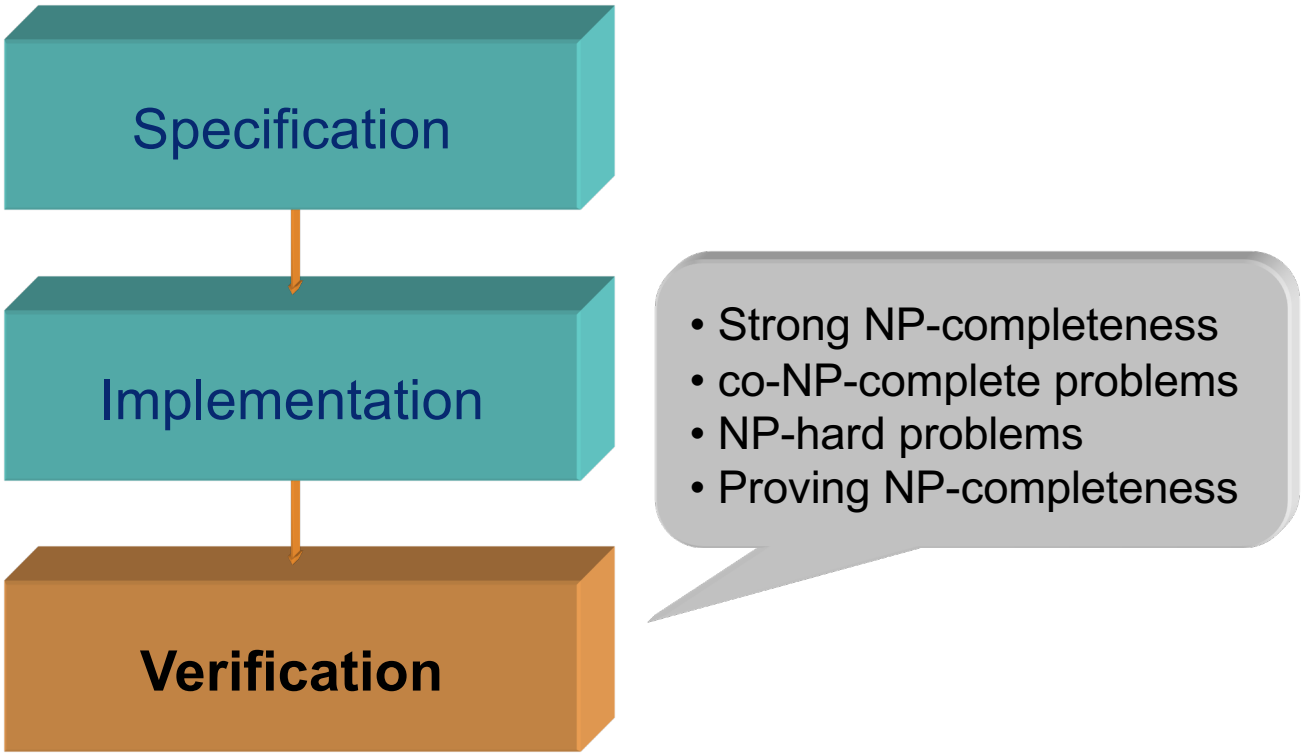
Real-Time Systems

Lecture #15

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Intractability and NP-completeness

Relationship between P and NP:

1. $P \subseteq NP$
   - Proof: use a polynomial-time deterministic algorithm as the checking stage and ignore the guess ....
2. $P \neq NP$
   - This is a wide-spread belief, but …
   - … no proof of this conjecture exists!

The question of whether or not the NP-complete problems are intractable is now considered to be one of the foremost open questions of contemporary mathematics and computer science!

Strong NP-completeness

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.

Strong NP-completeness

If a decision problem Π is NP-complete and is not a number problem, then it cannot be solved by a pseudo-polynomial-time algorithm unless P = NP.

↓

Assuming P $\neq$ NP, the only NP-complete problems that are potential candidates for being solved by pseudo-polynomial-time algorithms are those that are number problems.

↓

A decision problem Π which cannot be solved by a pseudo-polynomial-time algorithm, unless P = NP, is said to be NP-complete in the strong sense.

Strong NP-completeness

NP-complete problems that are number problems ...

