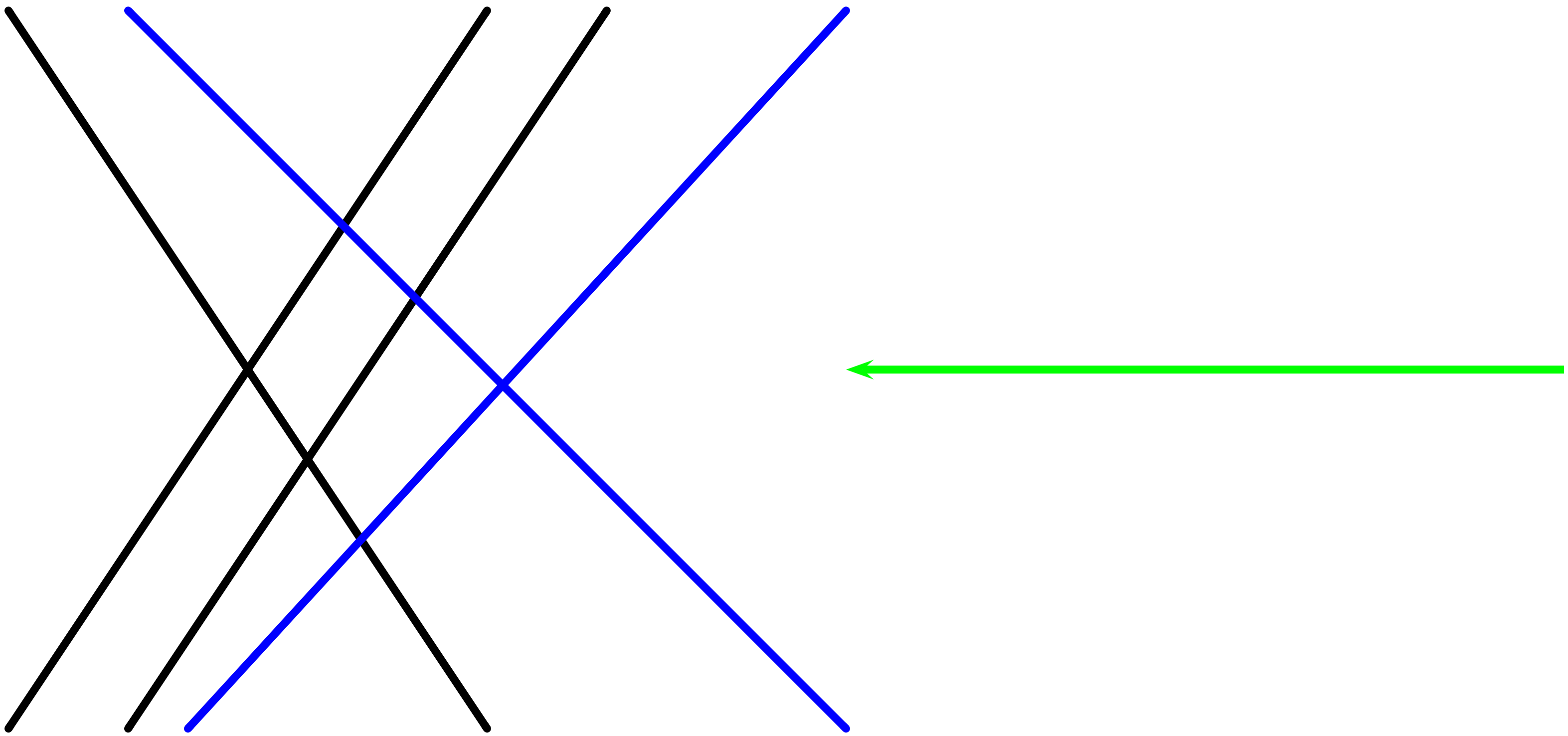
Linear Programming in Subexponential Time

