# Lecture 4 (part 4) — LP, Randomized Rounding & Randomized LP

> Course **2-INF-135/15 Pravdepodobnostné algoritmy**, LS 2025/26. Source slides: `04_metody78.pdf` (24 pages). Continues the **methods** block. This deck is the **randomized-rounding** method and its richer cousin **semidefinite rounding**, followed by a separate topic: using randomization to **solve the LP itself**.

## Where we are

The methods block has been a catalogue of *tricks*. Two more, both built on **linear programming**:

> **Method A — randomized rounding.** Hard combinatorial problems are integer programs. Drop the integrality (*relax*), solve the easy continuous problem, then turn the fractional answer back into an integral one by **flipping biased coins**.
>
> **Method B — randomized LP solving.** The LP we lean on is itself only "easy" in theory. In small dimension we can solve it *very* fast by feeding the constraints in **random order**.

The unifying object is the LP. Part A *uses* an LP oracle as a subroutine; part B *builds* one. Keep them mentally separate — they answer different questions.

---

## Part A — Relaxation and rounding

### A.0 The relaxation–rounding recipe

The basic LP task: minimize $c^T x$ subject to $Ax \leq b$, $x \geq 0$.

**Complexity of LP.**

- **Over** $\mathbb{R}$ (continuous $x$): the **simplex** method is polynomial *on average* but exponential in the worst case; ellipsoid/interior-point methods are polynomial. So a real LP is "easy".
- **Over** $\mathbb{Z}$ (integer programming): only exponential algorithms are known — integer programming is **NP-hard**.

That gap is the whole game. When a problem is naturally an *integer* program, we do:

1. **Relax** — replace each discrete constraint (e.g. $y_i \in \{0,1\}$) by a continuous one ($0 \le y_i \le 1$). Call it $\mathbf{LP}_{\text{rel}}$.
2. **Solve** the relaxation to get a fractional optimum $x^*_{\text{rel}}$.
3. **Round** $x^*_{\text{rel}}$ into a *feasible integral* $x$.

> **Randomized rounding** = step 3 done with coins: read each fractional value $x^*_i \in [0,1]$ as a **probability** and set the integer bit by a coin flip with that bias.

Two things must be checked after rounding: **feasibility** (is the rounded $x$ still a valid solution?) and **cost** (how much worse than $c(x^*)$, hence than the true integral optimum?). The relaxed optimum is always a *lower bound* on the integral optimum (the integral solutions are a subset of the feasible region), so any factor we lose against $c(x^*)$ is also a factor against the true OPT — that is exactly what an **approximation ratio** is.

---

### A.1 SetCover — the model problem

**Input.** A universe $X = \{x_1, \dots, x_n\}$, a family of sets $S = \{S_1, \dots, S_m\}$ with $S_i \subseteq X$, and weights $w : S \to \mathbb{R}^+$.

**Output.** A choice $I \subseteq \{1, \dots, m\}$ minimizing $\sum_{i \in I} w(S_i)$ such that $\bigcup_{i \in I} S_i = X$ — cover every element as cheaply as possible.

**As an integer program.** Let $y_i \in \{0,1\}$ indicate "set $S_i$ is chosen".

$$\min \; c(y) = \sum_{i=1}^{m} y_i \, w(S_i) \quad \text{subject to}$$

$$\textbf{(1)} \quad \forall x \in X : \sum_{i \,:\, x \in S_i} y_i \ge 1 \qquad\qquad \textbf{(2)} \quad y_i \in \{0,1\}.$$

Constraint **(1)** says *every element is in at least one chosen set*. The relaxation $\mathbf{LP}_{\text{rel}}$ replaces **(2)** with $0 \le y_i \le 1$.

---

### A.2 SetCover — deterministic rounding (factor $f$)

A warm-up before the randomized version, to see what "rounding" even means.

Let $y^* = (y^*_1, \dots, y^*_m)$ be the optimal relaxed solution and define the **frequency**

$$f = \max_i \{\text{number of sets that contain } x_i\}$$

(the most-covered element's multiplicity). Round by a fixed threshold:

$$y_i = 1 \iff y_i^* \geq \tfrac{1}{f}.$$

**Claim: $y$ is feasible.** Take any element $x$. It lies in at most $f$ sets, and by constraint (1) the fractional values on those sets sum to $\geq 1$. With $\leq f$ nonnegative numbers summing to $\geq 1$, **at least one** must be $\geq 1/f$ — so that set gets rounded up to $1$ and covers $x$.