- ... but are NP-complete in the strong sense regardless
  - Multiprocessor preemptive scheduling (partitioned and global)
  - Single-processor non-preemptive scheduling
  - 3-Partition, Simultaneous Congruences, Traveling Salesman

NP-complete problems that are number problems …

- ... and that do have pseudo-polynomial time complexity
  - Single-processor scheduling of synchronous constrained-deadline tasks with static priorities (using response-time analysis)
  - Single-processor scheduling of synchronous constrained-deadline tasks with dynamic priorities (using processor-demand analysis)

Strong NP-completeness

Proving pseudo-polynomial time complexity:

- It is quite easy to prove that response-time analysis (RTA) has pseudo-polynomial time complexity.
  - First show that RTA is a number problem
  - Then show that RTA has tractable time complexity
- It takes a little more effort to prove that processor-demand analysis (PDA) has pseudo-polynomial time complexity.
  - In the general case PDA may have exponential time complexity (due to length of the hyper period)
  - However, for the restricted case where task utilization $U < 1$ PDA can be shown to have tractable time complexity.

We now study the proofs for the RTA and PDA cases.

(see blackboard scribble)

Co-NP-complete problems

Class co-NP:

- Complement problem:
  - The complement of a decision problem Π is the problem Π^C having the same solution domain as Π, but with the outcome from solving the problem logically reversed.
  - That is, given the same problem instance, a "yes" outcome from solving problem Π would imply a "no" outcome from solving problem Π^C (and vice versa)

A decision problem $\Pi \in$ co-NP if and only if its complement problem $\Pi^C \in$ NP.

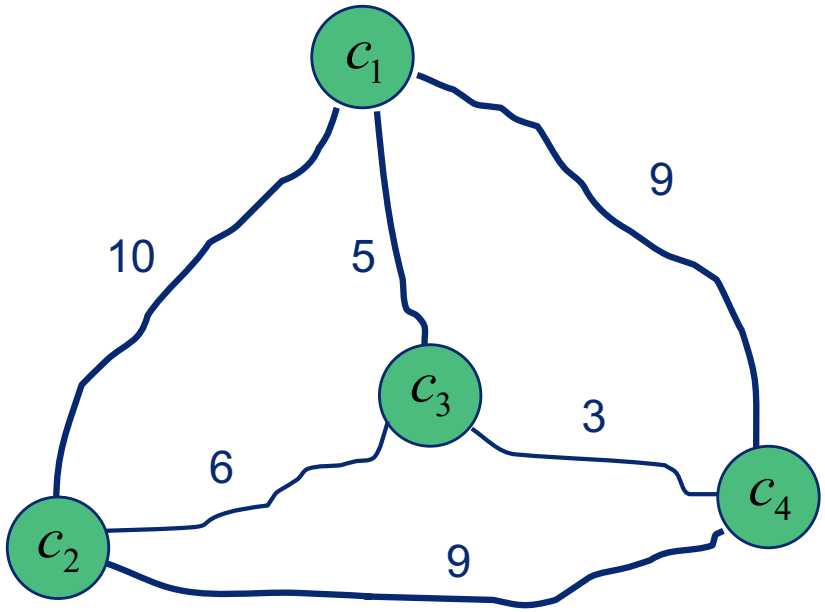
Co-NP-complete problems

NP vs co-NP:

- Problems in NP
  - The class of problems for which there exists a polynomial-time algorithm that can verify a solution that makes the binary problem statement true (“yes” outcome).
- Problems in co-NP
  - The class of problems for which there exists a polynomial-time algorithm that can verify a counterexample solution that makes the binary problem statement false (“no” outcome).
- Co-NP-complete problems
  - Decision problems for which it applies that their complement problem is an NP-complete problem.