Linear Programming Algebraic View

for $x, c \in \mathbb{R}^d$; $b \in \mathbb{R}^n$; $A \in \mathbb{R}^{(n \times d)}$

min $c^T x$

subject to $Ax \leq b$



Goals

The Holy Grail

- polynomial time algorithm in unit-cost model

Previous Work

- polynomial time algorithm in bit-cost model
- $O(d!n)$ time algorithm in unit-cost model (Seidel)

Today

- subexponential time algorithm in unit-cost model

Outline

- Algorithm SolveLP1 - Clarkson
- Algorithm SolveLP2 - Clarkson
- Algorithm SolveLP3 - Matousek, Sharir and Welzl; also Kalai

Result: $O(d^2 n + e^{O(\sqrt{d \log d})})$

Definitions

- H .... set of n linear constraints
- $v(H)$........... optimum point subject to H
- a basis for H . . . minimal set B of constraints s.t.

$$v(B) = v(H).$$

Note: the size of a basis is d.

Random Sampling

- Randomly reduce large problem to small subproblem.
- Solve small subproblem.
- If solution solves large problem, you win.
- Otherwise, get hints from wrong solution and try again.

Algorithm: SolveLP1

SolveLP1(H)

$G := \emptyset$

repeat

$R :=$ random subset of H of size r

$v :=$ SolveLP1($G \cup R$)

