# Pearls of Algorithms

## Part 2: Randomized Algorithms and Probabilistic Analysis

Prof. Dr. Heiko Röglin
Institut für Informatik I
universitätbonn

Winter 2011/12

# Efficient Algorithms

**When is an algorithm considered efficient?**

# Efficient Algorithms

**When is an algorithm considered efficient?**

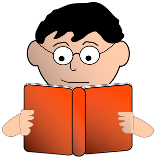
**Engineer**

The algorithm must be efficient in practice, i.e., it must solve practical instances in an appropriate amount of time.

# Efficient Algorithms

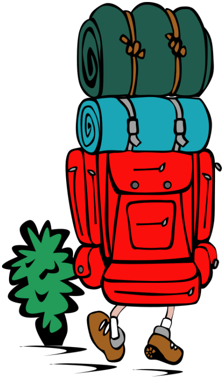
**When is an algorithm considered efficient?**

**Engineer**
The algorithm must be efficient in practice, i.e., it must solve practical instances in an appropriate amount of time.



**Theorist**
The algorithm must be efficient in the worst case, i.e., it must solve all instances in polynomial time.

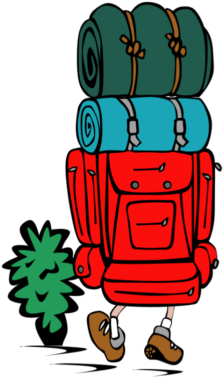
# The Knapsack Problem



## Knapsack problem (KP)

- **Input**
  - set of items $\{1, \ldots, n\}$
  - profits $p_1, \ldots, p_n$
  - weights $w_1, \ldots, w_n$
  - capacity $b$

# The Knapsack Problem

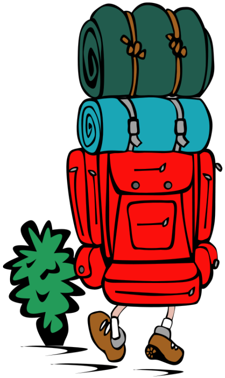


## Knapsack problem (KP)

- **Input**
  - set of items $\{1, \ldots, n\}$
  - profits $p_1, \ldots, p_n$
  - weights $w_1, \ldots, w_n$
  - capacity $b$
- **Goal**

  Find a subset of items that fits into the knapsack and maximizes the profit.

# The Knapsack Problem



## Knapsack problem (KP)

- **Input**
  - set of items $\{1, \dots, n\}$
  - profits $p_1, \dots, p_n$
  - weights $w_1, \dots, w_n$
  - capacity $b$
- **Goal**

  Find a subset of items that fits into the knapsack and maximizes the profit.
- **Formal description**

$$\begin{aligned} \text{max} \quad & p_1 x_1 + \dots + p_n x_n \\ \text{subject to} \quad & w_1 x_1 + \dots + w_n x_n \le b \\ & \text{and } x_i \in \{0, 1\} \end{aligned}$$

# Different Opinions

**Theorists say...**

- KP is NP-hard.
- FPTAS exists.

No efficient algorithm for KP, unless $P = NP$.

# Different Opinions

**Theorists say...**

- KP is NP-hard.
- FPTAS exists.

No efficient algorithm for KP, unless $P = NP$.

**Engineers say...**

- KP is easy to solve!
- Does not even require quadratic time.

There are very good heuristics for practical instances of KP.

# Reason for discrepancy

## Reason for discrepancy

- Worst-case complexity is too pessimistic!
- There are (artificial) worst-case instances for KP on which the heuristics are not efficient. These instances, however, do not occur in practice.
- This phenomenon occurs not only for KP, but also for many other problems.

# Reason for discrepancy

## Reason for discrepancy

- Worst-case complexity is too pessimistic!
- There are (artificial) worst-case instances for KP on which the heuristics are not efficient. These instances, however, do not occur in practice.
- This phenomenon occurs not only for KP, but also for many other problems.

## How to make theory more consistent with practice?

Find a more realistic performance measure.

# Reason for discrepancy

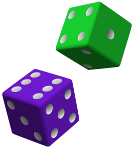
## Reason for discrepancy

- Worst-case complexity is too pessimistic!
- There are (artificial) worst-case instances for KP on which the heuristics are not efficient. These instances, however, do not occur in practice.
- This phenomenon occurs not only for KP, but also for many other problems.