**Claim: $c(y) \leq f \cdot c(y^*)$.** Whenever $y_i = 1$ we had $y_i^* \geq 1/f$, i.e. $1 \leq f y_i^*$; and when $y_i = 0$ trivially $0 \leq f y_i^*$. So termwise $y_i \leq f y_i^*$, hence

$$c(y) = \sum_i y_i w(S_i) \leq f \sum_i y_i^* w(S_i) = f\, c(y^*).$$

> **Deterministic rounding gives an $f$-approximation.** Clean, but $f$ can be as large as $m$. Randomization will replace $f$ by $O(\ln n)$ — far better when sets are large.

---

### A.3 SetCover — randomized rounding ($O(\ln n)$, and it is optimal)

Now read the fractional values as **probabilities**. With $y^*$ the relaxed optimum, define a random $y$ by

$$y_i = 1 \quad \text{with probability } y_i^*, \quad \text{independently.}$$

**Cost is right in expectation.** By linearity,

$$E[c(y)] = \sum_i \Pr[y_i = 1]\, w(S_i) = \sum_i y_i^*\, w(S_i) = c(y^*).$$

So one rounding matches the relaxed optimum *on average*. The danger is **feasibility** — a single rounding may leave some element uncovered.

**How likely is an element to be missed?** Let $S_{i_1}, \ldots, S_{i_k}$ be all sets containing $x_i$. Independence gives

$$\Pr[x_i \text{ not covered}] = \prod_{j=1}^{k} (1 - y_{i_j}^*) \leq \prod_{j=1}^{k} e^{-y_{i_j}^*} = e^{-\sum_j y_{i_j}^*} \leq e^{-1},$$

using $1 - t \leq e^{-t}$ and the LP constraint $\sum_j y_{i_j}^* \geq 1$. (Equivalently, fixing the sum to $1$, the product $\prod (1 - y_{i_j}^*)$ is largest when all are equal to $1/k$, giving $(1 - 1/k)^k \leq e^{-1}$.)

So one rounding covers each element with probability $\geq 1 - 1/e \approx 0.63$ — good but not enough.

**Amplify by stacking roundings.** Generate $c \ln n$ independent roundings $y^1, \dots, y^{c \ln n}$ and take their **union** (choose a set if *any* copy chose it). Then

$$\Pr[x_i \text{ covered by no } y^j] \le \left(e^{-1}\right)^{c \ln n} = n^{-c} \le \frac{1}{2n}$$

for a suitable constant $c$. Union-bounding over all $n$ elements,

$$\Pr[\exists\, x_i \text{ covered by no } y^j] \le n \cdot \frac{1}{2n} = \frac{1}{2}.$$

So with probability $\ge \frac{1}{2}$ the union is a **feasible** cover. Its expected weight is $\le c \ln n \cdot c(y^*)$ (each of the $c \ln n$ roundings has expected cost $c(y^*)$). Combining the two events (re-running the whole thing a constant number of times) yields a feasible cover of weight
$O(\ln n) \cdot c(y^*) = O(\ln n) \cdot \mathrm{OPT}$.

> **Punchline (professor-pleaser).** Randomized rounding gives an $O(\ln n)$-approximation for SetCover — and **this is essentially optimal**: unless $\mathrm{P} = \mathrm{NP}$, no polynomial algorithm beats ratio $(1 - o(1)) \ln n$ (Feige; Dinur–Steurer). The relaxation discards exactly the right amount of structure: the LP optimum is a *fractional cover*, and reading it as a probability distribution lands you on the inapproximability threshold.

---

### A.4 MaxCut — an integer *quadratic* program

Randomized rounding of a *linear* program (SetCover) was the easy case. MaxCut forces us to a richer relaxation.

**Input.** Undirected graph $G = (V, E)$, weights $w : E \to \mathbb{Q}^+$. **Output.** A cut $(S, \bar{S})$ of **maximum** total crossing weight $\max \sum_{e \in S \times \bar{S}} w(e)$.

