CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

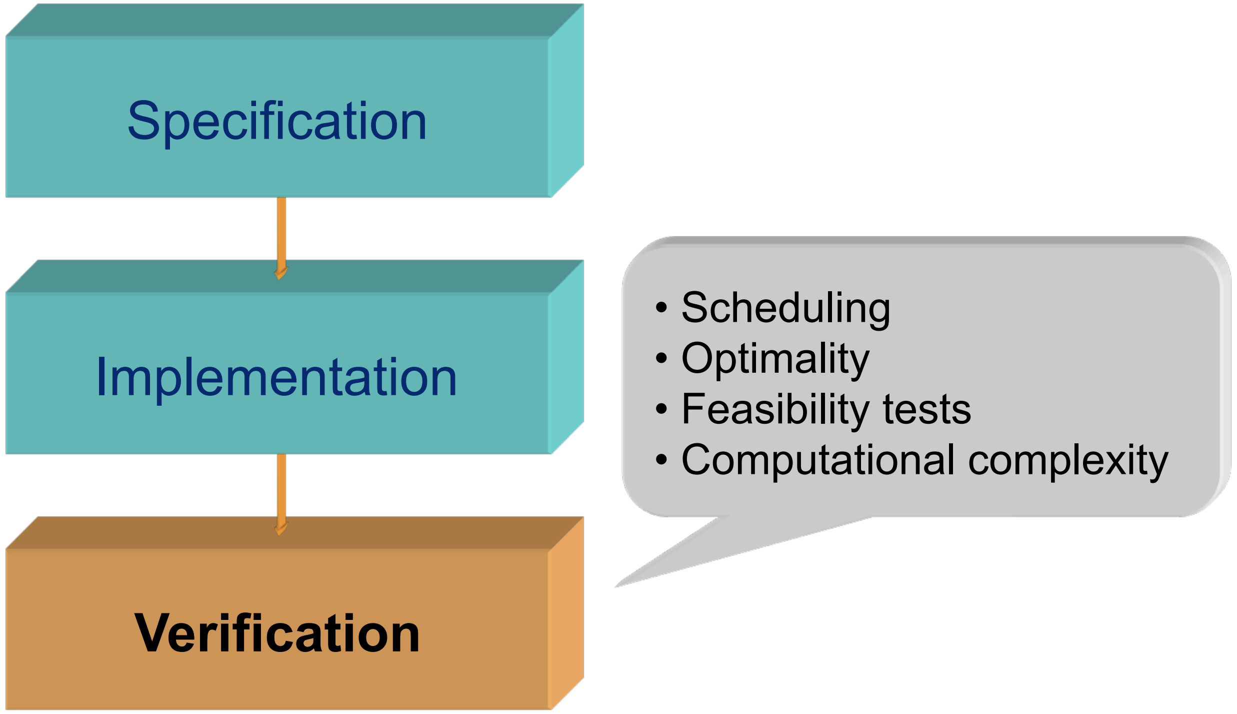
Real-Time Systems

Lecture #9

Professor Jan Jonsson

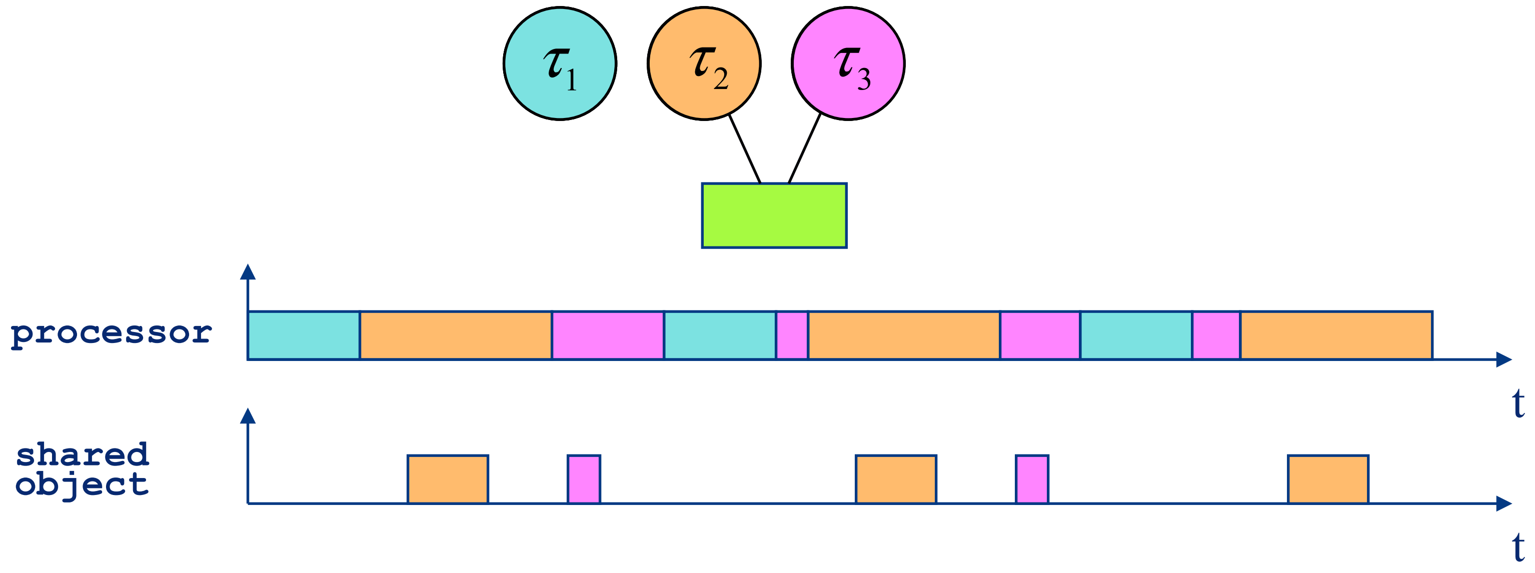
Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Scheduling