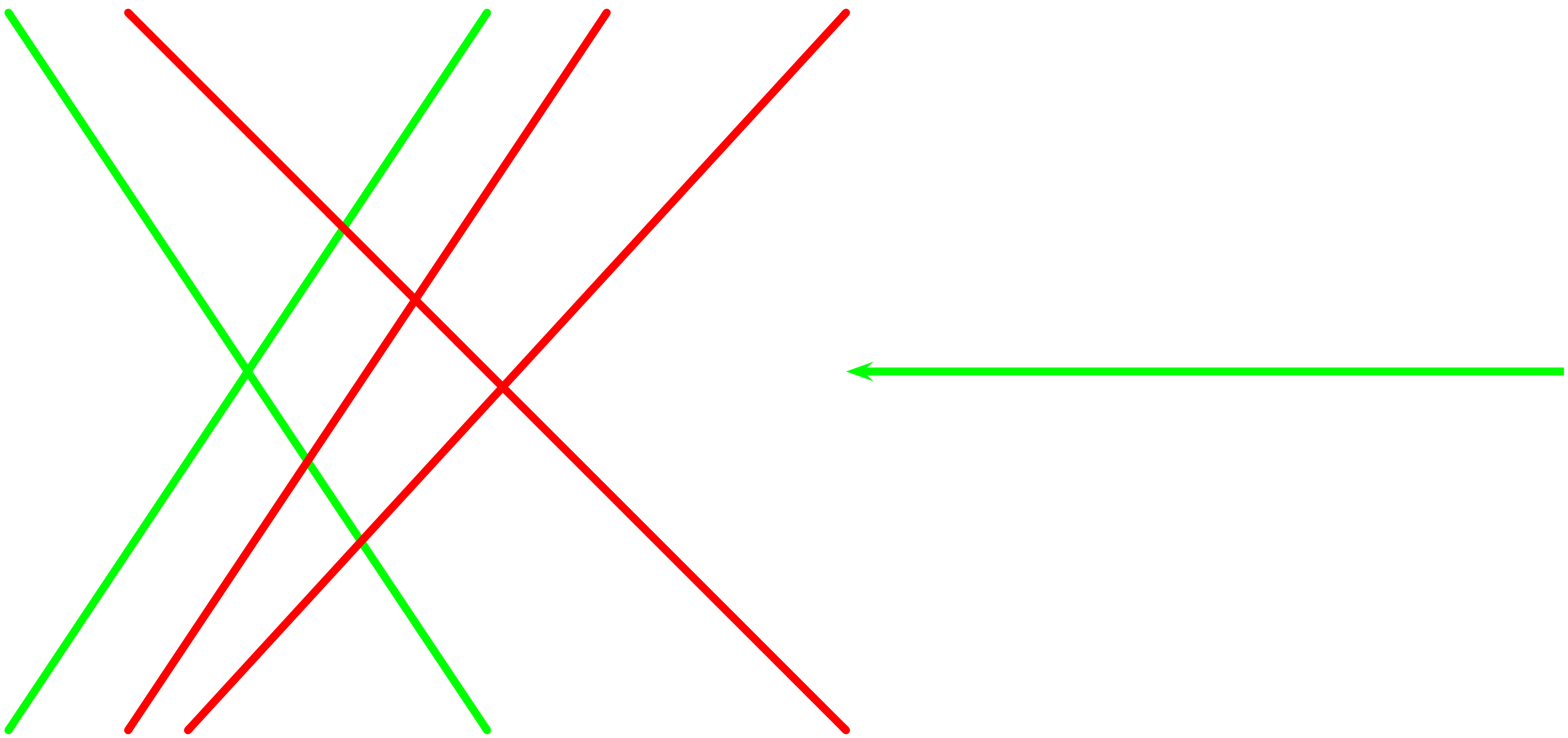
$V :=$ set of constraints violated by v

if $|V|$ is small enough, $G := G \cup V$

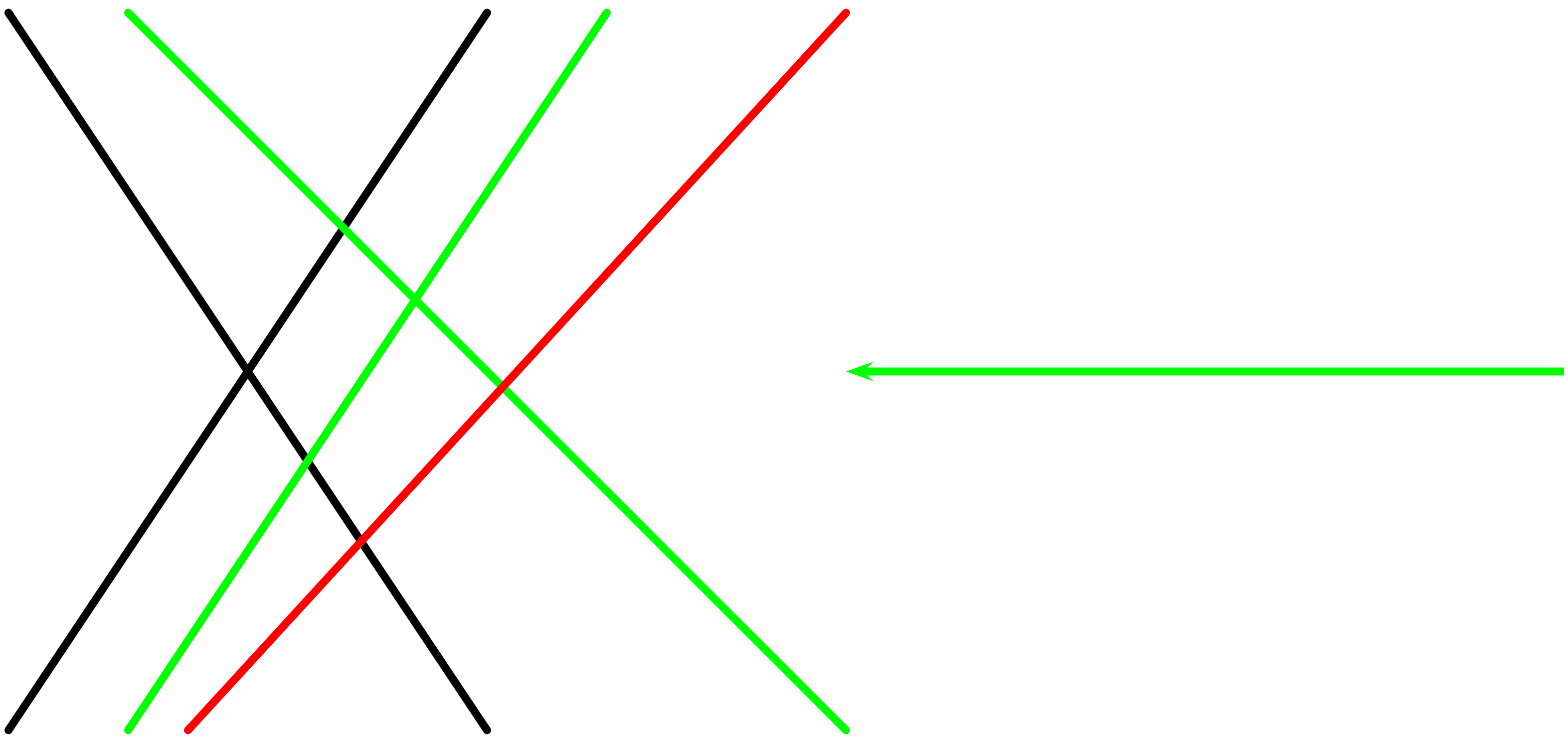
