\mathbb{F}_2$.
- $\mathbb{F}_2$ is the Galois field with 2 elements ($1+1=0$).
- Each equation is assigned a positive integer weight.
- We wish to find an assignment of values to the variables in order to maximize the total weight of satisfied equations.

$$\begin{array}{ll} z_1 = 1 & \text{Weight 1} \\ z_1 + z_2 = 0 & \text{Weight 2} \\ z_2 + z_3 = 1 & \text{Weight 1} \\ z_1 + z_2 + z_3 = 1 & \text{Weight 3} \end{array}$$

- **Known bound:** We can satisfy clauses of weight at least $W/2$, where W is the total weight of equations. Why? Because this is the average over all solutions

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

- MAX-LIN problem: given a system $\mathcal{I}$ of m linear equations over n variables over $\mathbb{F}_2$.
- $\mathbb{F}_2$ is the Galois field with 2 elements ($1 + 1 = 0$).
- Each equation is assigned a positive integer weight.
- We wish to find an assignment of values to the variables in order to maximize the total weight of satisfied equations.

$z_1 = 1$	Weight 1
$z_1 + z_2 = 0$	Weight 2
$z_2 + z_3 = 1$	Weight 1
$z_1 + z_2 + z_3 = 1$	Weight 3

- **Known bound:** We can satisfy clauses of weight at least $W/2$, where W is the total weight of equations. Why? Because this is the average over all solutions

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

- MAX-LIN problem: given a system $\mathcal{I}$ of m linear equations over n variables over $\mathbb{F}_2$.
- $\mathbb{F}_2$ is the Galois field with 2 elements $(1+1=0)$.
- Each equation is assigned a positive integer weight.
- We wish to find an assignment of values to the variables in order to maximize the total weight of satisfied equations.

$z_1 = 1$	Weight 1
$z_1 + z_2 = 0$	Weight 2
$z_2 + z_3 = 1$	Weight 1
$z_1 + z_2 + z_3 = 1$	Weight 3

- **Known bound:** We can satisfy clauses of weight at least $W/2$, where W is the total weight of equations. Why?
  Because this is the average over all solutions

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

- MAX-LIN problem: given a system $\mathcal{I}$ of m linear equations over n variables over $\mathbb{F}_2$.
- $\mathbb{F}_2$ is the Galois field with 2 elements $(1+1=0)$.
- Each equation is assigned a positive integer weight.
- We wish to find an assignment of values to the variables in order to maximize the total weight of satisfied equations.

$z_1 = 1$	Weight 1
$z_1 + z_2 = 0$	Weight 2
$z_2 + z_3 = 1$	Weight 1
$z_1 + z_2 + z_3 = 1$	Weight 3

- **Known bound:** We can satisfy clauses of weight at least $W/2$, where W is the total weight of equations. Why? Because this is the average over all solutions

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

MAX-r-LIN is equivalent to MAX-LIN except all equations have at most r variables.

MAX-LIN and MAX-r-LIN are important, difficult, problems.

How do we attack it?

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

MAX-r-LIN is equivalent to MAX-LIN except all equations have at most r variables.

MAX-LIN and MAX-r-LIN are important, difficult, problems.

How do we attack it?

Satisfying Linear Equations Over $\mathbb{F}_2$, Definition

MAX-r-LIN is equivalent to MAX-LIN except all equations have at most r variables.

MAX-LIN and MAX-r-LIN are important, difficult, problems.

How do we attack it?

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

Is MAX-LIN FPT, if the weight of a solution is the parameter?

Yes. If the number of equations, m, is more than $2k$ the answer is YES.

If $m < 2k$ then for all of the $2^m < 2^{2k} = 4^k$ subsets of equations we can check if we can satisfy all and then pick the best result found (in $O(4^k \times pol(n, m))$ time).

So we are done? No. We did not pick the right parameter!

We want the parameter to be small, which it never is (in non-trivial instances), due to the lower bound $W/2$.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

Is MAX-LIN FPT, if the weight of a solution is the parameter?

Yes. If the number of equations, m, is more than $2k$ the answer is YES.

If $m < 2k$ then for all of the $2^m < 2^{2k} = 4^k$ subsets of equations we can check if we can satisfy all and then pick the best result found (in $O(4^k \times pol(n, m))$ time).

So we are done? No. We did not pick the right parameter!

We want the parameter to be small, which it never is (in non-trivial instances), due to the lower bound $W/2$.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

Is MAX-LIN FPT, if the weight of a solution is the parameter?

Yes. If the number of equations, m, is more than $2k$ the answer is YES.

If $m < 2k$ then for all of the $2^m < 2^{2k} = 4^k$ subsets of equations we can check if we can satisfy all and then pick the best result found (in $O(4^k \times pol(n, m))$ time).

So we are done? No. We did not pick the right parameter!

We want the parameter to be small, which it never is (in non-trivial instances), due to the lower bound $W/2$.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

Is MAX-LIN FPT, if the weight of a solution is the parameter?

Yes. If the number of equations, m, is more than $2k$ the answer is YES.

If $m < 2k$ then for all of the $2^m < 2^{2k} = 4^k$ subsets of equations we can check if we can satisfy all and then pick the best result found (in $O(4^k \times pol(n, m))$ time).

So we are done? No. We did not pick the right parameter!

We want the parameter to be small, which it never is (in non-trivial instances), due to the lower bound $W/2$.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

Is MAX-LIN FPT, if the weight of a solution is the parameter?

