Chapter 34: P versus NP, A Gentle Introduction
(Version of 18th December 2022)

Pierre Flener

Department of Information Technology
Computing Science Division
Uppsala University
Sweden

Course 1DL231:
Algorithms and Data Structures 2 (AD2)

Outline

Outline

$P \stackrel{?}{=} NP$ (Cook, 1971; Levin, 1973)

This is one of the seven Millennium Prize problems of the Clay Mathematics Institute (Massachusetts, USA), each worth 1 million US$.

$P \stackrel{?}{=} NP$ (Cook, 1971; Levin, 1973)

This is one of the seven Millennium Prize problems of the Clay Mathematics Institute (Massachusetts, USA), each worth 1 million US$.

Informally:

- P = class of problems that need no search to be solved
  NP = class of problems that might need search to solve

$P \stackrel{?}{=} NP$ (Cook, 1971; Levin, 1973)

This is one of the seven Millennium Prize problems of the Clay Mathematics Institute (Massachusetts, USA), each worth 1 million US$.

Informally:

- P = class of problems that need no search to be solved
  NP = class of problems that might need search to solve
- P = class of problems with easy-to-compute solutions
  NP = class of problems with easy-to-check solutions

$P \stackrel{?}{=} NP$ (Cook, 1971; Levin, 1973)

This is one of the seven Millennium Prize problems of the Clay Mathematics Institute (Massachusetts, USA), each worth 1 million US$.

Informally:

- P = class of problems that need no search to be solved
  NP = class of problems that might need search to solve
- P = class of problems with easy-to-compute solutions
  NP = class of problems with easy-to-check solutions

Thus: Can search always be avoided ($P = NP$), or is search sometimes necessary ($P \neq NP$)?

$P \stackrel{?}{=} NP$ (Cook, 1971; Levin, 1973)

This is one of the seven Millennium Prize problems of the Clay Mathematics Institute (Massachusetts, USA), each worth 1 million US$.

Informally:

- P = class of problems that need no search to be solved
  NP = class of problems that might need search to solve
- P = class of problems with easy-to-compute solutions
  NP = class of problems with easy-to-check solutions

Thus: Can search always be avoided ($P = NP$), or is search sometimes necessary ($P \neq NP$)?

Problems that are solvable in polynomial time (in the input size) are considered tractable, or easy. Problems requiring non-polynomial time are considered intractable, or hard.

Outline

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is ‘yes’ or ‘no’, for inputs of size n:

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is 'yes' or 'no', for inputs of size n:

- P = the class of easy problems, whose solutions can be computed in *p*olynomial time: $\mathcal{O}(n^k)$ for some fixed k.

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is 'yes' or 'no', for inputs of size n:

- P = the class of easy problems, whose solutions can be computed in *p*olynomial time: $\mathcal{O}(n^k)$ for some fixed k.
  Examples: sorting; almost all problems in this course.

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is 'yes' or 'no', for inputs of size n:

- P = the class of easy problems, whose solutions can be computed in *p*olynomial time: $\mathcal{O}(n^k)$ for some fixed k. **Examples**: sorting; almost all problems in this course.
- NP = the class of problems for which a witness can be checked in polynomial time, when the answer is 'yes'. NP stands for "*non-deterministic polynomial time*", not for "*n*on-*p*olynomial time". We trivially have P $\subseteq$ NP.

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is 'yes' or 'no', for inputs of size n:

- P = the class of easy problems, whose solutions can be computed in *p*olynomial time: $\mathcal{O}(n^k)$ for some fixed k. **Examples**: sorting; almost all problems in this course.
- NP = the class of problems for which a witness can be checked in polynomial time, when the answer is 'yes'. NP stands for "*non-deterministic polynomial time*", not for "*n*on-*p*olynomial time". We trivially have P $\subseteq$ NP. **Example** (factoring): Given an n-digit number, does it have a divisor ending in 7? Computing such a divisor seems hard, but checking a candidate divisor is easy.

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is ‘yes’ or ‘no’, for inputs of size n:

- P = the class of easy problems, whose solutions can be computed in *p*olynomial time: $\mathcal{O}(n^k)$ for some fixed k.
  Examples: sorting; almost all problems in this course.
- NP = the class of problems for which a witness can be checked in polynomial time, when the answer is ‘yes’.
  NP stands for “*non-deterministic polynomial time*”, not for “*n*on-*p*olynomial time”. We trivially have $\mathrm{P} \subseteq \mathrm{NP}$.
  Example (factoring): Given an n-digit number, does it have a divisor ending in 7? Computing such a divisor seems hard, but checking a candidate divisor is easy.
- Undecidable problems cannot be solved by any algorithm, no matter how much time is allocated.