A <u>schedule</u> is a reservation of spatial (e.g., processor, shared objects) and temporal (time) resources for a given set of tasks.



Scheduling

A scheduling algorithm is used for generating a schedule for a given set of tasks for a particular type of run-time system.

- The scheduling algorithm is implemented by a scheduler in the run-time system, that decides in what order the tasks should be executed.
- Note that the scheduler decides which task should be executed next, whereas the dispatcher is responsible for starting the selected task.

Scheduling

How is scheduling implemented?

- Cyclic executive:
  - The schedule is generated ”off-line” before the tasks becomes ready, sometimes even before the system is in mission.
  - The schedule consists of a time table, containing explicit start and completion times for each task instance, that controls the order of execution at run-time.
- Pseudo-parallel execution:
  - The schedule is generated ”on-line” as a side effect of tasks being executed, that is, when the system is in mission.
  - Ready tasks are sorted in a queue and receive access to the processor based on priority and/or time quanta (”round-robin”).

Scheduling

How is scheduling implemented? (cont'd)

- Cyclic executive:
  - The implementation of the scheduler is relatively simple because the next task is chosen with a table look-up.
  - However, the time table must be generated off-line (before the system is in mission) by a more advanced algorithm.
- Pseudo-parallel execution:
  - The implementation of the scheduler is more sophisticated because it consists of a decision algorithm that must be activated regularly (at each system event).
  - If shared resources are used the scheduler must also handle protocols for avoiding priority/deadline inversion.

Scheduling

When are scheduling decisions taken?

- Non-preemptive scheduling:
  - Scheduling decisions are taken when no task executes.
  - Mutual exclusion can be automatically guaranteed.
  - Corresponds to fundamental assumption in WCET analysis.
- Preemptive scheduling:
  - Scheduling decisions may be taken as soon as the system state changes (that is, even during an ongoing task execution).
  - Mutual exclusion may have to be guaranteed with semaphores (or similar primitives).
  - WCET analysis becomes more complicated, because the state in caches and pipelines will change at a task switch.

Scheduling

When are scheduling decisions taken? (cont'd)

- Myopic scheduling:
  - Scheduling algorithm only knows about tasks that are ready.
  - Scheduling decisions are only taken when system state changes.
  - On-line myopic scheduling is state-of-the-art in run-time systems.
- Clairvoyant scheduling:
  - Scheduling algorithm ”knows the future”; that is, it knows in advance all the arrival times of all the tasks.
  - Scheduling decisions may be taken at <u>any</u> time, not necessarily when system state changes.
  - On-line clairvoyant scheduling is very difficult (often impossible) to realize in practice.

Scheduling

A schedule is said to be <u>feasible</u> if it fulfills all application constraints for a given set of tasks.

A set of tasks is said to be <u>schedulable</u> if there exists at least one scheduling algorithm that can generate a feasible schedule.

Scheduling

A scheduling algorithm is said to be optimal with respect to schedulability if it can always find a feasible schedule whenever any other scheduling algorithm can do so.

A scheduling algorithm is said to be optimal with respect to a performance metric if it can always find a schedule that maximizes/minimizes that metric value.

Feasibility tests

A feasibility test is used for deciding whether a set of tasks is feasible or not for a given scheduler.

