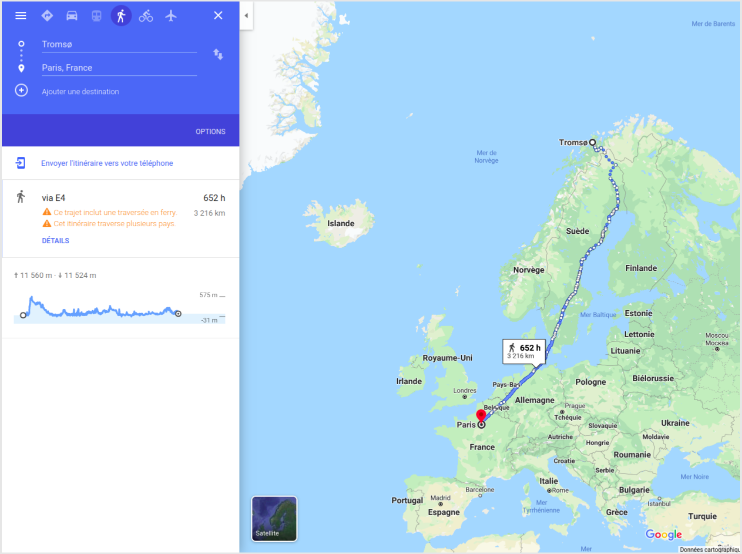
Exact semidefinite programming bounds for packing problems

Philippe Moustrou, UiT The Arctic University of Norway
Joint work with M. Dostert (EPFL) and D. de Laat (TU Delft).

Online Summer School on Optimization, Interpolation and Modular Forms
August 28, 2020

Tromsø: the Paris of the North



Tromsø: the Paris of the North

Menu

Exact semidefinite programming bounds for packing problems

Menu

Exact semidefinite programming bounds for packing problems

- Packing problems: What kind of problems?

Menu

Exact semidefinite programming bounds for packing problems

- Packing problems: What kind of problems?
- Semidefinite programming bounds: Applications of David's lectures.

Menu

Exact semidefinite programming bounds for packing problems

- Packing problems: What kind of problems?
- Semidefinite programming bounds: Applications of David's lectures.
- Exact: Why do we want exact bounds?

Menu

Exact semidefinite programming bounds for packing problems

- Packing problems: What kind of problems?
- Semidefinite programming bounds: Applications of David's lectures.
- Exact: Why do we want exact bounds?

Problem:

Usually semidefinite programming provides approximate numerical bounds.

Menu

Exact semidefinite programming bounds for packing problems

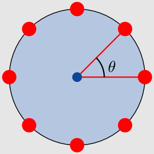
- Packing problems: What kind of problems?
- Semidefinite programming bounds: Applications of David's lectures.
- Exact: Why do we want exact bounds?

Problem:

Usually semidefinite programming provides approximate numerical bounds.

How can we turn these bounds into exact bounds?

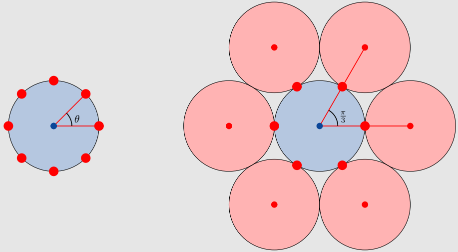
Spherical codes and variants



Spherical codes:

$$\max\{|C|, \quad C \subset S^{n-1}, \quad x \cdot y \leq \cos\theta \text{ for all } x \neq y \in C\}$$

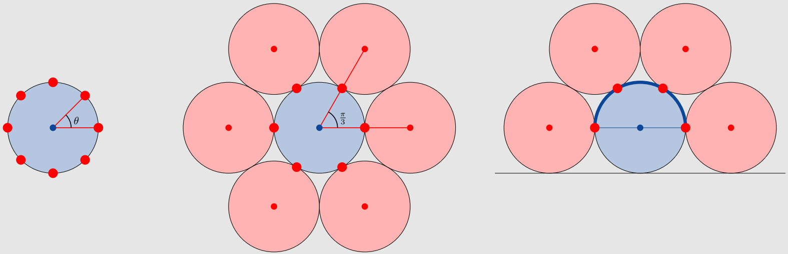
Spherical codes and variants



Kissing number:

$$\max\{|C|, \quad C \subset S^{n-1}, \quad x \cdot y \leq 1/2 \text{ for all } x \neq y \in C\}$$

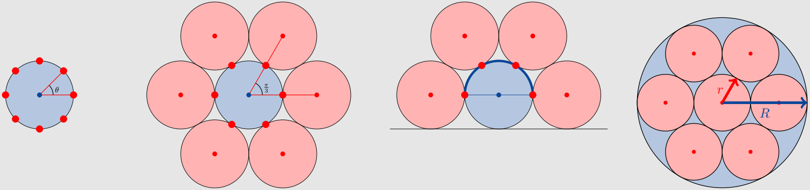
Spherical codes and variants



Kissing number of the hemisphere:

$$\max\{|C|, \quad C \subset \mathrm{H}^{n-1}, \quad x \cdot y \leq 1/2 \text{ for all } x \neq y \in C\}$$

Spherical codes and variants



Packing spheres in spheres:

$$\max\{|C| : C \subset B(0, R-r), \|x-y\| \geq 2r \text{ for all } x \neq y \in C\}$$

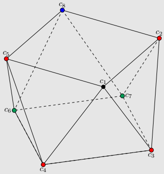
Examples

We are interested in special rigid structures, like:

Examples

We are interested in special rigid structures, like:

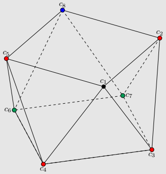
- The square antiprism, the unique optimal θ-spherical code in dimension 3 with $\cos\theta = (2\sqrt{2}-1)/7$ (Schütte-van der Waerden 1951, Danzer 1986).



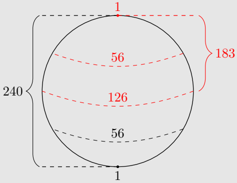
Examples

We are interested in special rigid structures, like:

- The square antiprism, the unique optimal θ-spherical code in dimension 3 with $\cos\theta = (2\sqrt{2}-1)/7$ (Schütte-van der Waerden 1951, Danzer 1986).



