Chapter 3

The Power of Increasing Resources

Now that the models and complexity measures have been defined and understood, we are in a position to pose some typical motivating questions in complexity theory.

1. More vs. Less: Given more time or space, can machines in a fixed model necessarily solve more difficult problems? This is the topic of this chapter.
2. Resource and Model comparison: What is the relationship between time and space? What are the relationships among deterministic, nondeterministic, and alternating Turing machines? This is the topic of Chapter 4.
3. Feature comparison: Are $k+1$ tape machines more powerful than k tape machines? Are indexing machines more powerful than non-indexing machines? Some of these questions will be addressed in each of these chapters.

3.1. Constant Factor Speedup Theorems

This section answers the simplest questions of the first type: doubling the space or time adds no computational power.

Theorem 3.1: For every $\epsilon > 0$ and every space bound $S(n)$,

$$\text{DSPACE}(S(n)) \subseteq \text{DSPACE}(\lceil \epsilon S(n) \rceil),$$

and similarly for NSPACE and ASPACE.

Proof: Let $c = \lceil 1/\epsilon \rceil$. Let M be a deterministic Turing machine with worktape alphabet Γ that runs in space $S(n)$. Construct a deterministic Turing machine N with worktape alphabet Γ^c, and compress every c symbols on each worktape of M into one symbol on the corresponding worktape of N. (See Figure 3.1 for an example with $\epsilon = 1/3$.) The remaining details are left as an exercise. □

Theorem 3.2: For every $\epsilon > 0$ and every time bound $T(n) \geq n$,

$$\text{DTIME}(T(n)) \subseteq \text{DTIME}(n + \lceil \epsilon T(n) \rceil),$$

and similarly for NTIME.

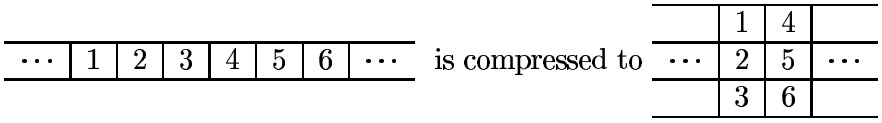

Figure 3.1: Compressing a Worktape for Constant Space Reduction

Proof: One might be inclined to use the same proof as in Theorem 3.1, but this falls a bit short. If the symbols are grouped as in that proof, the Turing machine might oscillate between adjacent cells on the uncompressed tape that were in different cells on the compressed tape (for instance, cells 3 and 4 in Figure 3.1), so the construction fails to yield any speedup. The solution is to overlap the contents of adjacent compressed cells.

Let $c = 2\lceil 1/\epsilon \rceil$. Let M be a deterministic Turing machine with worktape alphabet Γ that runs in time $T(n) \geq n$. We construct a deterministic Turing machine N with worktape alphabet Γ^{3c-2}, whose compressed tapes are illustrated in Figure 3.2 for the case $\epsilon = 1/2$.

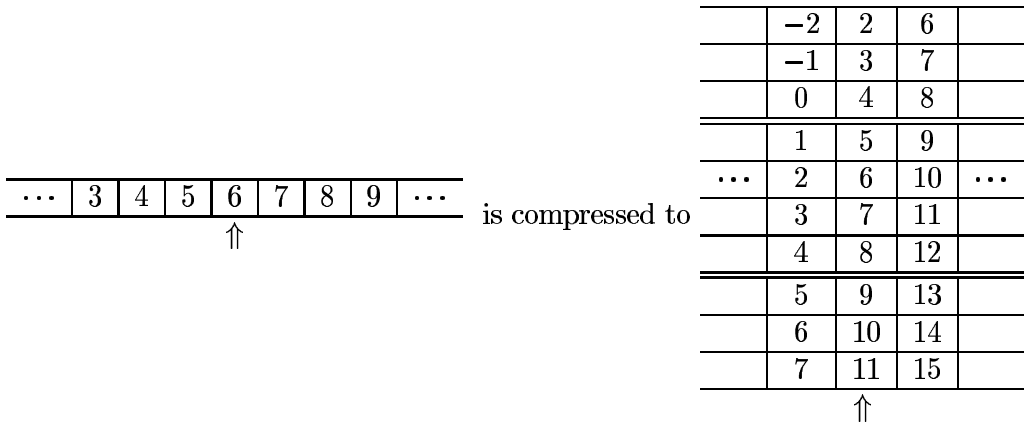

Figure 3.2: Compressing a Worktape for Constant Time Reduction

If M begins the next c steps by reading tape cell i, then N's head is reading the tape cell that has i in its middle c tracks. In one step of N we can simulate the next c moves of M before having to move N's heads. After these c steps of M, N repositions its head if necessary, carrying in its finite control the updated values of the $2c-2$ overlapping cells of M. Any inconsistent values in the opposite direction on this tape will be corrected when N's head passes through the cell in that direction.

