Integer Linear-Exponential Programming

Alessio Mansutti

IMDEA Software Institute

SC^2 Workshop 2025

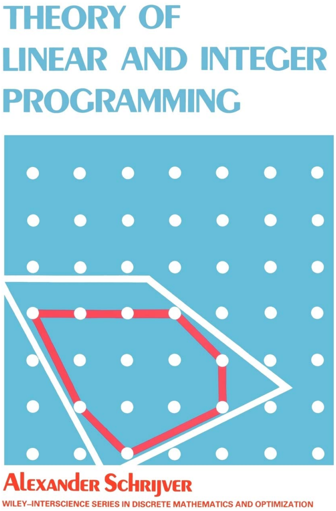
Based on an **ICALP'24** paper with Dmitry Chistikov and Mikhail Starchak, and a **preprint** with S Hitarth and Guru Shabadi

Integer Linear Programming (ILP)

$$\text{maximize} \quad a_d \cdot x_d + \cdots + a_1 \cdot x_1 + a_0$$

$$\text{subject to} \quad \begin{bmatrix} * & \cdots & * \\ \vdots & \ddots & \vdots \\ * & \cdots & * \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \leq \begin{bmatrix} * \\ \vdots \\ * \end{bmatrix}$$

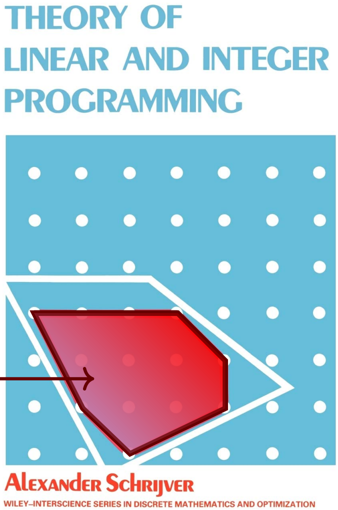
$$x_1, \ldots, x_d \in \mathbb{N}$$



Integer Linear Programming (ILP)

$$\begin{aligned} \text{maximize} \quad & a_d \cdot x_d + \cdots + a_1 \cdot x_1 + a_0 \\ \text{subject to} \quad & \begin{bmatrix} * & \cdots & * \\ \vdots & \ddots & \vdots \\ * & \cdots & * \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \leq \begin{bmatrix} * \\ \vdots \\ * \end{bmatrix} \\ & x_1, \ldots, x_d \in \mathbb{N} \end{aligned}$$

maximize $x + y$ subject to $\left.\begin{aligned} y &\leq 3 \\ x + y &\leq 6 \\ x &\leq 4 \\ x - 2 \cdot y &\leq 2 \\ -x - y &\leq -2 \\ -2 \cdot x - y &\leq -3 \end{aligned}\right\}$



Integer Linear-Exponential Programming (ILEP)

Linear-exponential term: $a_0 + \sum_{i=1}^{d} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{d} c_{i,j} \cdot (x_i \bmod 2^{x_j}) \right)$

Integer Linear-Exponential Programming (ILEP)

Linear-exponential term: $a_0 + \sum_{i=1}^{d} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{d} c_{i,j} \cdot (x_i \bmod 2^{x_j}) \right)$

Integer linear-exponential programming:

$$
\begin{aligned}
\text{maximize} \quad & \tau(\boldsymbol{x}) \\
\text{subject to} \quad & \rho_i(\boldsymbol{x}) \leq 0 \quad \text{for } i \in [1..n] \\
& \boldsymbol{x} \in \mathbb{N}^d
\end{aligned}
$$

where $\tau, \rho_1, \dots, \rho_n$ are linear-exponential terms.

Integer Linear-Exponential Programming (ILEP)

Linear-exponential term: $a_0 + \sum_{i=1}^{d} \Big(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{d} c_{i,j} \cdot (x_i \bmod 2^{x_j}) \Big)$

Integer linear-exponential programming:

$$
\begin{aligned}
\text{maximize} \quad & \tau(\boldsymbol{x}) \\
\text{subject to} \quad & \rho_i(\boldsymbol{x}) \leq 0 \quad \text{for } i \in [1..n] \\
& \boldsymbol{x} \in \mathbb{N}^d
\end{aligned}
$$

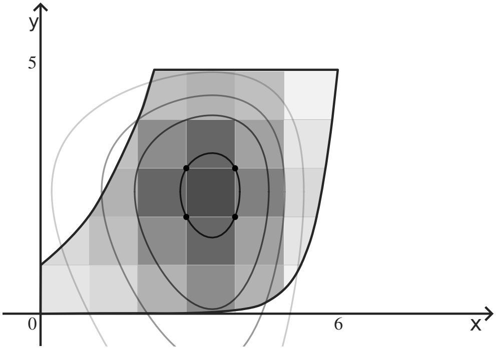
where $\tau, \rho_1, \dots, \rho_n$ are linear-exponential terms.

Question: Is there an algorithm to find an (optimal) solution?

An example

$$\text{maximize } \tau(x,y) := 8x + 4y - (2^x + 2^y)$$

$$\text{subject to } \varphi(x,y,z) := \begin{aligned} & y \leq 5 \\ & y \leq 2^x \\ & 2^z \leq 2^{16} y \\ & z = 3 \cdot x \end{aligned}$$



Setting (x, y) to any point in $\{3,4\} \times \{2,3\}$ yields an optimal solution.