Co-NP-complete problems

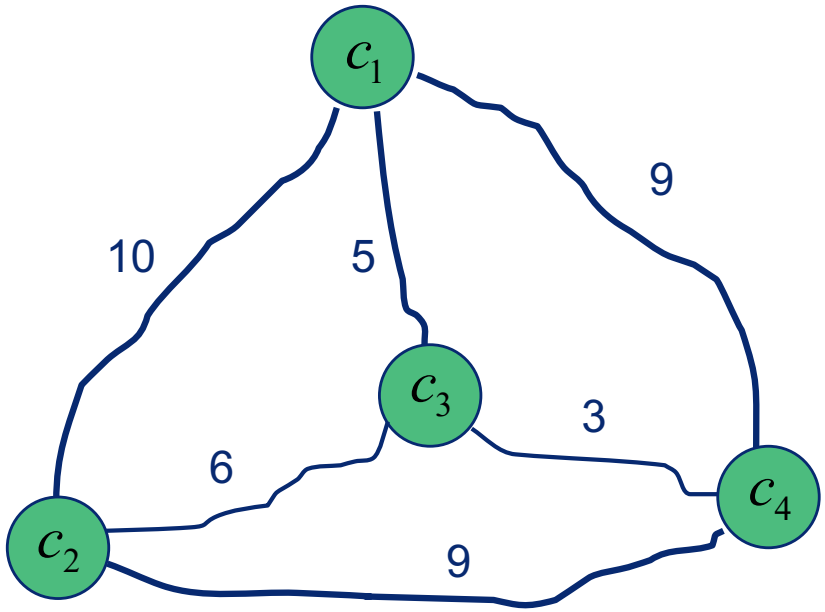
The Original Traveling Salesman Problem:



Does there exist one "tour" of all the cities in C having a total length of no more than B?

Co-NP-complete problems

The Complement Traveling Salesman Problem:



Does every "tour" of all the cities in C have a total length that exceeds B?

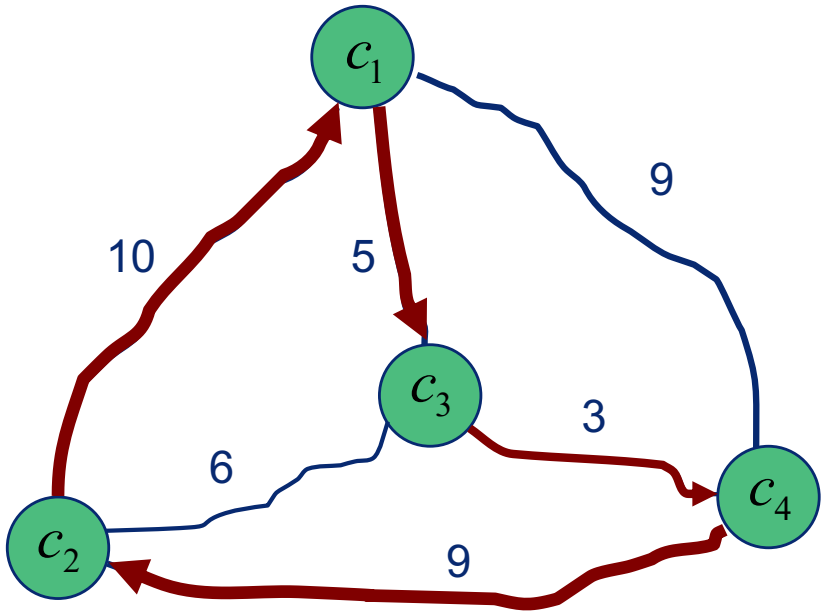
Co-NP-complete problems

The Complement Traveling Salesman Problem:

- Verifying a “yes” outcome
  - Requires checking that all possible solutions (”tours”) to the problem instance fulfills the problem statement. Can in general only be done in exponential time (need to show that every possible “tour” length > B).
- Verifying a “no” outcome
  - Requires checking that one solution (the counterexample “tour”) to the problem instance does not fulfill the problem statement. Can be done in polynomial time (only need to show that the counterexample “tour” length ≤ B).
  - This corresponds exactly to verifying a “yes” outcome in the original Traveling Salesman Problem (which is NP-complete).

Co-NP-complete problems

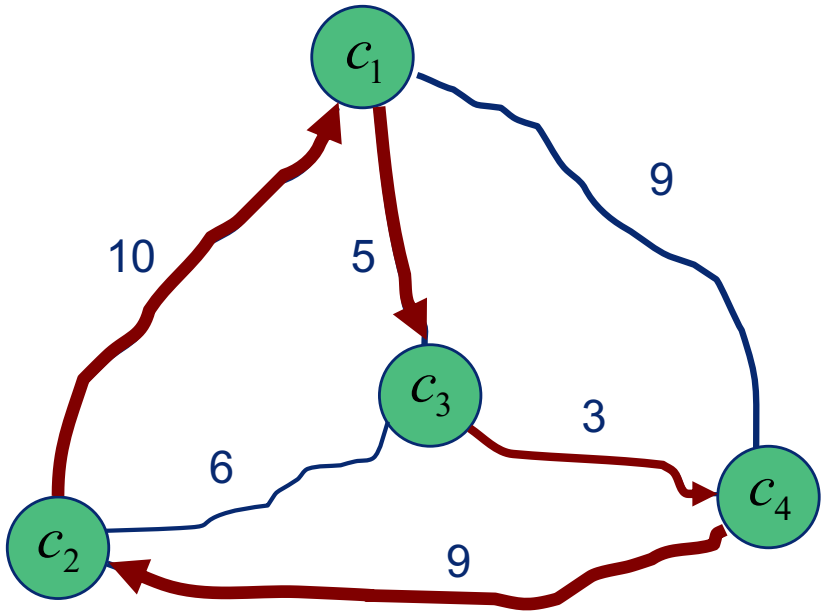
The Complement Traveling Salesman Problem:



Does every "tour" of all the cities in C have a total length that exceeds **30**? No! One has length 27.

Co-NP-complete problems

The Original Traveling Salesman Problem:



Does there exist one "tour" of all the cities in C having a total length of no more than **30**? Yes! One has length 27.

Co-NP-complete problems

Why is co-NP relevant for scheduling problems?

- The original scheduling problem formulation:
  - Does there exist one schedule for the given task set and run-time system that is feasible?
- The complement scheduling problem formulation:
  - Is every schedule for the given task set and run-time system infeasible?
- Reducibility of scheduling problems:
  - For many well-known scheduling problems it turns out that it is only the complement problem that can be reduced to a known NP-complete problem.

Co-NP-complete problems

The Complement Scheduling Problem:

- Verifying a “yes” outcome
  - Requires checking that all possible solutions (“schedules”) will contain at least one missed deadline. Can in general only be done in exponential time (need to show that every possible “schedule” is infeasible).
- Verifying a “no” outcome
  - Only requires checking that one solution (the counterexample “schedule”) will meet all deadlines. Can be done in polynomial time (show that the counterexample “schedule” is feasible).
  - This corresponds exactly to verifying a “yes” outcome in the original Scheduling Problem.

NP-hard problems

NP-hard problems:

Problems that are “at least as hard” as the hardest problems in class NP.

NP-hard problems

Turing reducibility:

- A problem Π' is Turing reducible to problem Π if there exists an algorithm A that solves Π' by using a hypothetical subroutine S for solving Π such that, if S were a polynomial time algorithm for Π, then A would be a polynomial time algorithm for Π' as well.

  When Π' is Turing reducible to Π, we write $\Pi' \propto_T \Pi$

A search problem Π is said to be NP-hard if there exists some NP-complete problem Π' that Turing-reduces to Π.

NP-hard problems

Observations:

- All NP-complete problems are NP-hard
- All co-NP-complete problems are NP-hard

  Turing reduction from Π to Π^C (and vice versa) is trivial.

- Given an NP-complete decision problem, the corresponding optimization problem is NP-hard

  To see this, imagine that the optimization problem (that is, finding the optimal cost) could be solved in polynomial time.

  The corresponding decision problem (i.e., determining whether there exists a solution with a cost no more than B) could then be solved by simply comparing the found optimal cost to the bound B. This comparison is a constant-time operation.

Reference only, not examined

History of NP-completeness

S. Cook: (1971)

"The Complexity of Theorem Proving Procedures"

Every problem in the class NP of decision problems polynomially reduces to the SATISFIABILITY problem.

R. Karp: (1972)

"Reducibility among Combinatorial Problems"

Decision problem versions of many well-known combinatorial optimization problems are "just as hard" as SATISFIABILITY.

Reference only, not examined

History of NP-completeness

D. Knuth: (1974)

"A Terminological Proposal"

Initiated a researcher's poll in search of a better term for "at least as hard as the polynomial complete problems".

The winning suggestion was "NP-complete" problems.

One (rejected, but smart) suggestion was "PET" problems:

- "Probably Exponential Time" (if P = NP remain open question)
- "Provably Exponential Time" (if $P \neq NP$)
- "Previously Exponential Time" (if P = NP)

History of NP-completeness

SATISFIABILITY: The original NP-complete problem

- Variables and literals
  - Let U be a set of Boolean variables.
  - If u is a variable in U then u and u' are literals over U.
- Conjunctive normal form
  - A formula is in conjunctive normal form (CNF) if it is a conjunction of one or more clauses, where a clause is a disjunction of literals.
- SATISFIABILITY question:
  - Given a formula in CNF does there exist a truth assignment for the variables in U that yields a True statement?