until $V = \emptyset$

Example

$r = 2$, "small enough" $= 1$

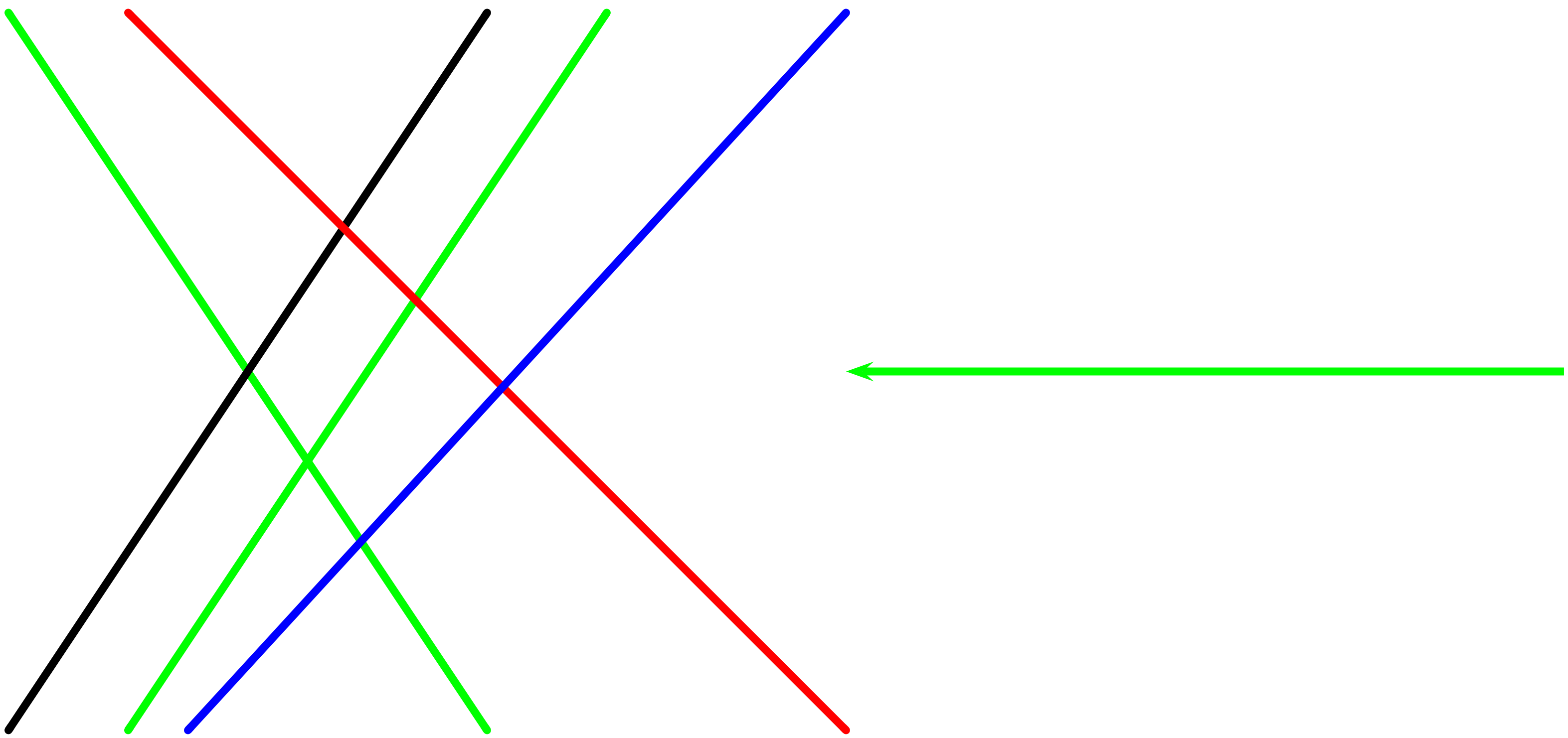


$r = 2$, “small enough” $= 1$