Use a $\pm 1$ indicator per vertex, $y_i \in \{-1, +1\}$:

$$(S, \bar{S}) = \big(\{v_i : y_i = 1\},\ \{v_i : y_i = -1\}\big).$$

The product $y_i y_j$ reads off the relationship:

$$y_i y_j = \begin{cases} -1 & v_i, v_j \text{ on different sides (edge cut)}, \\ +1 & v_i, v_j \text{ on the same side.} \end{cases}$$

So $\frac{1}{2}(1 - y_i y_j)$ is exactly the indicator "edge $ij$ is cut", and MaxCut becomes a **quadratic** integer program:

$$\max \frac{1}{2} \sum_{1 \le i < j \le n} w_{i,j}\,(1 - y_i y_j) \quad \text{s.t.} \quad y_i^2 = 1,\ \ y_i \in \mathbb{Z}.$$

The constraint $y_i^2 = 1$ is what makes this *not* an LP — it is genuinely quadratic, and MaxCut is NP-hard. We need a relaxation that respects the quadratic structure. Enter **semidefinite programming**.

---

### A.5 Semidefinite programming (SDP) — the toolbox

An SDP optimizes over a **matrix** of variables instead of a vector:

- $Y$ is an $n \times n$ matrix of real variables;
- maximize a **linear** function of the entries $y_{i,j}$;
- subject to **linear** constraints on the $y_{i,j}$;
- **plus** $Y$ symmetric and **positive semidefinite** (PSD):

$$Y \succeq 0 \iff \forall X \in \mathbb{R}^n :\ X^T Y X \ge 0 \iff Y \text{ has all eigenvalues} \ge 0 \iff \exists U :\ Y = U^T U.$$

The last form is the one we exploit: a PSD matrix is a **Gram matrix** $Y = U^T U$, so its entries $y_{i,j} = u_i \cdot u_j$ are inner products of vectors $u_i$ (the columns of $U$). *PSD = "is realizable as a set of vectors".*

> **Solvability.** An SDP can be solved to within **additive error** $\varepsilon$ in time $\mathrm{poly}(n, \log(1/\varepsilon))$ (ellipsoid method — the PSD cone is convex). So, like LP, SDP is "easy".

**Standard form $S$.** Given symmetric real matrices $C, D_1, \dots, D_k \in M_n$ and scalars $d_1, \dots, d_k$:

$$S: \quad \max\ C \circ Y \quad \text{s.t.} \quad D_i \circ Y = d_i\ (1 \le i \le k), \quad Y \succeq 0,$$

where $A \circ B = \sum_{i,j} A[i,j]\,B[i,j]$ is the **Frobenius (entrywise) inner product**.

> **LP is a special case.** If $C, D_1, \dots, D_k$ are all **diagonal**, only the diagonal of $Y$ matters and $S$ degenerates to an ordinary LP. SDP strictly generalizes LP — it sees correlations between variables, which is exactly what a quadratic objective needs.

**Quadratic and vector programs.**

- A **quadratic program** optimizes a quadratic function (integer-valued variables) under quadratic constraints.
- A **vector program** has vector variables $\vec{v}_1, \dots, \vec{v}_n \in \mathbb{R}^n$ and optimizes a *linear* function of the inner products $\vec{v}_i \cdot \vec{v}_j$, under *linear* constraints on those inner products.

The pipeline is: **strict quadratic program** → **vector program** → **SDP**, and the SDP we can solve.

---

### A.6 MaxCut as a vector program (the relaxation)

Take the quadratic program and perform the substitution "**scalar → vector, product → inner product**":

$$y_i \in \{\pm 1\} \;\rightsquigarrow\; \vec{v}_i \in \mathbb{R}^n, \qquad y_i y_j \;\rightsquigarrow\; \vec{v}_i \cdot \vec{v}_j.$$

$$\textbf{quadratic (integer):} \quad \max \tfrac{1}{2}\textstyle\sum_{i<j} w_{i,j}(1 - y_i y_j), \quad y_i^2 = 1,\ y_i \in \mathbb{Z}$$

$$\textbf{vector relaxation } v\textbf{:} \quad \max \tfrac{1}{2}\textstyle\sum_{i<j} w_{i,j}(1 - \vec{v}_i \cdot \vec{v}_j), \quad \vec{v}_i \cdot \vec{v}_i = 1,\ \vec{v}_i \in \mathbb{R}^n.$$