UPPSALA UNIVERSITET

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is 'yes' or 'no', for inputs of size n:

- P = the class of easy problems, whose solutions can be computed in *p*olynomial time: $\mathcal{O}(n^k)$ for some fixed k.
  Examples: sorting; almost all problems in this course.
- NP = the class of problems for which a witness can be checked in polynomial time, when the answer is 'yes'.
  NP stands for "*non-deterministic polynomial time*", not for "*n*on-*p*olynomial time". We trivially have P $\subseteq$ NP.
  Example (factoring): Given an n-digit number, does it have a divisor ending in 7? Computing such a divisor seems hard, but checking a candidate divisor is easy.
- Undecidable problems cannot be solved by any algorithm, no matter how much time is allocated.
  Examples: halting problem; disjointness of two CFLs.

UPPSALA UNIVERSITET

P and NP: Definitions and Examples

A bit more formally, and focussing on decision problems for NP, whose answer is 'yes' or 'no', for inputs of size n:

- P = the class of easy problems, whose solutions can be computed in *p*olynomial time: $\mathcal{O}(n^k)$ for some fixed k. **Examples**: sorting; almost all problems in this course.
- NP = the class of problems for which a witness can be checked in polynomial time, when the answer is 'yes'. NP stands for "*non-deterministic polynomial time*", not for "*n*on-*p*olynomial time". We trivially have P $\subseteq$ NP. **Example** (factoring): Given an n-digit number, does it have a divisor ending in 7? Computing such a divisor seems hard, but checking a candidate divisor is easy.
- Undecidable problems cannot be solved by any algorithm, no matter how much time is allocated. **Examples:** halting problem; disjointness of two CFLs.

So not all problems are in NP, independently of P versus NP.

Outline

Reduction and NP Hardness (Karp, 1972)

Informally:

- A problem Q reduces to a problem R, denoted $Q \leq_{\mathrm{P}} R$, if every instance of Q can be transformed in *p*oly time into an instance of R that has the same yes/no answer.

Reduction and NP Hardness (Karp, 1972)

Informally:

- A problem Q reduces to a problem R, denoted $Q \leq_{\mathrm{P}} R$, if every instance of Q can be transformed in *p*oly time into an instance of R that has the same yes/no answer. We also say that R is at least as hard as Q.

Reduction and NP Hardness (Karp, 1972)

Informally:

- A problem Q reduces to a problem R, denoted $Q \leq_{\mathrm{P}} R$, if every instance of Q can be transformed in *p*oly time into an instance of R that has the same yes/no answer. We also say that R is at least as hard as Q. Note that $\leq_{\mathrm{P}}$ is transitive: $\forall Q, E, R : Q \leq_{\mathrm{P}} E \leq_{\mathrm{P}} R \Rightarrow Q \leq_{\mathrm{P}} R$.

Reduction and NP Hardness (Karp, 1972)

Informally:

- A problem Q reduces to a problem R, denoted $Q \leq_{\mathrm{P}} R$, if every instance of Q can be transformed in *p*oly time into an instance of R that has the same yes/no answer. We also say that R is at least as hard as Q. Note that $\leq_{\mathrm{P}}$ is transitive: $\forall Q, E, R : Q \leq_{\mathrm{P}} E \leq_{\mathrm{P}} R \Rightarrow Q \leq_{\mathrm{P}} R$.
- Proving that a problem Q is in P is doable by showing that $Q \leq_{\mathrm{P}} E$ for some existing problem E in P.

Reduction and NP Hardness (Karp, 1972)

Informally:

- A problem Q reduces to a problem R, denoted $Q \leq_{\mathrm{P}} R$, if every instance of Q can be transformed in poly time into an instance of R that has the same yes/no answer. We also say that R is at least as hard as Q. Note that $\leq_{\mathrm{P}}$ is transitive: $\forall Q, E, R : Q \leq_{\mathrm{P}} E \leq_{\mathrm{P}} R \Rightarrow Q \leq_{\mathrm{P}} R$.
- Proving that a problem Q is in P is doable by showing that $Q \leq_{\mathrm{P}} E$ for some existing problem E in P.
- A problem is NP-hard if it is at least as hard as every problem in NP: every problem in NP reduces to it.