- For the Hemisphere in dimension 8: the E_8 lattice provides an optimal configuration (Bachoc-Vallentin, 2008). What about uniqueness?



Graph formulation

Let $G = (V, E)$ be the graph where:

Graph formulation

Let $G = (V, E)$ be the graph where:

- $V = S^{n-1}$ (or H^{n-1}),

Graph formulation

Let $G = (V, E)$ be the graph where:

- $V = S^{n-1}$ (or H^{n-1}),
- $\{x, y\} \in E$ if $x \cdot y > \cos\theta$.

Graph formulation

Let $G = (V, E)$ be the graph where:

- $V = S^{n-1}$ (or H^{n-1}),
- $\{x, y\} \in E$ if $x \cdot y > \cos\theta$.

Our problems boil down to computing the independence number of these graphs!

Graph formulation

Let $G = (V, E)$ be the graph where:

- $V = S^{n-1}$ (or H^{n-1}),
- $\{x, y\} \in E$ if $x \cdot y > \cos\theta$.

Our problems boil down to computing the independence number of these graphs!

- Lower bounds: Constructions.

Graph formulation

Let $G = (V, E)$ be the graph where:

- $V = S^{n-1}$ (or H^{n-1}),
- $\{x, y\} \in E$ if $x \cdot y > \cos\theta$.

Our problems boil down to computing the independence number of these graphs!

- Lower bounds: Constructions.
- Upper bounds: Hierarchies of semidefinite upper bounds (see David's lectures). In particular, for spherical codes:

Graph formulation

Let $G = (V, E)$ be the graph where:

- $V = S^{n-1}$ (or H^{n-1}),
- $\{x, y\} \in E$ if $x \cdot y > \cos\theta$.

Our problems boil down to computing the independence number of these graphs!

- Lower bounds: Constructions.
- Upper bounds: Hierarchies of semidefinite upper bounds (see David's lectures). In particular, for spherical codes:
  - 2-point bound (Delsarte-Goethals-Seidel 1977)

Graph formulation

Let $G = (V, E)$ be the graph where:

- $V = S^{n-1}$ (or H^{n-1}),
- $\{x, y\} \in E$ if $x \cdot y > \cos\theta$.

Our problems boil down to computing the independence number of these graphs!

- Lower bounds: Constructions.
- Upper bounds: Hierarchies of semidefinite upper bounds (see David's lectures). In particular, for spherical codes:
  - 2-point bound (Delsarte-Goethals-Seidel 1977)
  - 3-point bound (Bachoc-Vallentin 2008).

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Recall the two main ingredients:

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Recall the two main ingredients:

- Up to symmetry, a couple x, y of points in a θ-spherical code is uniquely determined by

$$u = x \cdot y, \quad \text{with } \begin{cases} u = 1 & x = y \\ u \in [-1, \cos\theta] & x \neq y \end{cases}$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Recall the two main ingredients:

- Up to symmetry, a couple x, y of points in a θ-spherical code is uniquely determined by

$$u = x \cdot y, \quad \text{with } \begin{cases} u = 1 & x = y \\ u \in [-1, \cos\theta] & x \neq y \end{cases}$$

- The normalized Gegenbauer polynomials $P_k^n(u)$ (with $P_k^n(1) = 1$), satisfying:

$$\text{For every } X \subset S^{n-1} \text{ finite, } \sum_{x,y \in X} P_k^n(x \cdot y) \geq 0.$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \ldots, \alpha_d \geq 0$ such that

$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$

- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \dots, \alpha_d \geq 0$ such that
$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$
- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

Then, if C is a θ-spherical code,

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \ldots, \alpha_d \geq 0$ such that
$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$
- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

Then, if C is a θ-spherical code,

$$\sum_{x,y \in C} f(x \cdot y)$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \ldots, \alpha_d \geq 0$ such that

$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$

- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

Then, if C is a θ-spherical code,

$$\sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y)$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \ldots, \alpha_d \geq 0$ such that

$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$

- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

Then, if C is a θ-spherical code,

$$0 \leq \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y)$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \dots, \alpha_d \geq 0$ such that
$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$
- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

Then, if C is a θ-spherical code,

$$0 \leq \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y) \leq |C| f(1) + \sum_{x \neq y} f(x \cdot y)$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \ldots, \alpha_d \geq 0$ such that

$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$

- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

Then, if C is a θ-spherical code,

$$0 \leq \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y) \leq |C| f(1) + \sum_{x \neq y} f(x \cdot y) = |C|(f(1) - |C| + 1)$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

Assume we have a polynomial f such that

- there exists coefficients $\alpha_0, \ldots, \alpha_d \geq 0$ such that

$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$

- $f(u) \leq -1$ for all $u \in [-1, \cos\theta]$

Then, if C is a θ-spherical code,

$$0 \leq \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y) \leq |C| f(1) + \sum_{x \neq y} f(x \cdot y) = |C|(f(1) - |C| + 1)$$

So

$$|C| \leq f(1) + 1$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

So for every $d \geq 0$, the size of a θ-spherical code is at most

$$\begin{aligned}\min\{M \in \mathbb{R} : \alpha_0, \ldots, \alpha_d &\geq 0,\\ f(1) &\leq M - 1,\\ f(u) &\leq -1 \text{ for all } u \in [-1, \cos\theta]\}\end{aligned}$$

where

$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$

2-point bound for spherical codes (Delsarte-Goethals-Seidel 1977)

So for every $d \geq 0$, the size of a θ-spherical code is at most

$$
\begin{aligned}
\min\{M \in \mathbb{R} : \alpha_0, \ldots, \alpha_d &\geq 0,\\
f(1) &\leq M - 1,\\
f(u) &\leq -1 \text{ for all } u \in [-1, \cos\theta]\}
\end{aligned}
$$

where

$$f(u) = \sum_{k=0}^{d} \alpha_k P_k^n(u).$$

This is a linear programming bound.