Reference only, not examined

History of NP-completeness

Proving NP-completeness

Proving NP-completeness for a decision problem Π:

1. Show that Π is in NP
2. Select a known NP-complete problem Π'
3. Construct a transformation $\propto$ from Π' to Π
4. Prove that $\propto$ is a (polynomial) transformation

The book "Computers and Intractability – A Guide to the Theory of NP-Completeness" (Garey and Johnson, 1979) contains a categorized list of 300+ NP-complete problems, with problem statements and how each problem was proven NP-complete.

Proving NP-completeness

Transformations for real-time scheduling problems:

In published results regarding the time complexity of known real-time scheduling problems, the following NP-complete problems are predominantly used for the transformations:

- 3-PARTITION
  - NP-complete in the strong sense.
  - Used in the proofs for multiprocessor scheduling, and in the proof for non-preemptive single-processor scheduling.
- SIMULTANEOUS CONGRUENCES (SCP)
  - NP-complete in the strong sense.
  - Used in the proofs for preemptive single-processor scheduling, by employing a reverse logic (co-NP) strategy.

Proving NP-completeness

3-PARTITION decision problem:

- Set of elements
  - Let $A = \{a_1, \ldots, a_{3m}\}$ be a set of $3m$ elements.
  - Each element $a_i \in A$ has a positive integer "size" $s(a_i)$.
- Element size constraints using a bound
  - Let B be a positive integer.
  - Each $s(a_i)$ satisfies $B/4 < s(a_i) < B/2$ and $\sum_{a_i \in A} s(a_i) = mB$.
- Question:
  - Can A be partitioned into m disjoint sets $S_1, \ldots, S_m$ such that, for each $1 \leq j \leq m$, it applies that $\sum_{a_i \in S_j} s(a_i) = B$?

Note: constraints dictate that each disjoint set must contain exactly 3 elements!

Proving NP-completeness

SIMULTANEOUS CONGRUENCES decision problem:

- Set of ordered pairs
  - Let $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ be a set of n ordered pairs.
  - Each pair $(a_i, b_i) \in A$ consists of positive integers.
- Minimum bound
  - Let B be a positive integer, such that $2 \leq B \leq n$.
- Question:
  - Does there exist a subset $A' \in A$ of at least B pairs and a positive integer x such that, for each $(a_i, b_i) \in A'$, it applies that $x \equiv a_i \pmod{b_i}$?

Note: $x \equiv a_i \pmod{b_i}$ means $x = a_i + k_i \cdot b_i$ for some non-negative integer k_i

Proving NP-completeness

Proving NP-completeness for a decision problem Π:

1. Show that Π is in NP
2. Select a known NP-complete problem Π'
3. Construct a transformation $\propto$ from Π' to Π
4. Prove that $\propto$ is a (polynomial) transformation

Example: the proof by Jeffay, Stanat and Martel (1991)

Transformation from 3-PARTITION is used in proving strong NP-completeness for non-preemptive single-processor EDF scheduling of asynchronous, periodic, implicit-deadline tasks (Theorem 5.2)

Proving NP-completeness

General complexity of real-time scheduling:

- Any type of scheduling of periodic tasks
  - NP-hard (exponential time)
- Any type of non-preemptive scheduling
  - NP-complete in the strong sense (reduction from 3-PARTITION)
- Any type of preemptive multiprocessor scheduling
  - NP-complete in the strong sense (reduction from 3-PARTITION)
  - Note: applies to both partitioned and global approaches
- Preemptive single-processor scheduling of asynchronous tasks
  - Co-NP-complete in the strong sense (reduction from SCP)

Proving NP-completeness

Complexity of preemptive single-processor scheduling:

- Scheduling of synchronous tasks w/ dynamic task priorities
  - Co-NP-complete in the strong sense (reduction from SCP)
  - Co-NP-complete in the weak ("normal") sense for $U < 1$
  - Special cases:
    - Pseudo-polynomial time for constrained-deadline tasks for $U < 1$
    - Polynomial time for implicit-deadline tasks
- Scheduling of synchronous tasks w/ static task priorities
  - NP-hard for arbitrary-deadline tasks (exponential time)
  - NP-complete in the weak sense for constrained-deadline tasks
  - Special cases:
    - Pseudo-polynomial time for constrained-deadline tasks
    - Polynomial time for implicit-deadline tasks for $U \leq \ln 2$