Reduction and NP Hardness (Karp, 1972)

Informally:

- A problem Q reduces to a problem R, denoted $Q \leq_{\mathrm{P}} R$, if every instance of Q can be transformed in *p*oly time into an instance of R that has the same yes/no answer. We also say that R is at least as hard as Q. Note that $\leq_{\mathrm{P}}$ is transitive: $\forall Q, E, R : Q \leq_{\mathrm{P}} E \leq_{\mathrm{P}} R \Rightarrow Q \leq_{\mathrm{P}} R$.
- Proving that a problem Q is in P is doable by showing that $Q \leq_{\mathrm{P}} E$ for some existing problem E in P.
- A problem is NP-hard if it is at least as hard as every problem in NP: every problem in NP reduces to it.
- On slide 19 is a wider definition of NP hardness.

UPPSALA UNIVERSITET

Outline

NP Completeness (Cook, 1971; Levin, 1973)

Formally:

- A problem is NP-complete if it is in NP and is NP-hard.

NP Completeness (Cook, 1971; Levin, 1973)

Formally:

- A problem is NP-complete if it is in NP and is NP-hard.

If some NP-complete problem is polynomial-time solvable, then every problem in NP is poly-time solvable: P $\supseteq$ NP.

NP Completeness (Cook, 1971; Levin, 1973)

Formally:

- A problem is NP-complete if it is in NP and is NP-hard.

If some NP-complete problem is polynomial-time solvable, then every problem in NP is poly-time solvable: $\mathrm{P} \supseteq \mathrm{NP}$.

An NP-complete problem is poly-time solvable iff $\mathrm{P} = \mathrm{NP}$.

NP Completeness (Cook, 1971; Levin, 1973)

Formally:

- A problem is NP-complete if it is in NP and is NP-hard.

If some NP-complete problem is polynomial-time solvable, then every problem in NP is poly-time solvable: P $\supseteq$ NP.

An NP-complete problem is poly-time solvable iff P $=$ NP.

If some problem in NP is not poly-time solvable (P $\neq$ NP), then no NP-complete problem is polynomial-time solvable.

NP Completeness (Cook, 1971; Levin, 1973)

Formally:

- A problem is NP-complete if it is in NP and is NP-hard.

If some NP-complete problem is polynomial-time solvable, then every problem in NP is poly-time solvable: P $\supseteq$ NP.

An NP-complete problem is poly-time solvable iff P $=$ NP.

If some problem in NP is not poly-time solvable (P $\neq$ NP), then no NP-complete problem is polynomial-time solvable.

The status of NP-complete problems is currently unknown: No polynomial-time algorithm was found for any of them, and no proof was made that no such algorithm can exist.

NP Completeness (Cook, 1971; Levin, 1973)

Formally:

- A problem is NP-complete if it is in NP and is NP-hard.

If some NP-complete problem is polynomial-time solvable, then every problem in NP is poly-time solvable: $\mathrm{P} \supseteq \mathrm{NP}$.

An NP-complete problem is poly-time solvable iff $\mathrm{P} = \mathrm{NP}$.

If some problem in NP is not poly-time solvable ($\mathrm{P} \neq \mathrm{NP}$), then no NP-complete problem is polynomial-time solvable.

The status of NP-complete problems is currently unknown: No polynomial-time algorithm was found for any of them, and no proof was made that no such algorithm can exist.

Most experts believe NP-complete problems are intractable, as the opposite would be truly amazing.

NP Completeness: Examples

Given a digraph (V, E):

Examples

- Finding a shortest path takes

NP Completeness: Examples

Given a digraph (V, E):

Examples

- Finding a shortest path takes $\mathcal{O}(V \cdot E)$ time and is in P.

NP Completeness: Examples

Given a digraph (V, E):

Examples

- Finding a shortest path takes $\mathcal{O}(V \cdot E)$ time and is in P.
- Determining the existence of a simple path (which has distinct vertices) that has *at least* a given number ℓ of edges is NP-complete.

NP Completeness: Examples

Given a digraph (V, E):

Examples

- Finding a shortest path takes $\mathcal{O}(V \cdot E)$ time and is in P.
- Determining the existence of a simple path (which has distinct vertices) that has *at least* a given number ℓ of edges is NP-complete.
  Hence finding a longest path seems hard: increase ℓ starting from a trivial lower bound, until answer is ‘no’.

