Tools of AI

Hard Computational Problems (2)

Marcin Sydow

Web Mining Lab/PJIIT
Optimisation Problem

An **instance** of an optimisation problem: a pair $(F, c)$, where $F$ is interpreted as a set of *feasible* solutions and $c$ is the *cost* function: $c : F \rightarrow Q$.

The task in *minimisation* is to find $f \in F$ such that $c(f)$ is minimum possible in $F$. $f$ is called *global optimal solution* to the instance $F$.

An **optimisation problem** is a set $I$ of instances of an optimisation problem.

---

1 Another variant is a *maximisation* problem, when we look for a feasible solution $f$ with maximum possible value of $c(f)$ – in this case we can call $c$ a *profit* function.
Example of an optimisation problem: SHORTEST-PATH

**Instance**: a directed graph $G=(V,A)$, two vertices $u, v \in V$ (source and target)

**Feasible set**: a set of all paths in $G$ that start in $u$ and end in $v$

**Cost function**: for a feasible solution $f$ (a path from $u$ to $v$) it is its length

**SHORTEST-PATH optimisation problem**: the set of all possible graphs and pairs of its vertices

This is a minimisation problem.

(is this problem hard or easy?)
Decision Problems

The solution is in the binary form: “yes” or “no”

Example: is the given boolean formula satisfiable?

An optimisation problem can be usually transformed into its decision version that is not harder.

For example, an instance of the SHORTEST-PATH optimisation problem with additional parameter $k \in \mathbb{N}$ can be viewed as a decision problem: “is there a path from $u$ to $v$ in $G$ of the length not exceeding $k$?”

(the problem is not harder since solution to the optimisation problem automatically solves the decision version, but not the opposite, in general)
To be solved by a computer, an abstract problem should be first **encoded** into binary form.

Let Q be an abstract problem represented by the set of instances I

encoding: \( e : I \rightarrow \{0, 1\}^* \)

\((* - \text{“Kleene star”})\)

Encoding transforms an abstract problem into a **concrete problem**, denoted as \( e(Q) \)

We say that an algorithm solves a concrete problem in time \( O(T(n)) \) iff for each instance \( i \) of size \( |i| = n \) (in bits) it finds a solution in time \( O(T(n)) \).
Definition

A complexity class $P$ is the set of concrete decision problems for which there exist algorithms that solve them with complexity upper-bounded by a polynomial of $n$ (the length of the concrete problem), i.e. the complexity is $O(n^k)$, for $n = |i|$ problem length, $k$ - a positive constant (for each problem).

Notice that a decision problem solvable, for example, with a $\Theta(n \log n)$ complexity ($n$ - size of encoded, concrete problem instance) is in $P$ (even if “$n \log n$” is not a polynomial of $n$ itself, but is upper-bounded by such)
Remark on encoding’s “compactness”

One can observe that the fact whether a concrete version of an abstract problem is in $P$ class depends on the “compactness” of encoding.

It is assumed that encoding is “reasonably” compact. In particular, it is assumed that binary encoding is used for numbers (and that all the numbers are rational) which results in $\lceil \log_2 n \rceil$ number of bits for a value $n$. \(^2\)

Notice that unary encoding (which is definitely not a compact one) can make some complex problems look as polynomially solvable due to “expensive” encoding.

\(^2\)It does not matter whether a binary positional system is used or another, since all the logarithms are asymptotically equivalent.
Definition

A function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is **polynomial-time computable** iff there exists an algorithm that for any input $x \in \{0, 1\}^*$ produces output $f(x)$ in the polynomially bounded time complexity $O(p(|x|))$ ($p()$ is a polynomial, $|x|$ is the size of concrete instance of the problem).

Given a set $I$ of problem instances, two encodings $e_1, e_2$ are **polynomially related** iff there exist two functions $f_{12}, f_{21}$ that are polynomial-time computable such that

$\forall i \in I \ f_{12}(e_1(i)) = e_2(i)$ and $f_{21}(e_2(i)) = e_1(i)$ (and strings not being instances are mapped to such in both directions)
Lemma

Assume $e_1$, $e_2$ are two polynomially related encodings of an abstract problem $Q$. Then, $e_1(Q) \in P \iff e_2(Q) \in P$ (i.e. membership in $P$ class is independent on polynomially related encodings)

Proof:

($\Leftarrow$) Assume that $e_1(Q)$ can be solved with $O(n^k)$ time for a constant $k$ and that the encoding $e_1(i)$ can be computed from $e_2(i)$ with $O(n^c)$ time for a constant $c$, where $n = |e_2(i)|$. To solve encoded instance $e_2(i)$ of the problem $e_2(Q)$ it suffices to first compute encoding $e_1(i)$ (that takes time $O(n^c)$ ) and then compute the solution on the output. Its size $|e_1(i)| = O(n^c)$ since output cannot be (asymptotically) bigger than running time. Thus the total complexity will be $O(|e_1(i)|^k) = O(n^{ck})$ a polynomial of $n$ (q.e.d.)

The proof of the other direction is symmetric.
“Standard” Encoding

A bit informally, we will assume some “standard” encoding of basic objects such as rational numbers\(^3\), sets, graphs, etc. that are “reasonable” (e.g. unary encoding of numbers is not reasonable in this sense)

More precisely, in the context of the lemma, we will assume that encoding for all the numbers will be polynomially related to binary encoding (notice: decimal encoding is such), for a set – to comma-separated list of elements, etc.

Such “standard” encoding will be denoted by \langle . \rangle symbols (e.g. \langle G \rangle for a graph \(G\)).

Now, we can talk directly about abstract problems without reference to any particular encoding.

---

\(^3\)We assume all the numbers used are rational, i.e. of finite precision
Kleinberg, Tardos “Algorithm Design”, chapter 8
Garey, Johnson “Computers and Intractability” (1979, difficult to get nowadays)
Papadimitriou “Computational Complexity” (first chapters) (more advanced textbook)
Thank you for attention