

There are many practically important optimization problems that are NP-hard.

What can we do?

Heuristics

Exploit special structure of instances occurring in practice

Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimal

There are many practically important optimization problems that are NP-hard.

### What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

### What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

### What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

### What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

## Definition 57

An  $\alpha$ -approximation for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of  $\alpha$  of the value of an optimal solution.

## Why approximation algorithms?

- Approximation algorithms for hard problems.
- Approximation algorithms: theoretical base for solving hard problems.
- Provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithms.
- Approximation algorithms.

### Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.



## Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

### Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Definition 58

An optimization problem  $P = (\mathcal{I}, \text{sol}, m, \text{goal})$  is in **NPO** if

- ▶  $x \in \mathcal{I}$  can be **decided** in polynomial time
- ▶  $y \in \text{sol}(\mathcal{I})$  can be **verified** in polynomial time
- ▶  $m$  can be computed in polynomial time
- ▶  $\text{goal} \in \{\text{min}, \text{max}\}$

In other words: the decision problem **is there a solution  $y$  with  $m(x, y)$  at most/at least  $z$**  is in NP.

- ▶  $x$  is problem instance
- ▶  $y$  is candidate solution
- ▶  $m^*(x)$  cost/profit of an optimal solution

### Definition 59 (Performance Ratio)

$$R(x, y) := \max \left\{ \frac{m(x, y)}{m^*(x)}, \frac{m^*(x)}{m(x, y)} \right\}$$

### Definition 60 ( $r$ -approximation)

An algorithm  $A$  is an  $r$ -approximation algorithm iff

$$\forall x \in \mathcal{I} : R(x, A(x)) \leq r ,$$

and  $A$  runs in polynomial time.



## Definition 61 (PTAS)

A PTAS for a problem  $P$  from NPO is an algorithm that takes as input  $x \in \mathcal{I}$  and  $\epsilon > 0$  and produces a solution  $y$  for  $x$  with

$$R(x, y) \leq 1 + \epsilon .$$

The running time is polynomial in  $|x|$ .

approximation with arbitrary good factor... fast?

## Problems that have a PTAS

**Scheduling.** Given  $m$  jobs with known processing times; schedule the jobs on  $n$  machines such that the MAKESPAN is minimized.

## Definition 62 (FPTAS)

An FPTAS for a problem  $P$  from NPO is an algorithm that takes as input  $x \in \mathcal{I}$  and  $\epsilon > 0$  and produces a solution  $y$  for  $x$  with

$$R(x, y) \leq 1 + \epsilon .$$

The running time is polynomial in  $|x|$  and  $1/\epsilon$ .

approximation with arbitrary good factor... fast!

## Problems that have an FPTAS

**KNAPSACK.** Given a set of items with profits and weights choose a subset of total weight at most  $W$  s.t. the profit is maximized.

### Definition 63 (APX – approximable)

A problem  $P$  from NPO is in APX if there exist a constant  $r \geq 1$  and an  $r$ -approximation algorithm for  $P$ .

constant factor approximation...

## Problems that are in APX

**MAXCUT.** Given a graph  $G = (V, E)$ ; partition  $V$  into two disjoint pieces  $A$  and  $B$  s. t. the number of edges between both pieces is maximized.

**MAX-3SAT.** Given a 3CNF-formula. Find an assignment to the variables that satisfies the maximum number of clauses.

## Problems with polylogarithmic approximation guarantees

- ▶ Set Cover
- ▶ Minimum Multicut
- ▶ Sparsest Cut
- ▶ Minimum Bisection

There is an  $r$ -approximation with  $r \leq \mathcal{O}(\log^c(|x|))$  for some constant  $c$ .

Note that only for some of the above problem a matching lower bound is known.

## There are really difficult problems!

### Theorem 64

*For any constant  $\epsilon > 0$  there does not exist an  $\Omega(n^{1-\epsilon})$ -approximation algorithm for the maximum clique problem on a given graph  $G$  with  $n$  nodes unless  $P = NP$ .*

Note that an  $n$ -approximation is trivial.



## There are really difficult problems!

### Theorem 64

*For any constant  $\epsilon > 0$  there does not exist an  $\Omega(n^{1-\epsilon})$ -approximation algorithm for the maximum clique problem on a given graph  $G$  with  $n$  nodes unless  $P = NP$ .*

Note that an  $n$ -approximation is trivial.

**There are really difficult problems!**

### Theorem 64

*For any constant  $\epsilon > 0$  there does not exist an  $\Omega(n^{1-\epsilon})$ -approximation algorithm for the maximum clique problem on a given graph  $G$  with  $n$  nodes unless  $P = NP$ .*

Note that an  $n$ -approximation is trivial.

## There are weird problems!

Asymmetric  $k$ -Center admits an  $\mathcal{O}(\log^* n)$ -approximation.

There is no  $o(\log^* n)$ -approximation to Asymmetric  $k$ -Center unless  $NP \subseteq DTIME(n^{\log \log \log n})$ .

Class APX not important in practise.

Instead of saying **problem  $P$  is in APX** one says **problem  $P$  admits a 4-approximation**.

One only says that a problem is **APX-hard**.