Pina Kolling
piko0011

Summary: Lecture 9

Summary for the chapter *10.3.* [6, 2]

Function problems

Function problem

Finding a specific solution to a problem if possible, else return *no*.

In other words: A function problem is defined by a binary relation $R(x, y)$. For every input x, an algorithm that solves the problem must output a y such that $R(x, y)$. If there is no such y, the answer must be *no*.

- focus so far: languages deciding decision problems
- give *yes* or *no* as answer
- now: focus on finding a solution:
  - find satisfying truth assignment for a boolean expression
  - find optimal tour for TSP
  - → function problems

- decision problems are helpful for negative results of function problems
- complexity of the decision problem helps to specify the complexity of the corresponding function problem

SAT and FSAT

SAT

The SAT (satisfiability) problem is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. [7]

FSAT

The FSAT (satisfiability) problem is a function problem.

Given a boolean expression ϕ.
If ϕ is satisfiable, return a satisfying truth assignment and otherwise return *no*.

- for input ϕ there might be no satisying truth assignment $\phi \notin$ SAT
  - return *no*
- for input ϕ there might be more than one satisying truth assignment
  - return any satisfying truth assignment
- if SAT can be solved in polynomial time, FSAT can be solved in polynomial time, too

Algorithm for FSAT:

- expression ϕ with variables $x_1, ..., x_n$

- ask if ϕ is satisfiable ($\phi \in$ SAT):
  - if *no*: stop and return *no*
  - if *yes*: come up with satisying truth assignment
    - consider two expressions: $\phi[x_1 = \text{true}]$ and $\phi[x_1 = \text{false}]$
    - check which one is satisfiable ($\phi[x_1 = \text{true}] \in$ SAT or $\phi[x_1 = \text{false}] \in$ SAT) (if both are, chose one)
    - substitute the value of x_1 in ϕ
    - continue with x_2
    - at most $2n$ calls to find the satisfying truth assignment

Algorithm for FSAT as pseudo code:

An Algorithm for FSAT Using SAT

```
t := ε; {Truth assignment.}
if φ ∈ SAT then
  for i = 1, 2, . . . , n do
    if φ[ x_i = true ] ∈ SAT then
      t := t ∪ { x_i = true };
      φ := φ[ x_i = true ];
    else
      t := t ∪ { x_i = false };
      φ := φ[ x_i = false ];
    end if
  end for
  return t;
else
  return "no";
end if
```

Figure 1: FSAT algorithm as pseudo code [5]

Self-reducibilty
A function problem reduces to its corresponding decision problem.

- SAT is self-reducable

Questions:

- Book [6]:

  FSAT draws its difficulty precisely from the possibility that there may be no truth assignment satisfying the given expression.

→ Why is the difficulty coming from the possibility that there might be no truth assignment? It would in the first check of $\phi \in$ SAT return *no* and terminate? Is it because it takes longer for SAT to return *no* on every computation than it takes if a *yes* is found and all variables are substituted and checked with SAT?

Pina Kolling
piko0011

TSP and TSP(D)

TSP(D)

Given a list of cities and the distances between each pair of cities.

Is there a possible route of length k that visits each city exactly once and returns to the origin city?

TSP

Given a list of cities and the distances between each pair of cities.

What is the shortest possible route that visits each city exactly once and returns to the origin city?

- solve TSP with an algorithm for TSP(D)
- find optimum cost C of the tour with binary search (between 0 and 2^n) and using TSP(D)
- change one intercity distance at a time to $C + 1$ to check if it is part of the optimal tour
  - call TSP(D) C and the new distance entry
  - if *no*: restore old entry
    $\rightarrow$ distance is part of the tour
- after n^2 calls only entries of the distance matrix are $\leq C$ that are used for the optimum tour

FP and FNP

Lanugage L

$L = \{x : (x, y) \in R \text{ for some } y\}$
L gets an input x and finds a y with $((x, y) \in R$ and the relation $R \subseteq \Sigma^* \times \Sigma^*$.

NP

The language $L \subseteq \Sigma^*$ is in NP only if there is a polynomially decidable and polynomially balanced relation R such that $L = \{x : (x, y) \in R \text{ for some } y\}$.

Relationship between decision and function problems:

- L is a lanuage in NP
  - **Decision problem:**
    There is a string y with $R(x, y)$ only if $x \in L$.
  - **Function problem:**
    Given x, find a string y such that $R(x, y)$ if it exists, else return *no*.

FNP

Class of all function problems asspciated with languages in NP.

FP

FP is the subclass of FNP that contains function problems, that can be solved in polynomial time.

Examples:

- FSAT is in FNP but expected to be in FP
- HORNSAT is in FP
- BIPARTITEGRAPH is in FP

Reductions between function problems

Reductions between function problems

A function problem A reduces to a function problem B if the following holds:

- R and S are string functions, x and z are strings
- If x is an instance of A then $R(x)$ is an instance of B.
- If z is a correct output of $R(x)$, then $S(z)$ is a correct output of x.

- R produces an instance $R(x)$ of the function problem B
- $S(z)$ is an constructed output for x from any correct output z of $R(x)$
- translate answers back to the original problem
- reduction is a pair (R, S):
  - R translates input x to input x'
  - S translates result z' to result z
- a function problem A is complete for a class FC if it is in FC and all problems in that class reduce to A
- FP and FNP are closed under reduction
- reductions of function problems compose

How to prove FP = FNP?

- FP = FNP only if P = NP

$\rightarrow$ to prove the theorem above: show that SAT $\in P$ implies FSAT $\in FP$

- this can be shown by constructing a satisfying truth assignment
- SAT$'$ is a formular φ plus an assignment that satisfies φ
- assignment as clauses that connects the single variables or their negation with $\wedge$
- algorithm for FSAT with the help of SAT already described above

Cryptography

Cryptography argument [1, 6, 7]:

- P vs. NP problem is an unsolved problem
- currently clear: a correct solution to an NP problem can be checked for correctness in polynomial time
- experts wish NP problems to remain almost unsolvable because of cryptography
- complexity in cryptography is not only desirable, but necessary
- important to know that most encryption methods used today are based solely on the fact that the effort to *guess* the key is too high
  → problem of *guessing* is an NP problem
- proof of the solvability of NP problems means the end of all currently used encryption methods

→ cryptographic argument: if P=NP, no safe encoding exists

Total FNP

Total functions

Function problems in FNP that are quaranteed to never return *no* are called total problems.

In other words: A problem R in FNP is called total if for every input string x there is at least one string y such that $R(x, y)$.

- total problems sound like they are injective (for every input exists at least one output)

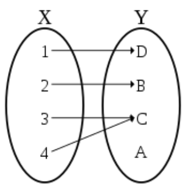

Figure 2: Visualization of injection [3]

FACTORING

Given an integer N.
Find its prime decomposition $N = p_1^{k_1}, p_2^{k_2}, ..., p_m^{k_m}$ together with the primality certificates of $p_1, ..., p_m$.

- FACTORING is a total function problem

Primality certificate:

- a primality certificate is a formal proof that a number is prime
- allows a number to be rapidly checked without having to run an expensive or unreliable primality test

Pina Kolling
piko0011

Example [4]:

- the factors of 15 are 3 and 5
- the factoring problem is to find 3 and 5 when given 15

- prime factorization requires splitting an integer into factors that are prime numbers
- every integer has a unique prime factorization
- FACTORING is in FNP
- no known polynomial algorithm for FACTORING

TFNP

The sucblass of FNP that contains all total functions problems is denoted as TFNP.

Questions:

- Can the terms *total function*, *total problem* and *total function problem* be used interchangeably?

References

[1] Dr Datenschutz (Website). *P vs. NP: Ein Geschenk der Informatik an die Mathematik.* Last opened 11.11.2022. URL: `https://www.dr-datenschutz.de/p-vs-np-ein-geschenk-der-informatik-an-die-mathematik/#:~:text=Hierbei%20werden%20von%20einem%20Computer,effizient%20l%C3%B6sen%20lassen%20oder%20nicht.`.

[2] Martin Berglund. *Lecture notes in Computational Complexity.*

[3] *Image source: Injective.* `https://en.wikipedia.org/wiki/Injective_function`.

[4] RSA Laboratories. *What is the factoring problem?* Website. Last opened 06.12.2022. URL: `http://security.nknu.edu.tw/crypto/faq/html/2-3-3.html`.

[5] Prof. Yuh-Dauh Lyuu. *Lecture slides on Complexity.* `https://www.csie.ntu.edu.tw/~lyuu/complexity/2010/20101130.pdf`. 2010.

[6] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[7] Prof. Dr. Thomas Schwentick. *Lecture notes in Grundbegriffe der theoretischen Informatik.* `https://www.cs.tu-dortmund.de/nps/de/Studium/Ordnungen_Handbuecher_Beschluesse/Modulhandbuecher/Archiv/Bachelor_LA_GyGe_Inf_Modellv/_Module/INF-BfP-GTI/index.html`.