Some stuff ILEP can express

x is of bit length y: $\quad 2^y \leq x < 2 \cdot 2^y$

Some stuff ILEP can express

x is of bit length y: $2^y \le x < 2 \cdot 2^y$

x is a tower of 2s of height n: $\exists x_1, \ldots, x_n : x = 2^{x_1} \wedge \left(\bigwedge_{i=1}^{n-1} x_i = 2^{x_{i+1}} \right) \wedge x_n = 1$

Some stuff ILEP can express

x is of bit length y: $2^y \leq x < 2 \cdot 2^y$

x is a tower of 2s of height n: $\exists x_1, \ldots, x_n : x = 2^{x_1} \wedge \left(\bigwedge_{i=1}^{n-1} x_i = 2^{x_{i+1}} \right) \wedge x_n = 1$

the yth digit of x is 1: $\exists z : z = y + 1 \wedge (x \bmod 2^z) - (x \bmod 2^y) \geq 1$

Some stuff ILEP can express

x is of bit length y: $2^y \leq x < 2 \cdot 2^y$

x is a tower of 2s of height n: $\exists x_1, \dots, x_n : x = 2^{x_1} \wedge \left(\bigwedge_{i=1}^{n-1} x_i = 2^{x_{i+1}} \right) \wedge x_n = 1$

the yth digit of x is 1: $\exists z : z = y + 1 \wedge (x \bmod 2^z) - (x \bmod 2^y) \geq 1$

	(msd)		y				(lsd)
x:	*	*	*	*	*	*	*

Some stuff ILEP can express

x is of bit length y: $2^y \leq x < 2 \cdot 2^y$

x is a tower of 2s of height n: $\exists x_1, \ldots, x_n : x = 2^{x_1} \wedge \left(\bigwedge_{i=1}^{n-1} x_i = 2^{x_{i+1}} \right) \wedge x_n = 1$

the yth digit of x is 1: $\exists z : z = y + 1 \wedge (x \bmod 2^z) - (x \bmod 2^y) \geq 1$

	(msd)		y				(lsd)
x:	0	0	*	*	*	*	*

$x \bmod 2^{y+1}$

Some stuff ILEP can express

x is of bit length y: $2^y \leq x < 2 \cdot 2^y$

x is a tower of 2s of height n: $\exists x_1, \ldots, x_n : x = 2^{x_1} \wedge \left(\bigwedge_{i=1}^{n-1} x_i = 2^{x_{i+1}}\right) \wedge x_n = 1$

the yth digit of x is 1: $\exists z : z = y + 1 \wedge (x \bmod 2^z) - (x \bmod 2^y) \geq 1$

	(msd)		y				(lsd)
x:	0	0	$*$	0	0	0	0

$$(x \bmod 2^{y+1}) - (x \bmod 2^y)$$

Some stuff ILEP can express

x is of bit length y: $2^y \le x < 2 \cdot 2^y$

x is a tower of 2s of height n: $\exists x_1, \dots, x_n : x = 2^{x_1} \wedge \left(\bigwedge_{i=1}^{n-1} x_i = 2^{x_{i+1}} \right) \wedge x_n = 1$

the yth digit of x is 1: $\exists z : z = y + 1 \wedge (x \bmod 2^z) - (x \bmod 2^y) \ge 1$

In relation with regular languages

Can express all regular languages of polynomial growth (Haase and Różycki, '21):

$$v_0 w_1^* v_1 w_2^* v_2 \dots v_{k-1} w_k^* v_k \qquad \text{where all } v_i, w_i \text{ are in } \{0,1\}^*$$

Can express non-regular languages (see all prior examples)

Cannot express all regular languages (Starchak, '24): ILEP cannot express $\{01, 10\}^*$

A comparison between ILP and ILEP

When solutions are encoded in binary:

ILP	ILEP
When feasible, is there a solution of polynomial size?	
✓	✗
When an optimum exists, is there one of polynomial size?	
✓	✗
Are optimal solutions located "near" the boundary of the feasible region?	
✓	✗
Can solutions be checked in polynomial time?	
✓	✓

A comparison between ILP and ILEP

When we give solutions in a compressed form:

ILP	ILEP
When feasible, is there a solution of polynomial size?	
✓	✓ *(ICALP'24)*
When an optimum exists, is there one of polynomial size?	
✓	✓ *(preprint)*
Are optimal solutions located "near" the boundary of the feasible region?	
✓	✗ **unproblematic**
Can solutions be checked in polynomial time?	
✓	✓ **kind of**

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

$$x_i \leftarrow 0, \quad x_i \leftarrow x_j + x_k, \quad x_i \leftarrow 2^{x_j}, \quad x_i \leftarrow a \cdot x_j, \qquad \text{where } a \in \mathbb{Q} \text{ and } j, k < i.$$

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

$$x_i \leftarrow 0, \quad x_i \leftarrow x_j + x_k, \quad x_i \leftarrow 2^{x_j}, \quad x_i \leftarrow a \cdot x_j, \qquad \text{where } a \in \mathbb{Q} \text{ and } j,k < i.$$

