Computational Complexity

计算复杂性

张方国

Email: isszhfg@mail.sysu.edu.cn

Lecture 13. More Resources, More Power?

Is it indeed the case that the more resources one has, the more one can achieve?

给图灵机更多的时间或空间是否就能扩大它所能求解的问题的类？

- Non-uniform complexity hierarchies
- Time Hierarchies and Gaps
- Space Hierarchies and Gaps

在20世纪50年代，Trahtenbrot和Rabin的论文被认为是计算复杂性理论这一领域最早的文献。(B Trahtenbrot, The impossibility of an algorithm for the decision problem for finite domains. Doklady Academii Nauk SSSR, 70, 569 – 572 (1950))

而一般说来，被公认为奠定了计算复杂性领域基础的是Hartmanis和Stearns的1960年代的论文On the computational complexity of algorithms。在这篇论文中，作者引入了时间复杂性类TIME(f(n))的概念，并利用对角线法证明了时间层级定理（Time Hierarchy Theorem）或时间谱系理论(这理论告诉我们图灵机在给予更多时间之后，保证能解决更多的问题。举例，必然存在问题是图灵机可以用n^2的时间解决，但是不能用n的时间解决)。

Non-uniform complexity hierarchies

The model of machines that use advice offers a very convenient setting for separation results.

Recall that every Boolean function is in $\mathcal{P}/2^n$ by virtue of a trivial algorithm that is given as advice the truth table of the function restricted to the relevant input length. An analogous algorithm underlies the following separation result.

Theorem 1. *For any two functions* $l', \delta : \mathbb{N} \to \mathbb{N}$ *such that* $l'(n) + \delta(n) \leq 2^n$ *and* δ *is unbounded, it holds that* $\mathcal{P}/l'$ *is strictly contained in* $\mathcal{P}/(l' + \delta)$.

Proof:

Time Hierarchies and Gaps

In this section we show that in the “reasonable cases” increasing time complexity allows for more problems to be solved, whereas in “pathological cases” it may happen that even a dramatic increase in the time complexity provides no additional computing power.

The “reasonable cases” correspond to time bounds that can be determined by the algorithm itself within the specified time complexity.

Time Hierarchies

Both theorems use the notion of a **time-constructible function**.

Definition 1. *(time constructible functions) A function* $t : \mathbb{N} \to \mathbb{N}$ *is called time constructible if there exists an algorithm that on input n outputs* $t(n)$ *using at most* $t(n)$ *steps.*

i.e., if there exists a deterministic Turing machine such that for every n, if the machine is started with an input of n ones, it will halt after precisely t(n) steps.

Equivalently, we may require that the mapping $1^n \to t(n)$ be computable within time complexity t.

For a fixed model of computation and for any function $t : \mathbb{N} \to \mathbb{N}$, we denote by $D_{TIME}(t)$ the class of decision problems that are solvable in time complexity t.

D_{TIME}或DTIME (或者TIME)是一个图灵机的计算资源或者计算时间的计量方式。它代表一个”普通”有实体的电脑解决特定计算问题，使用特定算法，所要花费的时间。这个计算资源是最被广泛研究的计算资源，因为它与真实世界所重视的资源(要花费多少时间才能计算出一个问题)息息相关。

DTIME这个资源常被使用来定义复杂度类，亦即，可以在特定时间内解决的决定性问题其集合。如果一个问题其输入的大小为n，并且可要求f(n)的计算时间来解决，那我们说这问题在DTIME(f(n)) (or TIME(f(n)))里面。

许多重要的复杂度类都使用DTIME来定义，这些类别包含需要花费特定时间才能解决的问题，来作为分类。

DTIME满足时间谱系理论，这代表在渐进分析内较大的时间，所产生的时间复杂度类严格大于(大于且不等于)其他时间复杂度类。

有名的复杂度类P代表所有可以在多项式内的DTIME解决的问题。我们可以正式定义为：

$$P = \bigcup_{k \in \mathbb{N}} DTIME(n^k)$$

使用DTIME的复杂度类是EXPTIME，包含了代表所有可以在指数时间内以图灵机解决的问题。正式定义为：

$$EXPTIME = \bigcup_{k \in \mathbb{N}} DTIME(2^{n^k})$$

The Time Hierarchy Theorem:

The time hierarchy theorem for deterministic Turing machines was proven by Richard Stearns and Juris Hartmanis。

Hartmanis, J.; Stearns, R. E. (1 May 1965). “On the computational complexity of algorithms”. Transactions of the American Mathematical Society (American Mathematical Society) 117: 285 – 306.

As a consequence, for every deterministic time-bounded complexity class, there is a strictly larger time-bounded complexity class, and so the time-bounded hierarchy of complexity classes does not completely collapse.

Theorem 2 (time hierarchy for two tape Turing machines)**.** *For any time constructible function* t_1 *and every function* t_2 *such that* $t_2(n) \geq (log t_1(n))^2 \cdot t_1(n)$ *and* $t_1(n) > n$ *it holds that*$D_{TIME}(t_1)$ *is strictly contained in* $D_{TIME}(t_2)$.

Proof of Theorem 4.3: The idea is constructing a Boolean function f such that all machines having time complexity t_1 fail to compute f. This is done by associating each possible machine M a different input x_M (e.g., $x_M = \langle M \rangle$) and making sure that $f(x_M) \neq M'(x_M)$, *where* $M'(x)$ *denotes an emulation of* $M(x)$ *that is suspended after* $t_1(|x|)$ *steps.* For example, we may define $f(x_M) = 1 - M'(x_M)$. We note that M' is used instead of M in order to allow computing f in time that is related to t_1. The point is that M is just an arbitrary machine that is associated to the input x_M, and so M does not necessarily run in time t_1 (but, by construction, the corresponding M' does run in time t_1).

Implementing the foregoing idea calls for an efficient association of machines to inputs as well as for a relatively efficient emulation of t_1 steps of an arbitrary machine. As shown next, both requirements can be met easily. Actually, we are going to use a mapping μ of inputs to machines (i.e., μ will map the aforementioned x_M to M) such that each machine is in the range of μ and μ is very easy to compute (e.g., indeed, for starters, assume that μ is the identity mapping). Thus, by construction, $f \notin \text{DTIME}(t_1)$. The issue is presenting a relatively efficient algorithm for computing f; that is, showing that $f \in \text{DTIME}(t_2)$.

Corollary 1 (time hierarchy for any reasonable and general model)**.** *For any reasonable and general model of computation there exists a positive polynomial p such that for any time-computable function t_1 and every function t_2 such that $t_2(n) \geq p(t_1)$ and $t_1(n) > n$ it holds that$D_{TIME}(t_1)$ is strictly contained in $D_{TIME}(t_2)$.*

Proof of Corollary 4.4: Letting DTIME_2 denote the classes that correspond to two-tape Turing machines, we note that $\text{DTIME}(t_1) \subseteq \text{DTIME}_2(t'_1)$ and $\text{DTIME}(t_2) \supseteq \text{DTIME}_2(t'_2)$, where $t'_1 = \text{poly}(t_1)$ and t'_2 is defined such that $t_2(n) = \text{poly}(t'_2(n))$. The latter unspecified polynomials, hereafter denoted p_1 and p_2 respectively, are the ones guaranteed by the Cobham-Edmonds Thesis. Also, the hypothesis that t_1 is time-computable implies that $t'_1 = p_1(t_1)$ is time-constructible with respect to the two-tape Turing machine model. Thus, for a suitable choice of the polynomial p (i.e., $p(p_1^{-1}(m)) \geq p_2(m^2)$), it holds that

$$t'_2(n) = p_2^{-1}(t_2(n)) > p_2^{-1}(p(t_1(n))) = p_2^{-1}(p(p_1^{-1}(t'_1(n)))) > t'_1(n)^2 ,$$

where the last inequality holds by the choice of p and the first inequality holds by the corollary's hypothesis (i.e., $t_2 > p(t_1)$). Invoking Theorem 4.3 (while noting that $t'_2(n) > t'_1(n)^2$), we have $\text{DTIME}_2(t'_2) \supset \text{DTIME}_2(t'_1)$. Combining this with the aforementioned relations between DTIME and DTIME_2, the corollary follows. ■

确定性时间分层定理

TH: 如果f,g是满足$f(n)\log f(n)=o(g(n))$的时间可构造函数，则

$$\mathbf{DTIME(f(n)) \subsetneq DTIME(g(n))}$$

Proof.

采用对角线方法，将函数表中第x行x列取反（使用通用图灵机$\mathcal{M}$模拟，如果不能在$g(n)$的时间内算出来则停机并输出0），则：该函数不可在$f(n)\log f(n)$的时间内计算，但可在$g(n)$的时间内计算。

采用反证法，如果存在$f(n)\log f(n)$时间内计算上述函数的机器，设该机器为M_a，则M_a输入a如果能在$f(a)\log a < g(a)$的时间内停机，则与取反的条件矛盾，否则不能在$f(a)\log a$步内停机，产生矛盾。

Impossibility of speed-up for universal computation

The Time Hierarchy Theorem implies that the computation of a universal machine cannot be significantly sped up. That is, consider the function $u'(< M >, x, t) = y$ if on input x machine M halts within t steps and outputs the string y, and $u'(< M >, x, t) = \perp$ if on input x machine M makes more than t steps. Recall that the value of $u'(< M >, x, t)$ can be computed in $\widetilde{O}(|x| + | < M > |t)$ steps. The Time Hierarchy Theorem implies that $u'(< M >, x, t)$ cannot be computed within significantly less steps, i.e.,

Theorem 3. *There exists no two-tape Turing machine that, on input* $< M >, x$ *and* t*, computes* $u'(< M >, x, t)$ *in* $o((t + |x|) f(M)/log^2(t + |x|))$ *steps, where* f *is an arbitrary function.*

Proof: Suppose (towards the contradiction) that, for every fixed M, given x and $t > |x|$, the value of $\mathtt{u}'(\langle M\rangle, x, t)$ can be computed in $o(t/\log^2 t)$ steps, where the o-notation hides a constant that may depend on M. Consider an arbitrary time constructible t_1 (s.t. $t_1(n) > n$) and an arbitrary set $S \in \text{DTIME}(t_2)$, where $t_2(n) = t_1(n) \cdot \log^2 t_1(n)$. Let M be a machine of time complexity t_2 that decides membership in S, and consider an algorithm that, on input x, first computes $t = t_1(|x|)$, and then computes (and outputs) the value $\mathtt{u}'(\langle M\rangle, x, t\log^2 t)$. By the time constructibility of t_1, the first computation can be implemented in t steps, and by the contradiction hypothesis the same holds for the second computation. Thus, S can be decided in $\text{DTIME}(2t_1) = \text{DTIME}(t_1)$, implying that $\text{DTIME}(t_2) = \text{DTIME}(t_1)$, which in turn contradicts Theorem 4.3. ■

Hierarchy Theorem for Non-deterministic Time:

The time hierarchy theorem for nondeterministic Turing machines was proven by Stephen Cook.

Stephen Cook (1972). A hierarchy for nondeterministic time complexity. Proceedings of the fourth annual ACM symposium on Theory of computing, pp. 187 – 192.

复杂度类NTIME(f(n))是一种可以用非确定型图灵机使用O(f(n))的时间和无限制的空间所能解决的所有决定性问题的集合。NP这个有名的复杂度类，可以用NTIME来定义如下：

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

NEXPTIME这个复杂度类是由NTIME定义出来的，非决定型的时间谱

系理论说明了非决定型的机器在使用更多时间的前提下可以解决更多的问题。

Analogously to DTIME, for a fixed model of computation (to be understood from the context) and for any function $t : \mathbb{N} \to \mathbb{N}$, we *denote by* NTIME$(t)$ *the class of sets that are accepted by some non-deterministic machine of time complexity* t. Alternatively, analogously to the definition of $\mathcal{NP}$, a set $S \subseteq \{0,1\}^*$ is in NTIME(t) if there exists a *linear-time* algorithm V such that the two conditions hold

1. For every $x \in S$ there exists $y \in \{0,1\}^{t(|x|)}$ such that $V(x,y) = 1$.
2. For every $x \notin S$ and every $y \in \{0,1\}^*$ it holds that $V(x,y) = 0$.

Theorem 4. *(non-deterministic time hierarchy for two-tape Turing machines:) For any time constructible and monotonicly non-decreasing function* t_1 *and every function* t_2 *such that* $t_2(n) \geq (logt_1(n+1))^2 \cdot t_1(n+1)$ *and* $t_1(n) > n$ *it holds that*$N_{TIME}(t_1)$ *is strictly contained in* $N_{TIME}(t_2)$.

非确定性时间分层定理

TH:如果f,g是满足f(n+1)=o(g(n))的时间可构造函数，则

$$\mathbf{NTIME(f(n)) \subseteq NTIME(g(n))}$$

首先找函数h，使得$f(n+1)=o(h(n+1)), h(n+1)=o(g(n))$（比如$h(n+1)=f(n+1)\log\frac{g(n)}{f(n+1)}$），然后令$T(1)=1, T(n)=2^{h(T(n-1))}$. 将函数表第i行的$[T(i)+1, T(i+1)]$标记，考虑如下函数：
$\forall n$，找到标记第n列的行i，在$g(n)$的时间内计算$M_i(n+1)$，如果输出1，则$f(n)=1$，如果输出0或没有停机，则输出0. 如果$n=T(i+1)$，则指数时间对$M_i(T(i))$取反，则该函数可$g(n)$时间内计算，但不能在$f(n)$时间内计算，否则假设该图灵机为M，在第i行可得出矛盾（见P55图）。没有log的是因为通用非确定性图灵机可在常数倍时间内模拟其他机器。

Time Gaps and Speedup

In contrast to Theorem 4.3, there exists functions $t : \mathbb{N} \to \mathbb{N}$ such that $\text{DTIME}(t) = \text{DTIME}(t^2)$ (or even $\text{DTIME}(t) = \text{DTIME}(2^t)$). Needless to say, these functions are not time-constructible (and thus the aforementioned fact does not contradict Theorem 4.3). The reason for this phenomenon is that, for such functions t, there exists not algorithms that have time complexity above t but below t^2 (resp., 2^t).

Theorem 4.7 (the time gap theorem): *For every non-decreasing computable function $g : \mathbb{N} \to \mathbb{N}$ there exists a non-decreasing computable function $t : \mathbb{N} \to \mathbb{N}$ such that* $\text{DTIME}(t) = \text{DTIME}(g(t))$.

The forgoing examples referred to $g(m) = m^2$ and $g(m) = 2^m$. Since we are mainly interested in dramatic gaps (i.e., super-polynomial functions g), the model of computation does not matter here (as long as it is reasonable and general).

Speedup Theorems

Theorem 5. *(the time Speed-up Theorems:) For every computable(and super-linear) function g there exists a decidable set S such that if* $S \in D_{TIME}(t)$ *then* $S \in D_{TIME}(t')$ *for* t' *satisfying* $g(t'(n)) < t(n)$.

Time Gaps and Speedup

There are analogous of Hierarchy and Gap Theorems in space complexity.

For any function $s : \mathbb{N} \to \mathbb{N}$ we denote by $D_{SPACE}(s)$ the class of decision problems that are solvable in space complexity s.

A function $s : \mathbb{N} \to \mathbb{N}$ **space constructible** if there exists an algorithm that on input n outputs $s(n)$ using at most $s(n)$ cells of the work-tape.

Theorem 6. *(space hierarchy for three-tape Turing machines) For any space constructible function s_2 and every function s_1 such that $s_2 = \omega(s_1)$ and $s_1(n) > \log n$ it holds that $D_{SPACE}(s_1)$ is strictly contained in $D_{SPACE}(s_2)$.*

Thank You Very Much!