The constraint $\vec{v}_i \cdot \vec{v}_i = 1$ says each $\vec{v}_i$ is a **unit vector** on the sphere $S^{n-1}$. We have relaxed "*each vertex gets a sign* $\pm 1$" (two antipodal points on a line) to "*each vertex gets a unit vector in* $\mathbb{R}^n$" — strictly more freedom, so

$$\mathrm{OPT}_v \geq \mathrm{OPT}.$$

As an SDP: set $x_{ij} = \vec{v}_i \cdot \vec{v}_j$, so the Gram matrix $Y = (x_{ij}) = U^T U$ (column $i$ of $U$ is $\vec{v}_i$) is PSD. Solve the SDP, recover the vectors $\vec{v}_i$ from $Y = U^T U$. **Now we must round vectors back to $\pm 1$.**

---

### A.7 Goemans–Williamson rounding — the random hyperplane

We have a fractional optimum: unit vectors $x_1^*, \ldots, x_n^*$ with value $\mathrm{OPT}_v$. Let $\Theta_{i,j}$ be the **angle** between $x_i^*$ and $x_j^*$; since they are unit vectors,

$$\cos\Theta_{i,j} = \frac{x_i^* \cdot x_j^*}{\|x_i^*\|\,\|x_j^*\|} = x_i^* \cdot x_j^*.$$

The pair's contribution to the relaxed objective is $\frac{w_{i,j}}{2}(1 - \cos\Theta_{i,j})$. Reading the geometry:

$$\Theta = 0 \Rightarrow (1 - \cos\Theta) = 0, \qquad \Theta = \tfrac{\pi}{2} \Rightarrow 1, \qquad \Theta = \pi \Rightarrow 2.$$

So the relaxation *rewards* pairs that point in **opposite** directions (large angle). A good rounding should therefore put vertices with a large angle on **opposite sides** of the cut. The trick:

> **Cut with a random hyperplane through the origin.** Pick a uniformly random unit vector $p \in_R S^{n-1}$ (sample each coordinate as a standard Gaussian, then normalize) and set
> $S = \{v_i : x_i^* \cdot p \geq 0\}, \qquad \bar{S} = V \setminus S.$

The hyperplane with normal $p$ slices the sphere; vertices land in $S$ or $\bar{S}$ by which side their vector falls on.

**Separation probability.**

$$\boxed{\Pr[v_i, v_j \text{ separated}] = \frac{\Theta_{i,j}}{\pi}.}$$

*Why.* Project $p$ onto the 2-D plane spanned by $x_i^*, x_j^*$ and let $r$ be that projection; only this plane matters for separating $i, j$. The two vectors split the circle, and the hyperplane separates them exactly when $r$ lands in one of the two arcs "between" them — total arc $2\Theta_{i,j}$ out of $2\pi$, giving $2\Theta_{i,j}/2\pi = \Theta_{i,j}/\pi$.

---

### A.8 The $0.878$ ratio

Let $W$ be the (random) weight of the cut produced. By linearity and the separation probability,

$$E[W] = \sum_{i<j} w_{i,j} \frac{\Theta_{i,j}}{\pi}, \qquad \text{compare} \qquad \mathrm{OPT}_v = \sum_{i<j} \frac{w_{i,j}}{2}(1 - \cos\Theta_{i,j}).$$

We want to compare the **per-pair** ratio of "what rounding gets" to "what the relaxation counts". Define the worst-case ratio over all angles:

$$\alpha = \min_{0<\Theta\le\pi} \frac{\Theta/\pi}{\frac{1}{2}(1-\cos\Theta)} = \min_{z\in[-1,1)} \frac{2\arccos z}{\pi(1-z)} \geq 0.87856\ldots$$

(substituting $z = \cos\Theta$). By the definition of $\alpha$, **every** term satisfies $\frac{\Theta_{i,j}}{\pi} \geq \alpha \cdot \frac{1-\cos\Theta_{i,j}}{2}$, so summing,

$$E[W] = \sum_{i<j} w_{i,j}\frac{\Theta_{i,j}}{\pi} \geq \alpha \sum_{i<j} \frac{w_{i,j}}{2}(1-\cos\Theta_{i,j}) = \alpha\,\mathrm{OPT}_v \geq \alpha\,\mathrm{OPT}.$$

> **The Goemans–Williamson constant** $\alpha \approx 0.87856$ is the minimum of a one-variable function — it comes purely from the geometry of "angle vs. chord", not from the graph. The deep point: a *random* hyperplane recovers a constant fraction $\alpha$ of the relaxed value pair-by-pair, with no dependence on $n$.

