How Some Problems Are More Complete than the Others

Yi Ren (002269013)
Wentao Lu (002276355)

Contents

1 Preliminaries

1.1 Basic notion

Problem: a problem is the abstract question to be solved.

Problem instance: a instance is the concrete input ω to a computational problem π.

Problem solution: the solution is the output of the problem with the given input.

Decision problems: decision problem [1] is a fundamental computational problem whose solution is always TRUE or FALSE (1 or 0), with given input and algorithm. We mainly focus on decision problems because other problems could be converted into decision problems.

Time: time depends on the number of steps that an algorithm uses with a particular input on the Turing Machine. Basically, it means the time it takes for a program to execute[2].

Space: space is the amount of storage used during computation or the locations on TM's work tapes (excluding the input tape) is ever visited by Machine's head[2].

Complexity: intuitively, complexity[2] is difficulty to solve a problem, but we could have some more precise measures with computational models, such as deterministic Turing Machine. We have various measures to quantify the complexity of a particular problem, such as time, space(storage) and amount of communication.

Deterministic Turing machine: DTM [4] is a sequence of moves, as determined by its transition function.

Nondeterministic Turing machine: NTM [4] is a TM that allows for a choice of next moves.

Offline Turing machines: an offline Turing machine is a multi tape TM with a separate read-only input tape [4], which defines the space complexity.

1.2 Complexity classes

A complexity class is a collection of sets that can be accepted by Turing machines with the same resources [4]. Some complexity classes could be defined by their resources constraint, including time and space.

T(n): M is a T(n) time-bounded Turing machine if for every input of length n, M makes at most T(n) moves before halting[2] .

S(n): M is an S(n) space-bounded Turing machine if, for every input of length n, M scans at most S(n) cells on any storage tape[2] .

DTIME(T(n)): the set of all languages having time-complexity T(n).

NTIME(T(n)): the set of all languages accepted by nondeterministic T(n) time-bounded Turing machines.

DSPACE(S(n)): the set of all languages having space-complexity S(n).

NSPACE(S(n)): the set of all languages accepted by nondeterministic S(n) time-bounded Turing machines.

P: class P contains problems that a deterministic Turing machine could solve within polynomial time. That is, T(n) = {DTIME(n^k) | k $\geq$ 1}

NP: class NP contains problems that a nondeterministic Turing machine could solve with polynomial time. That is, T(n) = {NTIME(n^k) | k $\geq$ 1}

L: class L contains problems that a deterministic Turing machine could solve within logarithmic space. That is, S(n) = DSPACE(log(n))

NL: class NL contains problems that a nondeterministic Turing machine could solve with logarithmic space. That is, S(n) = NSPACE(log(n))

POLYLOGSPACE: problems of class POLYLOGSPACE [2] could be solved by a deterministic Turing machine within polylogarithmic space. That is, S(n) = {DSPACE($log(n)^k$) | k $\geq$ 1}

1.3 Hardest Problems

To find the hardest problems in a complexity class, it could be unconvincing to compare one specific measure of a problem, such as time or space. Under such circumstances, it could be insignificant to compare one problem of less running time but more space with another problem of less space but more running time. Thus, we have some definitions for the hardness of a problem.

Definition: If π is the hardest [4] problem in a complexity class, any other problem $\pi^{'}$ in the same class could be reducted to it.

1.4 P versus NP

P versus NP problem is one of the seven well-known Millennium Prize Problems, and it focuses on whether the NP problem(could be verified in polynomial time) can also be solved within polynomial time(P problem). We regard it as a significant problem for the conjecture $P = NP$, which means all the NP problems, even the hardest (NPC problems[1]), can be solved efficiently (in polynomial time). That is, we can solve all the NP problems in polynomial time. Conversely, if $P \neq NP$ (or $P \subseteq NP$), some NP problems, particularly some hardest problems in NP, would have no solution within polynomial time.

1.5 Parallel Computation

With the rapid development of CPU manufacturing technology, it has become more and more challenging to improve single-core performance, so CPU manufacturers turned to develop multicore processors. Consequently, people began to put more effort into parallel computation research, which means a problem can be solved by more than one processor simultaneously. One traditional parallel model is the PRAM(parallel random access memory)[6] , which is a shared memory machine deployed with some parallel algorithms. To deal with read/write conflicts, there are some strategies for PRAM, which include EREW[5](Exclusive read exclusive write), CREW(Concurrent read exclusive write), ERCW(Exclusive read concurrent write), and CRCW(Concurrent read concurrent write). In recent years, people find it hard to keep processors synchronized on PRAM, so reconfiguration was raised to satisfy the need for this kind of problem. Compared with traditional models, reconfigurable models can make better use of hardware resources because the processors could be used to run the tasks when available. As a result, reconfigurable models[6] make it possible to solve the problem more efficiently.