Example: $x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x_2 \leftarrow -1 \cdot x_1, \quad x_3 \leftarrow 2^{x_2}, \quad x_4 \leftarrow 2^{x_3}$

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

$$x_i \leftarrow 0, \quad x_i \leftarrow x_j + x_k, \quad x_i \leftarrow 2^{x_j}, \quad x_i \leftarrow a \cdot x_j, \qquad \text{where } a \in \mathbb{Q} \text{ and } j, k < i.$$

Example: $x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x_2 \leftarrow -1 \cdot x_1, \quad x_3 \leftarrow \frac{1}{2}, \qquad x_4 \leftarrow 2^{x_3}$

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

$$x_i \leftarrow 0, \quad x_i \leftarrow x_j + x_k, \quad x_i \leftarrow 2^{x_j}, \quad x_i \leftarrow a \cdot x_j, \qquad \text{where } a \in \mathbb{Q} \text{ and } j, k < i.$$

Example: $x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x_2 \leftarrow -1 \cdot x_1, \quad x_3 \leftarrow \frac{1}{2}, \qquad x_4 \leftarrow \sqrt{2}$

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

$$x_i \leftarrow 0, \quad x_i \leftarrow x_j + x_k, \quad x_i \leftarrow 2^{x_j}, \quad x_i \leftarrow a \cdot x_j, \qquad \text{where } a \in \mathbb{Q} \text{ and } j, k < i.$$

Example: $x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x_2 \leftarrow -1 \cdot x_1, \quad x_3 \leftarrow \frac{1}{2}, \qquad x_4 \leftarrow \sqrt{2}$

ILESLP: An LESLP in which **all** variables evaluate to an integer.

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

$$x_i \leftarrow 0, \quad x_i \leftarrow x_j + x_k, \quad x_i \leftarrow 2^{x_j}, \quad x_i \leftarrow a \cdot x_j, \quad \text{where } a \in \mathbb{Q} \text{ and } j,k < i.$$

Example: $x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x_2 \leftarrow -1 \cdot x_1, \quad x_3 \leftarrow \frac{1}{2}, \quad x_4 \leftarrow \sqrt{2}$

ILESLP: An LESLP in which **all** variables evaluate to an integer.

Theorem (Hitarth, M., Shabadi. *preprint*)

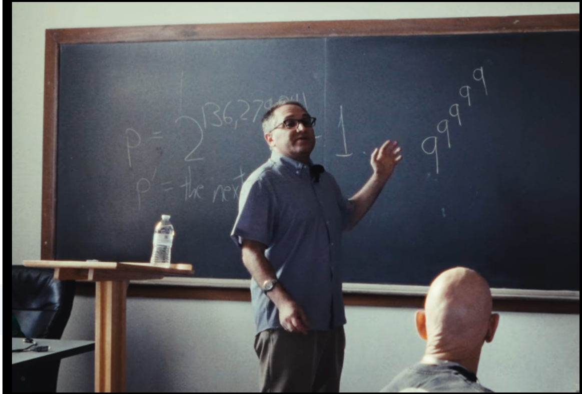
If an ILEP has optimal solutions, then one is encoded by a polynomial-size ILESLP.

(Proven by giving an algorithm for constructing optimal solutions.)

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

Prof. Scott Aaronson: Why Philosophers Should Care About Computational Complexity @ UT Austin



$\cdot x_j$, where $a \in \mathbb{Q}$ and $j, k < i$.

$x_3 \leftarrow \frac{1}{2}$, $x_4 \leftarrow \sqrt{2}$

ate to an integer.

nt)

coded by a polynomial-size ILESLP.

(Proven by giving an algorithm for constructing optimal solutions.)

Integer Linear-Exponential Straight-Line Programs (ILESLPs)

Linear-Exponential Straight-Line Program (LESLP): A sequence of assignments

$a \cdot x_j$, where $a \in \mathbb{Q}$ and $j, k < i$.

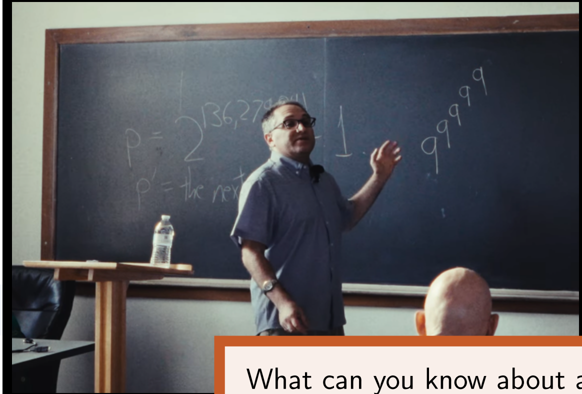
$x_3 \leftarrow \frac{1}{2}$, $x_4 \leftarrow \sqrt{2}$

ate to an integer.

nt)

(Proven by giving

Prof. Scott Aaronson: Why Philosophers Should Care About Computational Complexity @ UT Austin



What can you know about an ILESLP? What can you do with it?