Yes. If the number of equations, m, is more than $2k$ the answer is YES.

If $m < 2k$ then for all of the $2^m < 2^{2k} = 4^k$ subsets of equations we can check if we can satisfy all and then pick the best result found (in $O(4^k \times pol(n, m))$ time).

So we are done? No. We did not pick the right parameter!

We want the parameter to be small, which it never is (in non-trivial instances), due to the lower bound $W/2$.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

Is MAX-LIN FPT, if the weight of a solution is the parameter?

Yes. If the number of equations, m, is more than $2k$ the answer is YES.

If $m < 2k$ then for all of the $2^m < 2^{2k} = 4^k$ subsets of equations we can check if we can satisfy all and then pick the best result found (in $O(4^k \times pol(n, m))$ time).

So we are done? No. We did not pick the right parameter!

We want the parameter to be small, which it never is (in non-trivial instances), due to the lower bound $W/2$.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

So we consider the parameter k, where we want to decide if $\max(\mathcal{I}) \geq W/2 + k$. Is this interesting for small k?

Yes. There exists (large) examples where the answer is NO even for very small k.

$$z_1 = 0$$
$$z_1 = 1$$
$$z_2 + z_3 = 0$$
$$z_2 + z_3 = 1$$
$$\ldots$$

- Mahajan, Raman & Sikdar (2006) asked if this problem is FPT.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

So we consider the parameter k, where we want to decide if $\max(\mathcal{I}) \geq W/2 + k$. Is this interesting for small k?

Yes. There exists (large) examples where the answer is NO even for very small k.

$$z_1 = 0$$
$$z_1 = 1$$
$$z_2 + z_3 = 0$$
$$z_2 + z_3 = 1$$
$$\ldots$$

- Mahajan, Raman & Sikdar (2006) asked if this problem is FPT.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

So we consider the parameter k, where we want to decide if $\max(\mathcal{I}) \geq W/2 + k$. Is this interesting for small k?

Yes. There exists (large) examples where the answer is NO even for very small k.

$$z_1 = 0$$
$$z_1 = 1$$
$$z_2 + z_3 = 0$$
$$z_2 + z_3 = 1$$
$$\ldots$$

- Mahajan, Raman & Sikdar (2006) asked if this problem is FPT.

Satisfying Linear Equations Over $\mathbb{F}_2$, Parameterization

So we consider the parameter k, where we want to decide if $\max(\mathcal{I}) \geq W/2 + k$. Is this interesting for small k?

Yes. There exists (large) examples where the answer is NO even for very small k.

$$z_1 = 0$$
$$z_1 = 1$$
$$z_2 + z_3 = 0$$
$$z_2 + z_3 = 1$$
$$\ldots$$

- Mahajan, Raman & Sikdar (2006) asked if this problem is FPT.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Reduction Rule (LHS rule)

Suppose we have two equations, $\sum_{i \in S} z_i = b_1$ (weight w_1) and $\sum_{i \in S} z_i = b_2$ (weight w_2), where $w_1 \geq w_2$.

If $b_1 = b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 + w_2$).
If $b_1 \neq b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 - w_2$).

$$z_1 + z_2 = 1 \quad (w = 2) \quad \Rightarrow \quad z_1 + z_2 = 1 \quad (w = 3)$$
$$z_1 + z_2 = 1 \quad (w = 1)$$

$$z_2 + z_3 + z_4 = 0 \quad (w = 3) \quad \Rightarrow \quad z_2 + z_3 + z_4 = 0 \quad (w = 1)$$
$$z_2 + z_3 + z_4 = 1 \quad (w = 2)$$

- This allows us to assume no two equations have the same left-hand side.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Reduction Rule (LHS rule)

Suppose we have two equations, $\sum_{i \in S} z_i = b_1$ (weight w_1) and $\sum_{i \in S} z_i = b_2$ (weight w_2), where $w_1 \geq w_2$.

If $b_1 = b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 + w_2$).
If $b_1 \neq b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 - w_2$).

$$z_1 + z_2 = 1 \quad (w = 2) \quad \Rightarrow \quad z_1 + z_2 = 1 \quad (w = 3)$$
$$z_1 + z_2 = 1 \quad (w = 1)$$

$$z_2 + z_3 + z_4 = 0 \quad (w = 3) \quad \Rightarrow \quad z_2 + z_3 + z_4 = 0 \quad (w = 1)$$
$$z_2 + z_3 + z_4 = 1 \quad (w = 2)$$

- This allows us to assume no two equations have the same left-hand side.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Reduction Rule (LHS rule)

Suppose we have two equations, $\sum_{i \in S} z_i = b_1$ (weight w_1) and $\sum_{i \in S} z_i = b_2$ (weight w_2), where $w_1 \geq w_2$.

If $b_1 = b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 + w_2$).
If $b_1 \neq b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 - w_2$).

$$z_1 + z_2 = 1 \quad (w = 2) \quad \Rightarrow \quad z_1 + z_2 = 1 \quad (w = 3)$$
$$z_1 + z_2 = 1 \quad (w = 1)$$

$$z_2 + z_3 + z_4 = 0 \quad (w = 3) \quad \Rightarrow \quad z_2 + z_3 + z_4 = 0 \quad (w = 1)$$
$$z_2 + z_3 + z_4 = 1 \quad (w = 2)$$

- This allows us to assume no two equations have the same left-hand side.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Reduction Rule (LHS rule)

Suppose we have two equations, $\sum_{i \in S} z_i = b_1$ (weight w_1) and $\sum_{i \in S} z_i = b_2$ (weight w_2), where $w_1 \geq w_2$.