2 P versus POLYLOGSPACE

To compare P and NP, we know the hierarchy between these two complexity classes, that is, $P \subseteq NP$. However, when it comes to P vs POLYLOGSPACE, we do not have a clue about the relationship between them. Thus, we are trying to find a complexity class that is a subset of both two classes.

2.1 Prove that NL $\subseteq$ P and NL $\subseteq$ POLYLOGSPACE.

Here we introduce class NL, which has been defined in section 1. Intuitively, NL is a subset of P and POLYLOGSPACE.

Theorem 1: The problem of finding a path from the starting configuration to the accepting configuration(STCON), can be solved by a DTM within DSPACE($log(n)^2$).

Proof: With the algorithm below, we can recursively find a path from the starting configuration to the accepting configuration. Since the recursion need $log(n)$ levels and $log(n)$ space for each level, totally we need $log(n)^2$ space to solve it on a DTM.

Algorithm 1 STCON

Input: starting configuration s, accepting configuration a, edge constraint k
Output: a boolean value which indicates if there exists a path from s to a.

```
function FindPath($s, a, k$)
    if $k == 0$ then
        return $s == t$
    end if
    if $k == 1$ then
        $e = (s, a)$
        for $i$ in $edges$ do
            if $e == i$ then
                return $True$
            end if
        end for
        return $False$
    end if
    for $i$ in $vertices$ do
        if FindPath(s, i, floor(k/2)) and FindPath(i, a, ceil(k/2)) then
            return $True$
        end if
    end for
    return $False$
end function
```

Theorem 2 (Savitch)[4]: If π is fully space-constructible and $S(n) \geq log(n)$, then NSPACE($S(n)$) $\subseteq$ DSPACE($S(n)^2$).

Proof: Given a NTM with space S(n) and alphabet Γ. Considering that we have an Off-line Turing Machine M, we define a configuration as the storage of work tape at one moment.[4] To obtain the space needed on a DTM, we regard these configurations as a graph G(V, E), with each node c_i represents a certain configuration, there exist an edge between c_i and c_j iff we can convert c_i to c_j in a single step, which means they are close enough. Totally, we have C configurations,where

$$C = |\Gamma|^{S(n)}$$

Evidently, the problem π could be solved on a DTM iff there exists a path in G(V, E) from the starting configuration to the accepting configuration. And according to Theorem 1, the space S of DTM is

$$S = log(|\Gamma|^{S(n)})^2 = S(n)^2$$

Finally, if a problem π can be solved on NTM within space $S(n)$, a DTM can solve the same problem in space $S(n)^2$.

Theorem 3: NL $\subseteq$ POLYLOGSPACE

For NL, we have

$$S(n) = log(n)$$

With respect to POLYLOGSPACE, we know that

$$POLYLOGSPACE = \cup\{DSPACE(log(n)^k) | k \geq 1\}$$

Hence, we have

$$NL = NSPACE(log(n)) \subseteq DSPACE(log(n)^2) \subseteq DSPACE(log(n)^k) \subseteq POLYLOGSPACE$$

That is,

$$NL \subseteq POLYLOGSPACE.$$

Theorem 4: The time complexity of DFS or BFS is $\mathbf{O(n^2)}$.

Theorem 5: NL $\subseteq$ P

For NL, we have

$$S(n) = log(n)$$

Which means the problem π can be solved with at most $log(n)$ space on a NTM. That is, the amount of storage used on the work tape is $log(n)$, given the input size n. Thus, we have C configurations, where

$$C = |\Gamma|^{log(n)} = |\Gamma|^{log_{|\Gamma|}(n)} = POLY(n)$$

Hence, NL can be solved within time T, where

$$T = C^2 = (POLY(n))^2 = POLY(n)$$

In this way, we can show that

$$NL = DTIME(POLY(n)) \subseteq P = DTIME(POLY(n))$$

That is,

$$NL \subseteq P$$

Finally, NL is the subset of both POLYLOGSPACE and P, which means there exists an intersection between two classes.