---

### A.9 From expectation to high probability

$E[W] \geq \alpha \operatorname{OPT}$ is only an *average*. For a maximization Monte Carlo we must show a single run lands near its mean with constant probability — then repeat.

Let $T = \sum_{e \in E} w(e)$ be the total weight, and write $E[W] = aT$ (so $a = E[W]/T$). We bound $p = \Pr[W < (1-\varepsilon)aT]$. Since always $W \leq T$, split the expectation by the bad event:

$$aT = E[W] \leq p \cdot (1-\varepsilon)aT + (1-p) \cdot T.$$

Solving for $p$,

$$p \leq \frac{1-a}{1-a+a\varepsilon} = 1 - \frac{a\varepsilon}{1-a+a\varepsilon}.$$

Now lower-bound $a$. A random/greedy cut already gives $\geq T/2$, and $aT = E[W] \geq \alpha \operatorname{OPT}_v \geq \alpha \operatorname{OPT} \geq \alpha \frac{T}{2}$, so $a \geq \alpha/2$. Plugging in, $p \leq 1 - c$ for a constant $c = \Theta(\varepsilon) > 0$.

**Amplify.** Run $1/c$ independent rounds and keep the best cut $W'$:

$$\Pr\left[W' \geq (1-\varepsilon)aT\right] \geq 1 - (1-c)^{1/c} \geq 1 - e^{-1}.$$

Finally, since $aT \geq \alpha \operatorname{OPT} > 0.87856 \operatorname{OPT}$ and $\alpha = 0.878567\ldots$ leaves slack, we can pick $\varepsilon$ so small that $(1-\varepsilon)aT \geq 0.87856 \operatorname{OPT}$.

> **Theorem (Goemans–Williamson, 1995).** There is a randomized approximation algorithm for MaxCut with approximation ratio $0.87856$.

> **Professor-pleasers.**
>
> - The relaxation is the engine: an *SDP* relaxation is strictly stronger than any *LP* relaxation for MaxCut, because the quadratic objective lives on inner products, and only PSD matrices encode "these are real vectors".
> - The rounding is one random hyperplane — astonishingly, it loses only a factor $0.878$ uniformly.
> - **It is conjecturally optimal:** under the *Unique Games Conjecture* (Khot et al.), no polynomial algorithm beats $0.87856$ for MaxCut. As with SetCover, randomized rounding lands exactly on the hardness threshold.

---

## Part B — Solving the LP itself by randomization

Part A *used* an LP/SDP solver as a black box. Part B *builds* a fast solver for the case that matters in computational geometry: **few dimensions $d$, many constraints $n$.** The goal is running time **linear in $n$**, with the (possibly bad) dependence isolated in $d$.

Geometric picture: we seek the *lowest* point (minimize the objective, say $x_1$) in the intersection of $n$ halfspaces in $\mathbb{R}^d$. Adding a constraint can only **raise** the optimum (the feasible region shrinks) — monotonicity we lean on throughout.

### B.0 Vocabulary (read once, refer back)

- $H$ — the set of constraints; $H_+$ — the trivial constraints $x \geq 0$.
- $F_H$ — feasible region for the constraints in $H$.
- $v_H$ — the optimal point in $F_H$ (and, by abuse, its objective value).
- **Basis** $B$ — a *minimal* set of constraints that determines $v_B$: $\forall B' \subsetneq B:\ v_{B'} < v_B$. In $\mathbb{R}^d$ a basis has $\leq d$ constraints (a vertex is pinned by $d$ tight constraints).
- **Basis for** $H$ — the minimal basis defining $v_H$.
- $x$ **violates** $h$: $x$ fails constraint $h$ (write $x \notin h$, viewing $h$ as a halfspace).
- $h$ is **extreme** in $H$: $v_{H-h} < v_H$ (dropping $h$ strictly lowers the optimum — $h$ is "binding").

Assumptions: minimize w.r.t. $x_1$; $F_H \neq \emptyset$; each vertex is determined by exactly $d$ constraints (non-degeneracy).