If $b_1 = b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 + w_2$).
If $b_1 \neq b_2$, replace with one equation $\sum_{i \in S} z_i = b_1$ (weight $w_1 - w_2$).

$$z_1 + z_2 = 1 \quad (w = 2) \quad \Rightarrow \quad z_1 + z_2 = 1 \quad (w = 3)$$
$$z_1 + z_2 = 1 \quad (w = 1)$$

$$z_2 + z_3 + z_4 = 0 \quad (w = 3) \quad \Rightarrow \quad z_2 + z_3 + z_4 = 0 \quad (w = 1)$$
$$z_2 + z_3 + z_4 = 1 \quad (w = 2)$$

- This allows us to assume no two equations have the same left-hand side.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Reduction Rule (Rank rule)

Let A be the matrix over $\mathbb{F}_2$ corresponding to the set of equations in $\mathcal{I}$, such that $a_{ji} = 1$ if z_i appears in equation j, and 0 otherwise. Let $t = \mathrm{rank}A$ and suppose columns $a^{i_1}, \ldots, a^{i_t}$ of A are linearly independent. Then delete all variables not in $\{z_{i_1}, \ldots, z_{i_t}\}$ from the equations of S.

$$\begin{aligned} z_1 + z_3 + z_4 &= 1 \\ z_2 + z_3 + z_4 &= 0 \\ z_2 + z_3 &= 0 \\ z_1 + z_2 &= 1 \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad \Rightarrow \quad \begin{aligned} z_1 + z_4 &= 1 \\ z_2 + z_4 &= 0 \\ z_2 &= 0 \\ z_1 + z_2 &= 1 \end{aligned}$$

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Reduction Rule (Rank rule)

Let A be the matrix over $\mathbb{F}_2$ corresponding to the set of equations in $\mathcal{I}$, such that $a_{ji} = 1$ if z_i appears in equation j, and 0 otherwise. Let $t = \mathrm{rank}A$ and suppose columns $a^{i_1}, \ldots, a^{i_t}$ of A are linearly independent. Then delete all variables not in $\{z_{i_1}, \ldots, z_{i_t}\}$ from the equations of S.

$$\begin{aligned} z_1 + z_3 + z_4 &= 1 \\ z_2 + z_3 + z_4 &= 0 \\ z_2 + z_3 &= 0 \\ z_1 + z_2 &= 1 \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} \mathbf{1} & \mathbf{0} & 1 & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & 1 & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & 1 & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & 0 & \mathbf{0} \end{pmatrix} \quad \Rightarrow \quad \begin{aligned} z_1 + z_4 &= 1 \\ z_2 + z_4 &= 0 \\ z_2 &= 0 \\ z_1 + z_2 &= 1 \end{aligned}$$

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Reduction Rule (Rank rule)

Let A be the matrix over $\mathbb{F}_2$ corresponding to the set of equations in $\mathcal{I}$, such that $a_{ji} = 1$ if z_i appears in equation j, and 0 otherwise. Let $t = \mathrm{rank}A$ and suppose columns $a^{i_1}, \ldots, a^{i_t}$ of A are linearly independent. Then delete all variables not in $\{z_{i_1}, \ldots, z_{i_t}\}$ from the equations of S.

$$\begin{aligned} z_1 + z_3 + z_4 &= 1 \\ z_2 + z_3 + z_4 &= 0 \\ z_2 + z_3 &= 0 \\ z_1 + z_2 &= 1 \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} \mathbf{1} & \mathbf{0} & 1 & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & 1 & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & 1 & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & 0 & \mathbf{0} \end{pmatrix} \quad \Rightarrow \quad \begin{aligned} z_1 + z_4 &= 1 \\ z_2 + z_4 &= 0 \\ z_2 &= 0 \\ z_1 + z_2 &= 1 \end{aligned}$$

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Why does the Rank Rule work?

$$\mathcal{I}: \begin{aligned} z_1 + z_3 + z_4 &= 1 \\ z_2 + z_3 + z_4 &= 0 \\ z_2 + z_3 &= 0 \\ z_1 + z_2 &= 1 \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad \Rightarrow \quad \mathcal{I}': \begin{aligned} z_1 + z_4 &= 1 \\ z_2 + z_4 &= 0 \\ z_2 &= 0 \\ z_1 + z_2 &= 1 \end{aligned}$$

- Set $z_3 = 0$ and add a solution for $\mathcal{I}'$ to get a solution of equal weight for $\mathcal{I}$.
- Consider a solution for $\mathcal{I}$.
  If $z_3 = 1$, then change the values of z_1, z_2, z_3 to get an equivalent solution with $z_3 = 0$. Why does this work?
  So $z_3 = 0$, and we have a solution for $\mathcal{I}'$ of equal weight.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Why does the Rank Rule work?

$$\mathcal{I} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{I}'$$

$$\begin{array}{l} z_1 + z_3 + z_4 = 1 \\ z_2 + z_3 + z_4 = 0 \\ z_2 + z_3 = 0 \\ z_1 + z_2 = 1 \end{array} \quad \Rightarrow \quad \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad \Rightarrow \quad \begin{array}{l} z_1 + z_4 = 1 \\ z_2 + z_4 = 0 \\ z_2 = 0 \\ z_1 + z_2 = 1 \end{array}$$