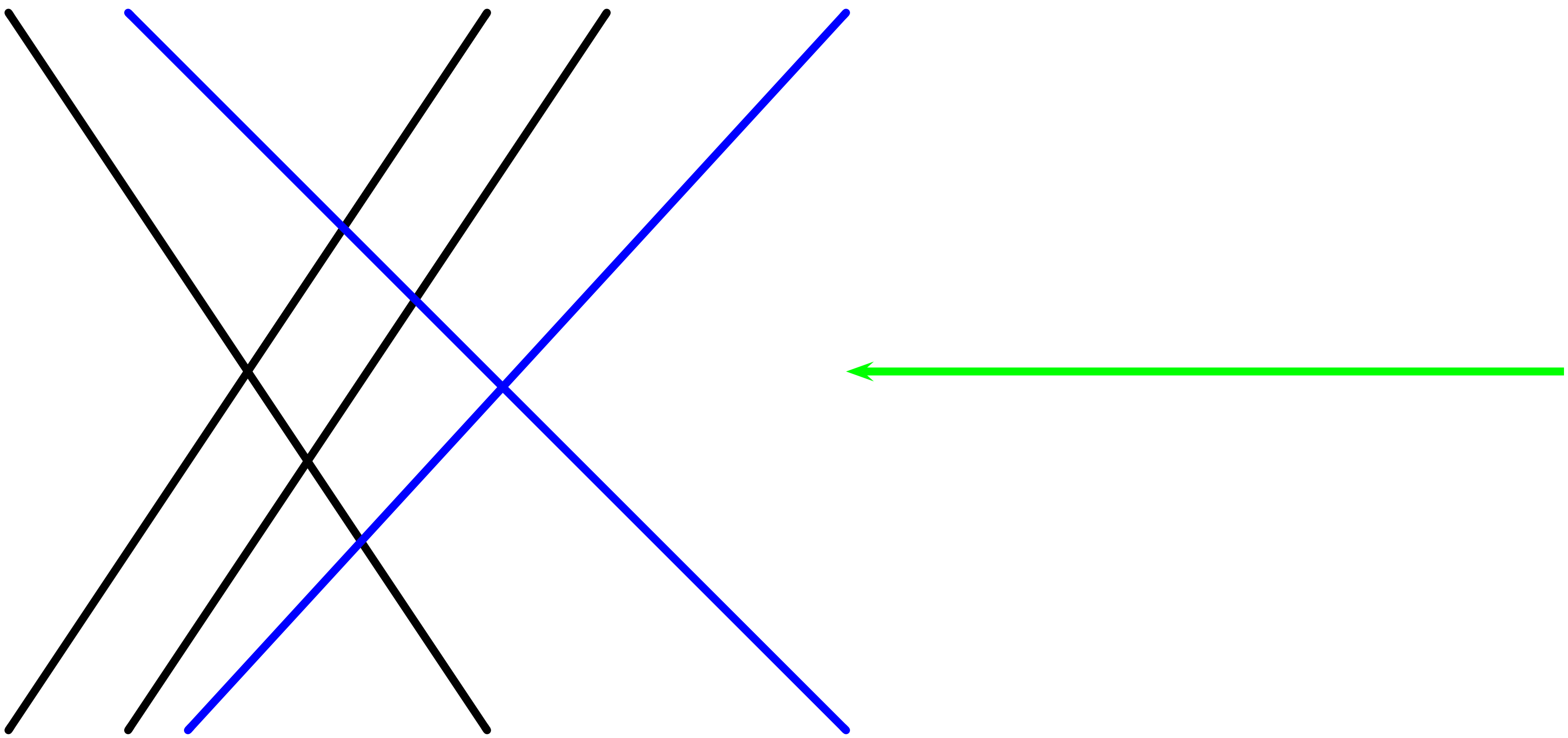


Example

$r = 2$, “small enough” $= 1$



$r = 2$, "small enough" $= 1$



Analysis

Bounding the number of iterations until a successful one

- number of violated constraints is nd/r

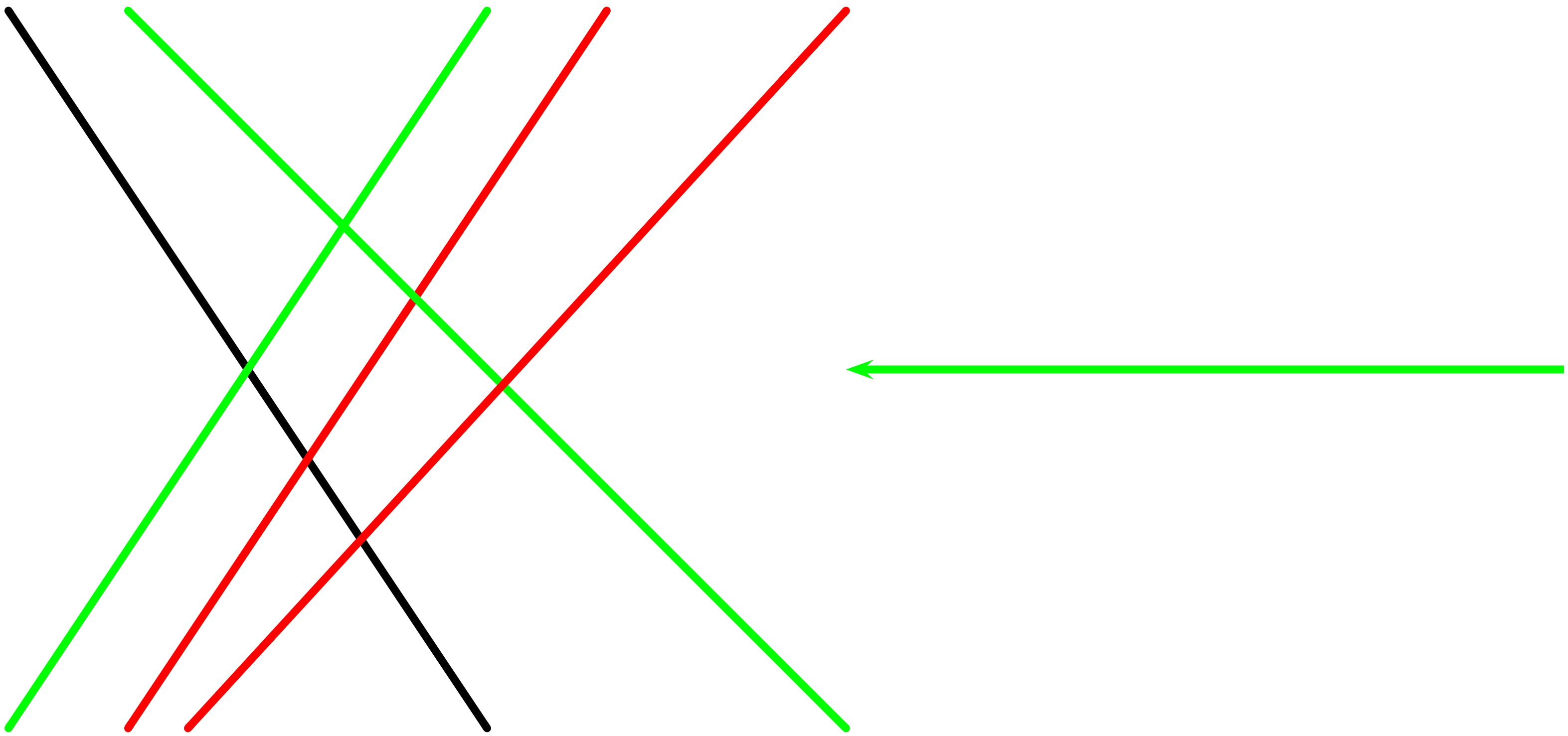
$$E[|V|] = n \,\mathrm{Pr}(constraint\ is\ violated) \leq n(d/(r+1))$$