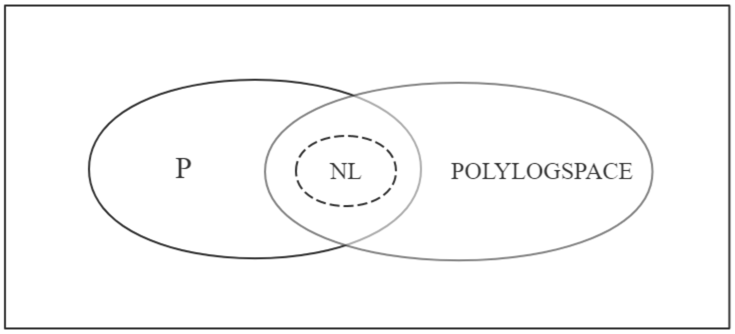

Figure 1: Relation of P, POLYLOGSPACE and NL

2.2 Define reductions for class P

Define reductions for the purpose of defining complete problems for the class P. Explain carefully how are your reductions closed under functional composition.

For class P, we define a PARALLEL-LOG reduction:

Definition: Let PARALLEL-LOG be a reduction that is computable in time $O(log(n)^c)$ using $O(n^k)$ parallel processors.

It's evident that PARALLEL-LOG reduction can be simulated by a single processor machine with time T, where

$$T = (log(n)^c) * (n^k) = POLY(n)$$

In this way, PARALLEL-LOG reduction also features polynomial-time restriction. If problem π_1 is in class P, and class π_1 can be PARALLEL-LOG reducted to π_2 , that is,

$$\tau_{\pi_1 \to \pi_2}$$

Then problem π_2 is in Class P, because π_1 is polynomial-time solvable and the reduction from π_1 to π_2 also features polynomial-time restriction. Thus we have

$$\pi_2 \in P$$

Finally, we can come to the conclusion that whenever π_1 is in class P and π_2 PARALLEL-LOG reduces to π_1, π_2 is in class P. Concerning a third problem π_3, if π_2 PARALLEL-LOG reduces to π_3, we can affirm that π_3 is also in class P under functional composition reduction.

Proof: If reduction $\tau_{\pi_1 \to \pi_2}(\omega)$ and $\tau_{\pi_2 \to \pi_3}(\omega)$ features PARALLEL-LOG on input ω, let $f(w)$, $g(w)$ be the reduction function of $\pi_1 \to \pi_2$ and $\pi_2 \to \pi_3$. We have $f(g(w))$ being polynomial, because the composition of two polynomials is another polynomial[8], which indicates that PARALLEL-LOG reduction is closed under functional composition for class P.

2.3 Define the class of P-complete problems

Define the class of P-complete problems. Explain carefully how showing that a P-complete problem is in PLOYLOGSPACE results in the whole class P being included in POLYLOGSPACE. Explain the practical consequences of finding that P-complete problems are or are not in POLYLOGSPACE.

Intuitively, P-complete problems are the hardest problems in class P. According to the definition of the hardest problem in section 1.3.

Definition: a problem π is P-complete iff

1. π is in P
2. Every problem π' in P can be reduced to π with PARALLEL-LOG reduction

Since P-complete problems are the most unsolvable individuals in class P, if any P-complete problem can be solved fast on a parallel machine, P-complete problems are in PLOYLOGSPACE, then every problem in class P will be included in PLOYLOGSPACE. In this way, we will have

$$P \subseteq PLOYLOGSPACE$$

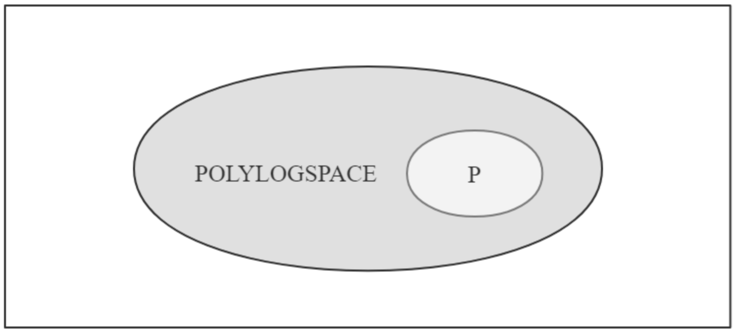

Figure 2: P in POLYLOGSPACE

Conversly, if no P-complete problem can be solved efficiently on a parallel machine, then P-complete problems are obviously lying outside POLYLOGSPACE, thus