- Set $z_3 = 0$ and add a solution for $\mathcal{I}'$ to get a solution of equal weight for $\mathcal{I}$.
- Consider a solution for $\mathcal{I}$.
  If $z_3 = 1$, then change the values of z_1, z_2, z_3 to get an equivalent solution with $z_3 = 0$. Why does this work?
  So $z_3 = 0$, and we have a solution for $\mathcal{I}'$ of equal weight.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Why does the Rank Rule work?

$$\mathcal{I}: \begin{array}{l} z_1 + z_3 + z_4 = 1 \\ z_2 + z_3 + z_4 = 0 \\ z_2 + z_3 = 0 \\ z_1 + z_2 = 1 \end{array} \Rightarrow \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \Rightarrow \mathcal{I}': \begin{array}{l} z_1 + z_4 = 1 \\ z_2 + z_4 = 0 \\ z_2 = 0 \\ z_1 + z_2 = 1 \end{array}$$

- Set $z_3 = 0$ and add a solution for $\mathcal{I}'$ to get a solution of equal weight for $\mathcal{I}$.
- Consider a solution for $\mathcal{I}$.
  If $z_3 = 1$, then change the values of z_1, z_2, z_3 to get an equivalent solution with $z_3 = 0$. Why does this work?
  So $z_3 = 0$, and we have a solution for $\mathcal{I}'$ of equal weight.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Why does the Rank Rule work?

$$\mathcal{I} \qquad\qquad \mathcal{I}'$$

$$\begin{aligned} z_1 + z_3 + z_4 &= 1 \\ z_2 + z_3 + z_4 &= 0 \\ z_2 + z_3 &= 0 \\ z_1 + z_2 &= 1 \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad \Rightarrow \quad \begin{aligned} z_1 + z_4 &= 1 \\ z_2 + z_4 &= 0 \\ z_2 &= 0 \\ z_1 + z_2 &= 1 \end{aligned}$$

- Set $z_3 = 0$ and add a solution for $\mathcal{I}'$ to get a solution of equal weight for $\mathcal{I}$.
- Consider a solution for $\mathcal{I}$.
  If $z_3 = 1$, then change the values of z_1, z_2, z_3 to get an equivalent solution with $z_3 = 0$. Why does this work?
  So $z_3 = 0$, and we have a solution for $\mathcal{I}'$ of equal weight.

Satisfying Linear Equations Over $\mathbb{F}_2$, Reduction Rules

Why does the Rank Rule work?

$$\mathcal{I} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{I}'$$

$$\begin{array}{l} z_1 + z_3 + z_4 = 1 \\ z_2 + z_3 + z_4 = 0 \\ z_2 + z_3 = 0 \\ z_1 + z_2 = 1 \end{array} \Rightarrow \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \Rightarrow \begin{array}{l} z_1 + z_4 = 1 \\ z_2 + z_4 = 0 \\ z_2 = 0 \\ z_1 + z_2 = 1 \end{array}$$

- Set $z_3 = 0$ and add a solution for $\mathcal{I}'$ to get a solution of equal weight for $\mathcal{I}$.
- Consider a solution for $\mathcal{I}$.
  If $z_3 = 1$, then change the values of z_1, z_2, z_3 to get an equivalent solution with $z_3 = 0$. Why does this work?
  So $z_3 = 0$, and we have a solution for $\mathcal{I}'$ of equal weight.

Satisfying Linear Equations Over $\mathbb{F}_2$, Main Results

- Theorem A: The problem can be solved in time $O^*(n^{2k})$. The proof uses a branch-technique and a way of reducing the problem in each step.
- Theorem B: If the problem is reduced and $2k \leq m < 2^{n/2k}$, then it is a YES-instance.

The above results can be combined to show the following

Theorem (Main)

The problem is fixed-parameter tractable, and has a kernel with $O(k^2 \log k)$ variables.

So the problem is FPT (First use poly-time then brute-force on kernel).

Satisfying Linear Equations Over $\mathbb{F}_2$, Main Results

- Theorem A: The problem can be solved in time $O^*(n^{2k})$. The proof uses a branch-technique and a way of reducing the problem in each step.
- Theorem B: If the problem is reduced and $2k \leq m < 2^{n/2k}$, then it is a YES-instance.

The above results can be combined to show the following

Theorem (Main)

The problem is fixed-parameter tractable, and has a kernel with $O(k^2 \log k)$ variables.

So the problem is FPT (First use poly-time then brute-force on kernel).

Satisfying Linear Equations Over $\mathbb{F}_2$, Main Results

- Theorem A: The problem can be solved in time $O^*(n^{2k})$. The proof uses a branch-technique and a way of reducing the problem in each step.
- Theorem B: If the problem is reduced and $2k \leq m < 2^{n/2k}$, then it is a YES-instance.

The above results can be combined to show the following

Theorem (Main)

The problem is fixed-parameter tractable, and has a kernel with $O(k^2 \log k)$ variables.

So the problem is FPT (First use poly-time then brute-force on kernel).

Proof of Theorem A

Theorem A There exists an $O^*(n^{2k})$-time algorithm for Max-Lin-AA.