3-point bound for spherical codes (Bachoc-Vallentin 2008)

3-point bound for spherical codes (Bachoc-Vallentin 2008)

- Up to symmetry, a triple of points x, y, z in a θ-spherical code is uniquely determined by

$$u = x \cdot y, \quad v = x \cdot z, \quad t = y \cdot z,$$

with (u, v, t) in

$$\begin{cases} \{(1,1,1)\} & x = y = z \\ \Delta_0 = \{(u,u,1) : u \in [-1, \cos\theta]\} & x \neq y = z \\ \Delta & x, y, z \text{ distinct} \end{cases}$$

where

$$\Delta = \{(u, v, t) : u, v, t \in [-1, \cos\theta], 1 + 2uvt - u^2 - v^2 - t^2 \geq 0\}$$

3-point bound for spherical codes (Bachoc-Vallentin 2008)

- Up to symmetry, a triple of points x, y, z in a θ-spherical code is uniquely determined by

$$u = x \cdot y, \quad v = x \cdot z, \quad t = y \cdot z,$$

with (u, v, t) in

$$\begin{cases} \{(1,1,1)\} & x = y = z \\ \Delta_0 = \{(u,u,1) : u \in [-1, \cos\theta]\} & x \neq y = z \\ \Delta & x, y, z \text{ distinct} \end{cases}$$

where

$$\Delta = \{(u,v,t) : u,v,t \in [-1,\cos\theta], 1 + 2uvt - u^2 - v^2 - t^2 \geq 0\}$$

- Matrix polynomials $S_k^n(u, v, t)$ satisfying:

For every $X \subset S^{n-1}$ finite, $\displaystyle\sum_{x,y,z \in X} S_k^n(x \cdot y, x \cdot z, y \cdot t) \succeq 0.$

3-point bound for spherical codes (Bachoc-Vallentin 2008)

So for every $d \geq 0$, the size of a θ-spherical code is at most

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, F_k \succeq 0$$

$$\sum_{k=0}^{d} \alpha_k + F(1,1,1) \leq M - 1,$$

$$\sum_{k=0}^{d} \alpha_k P_k^n(u) + 3F(u,u,1) \leq -1 \text{ for all } u \in [-1, \cos\theta],$$

$$F(u,v,t) \leq 0 \text{ for all } (u,v,t) \in \Delta\}$$

where

$$F(u,v,t) = \sum_{k=0}^{d} \langle F_k, S_k^n(u,v,t) \rangle.$$

3-point bound for spherical codes (Bachoc-Vallentin 2008)

So for every $d \geq 0$, the size of a θ-spherical code is at most

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, F_k \succeq 0$$

$$\sum_{k=0}^{d} \alpha_k + F(1,1,1) \leq M - 1,$$

$$\sum_{k=0}^{d} \alpha_k P_k^n(u) + 3F(u,u,1) \leq -1 \text{ for all } u \in [-1, \cos\theta],$$

$$F(u,v,t) \leq 0 \text{ for all } (u,v,t) \in \Delta\}$$

where

$$F(u,v,t) = \sum_{k=0}^{d} \langle F_k, S_k^n(u,v,t) \rangle.$$

This leads to semidefinite programming upper bounds using sums of squares.

Less symmetry makes it harder

- These bounds work for spherical codes.

Less symmetry makes it harder

- These bounds work for spherical codes.
- They rely on the action of the orthogonal group $\mathcal{O}(n)$ on S^{n-1}.

Less symmetry makes it harder

- These bounds work for spherical codes.
- They rely on the action of the orthogonal group $\mathcal{O}(n)$ on S^{n-1}.
- For spherical codes in spherical caps, the symmetry group is $\mathcal{O}(n-1)$.

Less symmetry makes it harder

- These bounds work for spherical codes.
- They rely on the action of the orthogonal group $\mathcal{O}(n)$ on S^{n-1}.
- For spherical codes in spherical caps, the symmetry group is $\mathcal{O}(n-1)$.
- Delsarte linear programming bound does not apply anymore!

Less symmetry makes it harder

- These bounds work for spherical codes.
- They rely on the action of the orthogonal group $\mathcal{O}(n)$ on S^{n-1}.
- For spherical codes in spherical caps, the symmetry group is $\mathcal{O}(n-1)$.
- Delsarte linear programming bound does not apply anymore!
- Nevertheless, one can still compute the 2-point bound for these problems.

Less symmetry makes it harder

- These bounds work for spherical codes.
- They rely on the action of the orthogonal group $\mathcal{O}(n)$ on S^{n-1}.
- For spherical codes in spherical caps, the symmetry group is $\mathcal{O}(n-1)$.
- Delsarte linear programming bound does not apply anymore!
- Nevertheless, one can still compute the 2-point bound for these problems.
- These bounds look like the 3-point bound for spherical codes. In particular they are semidefinite programming bounds.

Why exact bounds?

Assume we know a configuration C with $|C| = N$.

Why exact bounds?

Assume we know a configuration C with $|C| = N$.

- Any upper bound $< N + 1$ is enough to prove that C is optimal.

Why exact bounds?

Assume we know a configuration C with $|C| = N$.

- Any upper bound $< N + 1$ is enough to prove that C is optimal.
- Even if we do not solve the SDP exactly, if the numerical output of the solver is very close to N, it is not hard to prove a rigorous upper bound of the form $N + \varepsilon$.

Why exact bounds?

Assume we know a configuration C with $|C| = N$.

- Any upper bound $< N + 1$ is enough to prove that C is optimal.
- Even if we do not solve the SDP exactly, if the numerical output of the solver is very close to N, it is not hard to prove a rigorous upper bound of the form $N + \varepsilon$.

So why do we want an exact sharp bound?

Why exact bounds?

Assume we know a configuration C with $|C| = N$.

- Any upper bound $< N + 1$ is enough to prove that C is optimal.
- Even if we do not solve the SDP exactly, if the numerical output of the solver is very close to N, it is not hard to prove a rigorous upper bound of the form $N + \varepsilon$.

So why do we want an exact sharp bound?

- Optimization: When does a bound give the independence number?