NP Completeness: Examples

Given a digraph (V, E):

Examples

- Finding a shortest path takes $\mathcal{O}(V \cdot E)$ time and is in P.
- Determining the existence of a simple path (which has distinct vertices) that has *at least* a given number ℓ of edges is NP-complete.
  Hence finding a longest path seems hard: increase ℓ starting from a trivial lower bound, until answer is 'no'.

Examples

- Finding an Euler tour (which visits each *edge* once) takes

NP Completeness: Examples

Given a digraph (V, E):

Examples

- Finding a shortest path takes $\mathcal{O}(V \cdot E)$ time and is in P.
- Determining the existence of a simple path (which has distinct vertices) that has *at least* a given number ℓ of edges is NP-complete.
  Hence finding a longest path seems hard: increase ℓ starting from a trivial lower bound, until answer is ‘no’.

Examples

- Finding an Euler tour (which visits each *edge* once) takes $\mathcal{O}(E)$ time and is thus in P.

NP Completeness: Examples

Given a digraph (V, E):

Examples

- Finding a shortest path takes $\mathcal{O}(V \cdot E)$ time and is in P.
- Determining the existence of a simple path (which has distinct vertices) that has *at least* a given number ℓ of edges is NP-complete.
  Hence finding a longest path seems hard: increase ℓ starting from a trivial lower bound, until answer is 'no'.

Examples

- Finding an Euler tour (which visits each *edge* once) takes $\mathcal{O}(E)$ time and is thus in P.
- Determining the existence of a Hamiltonian cycle (which visits each *vertex* once) is NP-complete.

NP Completeness: More Examples

Examples

- 2-SAT: Determining the satisfiability of a conjunction of disjunctions of 2 Boolean literals is in P.

NP Completeness: More Examples

Examples

- 2-SAT: Determining the satisfiability of a conjunction of disjunctions of 2 Boolean literals is in P.
- 3-SAT: Determining the satisfiability of a conjunction of disjunctions of 3 Boolean literals is NP-complete.

NP Completeness: More Examples

Examples

- 2-SAT: Determining the satisfiability of a conjunction of disjunctions of 2 Boolean literals is in P.
- 3-SAT: Determining the satisfiability of a conjunction of disjunctions of 3 Boolean literals is NP-complete.
- SAT: Determining the satisfiability of a formula over Boolean literals is NP-complete.

NP Completeness: More Examples

Examples

- 2-SAT: Determining the satisfiability of a conjunction of disjunctions of 2 Boolean literals is in P.
- 3-SAT: Determining the satisfiability of a conjunction of disjunctions of 3 Boolean literals is NP-complete.
- SAT: Determining the satisfiability of a formula over Boolean literals is NP-complete.
- Clique: Determining the existence of a clique (complete subgraph) of a given size in a graph is NP-complete.

NP Completeness: More Examples

Examples

- 2-SAT: Determining the satisfiability of a conjunction of disjunctions of 2 Boolean literals is in P.
- 3-SAT: Determining the satisfiability of a conjunction of disjunctions of 3 Boolean literals is NP-complete.
- SAT: Determining the satisfiability of a formula over Boolean literals is NP-complete.
- Clique: Determining the existence of a clique (complete subgraph) of a given size in a graph is NP-complete.
- Vertex Cover: Determining the existence of a vertex cover (a vertex subset with at least one endpoint for all edges) of a given size in a graph is NP-complete.

NP Completeness: More Examples

Examples

- 2-SAT: Determining the satisfiability of a conjunction of disjunctions of 2 Boolean literals is in P.
- 3-SAT: Determining the satisfiability of a conjunction of disjunctions of 3 Boolean literals is NP-complete.
- SAT: Determining the satisfiability of a formula over Boolean literals is NP-complete.
- Clique: Determining the existence of a clique (complete subgraph) of a given size in a graph is NP-complete.
- Vertex Cover: Determining the existence of a vertex cover (a vertex subset with at least one endpoint for all edges) of a given size in a graph is NP-complete.
- Subset Sum: Determining the existence of a subset, of a given set, that has a given sum is NP-complete.

Pseudo-Polynomial Algorithms

Example (Subset Sum)

Determining the existence of a subset, of a given set S of n numbers, that has a given sum t is NP-complete:

Pseudo-Polynomial Algorithms

Example (Subset Sum)

Determining the existence of a subset, of a given set S of n numbers, that has a given sum t is NP-complete:

- A dynamic programming algorithm takes $\mathcal{O}(n \cdot t)$ time, as each entry in its $n \times t$ table is found in $\mathcal{O}(1)$ time.