- **Proof (sketch):** Let $e_1, \ldots e_n$ be a set of equations in $\mathcal{I}$ which are 'independent'.
  (LHSs correspond to independent rows in matrix A.)
- Check unique assignment in which $e_1, \ldots e_n$ all false. If this assignment achieves excess $2k$, return Yes.
- Otherwise, one of $e_1, \ldots e_n$ must be true.
- Branch n ways. In branch i assume equation e_i is true and reduce.
- Since we can stop after $2k$ iterations of $\mathcal{H}$, search tree has n^{2k} leaves.

Proof of Theorem A

Theorem A There exists an $O^*(n^{2k})$-time algorithm for MAX-LIN-AA.

- **Proof (sketch):** Let $e_1, \ldots e_n$ be a set of equations in $\mathcal{I}$ which are 'independent'.
  (LHSs correspond to independent rows in matrix A.)
- Check unique assignment in which $e_1, \ldots e_n$ all false. If this assignment achieves excess $2k$, return YES.
- Otherwise, one of $e_1, \ldots e_n$ must be true.
- Branch n ways. In branch i assume equation e_i is true and reduce.
- Since we can stop after $2k$ iterations of $\mathcal{H}$, search tree has n^{2k} leaves.

Proof of Theorem A

Theorem A There exists an $O^*(n^{2k})$-time algorithm for Max-Lin-AA.

- **Proof (sketch):** Let $e_1, \ldots e_n$ be a set of equations in $\mathcal{I}$ which are 'independent'.
  (LHSs correspond to independent rows in matrix A.)
- Check unique assignment in which $e_1, \ldots e_n$ all false. If this assignment achieves excess $2k$, return Yes.
- Otherwise, one of $e_1, \ldots e_n$ must be true.
- Branch n ways. In branch i assume equation e_i is true and reduce.
- Since we can stop after $2k$ iterations of $\mathcal{H}$, search tree has n^{2k} leaves.

Proof of Theorem A

Theorem A There exists an $O^*(n^{2k})$-time algorithm for Max-Lin-AA.

- **Proof (sketch):** Let $e_1, \ldots e_n$ be a set of equations in $\mathcal{I}$ which are 'independent'.
  (LHSs correspond to independent rows in matrix A.)
- Check unique assignment in which $e_1, \ldots e_n$ all false. If this assignment achieves excess $2k$, return Yes.
- Otherwise, one of $e_1, \ldots e_n$ must be true.
- Branch n ways. In branch i assume equation e_i is true and reduce.
- Since we can stop after $2k$ iterations of $\mathcal{H}$, search tree has n^{2k} leaves.

Proof of Theorem A

Theorem A There exists an $O^*(n^{2k})$-time algorithm for Max-Lin-AA.

- **Proof (sketch):** Let $e_1, \ldots e_n$ be a set of equations in $\mathcal{I}$ which are 'independent'.
  (LHSs correspond to independent rows in matrix A.)
- Check unique assignment in which $e_1, \ldots e_n$ all false. If this assignment achieves excess $2k$, return Yes.
- Otherwise, one of $e_1, \ldots e_n$ must be true.
- Branch n ways. In branch i assume equation e_i is true and reduce.
- Since we can stop after $2k$ iterations of $\mathcal{H}$, search tree has n^{2k} leaves.

On Theorem B

Theorem B: If the problem is reduced and $2k \leq m < 2^{n/2k}$, then it is a YES-instance.

This is the most complicated part.....

The proof uses the notion of a M-free set, S, of vectors (ie $S \subseteq M$ and no two of more vectors in S sum up to a vector in M).

And an algorithm,.....

On Theorem B

Theorem B: If the problem is reduced and $2k \leq m < 2^{n/2k}$, then it is a YES-instance.

This is the most complicated part.....

The proof uses the notion of a M-free set, S, of vectors (ie $S \subseteq M$ and no two of more vectors in S sum up to a vector in M).

And an algorithm,.....

On Theorem B

Theorem B: If the problem is reduced and $2k \leq m < 2^{n/2k}$, then it is a YES-instance.

This is the most complicated part.....

The proof uses the notion of a M-free set, S, of vectors (ie $S \subseteq M$ and no two of more vectors in S sum up to a vector in M).

And an algorithm,.....

Proof of our main result

Theorem

MAX-LIN-AA *has a kernel with at most* $O(k^2 \log k)$ *variables.*

- **Proof:** Let $\mathcal{I}$ be a reduced system.
- Case 1: $m \geq n^{2k}$. Then using $O^*(n^{2k})$ algorithm, can solve in polynomial time.
- Case 2: $2k \leq m \leq 2^{n/2k}$. By earlier Theorem return YES.
- Case 3: $m < 2k$. Since $\mathcal{I}$ reduced by Rank Rule, $n \leq m$ so $n = O(k^2 \log k)$.
- Only remaining case is $2^{n/2k} < m < n^{2k}$.

Proof of our main result

Theorem

MAX-LIN-AA *has a kernel with at most* $O(k^2 \log k)$ *variables.*

- **Proof:** Let $\mathcal{I}$ be a reduced system.
- Case 1: $m \geq n^{2k}$. Then using $O^*(n^{2k})$ algorithm, can solve in polynomial time.
- Case 2: $2k \leq m \leq 2^{n/2k}$. By earlier Theorem return YES.
- Case 3: $m < 2k$. Since $\mathcal{I}$ reduced by Rank Rule, $n \leq m$ so $n = O(k^2 \log k)$.
- Only remaining case is $2^{n/2k} < m < n^{2k}$.