Important characteristics of feasibility tests:

- Exactness
  - What conclusions can be drawn regarding feasibility based on the outcome of the test?
- Computational complexity
  - How long time does it take for the test to produce an outcome with a decision regarding feasibility?

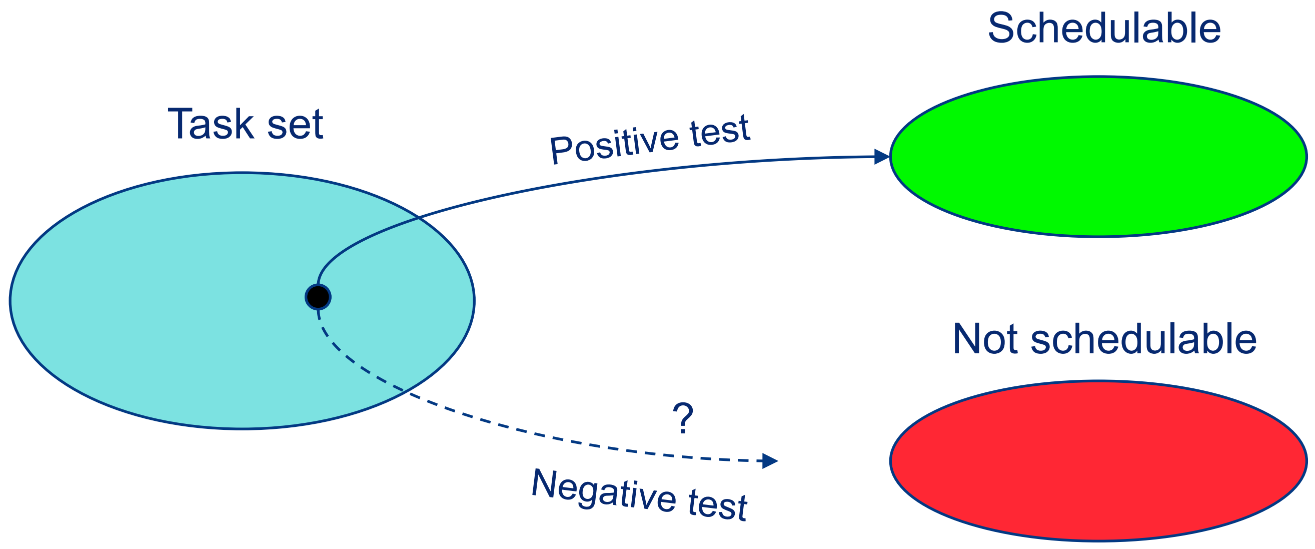
Feasibility tests

Exactness of a feasibility test

- The outcome of a feasibility test is binary:
  - Positive or Negative
  - True or False
  - Yes or No
- The conclusions that can be drawn depends on whether the test is:
  - Sufficient
  - Necessary
  - Exact (= sufficient and necessary)