UPPSALA UNIVERSITET

Pseudo-Polynomial Algorithms

Example (Subset Sum)

Determining the existence of a subset, of a given set S of n numbers, that has a given sum t is NP-complete:

- A dynamic programming algorithm takes $\mathcal{O}(n \cdot t)$ time, as each entry in its $n \times t$ table is found in $\mathcal{O}(1)$ time.
- This is polynomial in the size n of the input set S and polynomial in the magnitude of the input t, which can be large depending on n and the numbers in S.

Pseudo-Polynomial Algorithms

Example (Subset Sum)

Determining the existence of a subset, of a given set S of n numbers, that has a given sum t is NP-complete:

- A dynamic programming algorithm takes $\mathcal{O}(n \cdot t)$ time, as each entry in its $n \times t$ table is found in $\mathcal{O}(1)$ time.
- This is polynomial in the size n of the input set S and polynomial in the magnitude of the input t, which can be large depending on n and the numbers in S.
- This is exponential in the size $\lceil \log_b t \rceil$ of the base-b representation of t, since $t = b^{\log_b t}$ (usually: $b = 2$).

Pseudo-Polynomial Algorithms

Example (Subset Sum)

Determining the existence of a subset, of a given set S of n numbers, that has a given sum t is NP-complete:

- A dynamic programming algorithm takes $\mathcal{O}(n \cdot t)$ time, as each entry in its $n \times t$ table is found in $\mathcal{O}(1)$ time.
- This is polynomial in the size n of the input set S and polynomial in the magnitude of the input t, which can be large depending on n and the numbers in S.
- This is exponential in the size $\lceil \log_b t \rceil$ of the base-b representation of t, since $t = b^{\log_b t}$ (usually: $b = 2$).

Definition

An algorithm of complexity polynomial in the magnitude of its input numbers is said to be pseudo-polynomial.

NP Completeness: Proof by Reduction

Proving that a problem R of NP is NP-complete is doable by showing $E \leq_{\mathrm{P}} R$ for some existing NP-complete problem E, since by definition $Q \leq_{\mathrm{P}} E$ for every problem Q in NP.

NP Completeness: Proof by Reduction

Proving that a problem R of NP is NP-complete is doable by showing $E \leq_{\mathrm{P}} R$ for some existing NP-complete problem E, since by definition $Q \leq_{\mathrm{P}} E$ for every problem Q in NP.
If a poly-time algorithm for R existed, then we would have a poly-time algorithm for E, which would lead to $\mathrm{P} = \mathrm{NP}$.

NP Completeness: Proof by Reduction

Proving that a problem R of NP is NP-complete is doable by showing $E \leq_{\text{P}} R$ for some existing NP-complete problem E, since by definition $Q \leq_{\text{P}} E$ for every problem Q in NP.
If a poly-time algorithm for R existed, then we would have a poly-time algorithm for E, which would lead to $\text{P} = \text{NP}$.

Examples (exercises will be given in the AD3 course)

- SAT is NP-complete (Cook, 1971; Levin, 1973).

NP Completeness: Proof by Reduction

Proving that a problem R of NP is NP-complete is doable by showing $E \leq_{\mathrm{P}} R$ for some existing NP-complete problem E, since by definition $Q \leq_{\mathrm{P}} E$ for every problem Q in NP.
If a poly-time algorithm for R existed, then we would have a poly-time algorithm for E, which would lead to $\mathrm{P} = \mathrm{NP}$.

Examples (exercises will be given in the AD3 course)

- SAT is NP-complete (Cook, 1971; Levin, 1973).
- SAT reduces to 3-SAT, but not to 2-SAT.

UPPSALA UNIVERSITET

NP Completeness: Proof by Reduction

Proving that a problem R of NP is NP-complete is doable by showing $E \leq_{\text{P}} R$ for some existing NP-complete problem E, since by definition $Q \leq_{\text{P}} E$ for every problem Q in NP.
If a poly-time algorithm for R existed, then we would have a poly-time algorithm for E, which would lead to $\text{P} = \text{NP}$.

Examples (exercises will be given in the AD3 course)

- SAT is NP-complete (Cook, 1971; Levin, 1973).
- SAT reduces to 3-SAT, but not to 2-SAT.
- 3-SAT reduces to Clique and Subset Sum.

NP Completeness: Proof by Reduction

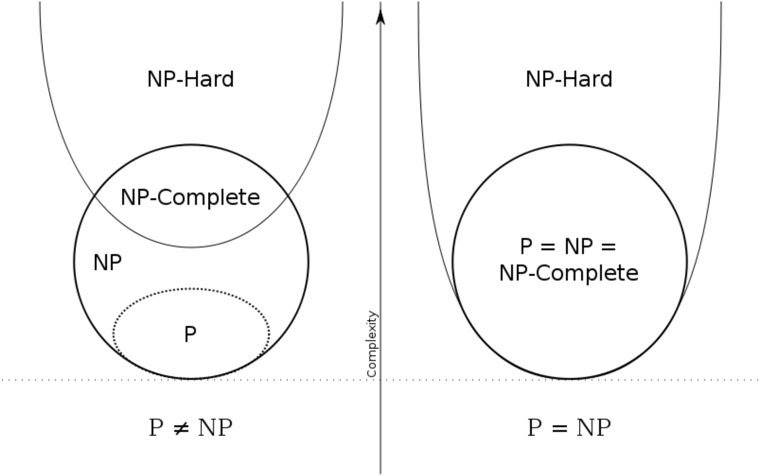
Proving that a problem R of NP is NP-complete is doable by showing $E \leq_{\mathrm{P}} R$ for some existing NP-complete problem E, since by definition $Q \leq_{\mathrm{P}} E$ for every problem Q in NP.
If a poly-time algorithm for R existed, then we would have a poly-time algorithm for E, which would lead to $\mathrm{P} = \mathrm{NP}$.

Examples (exercises will be given in the AD3 course)

- SAT is NP-complete (Cook, 1971; Levin, 1973).
- SAT reduces to 3-SAT, but not to 2-SAT.
- 3-SAT reduces to Clique and Subset Sum.
- Clique reduces to Vertex Cover, which reduces to Hamiltonian Cycle, which reduces to Travelling Salesperson (TSP), asking if there is a Hamiltonian cycle with cost at most k in a complete weighted graph.

Outline

Relationships



Remarks

- If P $\neq$ NP, then there exist problems in NP that are neither in P nor NP-complete. Artificial such problems can be constructed, but integer factorisation and graph isomorphism are practical problems in NP that are currently not known to be in P or to be NP-complete.

Remarks

- If P $\neq$ NP, then there exist problems in NP that are neither in P nor NP-complete. Artificial such problems can be constructed, but integer factorisation and graph isomorphism are practical problems in NP that are currently not known to be in P or to be NP-complete.
- There exist many other complexity classes, chartering the territory outside NP, some of them overlapping with the NP-hard class, and containing practical problems, such as planning. Determining a precise complexity map is contingent upon settling the P versus NP issue.

Remarks

- If P $\neq$ NP, then there exist problems in NP that are neither in P nor NP-complete. Artificial such problems can be constructed, but integer factorisation and graph isomorphism are practical problems in NP that are currently not known to be in P or to be NP-complete.
- There exist many other complexity classes, chartering the territory outside NP, some of them overlapping with the NP-hard class, and containing practical problems, such as planning. Determining a precise complexity map is contingent upon settling the P versus NP issue.
- The stable matching problem is believed by many to be hard, but it can be solved in $\mathcal{O}(n)$ time for n hospitals & n students, and is thus in P (Gale and Shapley, 1962). Shapley shared the Nobel Prize in Economics 2012.

Outline

What Now?

In a satisfaction problem, a ‘yes’ answer includes a witness.
In an optimisation problem, a ‘yes’ answer includes an optimal witness according to some cost function.

What Now?

In a satisfaction problem, a ‘yes’ answer includes a witness.
In an optimisation problem, a ‘yes’ answer includes an optimal witness according to some cost function.
Satisfaction and optimisation problems with NP-complete decision problems are often also said to be NP-hard.
(Recall the method on slide 11 for finding a longest path.)

What Now?

In a satisfaction problem, a 'yes' answer includes a witness. In an optimisation problem, a 'yes' answer includes an optimal witness according to some cost function. Satisfaction and optimisation problems with NP-complete decision problems are often also said to be NP-hard. (Recall the method on slide 11 for finding a longest path.)

Several courses at Uppsala University teach techniques for addressing NP-hard optimisation or satisfaction problems:

- Algorithms and Datastructures 3 (1DL481) (period 3)
- Continuous Optimisation (1TD184) (period 2)
- Modelling for Combinatorial Optim. (1DL451) (period 1)
- CO & Constraint Programming (1DL442) (periods 1+2)

☞ NP completeness is where the fun begins (not ends)!