Recognizing that an LESLP is an ILESLP

An LESLP is an ILESLP whenever:

- In assignments $x \leftarrow 2^y$, the variable y evaluates to a non-negative integer
- In assignments $x \leftarrow \frac{m}{g} \cdot y$, the variable y evaluates to a multiple of $\frac{g}{\gcd(m,g)}$

Recognizing that an LESLP is an ILESLP

An LESLP is an ILESLP whenever:

- In assignments $x \leftarrow 2^y$, the variable y evaluates to a non-negative integer
- In assignments $x \leftarrow \frac{m}{g} \cdot y$, the variable y evaluates to a multiple of $\frac{g}{\gcd(m,g)}$

$\text{NAT}_{\text{ILESLP}}$

Input: an ILESLP σ.
Question: does the last expression in σ evaluate to a non-negative integer?

$\text{DIV}_{\text{ILESLP}}$

Input: an ILESLP σ and a positive integer h.
Question: does the last expression in σ evaluate to a multiple of h?

Recognizing that an LESLP is an ILESLP

An LESLP is an ILESLP whenever:

- In assignments $x \leftarrow 2^y$, the variable y evaluates to a non-negative integer
- In assignments $x \leftarrow \frac{m}{g} \cdot y$, the variable y evaluates to a multiple of $\frac{g}{\gcd(m,g)}$

$\text{NAT}_{\text{ILESLP}}$

In polynomial time

Input: an <u>I</u>LESLP σ.
Question: does the last expression in σ evaluate to a non-negative integer?

$\text{DIV}_{\text{ILESLP}}$

Input: an <u>I</u>LESLP σ and a positive integer h.
Question: does the last expression in σ evaluate to a multiple of h?

Recognizing that an LESLP is an ILESLP

An LESLP is an ILESLP whenever:

- In assignments $x \leftarrow 2^y$, the variable y evaluates to a non-negative integer
- In assignments $x \leftarrow \frac{m}{g} \cdot y$, the variable y evaluates to a multiple of $\frac{g}{\gcd(m,g)}$

NAT$_{\text{ILESLP}}$ In polynomial time

Input: an <u>I</u>LESLP σ.
Question: does the last expression in σ evaluate to a non-negative integer?

DIV$_{\text{ILESLP}}$ In polynomial time with an oracle for factoring

(not in BPP, unless some crypto assumption breaks)

Input: an <u>I</u>LESLP σ and a positive integer h.
Question: does the last expression in σ evaluate to a multiple of h?

Recognizing that an LESLP is an ILESLP

An LESLP is an ILESLP whenever:

The set of all ILESLPs is not recognizable in polynomial time!

However, given an ILESLP σ, there is a small set $\mathbb{P}(\sigma)$ of small primes such that

$$\{(\sigma, \mathbb{P}(\sigma)) : \sigma \text{ is an ILESLP}\}$$

is recognizable in polynomial time.

We use $(\sigma, \mathbb{P}(\sigma))$ as certificates of solutions.

(not in BPP, unless some crypto assumption breaks)

Input: an ILESLP σ and a positive integer h.
Question: does the last expression in σ evaluate to a multiple of h?

Checking solutions

Input: An integer linear-exponential program φ, and $(\sigma, \mathbb{P}(\sigma))$ with σ ILESLP.
Question: Is σ a solution to φ?

Checking solutions

In polynomial time

Input: An integer linear-exponential program φ, and $(\sigma, \mathbb{P}(\sigma))$ with σ ILESLP.
Question: Is σ a solution to φ?

For inequalities without $(x \bmod 2^y)$:

$$-2 \cdot 2^x + 5 \cdot y + 10 \geq 0$$

1. Append $-2 \cdot 2^x + 5 \cdot y + 10$ to σ
2. Query the algorithm for $\text{NAT}_{\text{ILESLP}}$.

$\boxed{\sigma}$ (already defines x and y)

$z_0 \leftarrow 2^x$
$z_1 \leftarrow -2 \cdot z_0$

$z_2 \leftarrow 5 \cdot y$

$z_3 \leftarrow 0$
$z_4 \leftarrow 2^{z_3}$
$z_5 \leftarrow 10 \cdot z_4$

$z_6 \leftarrow z_1 + z_2$
$\mathbf{z_7} \leftarrow z_6 + z_5$

Checking solutions

In polynomial time

Input: An integer linear-exponential program φ, and $(\sigma, \mathbb{P}(\sigma))$ with σ ILESLP.
Question: Is σ a solution to φ?

Expressions $(x \bmod 2^y)$ can be eliminated:

Mod computation

Input: an ILESLP σ, two variables x and y in σ, and the set $\mathbb{P}(\sigma)$.
Output: an ILESLP σ' whose last expression evaluates to $(\sigma(x) \bmod 2^{\sigma(y)})$.

Algorithm runs in polynomial time.
If $\mathbb{P}(\sigma)$ is not provided in input, the algorithm requires an integer factoring oracle.

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.

✓: certificates checkable in polynomial time.

What about the objective function?

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.
✓: certificates checkable in polynomial time.

What about the objective function?

In ILP, the value of the objective τ can be computed in binary in polynomial time.
One can then compute the optimum of τ in $\mathsf{FP}^{\mathsf{NP}}$.

