CHALMERS

Computers and Intractability

A Guide to the Theory of NP-Completeness

The “Bible” of complexity theory

M. R. Garey and D. S. Johnson

W. H. Freeman and Company, 1979

CHALMERS

The ”Bandersnatch” problem

Background:

Find a good method for determining whether or not any given set of specifications for a new bandersnatch component can be met and, if so, for constructing a design that meets them.

CHALMERS

The "Bandersnatch" problem

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!

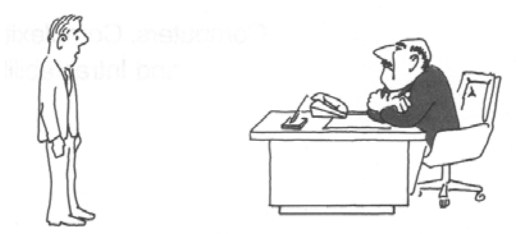
New problem:

How do you convey the bad information to your boss?

CHALMERS

The "Bandersnatch" problem

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

CHALMERS

The "Bandersnatch" problem

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!

CHALMERS

The "Bandersnatch" problem

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.

CHALMERS

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.

CHALMERS

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" $C = \{c_1, \ldots, c_n\}$ and a "distance" $d(c_i, c_j)$ between each pair of cities c_i and c_j

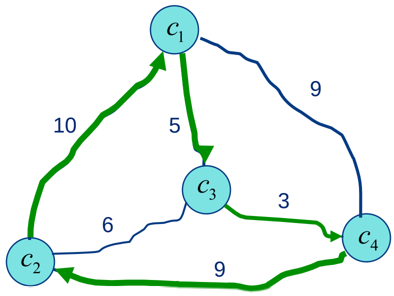
Instance:

- An instance of a problem is obtained by specifying particular values for all the problem parameters
  Example: $C = \{c_1, c_2, c_3, c_4\}, d(c_1, c_2) = 10, d(c_1, c_3) = 5, d(c_1, c_4) = 9,$ $d(c_2, c_3) = 6, d(c_2, c_4) = 9, d(c_3, c_4) = 3$

CHALMERS

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.

CHALMERS

Intractability

Input length:

- The number of information symbols needed for describing a problem instance using a "reasonable" encoding scheme

Largest number:

- The magnitude of the largest number in a problem instance

Time-complexity function:

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

CHALMERS

Intractability

Polynomial-time algorithm:

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

Exponential-time algorithm:

- Any algorithm whose time-complexity function cannot be so bounded.

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

CHALMERS

Intractability

Reasonable encoding scheme:

- Conciseness:
  - The encoding of an instance I should be concise and not “padded” with unnecessary information or symbols
  - Numbers occurring in I should be represented in binary (or decimal, or octal, or in any fixed base other than 1)
- Decodability:
  - It should be possible to specify a polynomial-time algorithm that can extract a description of any component of I.

CHALMERS

Decision problems

The theory of NP-completeness applies only to <u>decision problems</u>, where the solution is either a "Yes" or a "No".

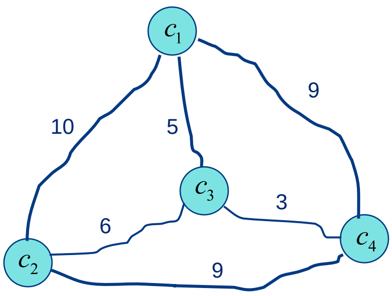
↓

If an optimization problem asks for a structure of a certain type that has minimum "cost" among such structures, 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 structure of the required type having cost <u>no more than</u> B.

CHALMERS

Decision problems

The Traveling Salesman <u>Decision</u> Problem:



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

CHALMERS

Class P

Deterministic algorithm: (Deterministic Turing Machine)

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

The class P is the class of all decision problems Π that, under reasonable encoding schemes, can be solved by polynomial-time deterministic algorithms.

CHALMERS

Class NP

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

1. Guessing stage:
   - Given a problem instance I, some structure 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, under reasonable encoding schemes, can be solved by polynomial-time non-deterministic algorithms.

CHALMERS

Relationship between P and NP

Observations:

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!

CHALMERS

NP-complete problems

Reducibility:

- A problem Π' is reducible 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.

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

A decision problem Π is said to be NP-complete if $\Pi \in$ NP and, for all other decision problems $\Pi' \in$ NP, Π' polynomially reduces to Π.

CHALMERS

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 decision problem $\Pi' \in NP$ that Turing-reduces to Π.