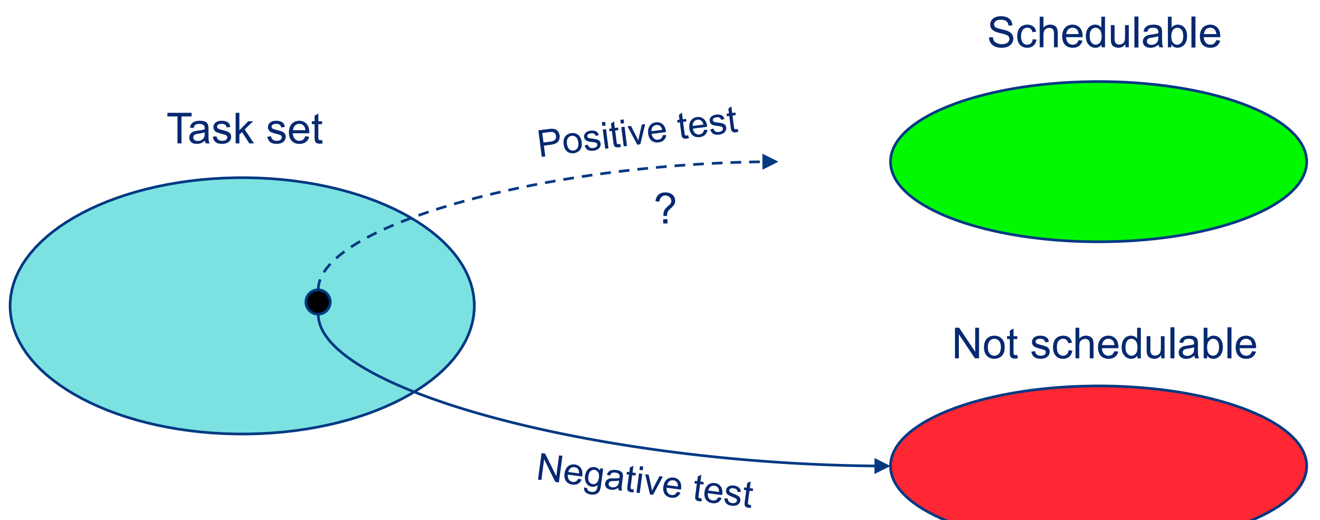
Feasibility tests

- A feasibility test is <u>sufficient</u> if it with a <u>positive</u> outcome shows that a set of tasks is definitely schedulable.
  - A negative outcome says nothing! A set of tasks can still be schedulable despite a negative outcome.



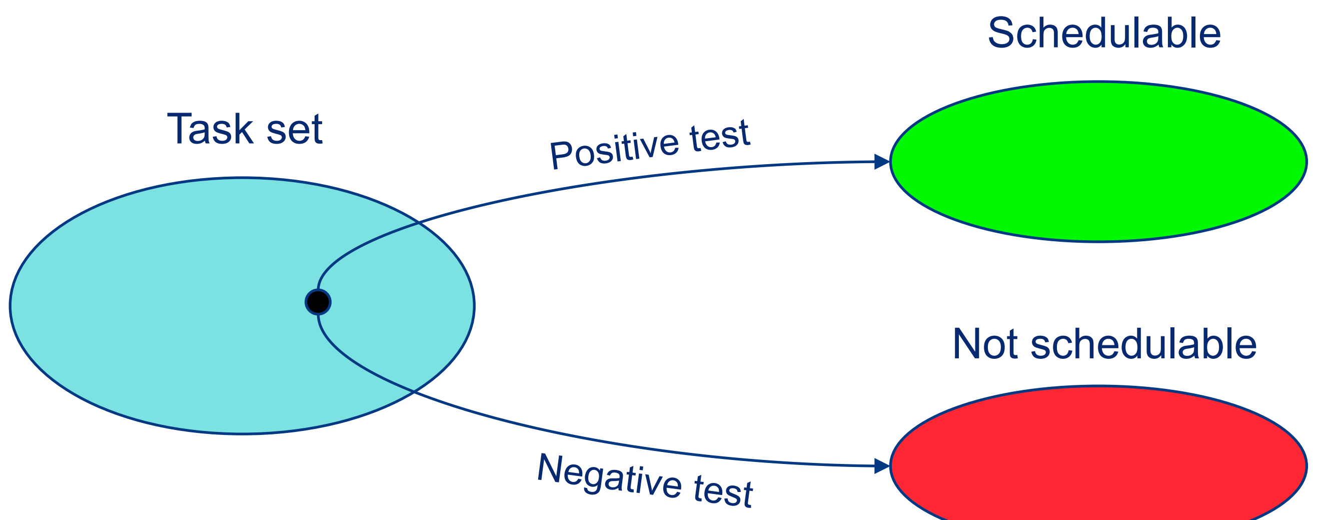
Feasibility tests

- A feasibility test is <u>necessary</u> if it with a <u>negative</u> outcome shows that a set of tasks is definitely <u>not</u> schedulable.
  - A positive outcome says nothing! A set of tasks can still be impossible to schedule despite a positive outcome.



Feasibility tests

- An <u>exact</u> feasibility test is both sufficient and necessary. If the outcome of the test is positive the set of tasks is definitely schedulable, and if the outcome is negative the set of tasks is definitely not schedulable.



Feasibility tests

Computational complexity of a feasibility test:

- For some schedulers the feasibility test can be done with polynomial time complexity.
  – These feasibility tests typically have relaxed assumptions regarding the task model, for example: independent tasks (with no shared resources) or tasks with deadline = period.
- For most schedulers the feasibility test cannot be done with polynomial time complexity.
  – These feasibility tests are either NP-complete problems or have exponential time complexity because all possible schedules must be considered in the worst case.

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (exponential time complexity)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (polynomial time complexity)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (NP-complete problem)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (NP-complete problem)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Why NP-completeness matters

Assume that your boss gives you the following problem:

Find a good algorithm (method) for determining whether or not any given set of specifications for your company's new bandersnatch component can be met and, if so, find a good algorithm for constructing a design that meets those specifications.

The bandersnatch example is taken from "A Guide to the Theory of NP-Completeness" by M. R. Garey and D. S. Johnson

Why NP-completeness matters

Initial attempt:

Pull down your reference books and plunge into the task with great enthusiasm.