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.

✓: certificates checkable in polynomial time.

What about the objective function?

In ILP, the value of the objective τ can be computed in binary in polynomial time.
One can then compute the optimum of τ in $\mathsf{FP}^{\mathsf{NP}}$.

τ : objective function

φ : integer linear program

$[0..100]$: range in which optimum lie

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.

✓: certificates checkable in polynomial time.

What about the objective function?

In ILP, the value of the objective τ can be computed in binary in polynomial time.
One can then compute the optimum of τ in $\mathsf{FP}^{\mathsf{NP}}$.

τ : objective function

φ : integer linear program

$[0..100]$: range in which optimum lie

Is $\varphi \wedge \tau \geq 50$ satisfiable?

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.

✓: certificates checkable in polynomial time.

What about the objective function?

In ILP, the value of the objective τ can be computed in binary in polynomial time.
One can then compute the optimum of τ in $\mathsf{FP}^{\mathsf{NP}}$.

τ : objective function

φ : integer linear program

$[0..49]$: range in which optimum lie

No.

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.

✓: certificates checkable in polynomial time.

What about the objective function?

In ILP, the value of the objective τ can be computed in binary in polynomial time.
One can then compute the optimum of τ in $\mathrm{FP}^{\mathrm{NP}}$.

τ	objective function
φ	integer linear program
$[0..49]$	range in which optimum lie

Is $\varphi \wedge \tau \geq 24$ satisfiable? →

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.

✓: certificates checkable in polynomial time.

What about the objective function?

In ILP, the value of the objective τ can be computed in binary in polynomial time.
One can then compute the optimum of τ in $\mathsf{FP}^{\mathsf{NP}}$.

τ : objective function

φ : integer linear program

$[24..49]$: range in which optimum lie

Yes.

Comparing values of the objective function

So far we have seen:

✓: polynomial-size certificates encoding of solutions and optimal solutions.

✓: certificates checkable in polynomial time.

What about the objective function?

In ILEP we do not know how to perform binary search on ILESLPs.

We can still **compare** values of the objective function τ in polynomial time:

For $(\sigma, \mathbb{P}(\sigma))$ and $(\eta, \mathbb{P}(\eta))$ solutions:

- build an ILESLP for $\tau(\sigma) - \tau(\eta)$
- call the algorithm for $\mathrm{NAT}_{\mathrm{ILESLP}}$.

This leads to a $\mathrm{FNP}^{\mathrm{NP}}$ upper bound for ILEP.

Recap and an open problem

Key properties of ILEP:

1. *If there are (optimal) solutions, one can be represented with a short ILESLP.*
2. *Checking if an ILESLP is a solution*
3. *Comparing values of the objective on two ILESLPs*

$\left.\begin{array}{l} \text{2.} \\ \text{3.} \end{array}\right\}$ in $\mathsf{P}^{\mathsf{FACTORING}}$…

…or in P if we add a small set of primes to the certificates

Recap and an open problem

Key properties of ILEP:

1. *If there are (optimal) solutions, one can be represented with a short ILESLP.*
2. *Checking if an ILESLP is a solution*
3. *Comparing values of the objective on two ILESLPs*

$\left.\begin{array}{l} 2. \\ 3. \end{array}\right\}$ in $\mathsf{P}^{\mathsf{FACTORING}}$...

...or in P if we add a small set of primes to the certificates

Open problem: binary search on ILESLPs.

Let S be the set of all ILESLPs of size at most k. Is there an algorithm with runtime polynomial in k that, given as input $L, U \in S$, computes $M \in S$ such that the size of both sets $S_1 := \{\sigma \in S : L \leq \sigma \leq M\}$ and $S_2 := \{\sigma \in S : M \leq \sigma \leq U\}$ is in $\Omega(\#S_1 + \#S_2)$?

Above, "$\sigma_1 \leq \sigma_2$" compares the value of the last expressions of σ_1 and σ_2.

An algorithm for finding a solution

$\theta(x, \boldsymbol{y})$	: ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$
$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$	: linear-exponential program

Input: $\varphi(\boldsymbol{x})$ linear-exponential program

As a preliminary step:

- **guess** an ordering θ:

$$2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$$

- initialize a vector $\boldsymbol{r} = \varnothing$ of remainder variables

  In the invariant of the loop that follows, $\boldsymbol{r} < 2^x$ and variables $\boldsymbol{r}$ do not occur in exponentials

- let $\boldsymbol{y} = (y, \ldots, x_0)$

An algorithm for finding a solution

$\theta(x, \boldsymbol{y})$	: ordering $2^x \geq 2^y \geq \dots \geq 2^{x_0} = 1$
$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$	: linear-exponential program

Step I

$\gamma(q_x, \boldsymbol{q}, u)$	: linear program
$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$	: linear-exponential program

Elimination of x: Step I

Divisions by 2^y

An algorithm for finding a solution

Side conditions: $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$

$\theta(x, \boldsymbol{y})$	: ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$	
$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$	: linear-exponential program	

$\downarrow$ *Step I*

$\gamma(q_x, \boldsymbol{q}, u)$	: linear program	
$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$	: linear-exponential program	