---

### B.1 Seidel's incremental algorithm — backward analysis

**Idea.** Insert the $n$ constraints **one at a time in random order**, maintaining the current optimum $v$.

- $d = 1$**:** solve directly in $O(n)$.
- $d = 2$**:** pick $h \in_R H$, let $v$ be the optimum of the constraints seen so far.
  - If $v \in h$ (satisfies the new constraint): $v$ is still optimal — **do nothing**.
  - If $v \notin h$ (violates it): the new optimum must lie **on the line of** $h$. Project all other constraints onto $h$ and solve a $1$-D problem in $O(n)$.
- $d \geq 3$**:** analogously — if $v$ violates $h$, the new optimum lies on $h$'s hyperplane, giving a $(d-1)$-dimensional subproblem.

**The cost of one insertion (the clever part).** Expected insertion cost is

$$E[\text{insert}] = \left(1 - \tfrac{2}{n}\right) O(1) + \tfrac{2}{n}\, O(n) = O(1),$$

so $E[T(n, 2)] = O(n)$. Where does the $2/n$ come from? **Backward analysis.**

> **Backward analysis.** Look at the *final* configuration of $n$ constraints. Its optimum $v_H$ is pinned by the basis — at most $d = 2$ constraints. Now imagine the random insertion run **in reverse**: the last-inserted constraint is a *uniformly random* one of the $n$. We pay the expensive $O(n)$ reprojection only if that last constraint is one of the $\leq 2$ basis constraints — probability $\leq 2/n$.

No need to track the actual history: by symmetry of a random permutation, the last element is uniform, and only the $\leq d$ basis constraints are expensive. The general recurrence:

$$T(n,d) \leq \begin{cases} O(n) & d = 1, \\ O(d) & n = 1, \\ T(n-1,d) + O(d) + \dfrac{d}{n}\Big[O(nd) + T(n-1,d-1)\Big] & \text{otherwise,} \end{cases}$$

which solves to

$$T(n,d) = O(d! \cdot n).$$

> Linear in $n$ (good), but the $d!$ in the dimension is brutal. Fixing that $d!$ is the whole point of the next two algorithms.

---

### B.2 Matoušek–Sharir–Welzl (MSW) — keep a candidate basis

**Weakness of Seidel.** When $v$ violates $h$, Seidel *forgets everything* and rebuilds from scratch. **Idea (MSW):** also carry a **candidate basis** $C \subseteq H$ with $v_C \leq v_H$ — a running hint, a lower bound on the optimum that we refine.

```
MSW(H, C):                          // initial call: MSW(H ∪ H+, H+)
  1: if H = C then return C
  2: h ∈_R H − C                    // random constraint not yet in the hint
  3: (v_B, B) ← MSW(H − h, C)       // solve without h
  4: if v_B violates h              // h matters → fold it into the basis
       then return MSW(H, basis(B ∪ {h}))   // recompute basis, O(d²)
       else return (v_B, B)
```

**The key quantity: $\Pr[\text{recompute at line 4}]$.** This is $p = \Pr[v_B \notin h]$ for a random $h$.

- **Trivial bound:** $p \leq \dfrac{d}{|H - C|} = \dfrac{d}{n-d}$ (at most $d$ basis constraints out of $|H - C|$ candidates can be "binding").

- **Better:** if $C$ already contains $k$ extreme constraints, $p \le \dfrac{d-k}{n-d}$.

**Hidden dimension** — the potential that drives the analysis:

$$\Delta(H, C) = d - \#\{\text{necessary constraints in } C\},$$

where $h$ is **necessary in** $C$ if $v_{H-h} < v_C$ (removing $h$ would drop the optimum below the current candidate value, so $h$ is locked into every basis). Then

$$\Pr[v \notin h,\ h \in_R H - C] \le \frac{\Delta(H, C)}{n-d}.$$

$\Delta$ counts *how many basis constraints we have not yet pinned down* — it only **decreases** as the algorithm runs ($v_C$ never falls, necessary constraints stay necessary). Ordering the extreme constraints by $v_{H-e_1} \le \cdots \le v_{H-e_{t'}}$, adding $e_{k+i}$ to $C$ drops $\Delta$ by $i$, and each is equally likely to be picked — this is what makes the recurrences solvable.

**Recurrences** (let $\Delta(H, C) \le k$):

$$t_k(n) \le t_k(n-1) + 1 + \frac{1}{n-d} \sum_{i=1}^{\min\{k,\, n-d\}} t_{k-i}(n-1), \qquad t_k(d) = 0,$$

$$b_k(n) \le b_k(n-1) + \frac{1}{n-d} \sum_{i=1}^{\min\{k,\, n-d\}} \big(1 + b_{k-i}(n-1)\big), \qquad b_k(d) = 0,$$

where $t_k$ counts **violation tests** ($O(d)$ each) and $b_k$ counts **basis computations** ($O(d^2)$ each). By induction,

$$b_k(n),\ t_k(n) \le 2^k(n-d) \implies b_d(n+d),\ t_d(n+d) \le 2^d n \implies \text{time } O(d^2\, 2^d\, n).$$

> **Punchline.** MSW replaces Seidel's $d!$ by $2^d$ — and a sharper analysis (the original article) gives a **subexponential** expectation $E[\#\text{ops}] = e^{\,2\sqrt{d \ln(n/\sqrt{d})} + O(\sqrt{d} + \ln n)}$, the best known bound for combinatorial LP. The hint $C$ is the whole trick: never throw away progress — carry a lower bound and refine it.

---

### B.3 Clarkson 1 — solve by sampling

A different idea: most constraints are irrelevant, so **sample a few, solve the small LP, collect the violators, and grow a kept set** $S$.

```
Clarkson1(H, r):                              // find S ⊇ B, B = basis for H
  1: S ← ∅
  2: repeat
  3:    R ← sample(H - S, r)                   // r = min{ d√n , |H \ S| }
  4:    v ← v_{S ∪ R}  with basis B            // if |S ∪ R| ≤ 9d², use another solver
  5:    V ← { h ∈ H : v violates h }
  6:    if |V| ≤ 2√n then S ← S ∪ V
  7:    until V = ∅
  8: return (v, B)
```

**The sampling lemma.** With $S \subseteq H$, $R$ a random $r$-subset of $H \setminus S$, $m = |H \setminus S|$, and $V$ the constraints violated by $v_{R \cup S}$:

$$\boxed{E[|V|] \le \frac{d\,(m - r + 1)}{r - d}.}$$

With $r = d\sqrt{n}$ this is $\le \sqrt{n}$. So each round the violator set is small (the test "$|V| \le 2\sqrt{n}$" almost always passes), and:

- each successful round adds $\ge 1$ constraint of the **final basis** $B$ to $S$; since $|B| \le d$, there are $\le d$ successful rounds;
- hence $|S| \le d \cdot 2\sqrt{n} = O(d\sqrt{n})$ throughout.

**Why the lemma holds (the counting argument).** Let $C_H = \{v_{T \cup S} : T \subseteq H \setminus S\}$ be all optima over subsets, and $C_R = \{v_{T \cup S} : T \subseteq R\}$. For $x \in C_H$, let $n_x = |\{h \in H : x \text{ violates } h\}|$ and $i_x = \mathbf{1}[x = v_{R \cup S}]$. Then $E[|V|] = \sum_x n_x\, E[i_x]$ with

$$E[i_x] = \Pr[x = v_{R \cup S}] = \frac{\binom{m - n_x - d}{r - d}}{\binom{m}{r}}$$

— $R$ must contain $x$'s $d$ defining constraints and **avoid** the $n_x$ it violates. The quantity $\binom{m-n_x-d}{r-d-1} / \binom{m}{r}$ that appears after a rearrangement is precisely $\Pr[x \in C_R \text{ violates exactly one } h \in R]$; summed over $C_R$ it is the **expected number of points of $C_R$ that violate exactly one sampled constraint**, and geometrically there are $\le d$ such points. That $\le d$ is the source of the bound:

$$E[|V|] \le \frac{d\,(m - r + 1)}{r - d} \le \sqrt{n}.$$

> **Intuition.** A sample of size $r$ "catches" the optimum so well that it leaves only a $d/r$ fraction of constraints violated. Oversampling by a factor $d$ shrinks the leftover work to $\sqrt{n}$.

**Time.** In expectation Clarkson 1 solves $2(d+1)$ linear programs each with $\le (2d^2+1)\sqrt{n}$ constraints; "small" instances go to simplex in $d^{O(d)}$. Overall $O(d^2 n) + (\log n)^{\log d} + 2\,d^{O(d)}$ — i.e. $O(d^2 n)$ plus lower-order terms.

---

### B.4 Clarkson 2 — reweighting (multiplicative weights)

The improvement: instead of permanently moving violators into $S$, **double their weight** and sample proportionally to weight. Important constraints quickly become heavy and get picked.

```
Clarkson2(H):
  1: ∀h ∈ H : w_h ← 1
  2: repeat
  3:   W ← Σ_{h∈H} w_h
  4:   R ← sample(H, 9d²)  with  Pr[h chosen] = w_h / W
  5:   find v_R                          // e.g. simplex
  6:   V ← { h ∈ H : v_R violates h }
  7:   if Σ_{h∈V} w_h ≤ 2W / (9d - 1)  then  ∀h ∈ V : w_h ← 2 w_h
  8:   until V = ∅
  9: return v_R
```

**Per-round bound.** With $r = 9d^2$ the sampling lemma gives

$$E[|V|] \le \frac{d(m-r+1)}{r-d} \le \frac{d(n-9d^2+1)}{9d^2-d} \le \frac{dn}{9d^2-d}, \qquad E[w(V)] \le \frac{dW}{9d^2-d} = \frac{W}{9d-1}.$$

By **Markov**, the doubling condition in line 7 ($w(V) \le 2W/(9d-1)$) holds in **at least every second round** — so we make $\Omega$(rounds) genuine doublings.

**Number of rounds is $O(d \log n)$.** Two opposing potentials on the basis $B$:

- Every round with $V \neq \emptyset$ has **at least one basis constraint violated**, so $\ge 1$ basis constraint gets doubled. After $kd$ doublings, $w(B) = \sum_{h\in B} 2^{n_h}$ where $\sum_{h\in B} n_h \ge kd$; by convexity this is minimized when balanced, so $w(B) \ge d\,2^k$ — the basis weight grows **exponentially**.
- Meanwhile the **total** weight barely grows: each round multiplies $W$ by at most $1 + \frac{2}{9d-1}$ (only the violated weight, $\le 2W/(9d-1)$, doubles), so after $kd$ rounds $W \le n\big(1+\frac{2}{9d-1}\big)^{kd} \le n\,e^{\,2kd/(9d-1)}$.

Since $B \subseteq H$ forces $w(B) \le W$ always,

$$d\,2^k \le n\,e^{\,2kd/(9d-1)},$$

and for $k > O(\log n)$ the left side overtakes the right — a contradiction. Hence $k = O(\log n)$ and the number of rounds is $kd = O(d \log n)$.

> **Punchline (the multiplicative-weights idea).** Doubling the weight of every violated constraint forces the *truly important* (basis) constraints to gain weight **exponentially**, while the total weight rises only by a bounded factor per round. Because the part can't outweigh the whole, only $O(d \log n)$ rounds are possible. This is the **weighted-majority / multiplicative-weights** potential argument, here driving an LP solver.

**Final time.** Combining Clarkson 2 with MSW for the small subproblems:

$$O\left(d^2 n + e^{O(\sqrt{d \log d})}\right).$$

Linear in $n$, subexponential in $d$ — the headline result for randomized LP.

---

## Recurring themes

| Theme | Where it appeared |
|---|---|
| **Relax → solve → round** | A.0; SetCover (A.2–A.3), MaxCut (A.5–A.8) |
| **Read a fraction as a probability** | A.3 randomized rounding; A.7 random hyperplane |
| $1 - t \le e^{-t}$ **+ union bound** | A.3 coverage failure $\le 1/2$ |
| **Hitting the inapproximability threshold** | A.3 SetCover $(1-o(1)) \ln n$; A.9 MaxCut UGC-optimal $0.878$ |
| **SDP / Gram matrix = "vectors exist"** ( $Y = U^T U$) | A.5–A.6 |
| **Expectation → high probability** ($W \le T$, then amplify) | A.9 |
| **Backward analysis** (last inserted element is uniform) | B.1 Seidel |
| **Carry a hint, never discard progress** | B.2 MSW candidate basis |
| **Sampling lemma** ($E[\#\text{violators}] \le d \cdot$ oversampling) | B.3–B.4 Clarkson |
| **Multiplicative weights** (part can't outweigh whole) | B.4 Clarkson 2 |

The single sentence tying Part A and Part B together:

> **Linear programming is the bridge between "hard discrete optimum" and "easy continuous optimum": randomness crosses it in *both directions* — coins round a fractional LP optimum into a near-optimal integral one (rounding), and a random order of constraints turns LP solving itself into expected-linear time (LP-type algorithms).**