Some weeks later ...

Your office is filled with crumpled-up scratch paper, and your enthusiasm has lessened considerable because …

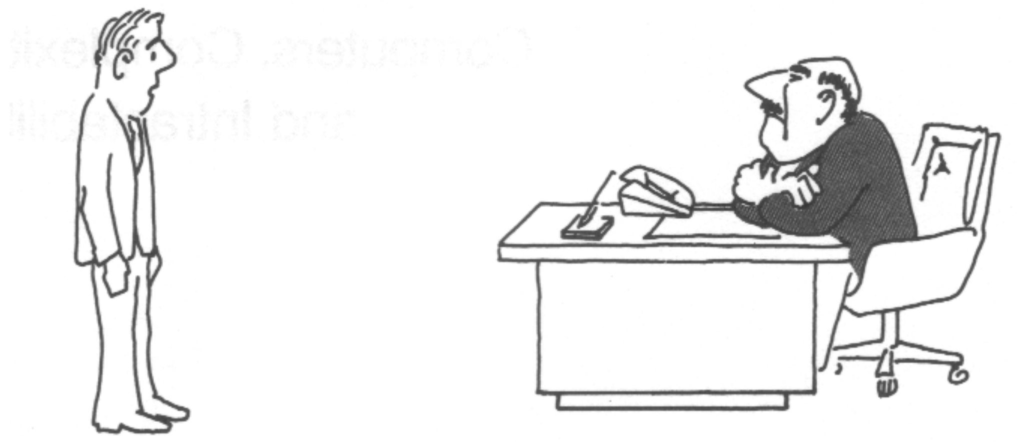
… the solution seems to be to examine all possible designs!

You now have a new problem:

How do you convey the bad information to your boss?

Why NP-completeness matters

Approach #1: Take the loser's way out

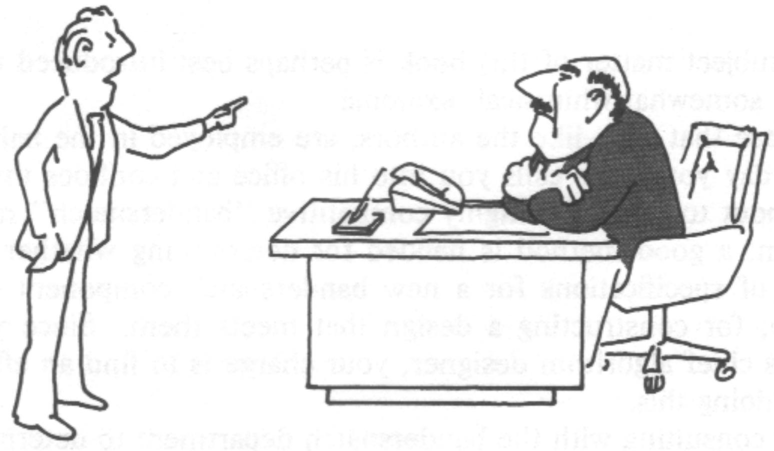


"I can't find an efficient algorithm, I guess I'm just too dumb."

Drawback: Could seriously damage your position within the company

Why NP-completeness matters

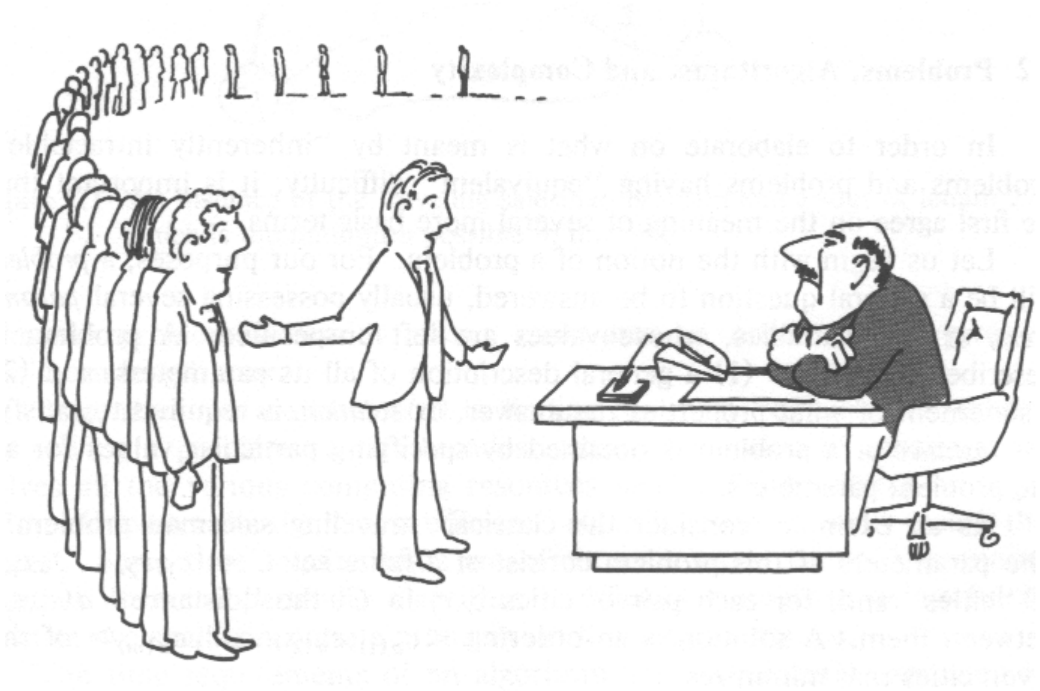
Approach #2: Prove that the problem is inherently intractable