Elimination of x: Step I

Divisions by 2^y

Change of variables:

$$x = q_x \cdot 2^y + r_x \quad \text{and} \quad \boldsymbol{r} = \boldsymbol{q} \cdot 2^y + \boldsymbol{r}'$$

+ constraints $r_x < 2^y$ and $\boldsymbol{r}' < 2^y$ (loop invariant)

An algorithm for finding a solution

Side conditions: $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$

$\theta(x, \boldsymbol{y})$: ordering $2^x \geq 2^y \geq \dots \geq 2^{x_0} = 1$

$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$: linear-exponential program

$\downarrow$ *Step I*

$\gamma(q_x, \boldsymbol{q}, u)$: linear program

$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$: linear-exponential program

Elimination of x: Step I

Divisions by 2^y

Change of variables:

$$x = q_x \cdot 2^y + r_x \quad \text{and} \quad \boldsymbol{r} = \boldsymbol{q} \cdot 2^y + \boldsymbol{r}'$$

+ constraints $r_x < 2^y$ and $\boldsymbol{r}' < 2^y$ (loop invariant)

Non-deterministic rewriting: guess a small $\ell \in \mathbb{Z}$

$$f(q_x, \boldsymbol{q}, u) \cdot 2^y + g(\boldsymbol{y}, r_x, \boldsymbol{r}') \leq 0$$

$$\downarrow$$

$$f(q_x, \boldsymbol{q}, u) + \ell \leq 0 \ \wedge \ (\ell - 1) \cdot 2^y \leq g(\boldsymbol{y}, r_x, \boldsymbol{r}') < \ell \cdot 2^y$$

An algorithm for finding a solution

Side conditions: $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$

$\theta(x, \boldsymbol{y})$	:	ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$
$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$	:	linear-exponential program

Step I

$\gamma(q_x, \boldsymbol{q}, u)$	:	linear program
$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$	:	linear-exponential program

Step II ⚠

$\gamma'(q_x, u)$	:	linear program

Elimination of x: Step II

Elimination of $\boldsymbol{q}$

γ is linear, so we can use quantifier elimination

$$\exists \boldsymbol{q}\, \gamma(q_x, \boldsymbol{q}, u) \iff \bigvee_{\beta} \gamma'_{\beta}(q_x, u)$$

An algorithm for finding a solution

Side conditions: $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$

$\theta(\boldsymbol{x}, \boldsymbol{y})$: ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$

$\varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{r})$: linear-exponential program

Step I

$\gamma(q_x, \boldsymbol{q}, u)$: linear program

$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$: linear-exponential program

Step II ⚠

$\gamma'(q_x, u)$: linear program

Elimination of x: Step II

Elimination of $\boldsymbol{q}$

γ is linear, so we can use quantifier elimination

$$\exists \boldsymbol{q}\, \gamma(q_x, \boldsymbol{q}, u) \iff \bigvee_{\beta} \gamma'_{\beta}(q_x, u)$$

⚠: For ∃PrA, only known to be in NEXPTIME.

⚠: It may lose all optimal solutions.

An algorithm for finding a solution

Side conditions: $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$

$\theta(x, \boldsymbol{y})$	: ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$
$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$	: linear-exponential program

Step I

$\gamma(q_x, \boldsymbol{q}, u)$	: linear program
$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$	: linear-exponential program

Step II ⚠

$\gamma'(q_x, u)$	: linear program

Elimination of x: Step II

Elimination of $\boldsymbol{q}$

γ is linear, so we can use quantifier elimination

$$\exists \boldsymbol{q}\, \gamma(q_x, \boldsymbol{q}, u) \iff \bigvee_{\beta} \gamma'_{\beta}(q_x, u)$$

⚠: For ∃PrA, only known to be in NEXPTIME.

✓: Can be improved to NP (ICALP'24)

⚠: It may lose all optimal solutions.

✓: No, if we decompose the search space (preprint)

An algorithm for finding a solution

Side conditions: $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$

$\theta(x, \boldsymbol{y})$	:	ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$
$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$	:	linear-exponential program

Step I

$\gamma(q_x, \boldsymbol{q}, u)$	:	linear program
$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$	:	linear-exponential program

Step II ⚠

$\gamma'(q_x, u)$	:	linear program

Step III

$\gamma''(q_x)$	:	linear program
$\psi''(y, r_x)$	:	linear-exponential program

Elimination of x: Step III

Elimination of u and x

Apply side conditions to $\gamma'(q_x, u)$

$$a \cdot 2^{(q_x \cdot 2^y + r_x - y)} + b \cdot q_x + c \leq 0$$

An algorithm for finding a solution

$\theta(x, \boldsymbol{y})$: ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$

$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$: linear-exponential program

$\downarrow$ *Step I*

$\gamma(q_x, \boldsymbol{q}, u)$: linear program

$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$: linear-exponential program

Step II ⚠

$\gamma'(q_x, u)$: linear program

$\downarrow$ *Step III*

$\gamma''(q_x)$: linear program

$\psi''(y, r_x)$: linear-exponential program

Elimination of x: Step III

Elimination of u and x

Apply side conditions to $\gamma'(q_x, u)$

$$a \cdot 2^{(q_x \cdot 2^y + r_x - y)} + b \cdot q_x + c \leq 0$$

For an extremely small $L \in \mathbb{N}$, **non-det. rewrite** as

- $q_x \cdot 2^y + r_x - y > L \ \wedge \ a \leq 0$ or
- $q_x \cdot 2^y + r_x - y = \ell \ \wedge \ a \cdot 2^\ell + q_x + c \leq 0$

for some guessed $\ell \in [0..L]$.

Follow-up with the split performed in Step I:

$$q_x \cdot 2^y + r_x - y - L > 0$$

An algorithm for finding a solution

$\theta(x, \boldsymbol{y})$: ordering $2^x \geq 2^y \geq \cdots \geq 2^{x_0} = 1$

$\varphi(x, \boldsymbol{y}, \boldsymbol{r})$: linear-exponential program

$\downarrow$ *Step I*

$\gamma(q_x, \boldsymbol{q}, u)$: linear program

$\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$: linear-exponential program

Step II ⚠

$\gamma'(q_x, u)$: linear program

$\downarrow$ *Step III*

$\gamma''(q_x)$: linear program

$\psi''(y, r_x)$: linear-exponential program

Step IV ⚠

is γ'' satisfiable?

$\Downarrow$ *no*

reject

Elimination of x: Step IV

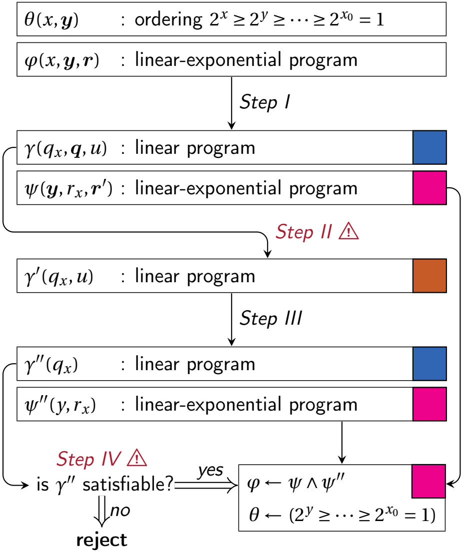
Only q_x is left

$\gamma''(q_x)$ is now completely indpendent from $\psi''(y, r_x)$

✓: For feasibility, simply check if γ'' is satisfiable

⚠: For optimization, decomposition not needed, but one has to chose a value for q_x carefully

An algorithm for finding a solution



Elimination of x: Step IV

Only q_x is left

$\gamma''(q_x)$ is now completely indpendent from $\psi''(y, r_x)$

✓: For feasibility, simply check if γ'' is satisfiable

⚠: For optimization, decomposition not needed, but one has to chose a value for q_x carefully

Continue with the next iteration!

All equations (e.g., $x = q_x \cdot 2^y + r$) are building an ILESLP!

Step II: Integer linear programming in NP through symbolic methods

Gaussian elimination

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau = 0) \leftarrow$ an arbitrary inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \text{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each equality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

Step II: Integer linear programming in NP through symbolic methods

Gaussian elimination

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau = 0) \leftarrow$ an arbitrary inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \mathrm{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each equality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

$\varphi[\frac{-\tau}{a} \,/\, x]$: $\quad b \cdot x + \quad \rho = 0$

Step II: Integer linear programming in NP through symbolic methods

Gaussian elimination

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau = 0) \leftarrow$ an arbitrary inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \text{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each equality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

$\varphi[\frac{-\tau}{a} \,/\, x]$: $\quad a \cdot b \cdot x + a \cdot \rho = 0$

Step II: Integer linear programming in NP through symbolic methods

Gaussian elimination

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau = 0) \leftarrow$ an arbitrary inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \text{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} / x]$
  divide each equality in φ by ℓ
  $\varphi \leftarrow \varphi \land (a \mid \tau)$
  $\ell \leftarrow a$
return φ

$\varphi[\frac{-\tau}{a} / x]$: $\quad -b \cdot \tau + a \cdot \rho = 0$

Step II: Integer linear programming in NP through symbolic methods

Gaussian elimination

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau = 0) \leftarrow$ an arbitrary inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \mathrm{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each equality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

$\varphi[\frac{-\tau}{a} \,/\, x]$: $\quad -b \cdot \tau + a \cdot \rho = 0$

Note: $(-b \cdot \tau + a \cdot \rho) = \det \begin{bmatrix} a & \tau \\ b & \rho \end{bmatrix}$

Step II: Integer linear programming in NP through symbolic methods

Bareiss's algorithm

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau = 0) \leftarrow$ an arbitrary inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \mathrm{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each equality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

Step II: Integer linear programming in NP through symbolic methods

Bareiss's algorithm

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau = 0) \leftarrow$ an arbitrary inequality in φ, with a non-zero
  $s \leftarrow$ guess an integer in $[0..|a| \cdot \text{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each equality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

Desnanot–Jacobi identity:

$$\begin{vmatrix} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

Step II: Integer linear programming in NP through symbolic methods

Presburger's quantifier elimination

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau \leq 0) \leftarrow$ **guess** an inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \mathrm{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} / x]$
  divide each inequality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

Step II: Integer linear programming in NP through symbolic methods

Presburger's quantifier elimination

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau \leq 0) \leftarrow$ **guess** an inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \mathrm{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each inequality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

Consider $\tau \leq a \cdot x \leq \tau'$ with $a > 0$.