Why exact bounds?

Assume we know a configuration C with $|C| = N$.

- Any upper bound $< N + 1$ is enough to prove that C is optimal.
- Even if we do not solve the SDP exactly, if the numerical output of the solver is very close to N, it is not hard to prove a rigorous upper bound of the form $N + \varepsilon$.

So why do we want an exact sharp bound?

- Optimization: When does a bound give the independence number?
- Geometry: Sharp bounds provide additional information on optimal configurations, leading to uniqueness proofs.

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, f(1) \leq M - 1, f(u) \leq -1 \text{ for all } u \in [-1, \cos\theta]\}$$

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, f(1) \leq M - 1, f(u) \leq -1 \text{ for all } u \in [-1, \cos\theta]\}$$

Take

$$f(u) = \frac{320}{3}(u+1)(u+1/2)^2 u^2 (u-1/2) - 1$$

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, f(1) \leq M - 1, f(u) \leq -1 \text{ for all } u \in [-1, \cos\theta]\}$$

Take

$$f(u) = \frac{320}{3}(u+1)(u+1/2)^2 u^2 (u-1/2) - 1 \quad \Rightarrow M = 240$$

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, f(1) \leq M - 1, f(u) \leq -1 \text{ for all } u \in [-1, \cos\theta]\}$$

Take

$$f(u) = \frac{320}{3}(u+1)(u+1/2)^2 u^2 (u-1/2) - 1 \quad \Rightarrow M = 240$$

Now if C is an optimal configuration,

$$0 \leq \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y) \leq |C| f(1) + \sum_{x \neq y} f(x \cdot y) = |C|(M - |C|)$$

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, f(1) \leq M - 1, f(u) \leq -1 \text{ for all } u \in [-1, \cos\theta]\}$$

Take

$$f(u) = \frac{320}{3}(u+1)(u+1/2)^2 u^2 (u-1/2) - 1 \quad \Rightarrow M = 240$$

Now if C is an optimal configuration,

$$0 = \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y) = |C| f(1) + \sum_{x \neq y} f(x \cdot y) = |C|(M - |C|)$$

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, f(1) \leq M-1, f(u) \leq -1 \text{ for all } u \in [-1, \cos\theta]\}$$

Take

$$f(u) = \frac{320}{3}(u+1)(u+1/2)^2 u^2 (u-1/2) - 1 \quad \Rightarrow M = 240$$

Now if C is an optimal configuration,

$$0 = \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y) \Big) = \sum_{x,y \in C} f(x \cdot y) = |C| f(1) + \sum_{x \neq y} f(x \cdot y) = |C|(M - |C|)$$

$$\Rightarrow \text{for all } x, y \in C, x \cdot y \in \{0, \pm 1/2, \pm 1\}$$

Example: Kissing number in dimension 8 (Bannai-Sloane 1981)

The E_8 lattice provides a configuration C_0 with 240 points.

$$\min\{M \in \mathbb{R} : \alpha_k \geq 0, f(1) \leq M - 1, f(u) \leq -1 \text{ for all } u \in [-1, \cos\theta]\}$$

Take

$$f(u) = \frac{320}{3}(u+1)(u+1/2)^2 u^2 (u-1/2) - 1 \quad \Rightarrow M = 240$$

Now if C is an optimal configuration,

$$0 = \sum_{k=0}^{d} \alpha_k \Big(\sum_{x,y \in C} P_k^n(x \cdot y)\Big) = \sum_{x,y \in C} f(x \cdot y) = |C| f(1) + \sum_{x \neq y} f(x \cdot y) = |C|(M - |C|)$$

$$\Rightarrow \text{for all } x, y \in C, x \cdot y \in \{0, \pm 1/2, \pm 1\} \quad \Rightarrow C = C_0$$

Results

Many examples of exact sharp LP bounds ...

Results

Many examples of exact sharp LP bounds ...

But very few cases in which SDP bound is proven to be sharp while LP is not:

Results

Many examples of exact sharp LP bounds ...

But very few cases in which SDP bound is proven to be sharp while LP is not:

- The Petersen code is the unique optimal $1/6$-code in dimension 4 (Bachoc-Vallentin 2009, Dostert-de Laat-M 2020).

Results

Many examples of exact sharp LP bounds ...

But very few cases in which SDP bound is proven to be sharp while LP is not:

- The Petersen code is the unique optimal $1/6$-code in dimension 4 (Bachoc-Vallentin 2009, Dostert-de Laat-M 2020).
- Numerically sharp for the square antiprism (Bachoc-Vallentin 2009) $\rightarrow$ Rigorous proof (Dostert-de Laat-M 2020)

Results

Many examples of exact sharp LP bounds ...

But very few cases in which SDP bound is proven to be sharp while LP is not:

- The Petersen code is the unique optimal 1/6-code in dimension 4 (Bachoc-Vallentin 2009, Dostert-de Laat-M 2020).
- Numerically sharp for the square antiprism (Bachoc-Vallentin 2009) $\to$ Rigorous proof (Dostert-de Laat-M 2020)
- E_8 gives an optimal configuration on the hemisphere in dimension 8 (Bachoc-Vallentin 2009)
  $\to$ Uniqueness (Dostert-de Laat-M 2020)

Solving an SDP: Rage against the machine precision

Solving an SDP: Rage against the machine precision

A semidefinite program:

$$\inf\{ \underbrace{c^t x}_{\text{objective}} : \underbrace{Ax = b}_{\text{linear constraints}} , \underbrace{\mathcal{B}_i(x) \succeq 0}_{\text{PSD constraints}} \}$$

with x the vector of unknowns, and $\mathcal{B}_i(x)$ the blocks of x.

Solving an SDP: Rage against the machine precision

A semidefinite program:

$$\inf\{ \underbrace{c^t x}_{\text{objective}} : \underbrace{Ax = b}_{\text{linear constraints}} , \underbrace{\mathcal{B}_i(x) \succeq 0}_{\text{PSD constraints}} \}$$

with x the vector of unknowns, and $\mathcal{B}_i(x)$ the blocks of x.

- Solving an SDP exactly is sometimes possible (Henrion-Naldi-Safey El Din 2018).

Solving an SDP: Rage against the machine precision

A semidefinite program:

$$\inf\{ \underbrace{c^t x}_{\text{objective}} : \underbrace{Ax = b}_{\text{linear constraints}} , \underbrace{\mathcal{B}_i(x) \succeq 0}_{\text{PSD constraints}} \}$$

with x the vector of unknowns, and $\mathcal{B}_i(x)$ the blocks of x.

- Solving an SDP exactly is sometimes possible (Henrion-Naldi-Safey El Din 2018).
- For larger problems, SDP solvers provide approximate solutions in floating point in polynomial time.

Solving an SDP: Rage against the machine precision

A semidefinite program:

$$\inf\{ \underbrace{c^t x}_{\text{objective}} \; : \; \underbrace{Ax = b}_{\text{linear constraints}} \; , \; \underbrace{\mathcal{B}_i(x) \succeq 0}_{\text{PSD constraints}} \}$$

with x the vector of unknowns, and $\mathcal{B}_i(x)$ the blocks of x.

- Solving an SDP exactly is sometimes possible (Henrion-Naldi-Safey El Din 2018).
- For larger problems, SDP solvers provide approximate solutions in floating point in polynomial time.

How can we turn an approximate solution into an exact one?

Solving an SDP: Rage against the machine precision

A semidefinite program:

$$\inf\{ \underbrace{c^t x}_{\text{objective}} : \underbrace{Ax = b}_{\text{linear constraints}} , \underbrace{\mathcal{B}_i(x) \succeq 0}_{\text{PSD constraints}} \}$$

with x the vector of unknowns, and $\mathcal{B}_i(x)$ the blocks of x.

- Solving an SDP exactly is sometimes possible (Henrion-Naldi-Safey El Din 2018).
- For larger problems, SDP solvers provide approximate solutions in floating point in polynomial time.

How can we turn an approximate solution into an exact one?

- Even if the SDP is defined over $\mathbb{Q}$, optimal solutions can require high algebraic degree (Nie-Ranestad-Sturmfels 2008).

Solving an SDP: Rage against the machine precision

A semidefinite program:

$$\inf\{ \underbrace{c^t x}_{\text{objective}} : \underbrace{Ax = b}_{\text{linear constraints}} , \underbrace{\mathcal{B}_i(x) \succeq 0}_{\text{PSD constraints}} \}$$

with x the vector of unknowns, and $\mathcal{B}_i(x)$ the blocks of x.

- Solving an SDP exactly is sometimes possible (Henrion-Naldi-Safey El Din 2018).
- For larger problems, SDP solvers provide approximate solutions in floating point in polynomial time.

How can we turn an approximate solution into an exact one?

- Even if the SDP is defined over $\mathbb{Q}$, optimal solutions can require high algebraic degree (Nie-Ranestad-Sturmfels 2008).
- Our context: The problems provide a candidate field to round over, either $\mathbb{Q}$ or $\mathbb{Q}(\sqrt{d})$.

Rounding over $\mathbb{Q}$: Preliminary steps

Rounding over $\mathbb{Q}$: Preliminary steps

- Once we know the optimal value, we can include the objective as a linear constraint.
  $\rightarrow$ Feasibility problem.

Rounding over $\mathbb{Q}$: Preliminary steps

- Once we know the optimal value, we can include the objective as a linear constraint.
  $\rightarrow$ Feasibility problem.

- Use symmetries to reduce the number of variables.
  ($110376 \rightarrow 37651$ for the Hemisphere in dimension 8)

Rounding over $\mathbb{Q}$: Preliminary steps

- Once we know the optimal value, we can include the objective as a linear constraint.
  $\rightarrow$ Feasibility problem.

- Use symmetries to reduce the number of variables.
  ($110376 \rightarrow 37651$ for the Hemisphere in dimension 8)

- Solve the SDP numerically in high precision (SDPA-GMP),
  $\rightarrow$ get an approximate solution x^*:

Rounding over $\mathbb{Q}$: Preliminary steps

- Once we know the optimal value, we can include the objective as a linear constraint.
  $\rightarrow$ Feasibility problem.

- Use symmetries to reduce the number of variables.
  ($110376 \rightarrow 37651$ for the Hemisphere in dimension 8)

- Solve the SDP numerically in high precision (SDPA-GMP),
  $\rightarrow$ get an approximate solution x^*:
  - $Ax^* \approx b$

Rounding over $\mathbb{Q}$: Preliminary steps

- Once we know the optimal value, we can include the objective as a linear constraint.
  $\rightarrow$ Feasibility problem.

- Use symmetries to reduce the number of variables.
  (110376 $\rightarrow$ 37651 for the Hemisphere in dimension 8)

- Solve the SDP numerically in high precision (SDPA-GMP),
  $\rightarrow$ get an approximate solution x^*:
  - $Ax^* \approx b$
  - The blocks $\mathcal{B}_i(x^*)$ might have negative near zero eigenvalues.

Rounding over $\mathbb{Q}$: the affine conditions

We want to find a solution x close to x^* and such that

$$Ax = b.$$

Rounding over $\mathbb{Q}$: the affine conditions

We want to find a solution x close to x^* and such that

$$Ax = b.$$

- Put the system into reduced row echelon form in rational arithmetic, (use Hecke in Julia, the system can be big)

Rounding over $\mathbb{Q}$: the affine conditions

We want to find a solution x close to x^* and such that

$$Ax = b.$$

- Put the system into reduced row echelon form in rational arithmetic, (use Hecke in Julia, the system can be big)
- Solve the system by backsubstitution.
  For every free variable, take a value close to the corresponding value in x^*.

Rounding over $\mathbb{Q}$: the affine conditions

We want to find a solution x close to x^* and such that

$$Ax = b.$$

- Put the system into reduced row echelon form in rational arithmetic, (use Hecke in Julia, the system can be big)
- Solve the system by backsubstitution.
  For every free variable, take a value close to the corresponding value in x^*.

The linear system is then satisfied... But what about the PSD conditions?