Proof of our main result

Theorem

MAX-LIN-AA *has a kernel with at most* $O(k^2 \log k)$ *variables.*

- **Proof:** Let $\mathcal{I}$ be a reduced system.
- Case 1: $m \geq n^{2k}$. Then using $O^*(n^{2k})$ algorithm, can solve in polynomial time.
- Case 2: $2k \leq m \leq 2^{n/2k}$. By earlier Theorem return YES.
- Case 3: $m < 2k$. Since $\mathcal{I}$ reduced by Rank Rule, $n \leq m$ so $n = O(k^2 \log k)$.
- Only remaining case is $2^{n/2k} < m < n^{2k}$.

Proof of our main result

Theorem

MAX-LIN-AA *has a kernel with at most* $O(k^2 \log k)$ *variables.*

- **Proof:** Let $\mathcal{I}$ be a reduced system.
- Case 1: $m \geq n^{2k}$. Then using $O^*(n^{2k})$ algorithm, can solve in polynomial time.
- Case 2: $2k \leq m \leq 2^{n/2k}$. By earlier Theorem return YES.
- Case 3: $m < 2k$. Since $\mathcal{I}$ reduced by Rank Rule, $n \leq m$ so $n = O(k^2 \log k)$.
- Only remaining case is $2^{n/2k} < m < n^{2k}$.

Proof of our main result

Theorem

MAX-LIN-AA *has a kernel with at most* $O(k^2 \log k)$ *variables.*

- **Proof:** Let $\mathcal{I}$ be a reduced system.
- Case 1: $m \geq n^{2k}$. Then using $O^*(n^{2k})$ algorithm, can solve in polynomial time.
- Case 2: $2k \leq m \leq 2^{n/2k}$. By earlier Theorem return YES.
- Case 3: $m < 2k$. Since $\mathcal{I}$ reduced by Rank Rule, $n \leq m$ so $n = O(k^2 \log k)$.
- Only remaining case is $2^{n/2k} < m < n^{2k}$.

Proof of our main result

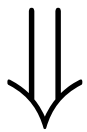
Theorem

MAX-LIN-AA *has a kernel with at most* $O(k^2 \log k)$ *variables.*

- **Proof:** Let $\mathcal{I}$ be a reduced system.
- Case 1: $m \geq n^{2k}$. Then using $O^*(n^{2k})$ algorithm, can solve in polynomial time.
- Case 2: $2k \leq m \leq 2^{n/2k}$. By earlier Theorem return YES.
- Case 3: $m < 2k$. Since $\mathcal{I}$ reduced by Rank Rule, $n \leq m$ so $n = O(k^2 \log k)$.
- Only remaining case is $2^{n/2k} < m < n^{2k}$.

Proof of our main result (continued)

- Suppose $2^{n/2k} < m < n^{2k}$. Then $n/2k < 2k \log n$.

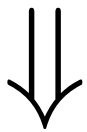


- So $n = O(k^2 \log k)$.

.... and this complete the part on Max-Lin.

Proof of our main result (continued)

- Suppose $2^{n/2k} < m < n^{2k}$. Then $n/2k < 2k \log n$.



- So $n = O(k^2 \log k)$.

.... and this complete the part on Max-Lin.

Proof of our main result (continued)

- Suppose $2^{n/2k} < m < n^{2k}$. Then $n/2k < 2k \log n$.

  - So $n < 4k^2 \log n$.
  - In order to bound $\log n$ we note that $\sqrt{n} < n/\log n < 4k^2$.
  - Therefore $n < (2k)^4$ and $\log n < 4 \log(2k)$
  - So $n < 4k^2 \log n < 16k^2(\log k + 1)$

- So $n = O(k^2 \log k)$.

.... and this complete the part on Max-Lin.

Proof of our main result (continued)

- Suppose $2^{n/2k} < m < n^{2k}$. Then $n/2k < 2k \log n$.

  - So $n < 4k^2 \log n$.
  - In order to bound $\log n$ we note that $\sqrt{n} < n/\log n < 4k^2$.
  - Therefore $n < (2k)^4$ and $\log n < 4 \log(2k)$
  - So $n < 4k^2 \log n < 16k^2(\log k + 1)$

- So $n = O(k^2 \log k)$.

.... and this complete the part on Max-Lin.

Max-r-Sat, definition

MAX-r-SAT parameterized above average (MAX-r-SAT-AA)

Instance: A CNF formula F with n variables, m clauses, such that each clause has r variables.

Parameter: k.

Question: Can we satisfy $\geq (1-1/2^r)m + k$ clauses?

$$(v_1 \vee \overline{v_3} \vee v_4) \wedge (\overline{v_2} \vee \overline{v_4} \vee v_5) \wedge \cdots \wedge (v_5 \vee v_6 \vee v_7)$$

- $(1-1/2^r)m$ is the expected number of clauses satisfied by a random assignment.
- **Pseudo-boolean function:** a function $f : \{-1,+1\}^n \to \mathbb{R}$

Max-r-Sat, definition

MAX-r-SAT parameterized above average (MAX-r-SAT-AA)

Instance: A CNF formula F with n variables, m clauses, such that each clause has r variables.

Parameter: k.