"I can't find an efficient algorithm, because no such algorithm is possible!"

Drawback: Proving inherent intractability can be as hard as finding efficient algorithms. Even the best theoreticians have failed!

Why NP-completeness matters

Approach #3: Prove that the problem is NP-complete

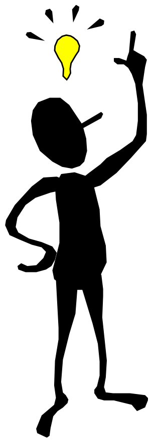


"I can't find an efficient algorithm, but neither can all these famous people."

Advantage: This would inform your boss that it is no good to fire you and hire another expert on algorithms.

NP-complete problems

NP-complete problems:
Problems that are "just as hard" as a large number of other problems that are widely recognized as being difficult by algorithmic experts.



NP-complete problems can (most likely) only be solved by an exponential-time algorithm in the general case.

NP-complete problems

Problem:

- A general question to be answered

  Example: The "traveling salesman optimization problem"

Parameters:

- Free problem variables, whose values are left unspecified

  Example: A set of "cities" $C = \{c_1, \dots, c_n\}$ and a "distance" $d(c_i, c_j)$ between each pair of cities c_i and c_j

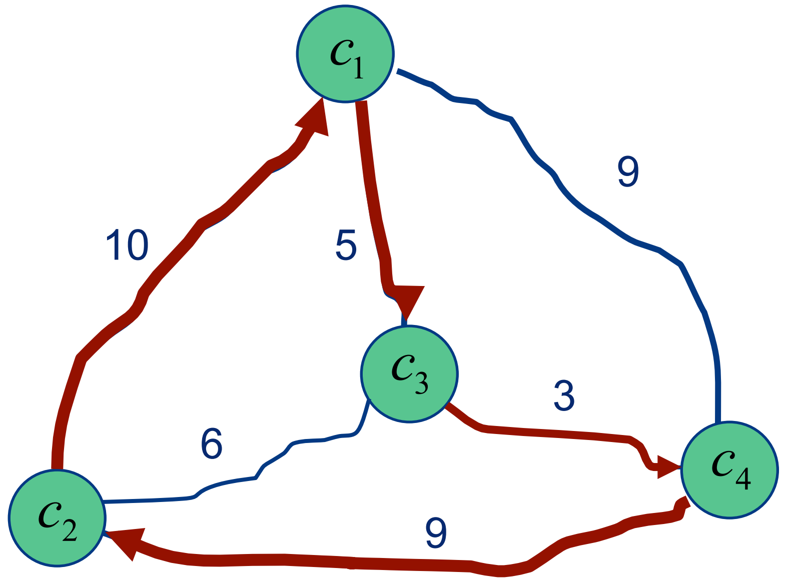
Instance:

- An instance of a problem is obtained by specifying particular values for all the problem parameters

  Example: $C = \{c_1, c_2, c_3, c_4\}, d(c_1, c_2) = 10, d(c_1, c_3) = 5, d(c_1, c_4) = 9,$
  $d(c_2, c_3) = 6, d(c_2, c_4) = 9, d(c_3, c_4) = 3$

NP-complete problems

The Traveling Salesman Optimization Problem:



Minimum “tour” length = 27

Minimize the length of the “tour” that visits each city in sequence, and then returns to the first city.