$$P \neq PLOYLOGSPACE$$

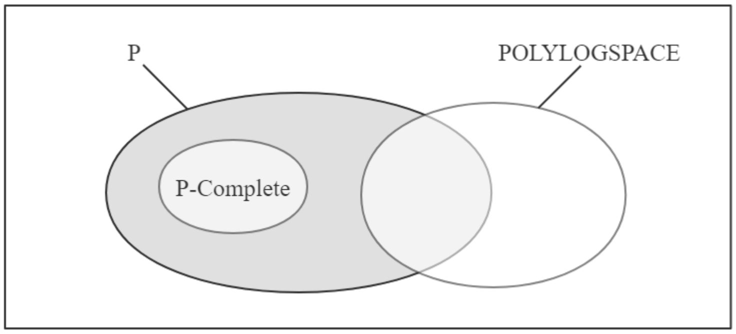

Figure 3: P-complete lies outside POLYLOGSPACE

So there exist some hard problems in class P that can not be solved fast with parallelism. That is, a parallel model cannot solve any of the P-complete problems within polylogarithmic time in parallel. Finally, some tractable problems remain inherently sequential.

3 P-complete problems

With the explicit definition of p-complete, we can introduce some examples of p-complete problems in this section. Basically, we know that p-complete problems cannot be parallelized with multi-processor models. So one basic problem is whether a Machine M halts in T steps, with given input ω. Until now, numerous problems in P have been proved P-hard.

3.1 Boolean formula value problem

The Boolean formula value problem is to determine the truth value of a variable-free Boolean formula[9].

Definition: Let Σ be the alphabet$\{\vee, \wedge, \neg, 0, 1, (,)\}$. The Boolean formulas are given by the following inductive definition:

(a) 0 and 1 are Boolean formulas.
(b) If and are Boolean formulas, then so are $(\neg\alpha)$, $(\alpha \wedge \beta)$ and $(\alpha \vee \beta)$.

The Boolean formula value problem is p-complete since it is a particular case of CVP(Circuit Value Problem) when the circuit is a tree. As proved by Ladner[7], all the CVP problems are p-complete, so does the Boolean formula value problem.

3.2 Linear programming

A general linear programming problem aims to optimize a linear function that is subject to a set of linear inequalities. Here we define as below,

Definition: Let f be the function and $c^T x$ be the constraints
$f(x_1, x_2, ..., x_n) = a_1x_1 + a_2x_2 + ... + a_nx_n = \sum_{i=1}^{n} a_ix_i$

$\{c^T x | x \in R^n \wedge Ax \le b \wedge x \ge 0\} =$

$$\begin{gathered} a_{11}x_1 + a_{12}x_2 \le b_1, \\ a_{21}x_1 + a_{22}x_2 \le b_2, \\ ... \\ a_{n1}x_1 + a_{n2}x_2 \le b_n \end{gathered}$$

Basically, linear programming is an essential technique for optimization problems; it is widely used in mathematics and economics fields. The completeness of Linear-programming has been proved by Serna[3]; he showed that approximating linear programming is P-complete under log-space reductions.

References

[1] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms,* MIT Press, Cambridge, MA, 3rd ed., 2009.

[2] R. Downey, *Turing's legacy: Developments from Turing's ideas in logic. In R. Downey (Ed.)*, Cambridge University Press, 2014, pp. 299-327.

[3] Maria Serna, *Approximating linear programming is log-space complete for P,* Information Processing Letters, Volume 37, Issue 4, 1991, pp. 233-236.

[4] M. SIPSER, *A introduction to the theory of computation,* Course Technology, 3rd ed., 2012.

[5] I. PARBERRY, *Parallel Complexity Theory,* John Wiley & Sons, New York, NY, 1987.

[6] R. Vaidyanathan AND J. L. Trahan *Dynamic Reconfiguration Architectures and Algorithms* Springer US, 2003.

[7] Richard E. Ladner. 1975. *The circuit value problem is log space complete for P.* S IGACT News 7, 1 (January 1975), 18–20.

[8] School Mathematics Study Group *Introduction to Algebra,* Yale University Press, 1965.

[9] Samuel R. Buss. *The Boolean formula value problem is in ALOGTIME,* In Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC'87), 1987, pp. 123-131.