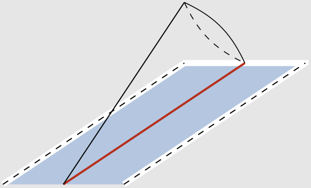
Rounding over $\mathbb{Q}$: the PSD conditions

Rounding over $\mathbb{Q}$: the PSD conditions

- If all the eigenvalues of $\mathcal{B}_i(x^*)$ are far away from zero, $\mathcal{B}_i(x)$ will be positive definite.

Rounding over $\mathbb{Q}$: the PSD conditions

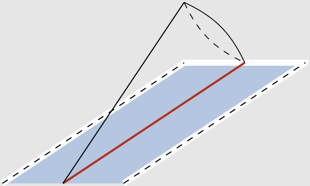
- If all the eigenvalues of $\mathcal{B}_i(x^*)$ are far away from zero, $\mathcal{B}_i(x)$ will be positive definite.



- If the dimension of the affine space is larger than that of the feasible set, we are in trouble. How to deal with near zero eigenvalues?

Rounding over $\mathbb{Q}$: the PSD conditions

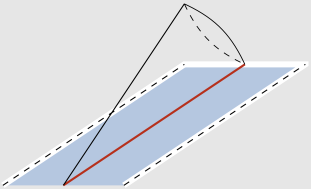
- If all the eigenvalues of $\mathcal{B}_i(x^*)$ are far away from zero, $\mathcal{B}_i(x)$ will be positive definite.



- If the dimension of the affine space is larger than that of the feasible set, we are in trouble. How to deal with near zero eigenvalues?
- Sometimes, zero eigenvalues can be forced by some additional affine constraints coming from an optimal configuration.
  This is sometimes enough... (Cohn-Woo 2012).

Rounding over $\mathbb{Q}$: the PSD conditions

- If all the eigenvalues of $\mathcal{B}_i(x^*)$ are far away from zero, $\mathcal{B}_i(x)$ will be positive definite.



- If the dimension of the affine space is larger than that of the feasible set, we are in trouble. How to deal with near zero eigenvalues?
- Sometimes, zero eigenvalues can be forced by some additional affine constraints coming from an optimal configuration.
  This is sometimes enough... (Cohn-Woo 2012).
- Sometimes not. How to force all these constraints?

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

- We expect a solution over $\mathbb{Q}$.

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

- We expect a solution over $\mathbb{Q}$.
- Intuition: There are some structural reasons for the zero eigenvalues. So the kernels should have nice rational bases.

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

- We expect a solution over $\mathbb{Q}$.
- Intuition: There are some structural reasons for the zero eigenvalues. So the kernels should have nice rational bases.
- Take a block $\mathcal{B}_i(x^*)$ of the approximate solution and compute its kernel in floating point with high precision.

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

- We expect a solution over $\mathbb{Q}$.
- Intuition: There are some structural reasons for the zero eigenvalues. So the kernels should have nice rational bases.
- Take a block $\mathcal{B}_i(x^*)$ of the approximate solution and compute its kernel in floating point with high precision.
- First example: one dimensional kernel.

$$\begin{pmatrix} 0.859374473300157 \\ -0.429687236650083 \\ -0.271381412621106 0 \\ -0.056537794296065 \end{pmatrix}$$

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

- We expect a solution over $\mathbb{Q}$.
- Intuition: There are some structural reasons for the zero eigenvalues. So the kernels should have nice rational bases.
- Take a block $\mathcal{B}_i(x^*)$ of the approximate solution and compute its kernel in floating point with high precision.
- First example: one dimensional kernel.

$$\begin{pmatrix} 0.859374473300157 \\ -0.429687236650083 \\ -0.271381412621106 0 \\ -0.056537794296065 \end{pmatrix} \rightarrow \begin{pmatrix} -15.19999999999925 \\ 7.5999999999997 \\ 4.79999999999982 \\ 1.0 \end{pmatrix}$$

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

- We expect a solution over $\mathbb{Q}$.
- Intuition: There are some structural reasons for the zero eigenvalues. So the kernels should have nice rational bases.
- Take a block $\mathcal{B}_i(x^*)$ of the approximate solution and compute its kernel in floating point with high precision.
- First example: one dimensional kernel.

$$\begin{pmatrix} 0.859374473300157 \\ -0.429687236650083 \\ -0.271381412621106 0 \\ -0.056537794296065 \end{pmatrix} \to \begin{pmatrix} -15.19999999999925 \\ 7.5999999999997 \\ 4.79999999999982 \\ 1.0 \end{pmatrix} \to v = \begin{pmatrix} -15.2 \\ 7.6 \\ 4.8 \\ 1.0 \end{pmatrix}$$

Rounding over $\mathbb{Q}$: detecting kernel vectors (one dimension)

- We expect a solution over $\mathbb{Q}$.
- Intuition: There are some structural reasons for the zero eigenvalues. So the kernels should have nice rational bases.
- Take a block $\mathcal{B}_i(x^*)$ of the approximate solution and compute its kernel in floating point with high precision.
- First example: one dimensional kernel.

$$\begin{pmatrix} 0.859374473300157 \\ -0.429687236650083 \\ -0.271381412621106 0 \\ -0.056537794296065 \end{pmatrix} \to \begin{pmatrix} -15.19999999999925 \\ 7.5999999999997 \\ 4.79999999999982 \\ 1.0 \end{pmatrix} \to v = \begin{pmatrix} -15.2 \\ 7.6 \\ 4.8 \\ 1.0 \end{pmatrix}$$

- Then $\mathcal{B}_i(x)v = 0$ provides new linear constraints on x!

Rounding over $\mathbb{Q}$: detecting kernel vectors (general case)

This is not enough in general. How to extract a nice basis from the numerical values?

$$\ker(\mathcal{B}_i(x^*)) \approx \left\langle \begin{pmatrix} 0.19550004741012542 \\ -0.10616756374846323 \\ -0.25700180101766007 \\ -0.33241916014721035 \end{pmatrix}, \begin{pmatrix} -0.8676883652023846 \\ -0.4321427618192919 \\ -0.2143699892153049 \\ -0.1054836185183479 \end{pmatrix} \right\rangle$$

Rounding over $\mathbb{Q}$: detecting kernel vectors (general case)

Key idea: use the LLL algorithm to detect an integer linear equation almost sastisfied by the kernel vectors...

$$\ker(\mathcal{B}_i(x^*)) \approx \left\langle \begin{pmatrix} 0.19550004741012542 \\ -0.10616756374846323 \\ -0.25700180101766007 \\ -0.33241916014721035 \end{pmatrix}, \begin{pmatrix} -0.8676883652023846 \\ -0.4321427618192919 \\ -0.2143699892153049 \\ -0.1054836185183479 \end{pmatrix} \right\rangle \begin{array}{c} \textbf{-1} \\ \textbf{3} \\ \textbf{-2} \\ \end{array}$$

$$\left\{ -u_1 + 3u_2 - 2u_3 = 0 \right.$$

Rounding over $\mathbb{Q}$: detecting kernel vectors (general case)

...and another one...

$$\ker(\mathcal{B}_i(x^*)) \approx \left\langle \begin{pmatrix} \cancel{0.19550004741012542} \\ -0.1061675637484632 3 \\ -0.2570018010176600 7 \\ -0.3324191601472103 5 \end{pmatrix}, \begin{pmatrix} \cancel{-0.8676883652023846} \\ -0.432142761819291 9 \\ -0.214369989215304 9 \\ -0.105483618518347 9 \end{pmatrix} \right\rangle \begin{array}{c} \\ \mathbf{1} \\ \mathbf{-3} \\ \mathbf{2} \end{array}$$

$$\begin{cases} -u_1 + 3u_2 - 2u_3 & = 0 \\ \qquad\quad u_2 - 3u_3 + 2u_4 & = 0 \end{cases}$$

Rounding over $\mathbb{Q}$: detecting kernel vectors (general case)

With enough equations, we can compute the expected kernel basis.

$$\ker(\mathcal{B}_i(x^*)) \approx \left\langle \begin{pmatrix} 0.19550004741012542 \\ -0.10616756374846323 \\ -0.25700180101766007 \\ -0.33241916014721035 \end{pmatrix}, \begin{pmatrix} -0.867688365202384 6 \\ -0.4321427618192919 \\ -0.2143699892153049 \\ -0.1054836185183479 \end{pmatrix} \right\rangle$$

$$\begin{cases} -u_1 + 3u_2 - 2u_3 & = 0 \\ u_2 - 3u_3 + 2u_4 & = 0 \end{cases}$$

$$\ker(\mathcal{B}_i(x)) = \left\langle \begin{pmatrix} 7 \\ 3 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -6 \\ -2 \\ 0 \\ 1 \end{pmatrix} \right\rangle$$

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.
3. Include the new linear constraints in the linear system $Ax = b$.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.
3. Include the new linear constraints in the linear system $Ax = b$.
4. Row reduce the linear system.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.
3. Include the new linear constraints in the linear system $Ax = b$.
4. Row reduce the linear system.
5. Solve it with backsubstitution using x^*.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.
3. Include the new linear constraints in the linear system $Ax = b$.
4. Row reduce the linear system.
5. Solve it with backsubstitution using x^*.
6. Check that the blocks of the rounded solution are indeed PSD.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.
3. Include the new linear constraints in the linear system $Ax = b$.
4. Row reduce the linear system.
5. Solve it with backsubstitution using x^*.
6. Check that the blocks of the rounded solution are indeed PSD.
7. Celebrate.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.
3. Include the new linear constraints in the linear system $Ax = b$.
4. Row reduce the linear system.
5. Solve it with backsubstitution using x^*.
6. Check that the blocks of the rounded solution are indeed PSD.
7. ~~Celebrate.~~ Restart from 1. and don't forget to save the solution.

Rounding over $\mathbb{Q}$: the complete procedure

1. Compute an approximate solution x^*.
2. Compute the kernels of the $\mathcal{B}_i(x^*)$'s and detect the expected kernels of the $\mathcal{B}_i(x)$'s.
3. Include the new linear constraints in the linear system $Ax = b$.
4. Row reduce the linear system.
5. Solve it with backsubstitution using x^*.
6. Check that the blocks of the rounded solution are indeed PSD.
7. Restart from 1. and don't forget to save the solution.
8. Celebrate.

Back to geometry

Once we get an exact optimal solution, we need to check that the function $F(u, v, t)$ gives enough information on possible optimal configurations:

Back to geometry

Once we get an exact optimal solution, we need to check that the function $F(u, v, t)$ gives enough information on possible optimal configurations:

- Check that the only possible inner products are the ones in the candidate optimal configuration (use Sturm sequences).

Back to geometry

Once we get an exact optimal solution, we need to check that the function $F(u, v, t)$ gives enough information on possible optimal configurations:

- Check that the only possible inner products are the ones in the candidate optimal configuration (use Sturm sequences).
- If needed compute the possible 3-point distance distribution of an optimal code.

Back to geometry

Once we get an exact optimal solution, we need to check that the function $F(u, v, t)$ gives enough information on possible optimal configurations:

- Check that the only possible inner products are the ones in the candidate optimal configuration (use Sturm sequences).
- If needed compute the possible 3-point distance distribution of an optimal code.
- Use this information and a bit of geometry to prove that the candidate optimal configuration is unique!

Generalizations (done or to be done)

Generalizations (done or to be done)

- We extended our rounding procedure to quadratic fields (needed for the square antiprism).

Generalizations (done or to be done)

- We extended our rounding procedure to quadratic fields (needed for the square antiprism).
- Besides spherical codes, we could apply our method for packing spheres in spheres (here also quadratic fields are needed).

Generalizations (done or to be done)

- We extended our rounding procedure to quadratic fields (needed for the square antiprism).
- Besides spherical codes, we could apply our method for packing spheres in spheres (here also quadratic fields are needed).
- There are natural related problems where this approach can be promising (energy minimization, codes in complex projective space,...)

Generalizations (done or to be done)

- We extended our rounding procedure to quadratic fields (needed for the square antiprism).
- Besides spherical codes, we could apply our method for packing spheres in spheres (here also quadratic fields are needed).
- There are natural related problems where this approach can be promising (energy minimization, codes in complex projective space,...)
- What about other applications?

Thank you!

Bonus: extension to quadratic fields (reformulation)

Multiply (but still conquer):

Bonus: extension to quadratic fields (reformulation)

Multiply (but still conquer):

- The semidefinite program is defined over $\mathbb{Q}(\sqrt{d})$, namely

$$A = A_1 + \sqrt{d} A_2, \quad b = b_1 + \sqrt{d} b_2$$

  where A_1, A_2, b_1, b_2 have coefficients in $\mathbb{Q}$.

Bonus: extension to quadratic fields (reformulation)

Multiply (but still conquer):

- The semidefinite program is defined over $\mathbb{Q}(\sqrt{d})$, namely

$$A = A_1 + \sqrt{d}A_2, \quad b = b_1 + \sqrt{d}b_2$$

  where A_1, A_2, b_1, b_2 have coefficients in $\mathbb{Q}$.
- We also expect a solution over $\mathbb{Q}(\sqrt{d})$, so write

$$x = x_1 + \sqrt{d}x_2$$

  and work over $\mathbb{Q}$:

$$\begin{pmatrix} A_1 & dA_2 \\ A_2 & A_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Bonus: extension to quadratic fields (finding good x_1^*,x_2^*)

- From the numerical x^* satisfying $Ax^* \approx b$ we need to find x_1^* and x_2^* such that $x^* \approx x_1^* + \sqrt{d}x_2^*$ and

$$\begin{pmatrix} A_1 & dA_2 \\ A_2 & A_1 \end{pmatrix} \begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} \approx \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Bonus: extension to quadratic fields (finding good x_1^*,x_2^*)

- From the numerical x^* satisfying $Ax^* \approx b$ we need to find x_1^* and x_2^* such that $x^* \approx x_1^* + \sqrt{d}x_2^*$ and

$$\begin{pmatrix} A_1 & dA_2 \\ A_2 & A_1 \end{pmatrix} \begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} \approx \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

- To do so, solve (in floating point) the linear system:

$$\begin{pmatrix} A_1 & dA_2 \\ A_2 & A_1 \end{pmatrix} \begin{pmatrix} y \\ \frac{1}{\sqrt{d}}(x^* - y) \end{pmatrix} \approx \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Bonus: extension to quadratic fields (kernel detection)

- Compute the approximate kernel of $\mathcal{B}_i(x^*)$

$$\ker(\mathcal{B}_i(x^*)) \approx \langle \begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \end{pmatrix}, \ldots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \end{pmatrix} \rangle$$

Bonus: extension to quadratic fields (kernel detection)

- Compute the approximate kernel of $\mathcal{B}_i(x^*)$

$$\ker(\mathcal{B}_i(x^*)) \approx \langle \begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \end{pmatrix}, \dots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \end{pmatrix} \rangle$$

- Look for integer relations in

$$\begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \\ \sqrt{d}u_1^1 \\ \vdots \\ \sqrt{d}u_l^1 \end{pmatrix}, \dots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \\ \sqrt{d}u_1^r \\ \vdots \\ \sqrt{d}u_l^r \end{pmatrix}$$

Bonus: extension to quadratic fields (kernel detection)

- Compute the approximate kernel of $\mathcal{B}_i(x^*)$

$$\ker(\mathcal{B}_i(x^*)) \approx \langle \begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \end{pmatrix}, \dots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \end{pmatrix} \rangle$$

- Look for integer relations in

$$\begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \\ \sqrt{d}u_1^1 \\ \vdots \\ \sqrt{d}u_l^1 \end{pmatrix}, \dots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \\ \sqrt{d}u_1^r \\ \vdots \\ \sqrt{d}u_l^r \end{pmatrix} \begin{array}{c} \lambda_1 \\ \vdots \\ \lambda_l \\ \mu_1 \\ \vdots \\ \mu_l^r \end{array}$$

Bonus: extension to quadratic fields (kernel detection)

- Compute the approximate kernel of $\mathcal{B}_i(x^*)$

$$\ker(\mathcal{B}_i(x^*)) \approx \left\langle \begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \end{pmatrix}, \ldots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \end{pmatrix} \right\rangle$$

- Look for integer relations in

$$\begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \\ \sqrt{d}u_1^1 \\ \vdots \\ \sqrt{d}u_l^1 \end{pmatrix}, \ldots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \\ \sqrt{d}u_1^r \\ \vdots \\ \sqrt{d}u_l^r \end{pmatrix} \begin{matrix} \lambda_1 \\ \vdots \\ \lambda_l \\ \mu_1 \\ \vdots \\ \mu_l^r \end{matrix} \to \sum_{i=1}^{l} (\lambda_i + \sqrt{d}\mu_i)u_i = 0$$

Bonus: extension to quadratic fields (kernel detection)

- Compute the approximate kernel of $\mathcal{B}_i(x^*)$

$$\ker(\mathcal{B}_i(x^*)) \approx \langle \begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \end{pmatrix}, \dots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \end{pmatrix} \rangle$$

- Look for integer relations in

$$\begin{pmatrix} u_1^1 \\ \vdots \\ u_l^1 \\ \sqrt{d}u_1^1 \\ \vdots \\ \sqrt{d}u_l^1 \end{pmatrix}, \dots, \begin{pmatrix} u_1^r \\ \vdots \\ u_l^r \\ \sqrt{d}u_1^r \\ \vdots \\ \sqrt{d}u_l^r \end{pmatrix} \begin{matrix} \lambda_1 \\ \vdots \\ \lambda_l \\ \mu_1 \\ \vdots \\ \mu_l^r \end{matrix} \to \sum_{i=1}^{l} (\lambda_i + \sqrt{d}\mu_i)u_i = 0$$

- Compute the expected kernel over $\mathbb{Q}$ and add the corresponding constraints on x_1 and x_2.