NP-complete problems

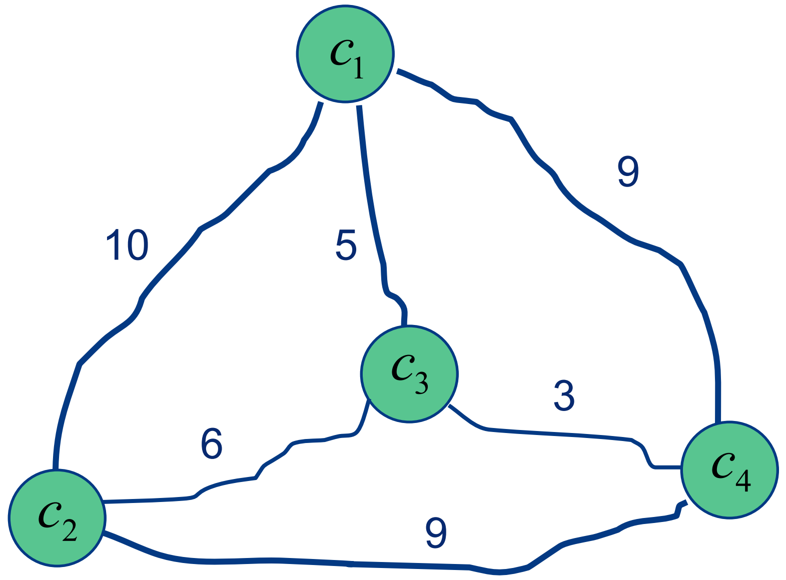
The theory of NP-completeness applies only to decision problems, where the solution is either a “Yes” or a “No”.

$\downarrow$

If an optimization problem asks for a solution that has minimum “cost”, we can associate with that problem a decision problem that includes a numerical bound B as an additional parameter and that asks whether there exists a solution having cost <u>no more than</u> B.

NP-complete problems

The Traveling Salesman Decision Problem:

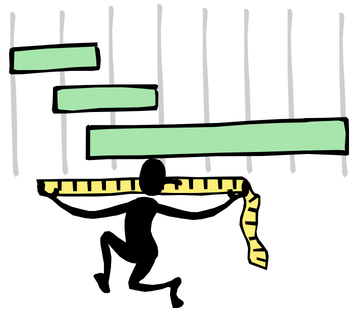


Is there a “tour” of all the cities in C having a total length of no more than B?

Intractability

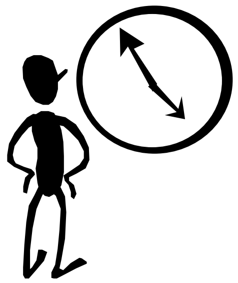
Input length:

- The number of information symbols (e.g. bits) needed for representing a problem instance of a given size.



Time-complexity function:

- Expresses an algorithm's worst-case run-time requirements giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size.



Intractability

Polynomial-time algorithm:

- An algorithm whose time-complexity function is proportional to $p(n)$ for some polynomial function p, where n is the input length.

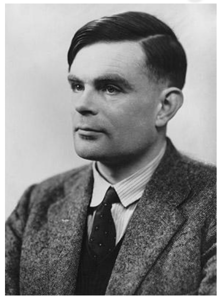
Exponential-time algorithm:

- Any algorithm whose time-complexity function cannot be bounded as above.

A problem is said to be <u>intractable</u> if it is so hard that no polynomial-time algorithm can possibly solve it.

Class P

Alan Turing



Deterministic algorithm: (Deterministic Turing Machine)

- Finite-state control:
  - The algorithm can pursue only one computation at a time
  - Given a problem instance I, some solution S is derived by the algorithm
  - The correctness of S is inherent in the algorithm

The <u>class P</u> is the class of all decision problems Π that can be solved by polynomial-time <u>deterministic</u> algorithms.

Class NP

Non-deterministic algorithm: (Non-Deterministic Turing Machine)

1. Guessing stage:
   - Given a problem instance I, some solution S is "guessed".
   - The algorithm can pursue an unbounded number of independent computational sequences in parallel.
2. Checking stage:
   - The correctness of S is verified in a normal deterministic manner

The class NP is the class of all decision problems Π that can be solved by polynomial-time non-deterministic algorithms.