$$\boxed{\text{“small enough”} \;:=\; 2nd/r}$$

- number of iterations until a successful one is 2

$$\mathrm{Pr}(|V| \geq 2E[|V|]) < 1/2$$

Bounding the number of successful iterations



number of successful iterations is d

Running time

- number of constraints in recursive call is $2nd^2/r$

$$r + |G| = r + d|V| = r + 2nd^2/r$$

- running time is

$$T_1(n) \leq 2dT_1(r + 2nd^2/r) + O(d^2n)$$

$$\boxed{r{:=}d\sqrt{2n}}$$

- base case is $n = O(d^2)$

$$n = d\sqrt{2n} \Longrightarrow n = 2d^2$$

Plug in Seidel

$$S(n,d) = O(d!n)$$

- running time of SloveLP1 is $O(d^2n + d^2d!\sqrt{n})$

$$\begin{aligned} T_1(n) &= 2dS(d\sqrt{n}, d) + O(d^2n) \\ &= O(d^2n + d^2d!\sqrt{n}) \end{aligned}$$

Improving the Algorithm

Reduce sample size

Instead of forcing the violated constraints,
include them in random sample with higher probability!

H is a multiset

SolveLP2(H)

repeat

$R :=$ random subset of H of size r

$v :=$ SolveLP2(R)

$V :=$ multiset of constraints violated by v

if $|V|$ is small enough, then $H := H \cup V$

until $V = \emptyset$

Bounding the number of iterations until a successful one

- number of violated constraints is $|H|d/r$

“small enough” $:= 2|H|d/r$

- number of iterations until a successful one is 2

Bounding the number of successful iterations

Fix a basis B of H.

- in one iteration, multiplicity of a constraint in B is doubled
- in kd iterations, $|B|$ is at least 2^k

- in one iteration, size of H increases by $2|H|d/r$
- in kd iterations, $|B|$ is at most $n\,\exp(2d^2/r)^k$

$$|B| \leq |H| \leq n(1+2d/r)^{kd} < n\,\exp(2kd^2/r)$$

Bounding the number of successful iterations

$$\boxed{2^k < n\,\mathrm{exp}(2d^2/r)^k}$$

- algorithm will terminate when inequality is violated
- minimize r subject to $2 \geq exp(2d^2/r)$

$$\boxed{r = O(d^2)}$$

- minimize k subject to $2^k \geq ne^{k/2}$

$$\boxed{k = O(logn)}$$

Running Time

- running time is

$$T_2(n) \le O(d \log n T_2(d^2) + d^2 n \log n)$$

- base case is $n = O(d^2)$

Plug in Seidel and plug into SolveLP1

$$S(n,d) = O(d!n)$$

running time is $O(d!d^3 \log n + d^3\sqrt{n} \log n + d^2 n)$

$$\begin{aligned} T(n) &= O(dT_2(d\sqrt{n}) + d^2 n) \\ &= O(d(d \log n S(d^2, d) + d^2\sqrt{n} \log n) + d^2 n) \\ &= O(d!d^3 \log n + d^3\sqrt{n} \log n + d^2 n) \end{aligned}$$

Modify Seidel!

Seidel(H)
  if $|H| = d$, return H
  else
    Pick a random constraint $h \in H$
    B := Seidel($H - \{h\}$)
    if B violates h, project constraints onto h and solve
    else return B

Algorithm: SolveLP3

SolveLP3(H,B)

Pick a random constraint in $h \in H - B$

B := SolveLP3($H - \{h\}$,B)

if h does not violate B, return B

else return SolveLP3(H,Basis($B \cup \{h\}$))