CHALMERS

NP-hard problems

Observations:

- All NP-complete problems are NP-hard
- 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 (that is, 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.
- While an NP-complete problem is solvable in polynomial time if and only if P = NP, an NP-hard problem cannot be solved in polynomial time unless P = NP.

CHALMERS

Strong NP-completeness

Pseudo-polynomial-time algorithm:

- An algorithm whose time-complexity function is $O(p(n,m))$ for some polynomial function p, where n is the input length and m is the largest number.

Number problem:

- A decision problem for which there exists no polynomial function p such that $m \leq p(n)$ for all instances of the problem.
- Examples:
  - PARTITION, KNAPSACK, TRAVELING SALESMAN
  - MULTIPROCESSOR SCHEDULING

CHALMERS

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.

CHALMERS

Avoiding NP-completeness

Tricks for circumventing the intractability:

1. Limiting the largest number *m*
2. Redefining the problem (e.g. edge vs vertex cover)
3. Exploiting problem structure (e.g. limits on vertex degrees, ”intree” vs ”outtree” task graphs)
4. Fixing problem parameters (e.g. fixed # of processors in multiprocessor scheduling)

CHALMERS

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:

Given a set U of Boolean variables and a collection C of clauses over U, is there a satisfying truth assignment for C ?

R. Karp: (1972)

“Reducibility among Combinatorial Problems”

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

CHALMERS

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".

One suggestion by S. Lin 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)

CHALMERS

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

CHALMERS

Basic NP-complete problems

SATISFIABILITY

3-DIMENSIONAL MATCHING

VERTEX COVER

HAMILTONIAN CIRCUIT

3-SATISFIABILITY

CLIQUE

PARTITION

KNAPSACK

3-PARTITION

ANNIHILATION

CLUSTERING

MINIMUM COVER

LONGEST PATH

BIN PACKING

MAX CUT

INTEGER PROGRAMMING

DEADLOCK AVOIDANCE

JOB-SHOP SCHEDULING

REGISTER SUFFICIENCY

MULTIPROCESSOR SCHEDULING

TRAVELING SALESMAN

GRAPH COLORABILITY

PREEMPTIVE SCHEDULING

CHALMERS

NP-complete scheduling problems

Uniprocessor scheduling with offsets and deadlines:

Independent tasks with individual offsets and deadlines.
Transformation from 3-PARTITION (Garey and Johnson, 1977)

NP-complete in the strong sense.
Solvable in pseudo-polynomial time if number of allowed values for offsets and deadlines is bounded by a constant.
Solvable in polynomial time if execution times are identical, preemptions are allowed, or all offsets are 0.

CHALMERS

NP-complete scheduling problems

Multiprocessor scheduling:

Independent tasks with an overall deadline.
Transformation from PARTITION (Garey and Johnson, 1979)

NP-complete in the strong sense for arbitrary number of processors.
NP-complete in the normal sense for two processors.
Solvable in pseudo-polynomial time for any fixed number of processors.
Solvable in polynomial time if execution times are identical.

CHALMERS

NP-complete scheduling problems

Precedence-constrained multiprocessor scheduling:

Precedence-constrained tasks with identical execution times and an overall deadline.
Transformation from 3-SATISFIABILITY (Ullman, 1975)

NP-complete in the normal sense for arbitrary number of processors.
Solvable in polynomial time for two processors, or for arbitrary number of processors and “forest-like” precedence constraints.
Remains an open problem for fixed number of processors (≥ 3).

CHALMERS

NP-complete scheduling problems

Multiprocessor scheduling with individual deadlines:

Precedence-constrained tasks with identical execution times and individual deadlines.

Transformation from VERTEX COVER (Brucker, Garey and Johnson, 1977)

NP-complete in the normal sense for arbitrary number of processors.

Solvable in polynomial time for two processors or “in-tree” precedence constraints.

CHALMERS

NP-complete scheduling problems

Preemptive uniprocessor scheduling of periodic tasks:

Independent tasks with individual offsets and periods, and preemptive dispatching.

Transformation from CLIQUE (Leung and Merrill, 1980)

NP-complete in the normal sense.

CHALMERS

NP-complete scheduling problems

Non-preemptive uniprocessor scheduling of periodic tasks:

Independent tasks with individual offsets and periods, and non-preemptive dispatching.

Transformation from 3-PARTITION (Jeffay, Stanat and Martel, 1991)

NP-complete in the strong sense.

Additional reading:

Read the paper by Jeffay, Stanat and Martel (RTSS'91)

Study particularly how the transformation from 3-PARTITION is used for proving strong NP-completeness (Theorem 5.2)