"between τ and τ' there is a multiple of a"

One such multiple has the form $a \cdot x = \tau + s$.

Step II: Integer linear programming in NP through symbolic methods

Presburger's quantifier elimination meets Bareiss's algorithm

Input: an integer linear program $\varphi(\boldsymbol{x}, \boldsymbol{z})$ with only inequalities
Ensure: all the variables in $\boldsymbol{x}$ are eliminated from the inequalities of φ

$\ell \leftarrow 1$
for x in $\boldsymbol{x}$ occurring in an inequality of φ **do**
  $(a \cdot x + \tau \leq 0) \leftarrow$ **guess** an inequality in φ, with a non-zero
  $s \leftarrow$ **guess** an integer in $[0..|a| \cdot \mathrm{mod}(\varphi) - 1]$
  $\tau \leftarrow \tau + s$
  $\varphi \leftarrow \varphi[\frac{-\tau}{a} \,/\, x]$
  divide each inequality in φ by ℓ
  $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
  $\ell \leftarrow a$
return φ

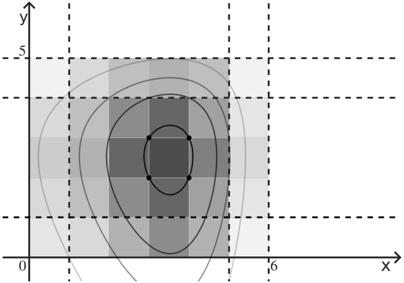
$$\boxed{\ell \cdot \rho + c \leq 0 \quad \to \quad \rho + \lceil \tfrac{c}{\ell} \rceil \leq 0}$$

Step II: Monotone decompositions

$$\text{maximize } f(x,y) := 8x + 4y - (2^x + 2^y)$$

$$\text{subject to } \varphi(x,y,z) := 0 \le x \le 6$$

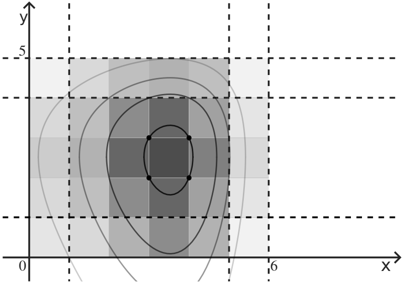
$$0 \le y \le 5$$



Dashed lines: subspaces explored by the quantifier elimination procedure.

Step II: Monotone decompositions

$$\begin{aligned} \text{maximize } f(x,y) &:= 8x + 4y - (2^x + 2^y) \\ \text{subject to } \varphi(x,y,z) &:= 0 \le x \le 6 \\ & \quad\;\; 0 \le y \le 5 \end{aligned}$$



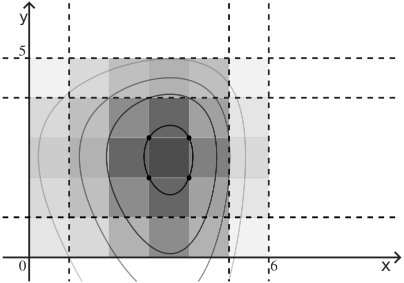
Dashed lines: subspaces explored by the quantifier elimination procedure.

Idea: when eliminating x, the objective f is monotone within $[0..3]$ and within $[4..6]$.

$(a \cdot x + \tau \le 0) \leftarrow$ **guess** an inequality in φ, with $a \neq 0$, or an inequality in $\{x \le 3, 4 \le x\}$.

Step II: Monotone decompositions

$$\text{maximize } f(x,y) := 8x + 4y - (2^x + 2^y)$$
$$\text{subject to } \varphi(x,y,z) := 0 \le x \le 6$$
$$0 \le y \le 5$$



Dashed lines: subspaces explored by the quantifier elimination procedure.

Theorem (Hitarth, M., Shabadi. *preprint*)

*One can always partition the search space into regions where the objective function is "**periodically monotone**". Inequalities defining the boundaries of these regions can be added to those guessed by quantifier elimination, without affecting the NP runtime.*

Recap and an open problem

Key properties of ILEP:

1. *If there are (optimal) solutions, one can be represented with a short ILESLP.*
2. *Checking if an ILESLP is a solution*
3. *Comparing values of the objective on two ILESLPs*

$\left.\begin{array}{l}2.\\3.\end{array}\right\}$ in $\mathsf{P}^{\mathsf{FACTORING}}$...

...or in P if we add a small set of primes to the certificates

Open problem: binary search on ILESLPs.

Let S be the set of all ILESLPs of size at most k. Is there an algorithm with runtime polynomial in k that, given as input $L, U \in S$, computes $M \in S$ such that the size of both sets $S_1 := \{\sigma \in S : L \leq \sigma \leq M\}$ and $S_2 := \{\sigma \in S : M \leq \sigma \leq U\}$ is in $\Omega(\#S_1 + \#S_2)$?

Above, "$\sigma_1 \leq \sigma_2$" compares the value of the last expressions of σ_1 and σ_2.