Question: Can we satisfy $\geq (1 - 1/2^r)m + k$ clauses?

$$(v_1 \vee \overline{v_3} \vee v_4) \wedge (\overline{v_2} \vee \overline{v_4} \vee v_5) \wedge \cdots \wedge (v_5 \vee v_6 \vee v_7)$$

- $(1 - 1/2^r)m$ is the expected number of clauses satisfied by a random assignment.
- **Pseudo-boolean function:** a function $f : \{-1, +1\}^n \to \mathbb{R}$

Max-r-Sat, definition

MAX-r-SAT parameterized above average (MAX-r-SAT-AA)

Instance: A CNF formula F with n variables, m clauses, such that each clause has r variables.

Parameter: k.

Question: Can we satisfy $\geq (1-1/2^r)m + k$ clauses?

$$(v_1 \vee \overline{v_3} \vee v_4) \wedge (\overline{v_2} \vee \overline{v_4} \vee v_5) \wedge \cdots \wedge (v_5 \vee v_6 \vee v_7)$$

- $(1-1/2^r)m$ is the expected number of clauses satisfied by a random assignment.
- **Pseudo-boolean function:** a function $f : \{-1,+1\}^n \to \mathbb{R}$

Max-r-Sat, definition

MAX-r-SAT parameterized above average (MAX-r-SAT-AA)

Instance: A CNF formula F with n variables, m clauses, such that each clause has r variables.

Parameter: k.

Question: Can we satisfy $\geq (1-1/2^r)m + k$ clauses?

$$(v_1 \vee \overline{v_3} \vee v_4) \wedge (\overline{v_2} \vee \overline{v_4} \vee v_5) \wedge \cdots \wedge (v_5 \vee v_6 \vee v_7)$$

- $(1-1/2^r)m$ is the expected number of clauses satisfied by a random assignment.
- **Pseudo-boolean function:** a function $f : \{-1,+1\}^n \to \mathbb{R}$

Max-r-Sat, Main Idea

- Write the Max-r-Sat as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of Max-Lin-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of Max-r-Lin-AA.
- Max-r-Lin-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve Max-r-Sat-AA in time $O^*(2^{(2^r k-1)r})$

Max-r-Sat, Main Idea

- Write the Max-r-Sat as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of Max-Lin-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of Max-r-Lin-AA.
- Max-r-Lin-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve Max-r-Sat-AA in time $O^*(2^{(2^r k-1)r})$

Max-r-Sat, Main Idea

- Write the MAX-r-SAT as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of MAX-LIN-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of MAX-r-LIN-AA.
- MAX-r-LIN-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve MAX-r-SAT-AA in time $O^*(2^{(2^r k-1)r})$

Max-r-Sat, Main Idea

- Write the Max-r-Sat as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of Max-Lin-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of Max-r-Lin-AA.
- Max-r-Lin-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve Max-r-Sat-AA in time $O^*(2^{(2^r k-1)r})$

Max-r-Sat, Main Idea

- Write the Max-r-Sat as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of Max-Lin-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of Max-r-Lin-AA.
- Max-r-Lin-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve Max-r-Sat-AA in time $O^*(2^{(2^r k-1)r})$

Max-r-Sat, Main Idea

- Write the MAX-r-SAT as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of MAX-LIN-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of MAX-r-LIN-AA.
- MAX-r-LIN-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve MAX-r-SAT-AA in time $O^*(2^{(2^r k-1)r})$

Max-r-Sat, creating f

- We can represent MAX-r-SAT-AA as a pseudo-boolean function, f, as follows.

Clause $(v_1 \vee \overline{v_3} \vee v_4)$ is equivalent to the following formula:

$$1 - (1 - x_1)(1 + x_3)(1 - x_4)$$

This is equal to 1 if the clause is true and otherwise it is -7, where $v_i = TRUE \Leftrightarrow x_i = 1$. and $v_i = FALSE \Leftrightarrow x_i = -1$.

$$\begin{aligned} f(x_1, \ldots, x_n) &= \text{true clause} - 7(m - (\text{true clauses})) \\ &= 8[\text{true clause} - 7m/8] \end{aligned}$$

Max-r-Sat, creating f

- We can represent MAX-r-SAT-AA as a pseudo-boolean function, f, as follows.

Clause $(v_1 \vee \overline{v_3} \vee v_4)$ is equivalent to the following formula:

$$1 - (1 - x_1)(1 + x_3)(1 - x_4)$$

This is equal to 1 if the clause is true and otherwise it is -7, where

$v_i = \mathit{TRUE} \Leftrightarrow x_i = 1.$ and $v_i = \mathit{FALSE} \Leftrightarrow x_i = -1.$

$$\begin{aligned} f(x_1, \ldots, x_n) &= \text{true clause} - 7(m - (\text{true clauses})) \\ &= 8[\text{true clause} - 7m/8] \end{aligned}$$

Max-r-Sat, creating f

- We can represent MAX-r-SAT-AA as a pseudo-boolean function, f, as follows.

Clause $(v_1 \vee \overline{v_3} \vee v_4)$ is equivalent to the following formula:

$$1 - (1 - x_1)(1 + x_3)(1 - x_4)$$

This is equal to 1 if the clause is true and otherwise it is -7, where

$v_i = TRUE \Leftrightarrow x_i = 1.$ and $v_i = FALSE \Leftrightarrow x_i = -1.$

$$\begin{aligned} f(x_1, \ldots, x_n) &= \text{true clause} - 7(m - (\text{true clauses})) \\ &= 8[\text{true clause} - 7m/8] \end{aligned}$$

Max-r-Sat, Recall the main idea

- Write the MAX-r-SAT as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of MAX-LIN-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of MAX-r-LIN-AA.
- MAX-r-LIN-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve MAX-r-SAT-AA in time $O^*(2^{(2^r k-1)r})$

Max-r-Sat, transform f into $\mathcal{I}$

- Suppose we know the Fourier expansion of $f(x)$

$$f(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$$

- For every $\emptyset \neq S \subseteq [n]$ with $c_S \neq 0$, construct equation $\sum_{i \in S} z_i = b_S$ with weight $|c_S|$, where $b_S = 0$ if c_S is positive and $b_S = 1$ if c_S is negative.

$$5x_1 - 3x_2x_3 + x_1x_2x_3 \quad \Rightarrow \quad \begin{array}{ll} z_1 = 0 & (w = 5) \\ z_2 + z_3 = 1 & (w = 3) \\ z_1 + z_2 + z_3 = 0 & (w = 1) \end{array}$$

- Let $z_i = 0$ if $x_i = 1$ and $z_i = 1$ if $x_i = -1$.
- $f(x) =$ weight of positive terms $-$ weight of negative terms $=$ weight of satisfied equations $-$ weight of falsified equations

Max-r-Sat, transform f into $\mathcal{I}$

- Suppose we know the Fourier expansion of $f(x)$

$$f(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$$

- For every $\emptyset \neq S \subseteq [n]$ with $c_S \neq 0$, construct equation $\sum_{i \in S} z_i = b_S$ with weight $|c_S|$, where $b_S = 0$ if c_S is positive and $b_S = 1$ if c_S is negative.

$$5x_1 - 3x_2x_3 + x_1x_2x_3 \quad \Rightarrow \quad \begin{array}{ll} z_1 = 0 & (w = 5) \\ z_2 + z_3 = 1 & (w = 3) \\ z_1 + z_2 + z_3 = 0 & (w = 1) \end{array}$$

- Let $z_i = 0$ if $x_i = 1$ and $z_i = 1$ if $x_i = -1$.
- $f(x)$ = weight of positive terms − weight of negative terms = weight of satisfied equations − weight of falsified equations

Max-r-Sat, transform f into $\mathcal{I}$

- Suppose we know the Fourier expansion of $f(x)$

$$f(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$$

- For every $\emptyset \neq S \subseteq [n]$ with $c_S \neq 0$, construct equation $\sum_{i \in S} z_i = b_S$ with weight $|c_S|$, where $b_S = 0$ if c_S is positive and $b_S = 1$ if c_S is negative.

$$5x_1 - 3x_2x_3 + x_1x_2x_3 \quad \Rightarrow \quad \begin{array}{ll} z_1 = 0 & (w = 5) \\ z_2 + z_3 = 1 & (w = 3) \\ z_1 + z_2 + z_3 = 0 & (w = 1) \end{array}$$

- Let $z_i = 0$ if $x_i = 1$ and $z_i = 1$ if $x_i = -1$.
- $f(x) =$ weight of positive terms $-$ weight of negative terms $=$ weight of satisfied equations $-$ weight of falsified equations

Max-r-Sat, transform f into $\mathcal{I}$

- Suppose we know the Fourier expansion of $f(x)$

$$f(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$$

- For every $\emptyset \neq S \subseteq [n]$ with $c_S \neq 0$, construct equation $\sum_{i \in S} z_i = b_S$ with weight $|c_S|$, where $b_S = 0$ if c_S is positive and $b_S = 1$ if c_S is negative.

$$5x_1 - 3x_2x_3 + x_1x_2x_3 \quad \Rightarrow \quad \begin{array}{ll} z_1 = 0 & (w = 5) \\ z_2 + z_3 = 1 & (w = 3) \\ z_1 + z_2 + z_3 = 0 & (w = 1) \end{array}$$

- Let $z_i = 0$ if $x_i = 1$ and $z_i = 1$ if $x_i = -1$.
- $f(x)$ = weight of positive terms − weight of negative terms = weight of satisfied equations − weight of falsified equations

Max-r-Sat, Recall the main idea

- Write the MAX-r-SAT as a pseudo-boolean function, f, such that $f/2^r$ gives us the number of satisfied clauses minus the average.
- Algebraically simplify f
- Transform f into an equivalent instance $\mathcal{I}$ of MAX-LIN-AA in time $O^*(2^r)$ with required excess $k' = 2^r k$.
- f is of degree r.
- Therefore $\mathcal{I}$ is an instance of MAX-r-LIN-AA.
- MAX-r-LIN-AA has a kernel with $(k'-1)r$ variables
  $\Rightarrow$ we can solve MAX-r-SAT-AA in time $O^*(2^{(2^r k-1)r})$

Conclusion

FPT is a very useful tool for attacking NP-hard problems.

It can give us a theoretical explanation why some problems are easier or more difficult than others.

It can also explain why sometimes heuristics work very well in practice on theoretically hard problems.

Conclusion

FPT is a very useful tool for attacking NP-hard problems.

It can give us a theoretical explanation why some problems are easier or more difficult than others.

It can also explain why sometimes heuristics work very well in practice on theoretically hard problems.

Conclusion

FPT is a very useful tool for attacking NP-hard problems.

It can give us a theoretical explanation why some problems are easier or more difficult than others.

It can also explain why sometimes heuristics work very well in practice on theoretically hard problems.

The End.

The End!

Thank you.

And thank you to the organizers for a great conference.

Any Questions?