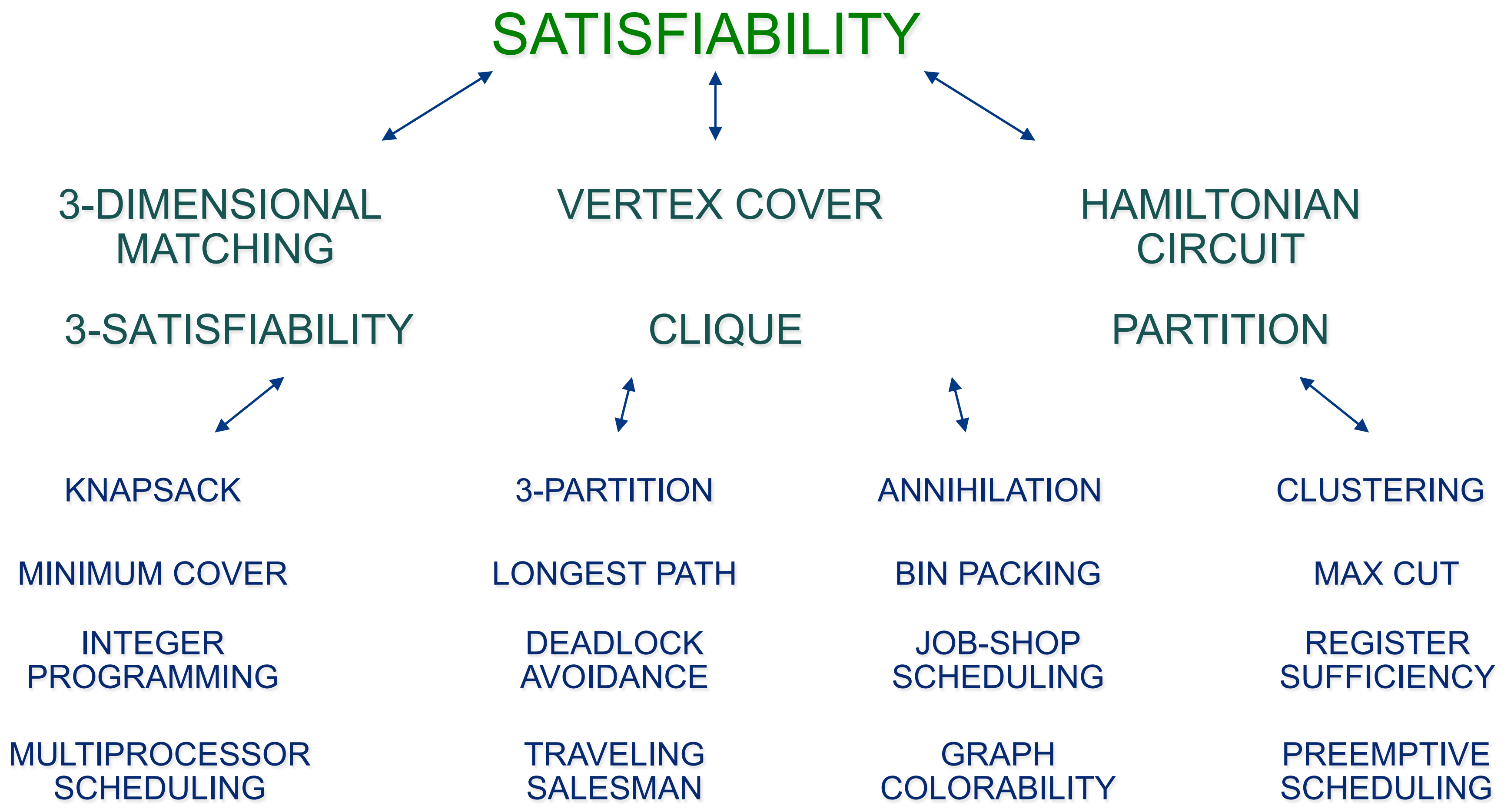
NP-complete problems

Reducibility:

- A problem Π' is <u>reducible</u> to problem Π if, for any instance of Π', an instance of Π can be constructed in polynomial time such that solving the instance of Π will solve the instance of Π' as well.

A decision problem Π is said to be <u>NP-complete</u> if $\Pi \in$ NP and, for all other decision problems $\Pi' \in$ NP, Π' reduces to Π in polynomial time .

Some original NP-complete problems



NP-completeness in practice

Pseudo-polynomial time complexity:

- Number problems
  - This is a special type of NP-complete problems for which the largest number (parameter value) in a problem instance is not bounded by the input length (size) of the problem.
- Number problems are often quite tractable
  - If the time complexity of a number problem can be shown to be a polynomial-time function of both the input length and the largest number, that number problem is said to have pseudo-polynomial time complexity.

    That is, the time-complexity function is proportional to $p(max,n)$ for some polynomial function p, where max is the largest number and n is the input length.

Feasibility tests

What types of feasibility tests exist? (revisited)

- Hyper period analysis (exponential time complexity)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (polynomial time complexity)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (pseudo-polynomial complexity)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (pseudo-polynomial complexity)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval