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 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.

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 2

An α-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 α of the value of an optimal solution.

Minimization Problem:

Let $\mathcal{I}$ denote the set of problem instances, and let for a given instance $I \in \mathcal{I}$, $\mathcal{F}(I)$ denote the set of feasible solutions. Further let $\mathrm{cost}(F)$ denote the cost of a feasible solution $F \in \mathcal{F}$.

Let for an algorithm A and instance $I \in \mathcal{I}$, $A(I) \in \mathcal{F}(I)$ denote the feasible solution computed by A. Then A is an approximation algorithm with approximation guarantee $\alpha \geq 1$ if

$$\forall I \in \mathcal{I} : \mathrm{cost}(A(I)) \leq \alpha \cdot \min_{F \in \mathcal{F}(I)} \{\mathrm{cost}(F)\} = \alpha \cdot \mathrm{OPT}(I)$$

Maximization Problem:

Let $\mathcal{I}$ denote the set of problem instances, and let for a given instance $I \in \mathcal{I}$, $\mathcal{F}(I)$ denote the set of feasible solutions. Further let $\mathrm{profit}(F)$ denote the profit of a feasible solution $F \in \mathcal{F}$.

Let for an algorithm A and instance $I \in \mathcal{I}$, $A(I) \in \mathcal{F}(I)$ denote the feasible solution computed by A. Then A is an approximation algorithm with approximation guarantee $\alpha \le 1$ if

$$\forall I \in \mathcal{I} : \mathrm{cost}(A(I)) \ge \alpha \cdot \max_{F \in \mathcal{F}(I)} \{\mathrm{profit}(F)\} = \alpha \cdot \mathrm{OPT}(I)$$

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.

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.

What can we hope for?

Definition 3

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\epsilon\}$, such that A_ϵ is a $(1+\epsilon)$-approximation algorithm (for minimization problems) or a $(1-\epsilon)$-approximation algorithm (for maximization problems).

Many NP-complete problems have polynomial time approximation schemes.

What can we hope for?

Definition 3

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\epsilon\}$, such that A_ϵ is a $(1+\epsilon)$-approximation algorithm (for minimization problems) or a $(1-\epsilon)$-approximation algorithm (for maximization problems).

Many NP-complete problems have polynomial time approximation schemes.

What can we hope for?

Definition 3

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\epsilon\}$, such that A_ϵ is a $(1+\epsilon)$-approximation algorithm (for minimization problems) or a $(1-\epsilon)$-approximation algorithm (for maximization problems).

Many NP-complete problems have polynomial time approximation schemes.

There are difficult problems!

The class MAX SNP (which we do not define) contains optimization problems like maximum cut or MAX-3SAT.

Theorem 4

For any MAX SNP*-hard problem, there does not exist a polynomial-time approximation scheme, unless* $\mathrm{P} = \mathrm{NP}$.

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.

There are difficult problems!

The class $\mathrm{MAX\,SNP}$ (which we do not define) contains optimization problems like maximum cut or MAX-3SAT.

Theorem 4

For any $\mathrm{MAX\,SNP}$*-hard problem, there does not exist a polynomial-time approximation scheme, unless* $\mathrm{P} = \mathrm{NP}$.

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.

There are difficult problems!

The class $\mathrm{MAX\,SNP}$ (which we do not define) contains optimization problems like maximum cut or MAX-3SAT.

Theorem 4

For any $\mathrm{MAX\,SNP}$-hard problem, there does not exist a polynomial-time approximation scheme, unless $\mathrm{P} = \mathrm{NP}$.

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.

There are difficult problems!

The class $\mathrm{MAX\,SNP}$ (which we do not define) contains optimization problems like maximum cut or MAX-3SAT.

Theorem 4

For any $\mathrm{MAX\,SNP}$*-hard problem, there does not exist a polynomial-time approximation scheme, unless* $\mathrm{P} = \mathrm{NP}$*.*

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.

There are really difficult problems!

Theorem 5

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

Note that an $1/n$-approximation is trivial.

There are really difficult problems!

Theorem 5
For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{\epsilon-1})$-approximation algorithm for the maximum clique problem on a given graph G with n nodes unless $\mathrm{P} = \mathrm{NP}$.

Note that an $1/n$-approximation is trivial.

There are really difficult problems!

Theorem 5

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

Note that an $1/n$-approximation is trivial.