## How to make theory more consistent with practice?

Find a more realistic performance measure.

## Average-case analysis

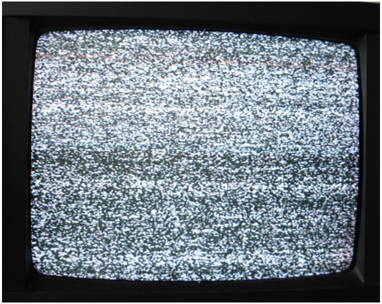


Is it realistic to consider the average case behavior instead of the worst case behavior?

# Random Inputs are not Typical

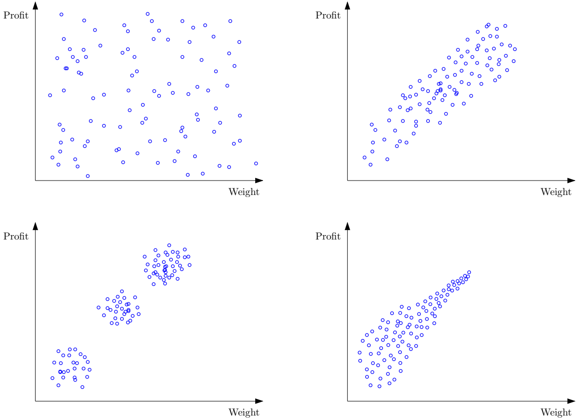
**Random inputs are not typical!**

If real-world data was random, watching TV would be very boring...



# What is a typical instance?

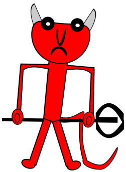
## What is a typical instance?

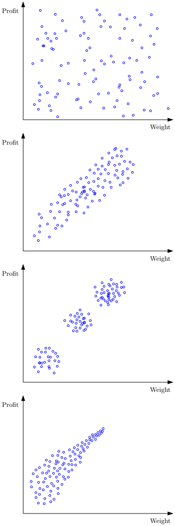



It depends very much on the concrete application. We cannot say in general what a typical instance for KP looks like.

# Smoothed Analysis

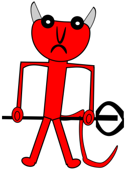
Step 1:
Adversary
chooses input $I$.

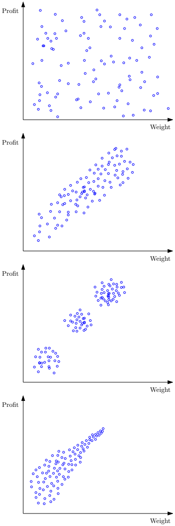




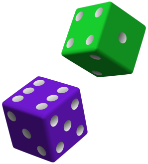
# Smoothed Analysis

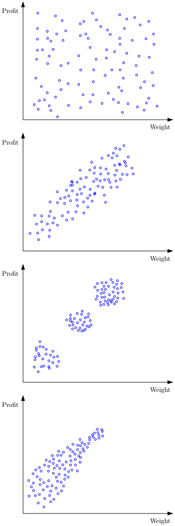
Step 1:
Adversary
chooses input $I$.



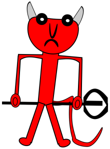



Step 2: Random
perturbation.
$I \to \mathrm{per}(I)$

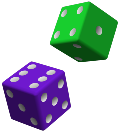




# Smoothed Analysis



Step 1:
Adversary
chooses input $I$.



Step 2: Random
perturbation.
$I \rightarrow \mathrm{per}(I)$

Smoothed Complexity = worst expected running time the adversary can achieve

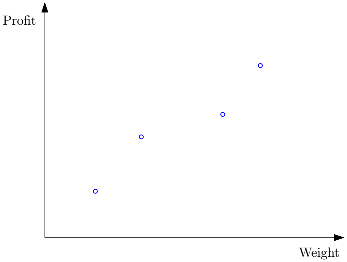
**Why do we consider this model?**

- First step alone: worst case analysis.
- Second step models random influences, e.g., measurement errors, numerical imprecision, rounding, ...
- So we have a combination of both: instances of any structure with some amount of random noise.

# Perturbation

## Example

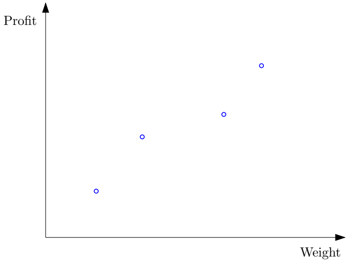
- Step 1: Adversary chooses all $p_i, w_i \in [0, 1]$ arbitrarily.

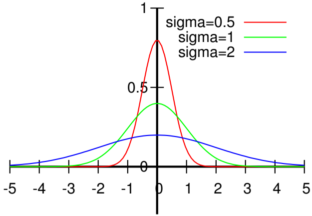


# Perturbation

## Example

- Step 1: Adversary chooses all $p_i, w_i \in [0, 1]$ arbitrarily.
- Step 2: Add an independent Gaussian random variable to each profit and weight.

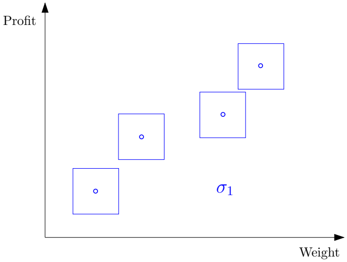


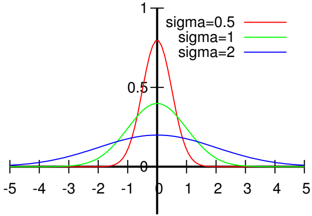



# Perturbation

## Example

- Step 1: Adversary chooses all $p_i, w_i \in [0, 1]$ arbitrarily.
- Step 2: Add an independent Gaussian random variable to each profit and weight.

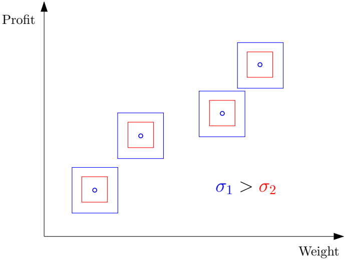


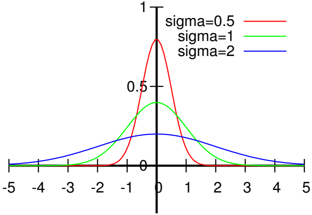



# Perturbation

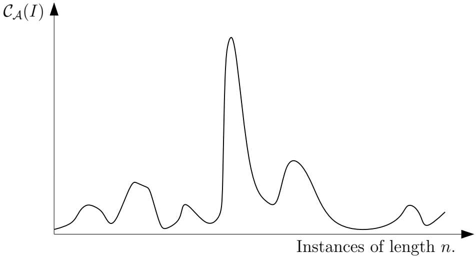
## Example

- Step 1: Adversary chooses all $p_i, w_i \in [0, 1]$ arbitrarily.
- Step 2: Add an independent Gaussian random variable to each profit and weight.



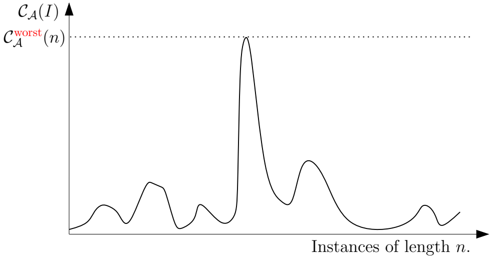



# Complexity Measures




$\mathcal{C}_{\mathcal{A}}(I)$ = running time of algorithm $\mathcal{A}$ on input $I$
$X_n$ = set of inputs of length $n$,

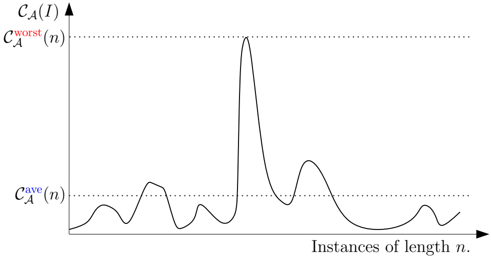
# Complexity Measures




$\mathcal{C}_{\mathcal{A}}(I) =$ running time of algorithm $\mathcal{A}$ on input $I$
$X_n =$ set of inputs of length $n$,

- $\mathcal{C}_{\mathcal{A}}^{\text{worst}}(n) = \max_{I \in X_n} (\mathcal{C}_{\mathcal{A}}(I))$

# Complexity Measures

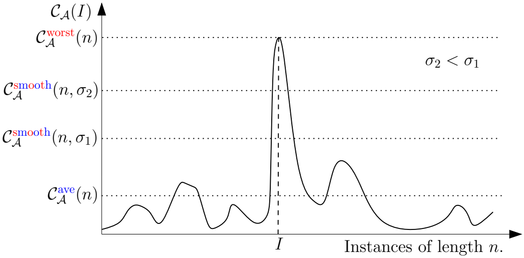



$\mathcal{C}_{\mathcal{A}}(I)$ = running time of algorithm $\mathcal{A}$ on input $I$

$X_n$ = set of inputs of length $n$, $\mu_n$ = prob. distribution on $X_n$

- $\mathcal{C}_{\mathcal{A}}^{\text{worst}}(n) = \max_{I \in X_n} (\mathcal{C}_{\mathcal{A}}(I))$
- $\mathcal{C}_{\mathcal{A}}^{\text{ave}}(n) = \mathbf{E}_{I \overset{\mu_n}{\leftarrow} X_n} [\mathcal{C}_{\mathcal{A}}(I)]$

# Complexity Measures

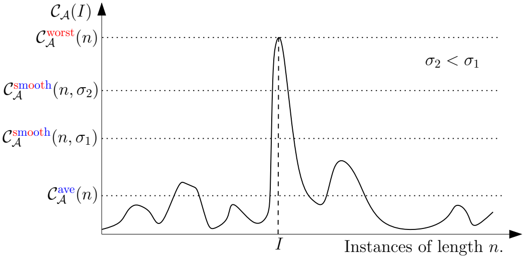



$\mathcal{C}_{\mathcal{A}}(I)$ = running time of algorithm $\mathcal{A}$ on input $I$

$X_n$ = set of inputs of length $n$, $\mu_n$ = prob. distribution on $X_n$

- $\mathcal{C}_{\mathcal{A}}^{\text{worst}}(n) = \max_{I \in X_n} (\mathcal{C}_{\mathcal{A}}(I))$
- $\mathcal{C}_{\mathcal{A}}^{\text{ave}}(n) = \mathbf{E}_{I \overset{\mu_n}{\leftarrow} X_n} [\mathcal{C}_{\mathcal{A}}(I)]$
- $\mathcal{C}_{\mathcal{A}}^{\text{smooth}}(n,\sigma) = \max_{I \in X_n} \mathbf{E} [\mathcal{C}_{\mathcal{A}} (\mathrm{per}_{\sigma}(I))]$

# Complexity Measures




$\mathcal{C}_{\mathcal{A}}(I)$ = running time of algorithm $\mathcal{A}$ on input $I$

$X_n$ = set of inputs of length $n$, $\mu_n$ = prob. distribution on $X_n$

- $\mathcal{C}_{\mathcal{A}}^{\text{worst}}(n) = \max_{I \in X_n} (\mathcal{C}_{\mathcal{A}}(I))$
- $\mathcal{C}_{\mathcal{A}}^{\text{ave}}(n) = \mathbf{E}_{I \overset{\mu_n}{\leftarrow} X_n} [\mathcal{C}_{\mathcal{A}}(I)]$
- $\mathcal{C}_{\mathcal{A}}^{\text{smooth}}(n, \sigma) = \max_{I \in X_n} \mathbf{E} [\mathcal{C}_{\mathcal{A}} (\text{per}_{\sigma}(I))]$

smoothed complexity low $\Rightarrow$ bad instances are isolated peaks

# Linear Programs

## Linear Programs (LPs)

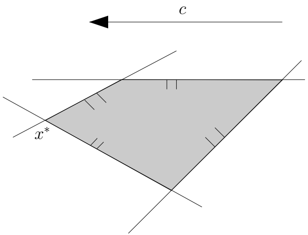
- variables: $x_1, \dots, x_d \in \mathbb{R}$.
- linear objective function:
  $\max c_1 x_1 + \ldots + c_n x_n$.

# Linear Programs

## Linear Programs (LPs)

- variables: $x_1, \ldots, x_d \in \mathbb{R}$.
- linear objective function:
  max $c_1 x_1 + \ldots + c_n x_n$.
- $n$ linear constraints:

$$\begin{array}{ccc} a_{1,1}x_1 & + \ldots + & a_{1,d}x_d \leq b_1 \\ & \vdots & \\ a_{n,1}x_1 & + \ldots + & a_{n,d}x_d \leq b_n \end{array}$$

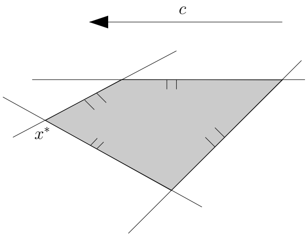


# Linear Programs

## Linear Programs (LPs)

- variables: $x_1, \ldots, x_d \in \mathbb{R}$.
- linear objective function:
  $\max c_1 x_1 + \ldots + c_n x_n$.
- $n$ linear constraints:

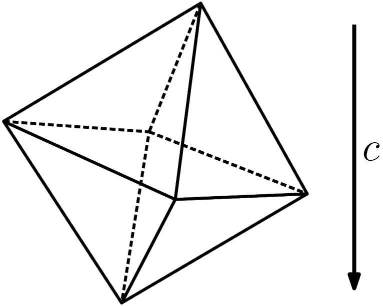
$$\begin{array}{ccc} a_{1,1}x_1 & + \ldots + & a_{1,d}x_d \leq b_1 \\ & \vdots & \\ a_{n,1}x_1 & + \ldots + & a_{n,d}x_d \leq b_n \end{array}$$




## Complexity of LPs

LPs can be solved in polynomial time by the ellipsoid method [Khachiyan 1979].

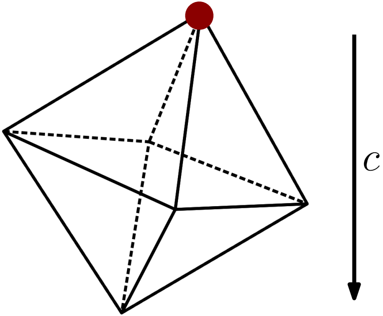
# Simplex Algorithm




**Simplex Algorithm**

- The simplex method walks along the vertices of the polytope in the direction of the objective function $c^T x$.

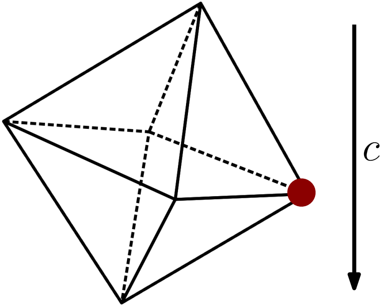
# Simplex Algorithm




## Simplex Algorithm

- The simplex method walks along the vertices of the polytope in the direction of the objective function $c^T x$.

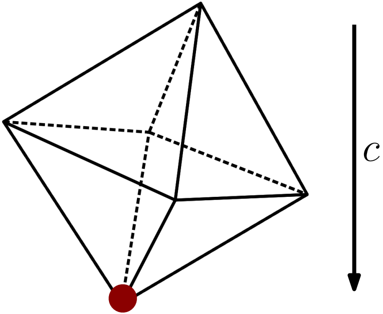
# Simplex Algorithm




**Simplex Algorithm**

- The simplex method walks along the vertices of the polytope in the direction of the objective function $c^T x$.

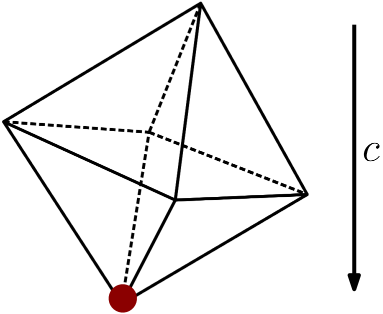
# Simplex Algorithm




**Simplex Algorithm**

- The simplex method walks along the vertices of the polytope in the direction of the objective function $c^T x$.

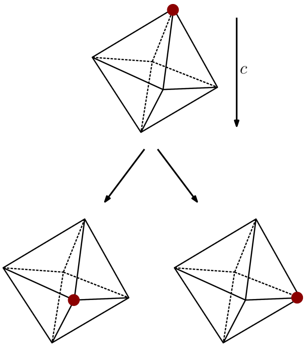
# Simplex Algorithm




**Simplex Algorithm**

- The simplex method walks along the vertices of the polytope in the direction of the objective function $c^T x$.
- Exponential in the worst case.
- Works well in practice.

# Pivot Rules



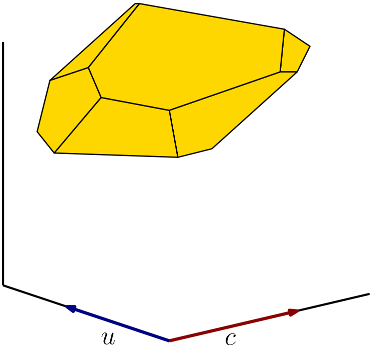

## Pivot Rules

- How is a better vertex on the polytope chosen if there are multiple options?
- Different pivot rules have been suggested:
  - random
  - steepest descent
  - shadow vertex pivot rule
  - ...

# Shadow Vertex Pivot Rule

## Shadow Vertex Pivot Rule

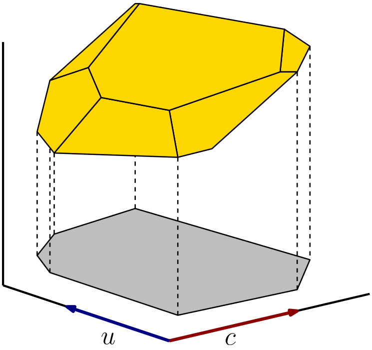
- Let $x_0$ be some vertex of the polytope.
- Compute $u \in \mathbb{R}^d$ such that $x_0$ maximizes $u^T x$.



# Shadow Vertex Pivot Rule

## Shadow Vertex Pivot Rule

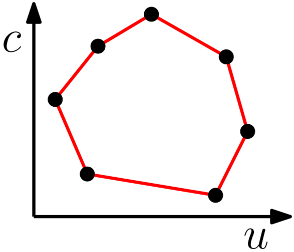
- Let $x_0$ be some vertex of the polytope.
- Compute $u \in \mathbb{R}^d$ such that $x_0$ maximizes $u^T x$.
- Project the polytope onto the plane spanned by $c$ and $u$.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

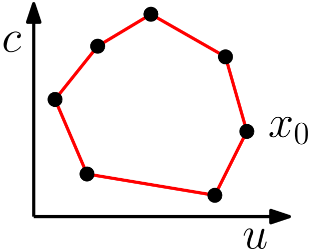
- The projection is 2-dimensional, that is, a polygon.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

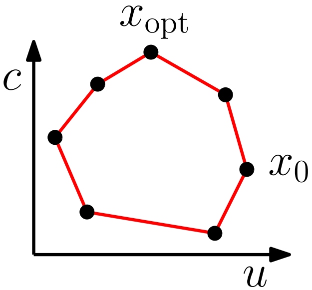
- The projection is 2-dimensional, that is, a polygon.
- $x_0$ is a vertex of the polygon.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

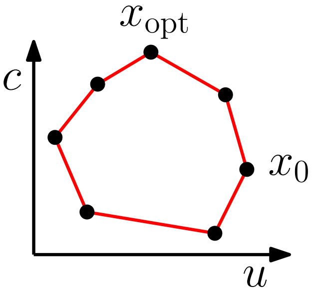
- The projection is 2-dimensional, that is, a polygon.
- $x_0$ is a vertex of the polygon.
- $x_{\mathrm{opt}}$ is a vertex of the polygon.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

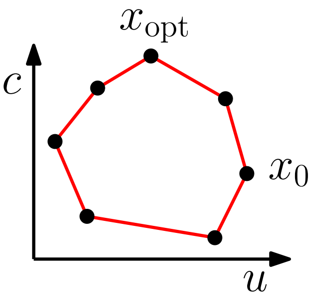
- The projection is 2-dimensional, that is, a polygon.
- $x_0$ is a vertex of the polygon.
- $x_{\mathrm{opt}}$ is a vertex of the polygon.
- Edges of the polygon correspond to edges of the polytope.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

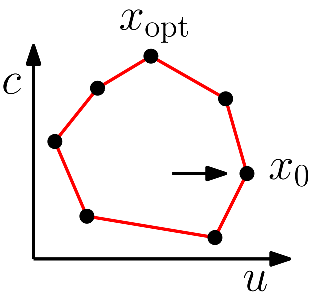
- In 2 dimension the simplex method is easy; it just follows the edges of the polygon.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

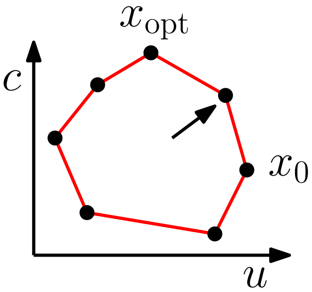
- In 2 dimension the simplex method is easy; it just follows the edges of the polygon.
- It starts at $x_0$...



# Shadow Vertex Pivot Rule

## 2-dimensional projection

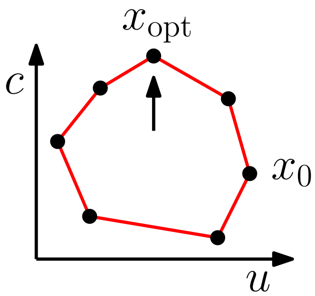
- In 2 dimension the simplex method is easy; it just follows the edges of the polygon.
- It starts at $x_0$...
- ... and follows the edges to $x_{\mathrm{opt}}$.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

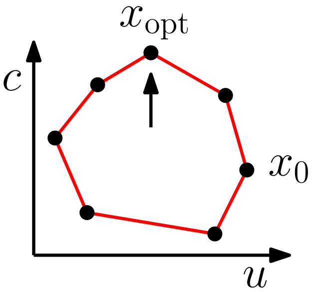
- In 2 dimension the simplex method is easy; it just follows the edges of the polygon.
- It starts at $x_0$...
- ... and follows the edges to $x_{\mathrm{opt}}$.



# Shadow Vertex Pivot Rule

## 2-dimensional projection

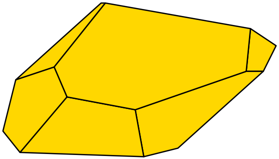
- In 2 dimension the simplex method is easy; it just follows the edges of the polygon.
- It starts at $x_0$...
- ... and follows the edges to $x_{\text{opt}}$.
- The polygon can have an exponential number of edges.



# Perturbed Linear Programs

## Perturbed LPs

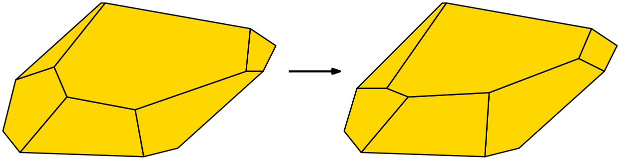
- Step 1: Adversary specifies arbitrary LP:
  $\max c^T x$ subject to $a_1^T x \leq b_1 \ldots a_n^T x \leq b_n$.
  W. l. o. g. $\|(a_i, b_i)\| = 1$.



# Perturbed Linear Programs

## Perturbed LPs

- Step 1: Adversary specifies arbitrary LP:
  $\max c^T x$ subject to $a_1^T x \leq b_1 \ldots a_n^T x \leq b_n$.
  W. l. o. g. $\|(a_i, b_i)\| = 1$.
- Step 2: Add Gaussian random variable with standard deviation $\sigma$ to each coefficient in the constraints.



# Smoothed Analysis of the Simplex Algorithm

**Theorem [Spielman and Teng 2001]**

The expected number of edges on the polygon is

$$O\left(\mathrm{poly}\left(n, d, \sigma^{-1}\right)\right) \quad .$$

# Smoothed Analysis of the Simplex Algorithm

**Theorem [Spielman and Teng 2001]**

The expected number of edges on the polygon is

$$O\left(\mathrm{poly}\left(n, d, \sigma^{-1}\right)\right) \quad .$$

The smoothed running time of the simplex algorithm with shadow vertex pivot rule is

$$O\left(\mathrm{poly}\left(n, d, \sigma^{-1}\right)\right) \quad .$$

# Smoothed Analysis of the Simplex Algorithm

**Theorem [Spielman and Teng 2001]**

The expected number of edges on the polygon is

$$O\left(\mathrm{poly}\left(n, d, \sigma^{-1}\right)\right) \quad .$$

The smoothed running time of the simplex algorithm with shadow vertex pivot rule is

$$O\left(\mathrm{poly}\left(n, d, \sigma^{-1}\right)\right) \quad .$$

Running time is polynomial in $n$, $d$, and $\sigma^{-1}$.

Already for small perturbation it is extremely unlikely to hit a bad instance.

# Improved Analysis

**Theorem [Vershynin 2006]**

The smoothed running time of the simplex algorithm with shadow vertex pivot rule is

$$O\left(\mathrm{poly}\left(\log n, d, \sigma^{-1}\right)\right) \quad .$$

Running time is only polylogarithmic in the number of constraints $n$.

# Overview of the coming Lectures

**Smoothed Analysis**

- 2-Opt heuristic for the traveling salesperson problem
- Nemhauser/Ullmann algorithm for the knapsack problem
- $k$-means clustering