N starts the simulation by compressing the input tape of M onto an extra worktape and "rewinding" that tape (i.e., moving its head to the left end). It takes n steps to compress the input tape, n/c to rewind, and $T(n)/c$ to simulate the $T(n)$ steps of M. Hence, the time used by N is at most

$$n + \frac{n}{c} + \frac{T(n)}{c} \leq n + \frac{2T(n)}{c} \leq n + \epsilon T(n).$$

□

Theorem 3.2 shows how to speed up deterministic and nondeterministic time by a constant factor. The same simulation does not work for alternating time, since the machine being simulated may alternate several times during the next c steps, whereas the simulating machine has only 1 corresponding step, which must be either existential or universal. Nonetheless, a different technique does work for alternating time:

Theorem 3.3: For every $\epsilon > 0$ and every time bound $T(n) \geq \log_2 n$,

$$\text{ATIME}(T(n)) \subseteq \text{ATIME}\left(\lceil \log_2 n \rceil + \lceil \epsilon T(n) \rceil\right).$$

Proof: The proof is left as an exercise. (Hint: simulate c steps by 2 steps rather than 1.) □

3.2. Hierarchy Theorems

The original motivation behind Theorems 3.1, 3.2, and 3.3 remains: is it possible to enhance the power of a Turing machine by allowing it more time or more space? In some sense the constant factor speedups are cheats, since there is a corresponding blowup in the alphabet size and transition function. They are interesting, however, because they tell us not to look for added power within a constant factor increase in time or space. In this section we will consider what happens when we increase the time or space by more than a constant factor.

3.2.1. The Deterministic Space Hierarchy Theorem

Definition 3.4: $S(n)$ is *(deterministic) space constructible* if and only if there is a deterministic Turing machine M with one worktape such that, on all inputs x, M halts after visiting (and marking) exactly $S(|x|)$ cells on its worktape.

Example 3.5: Any common function $S(n)$ that you will encounter is space constructible. For example, the functions $\lceil \log_2 n \rceil$, n, n^2, 2^n, and $n!$ are all space constructible. If $S(n)$ and $T(n)$ are space constructible, then so are $S(n) + T(n)$, $S(n)T(n)$, and $(S(n))^{T(n)}$. The proofs of these facts are left as exercises.

Example 3.6: The function $\lceil \log_2 \log_2 \log_2 n \rceil$ is not space constructible.

Example 3.7: The function $S(n)$ given below is not space constructible, since it is not even computable.

$$S(n) = \begin{cases} n & \text{if } n \text{ is the binary encoding of a deterministic Turing machine that always halts} \\ 2n & \text{otherwise} \end{cases}.$$

Theorem 3.8 is the deterministic space hierarchy theorem.

Theorem 3.8 (Hartmanis, Lewis, and Stearns [14]): For any space constructible function $S(n) = \Omega(\log n)$, and any $s(n) = o(S(n))$, [1]

$$\text{DSPACE}(S(n)) - \text{DSPACE}(s(n)) \neq \emptyset.$$

Proof: By Theorem 3.1, we can assume without loss of generality that $S(n) \geq 2\log_2 n + 2$ for all sufficiently large n. We will construct a deterministic Turing machine M that runs in space $O(S(n))$ and disagrees on at least one input with every possible deterministic Turing machine that runs in space $s(n) = o(S(n))$.

Construct M as follows. On input $x = 0^i 1 j$, where $i \geq 0$ and $j \in \{0,1\}^*$, with $|x| = n$,

1. M marks off $S(n)$ cells on each of two worktapes. This can be done in space $S(n)$, by Definition 3.4.
2. M treats j as the binary encoding of some deterministic Turing machine T_j and, except as modified below, simulates T_j on input x, recording successive binary encodings of T_j's configurations within the marked cells of one of its worktapes. (If j does not encode a deterministic Turing machine, then M rejects x.)
   (a) If T_j halts in a final configuration, then M halts in a nonfinal configuration.
   (b) If T_j halts in a nonfinal configuration, then M halts in a final configuration.
   (c) If the encoding of one of T_j's configuration runs over the $S(n)$ marked cells, then M accepts.
   (d) If T_j runs for more than $2^{S(n)}$ steps, as timed on the second marked-off tape, then M accepts.

By construction, M uses space $O(S(n))$.

All that remains to show is that, for any deterministic Turing machine T running in space $s(n) = o(S(n))$, $L(T) \neq L(M)$.

Let $T = T_j$. There is a constant c_j such that any configuration of T_j on any accepted input can be encoded in binary using $c_j s(n) + \lceil \log_2 n \rceil$ bits, where c_j depends only on the size of T_j's worktape alphabet, the number of tapes in T_j, and the number of states in T_j, and the $\lceil \log_2 n \rceil$ bits encode the input head position. Note that T_j has at most $2^{c_j s(n) + \lceil \log_2 n \rceil}$ distinct configurations.

Since $S(n) \geq 2\log_2 n + 2$ and $s(n) = o(S(n))$,

$$\lim_{n\to\infty} \frac{s(n)}{S(n) - \lceil \log_2 n \rceil} \leq \lim_{n\to\infty} \frac{s(n)}{S(n)/2} = 0 < \frac{1}{c_j}.$$

Thus, for sufficiently large n,

$$\frac{s(n)}{S(n) - \lceil \log_2 n \rceil} \leq \frac{1}{c_j},$$

[1] Actually, $s(n) = o(S(n))$ can be replaced by the weaker condition

$$\lim_{n\to\infty} \inf \frac{s(n)}{S(n)} = 0,$$

but in practice the condition given always suffices.

which implies that, for sufficiently large n,

$$c_j s(n) + \lceil \log_2 n \rceil \leq S(n). \tag{3.1}$$

Choose any $n > |j|$ satisfying Inequality (3.1), and let $x = 0^{n-|j|-1}1j$. Then M accepts x if and only if T_j does not accept x: If T_j accepts or rejects x while M uses space at most $S(n)$ and T_j uses time at most $2^{S(n)}$ (cases 2a and 2b), then M halts with the opposite answer. If M uses more space than $S(n)$ on x then, by Inequality (3.1), T_j uses more space than $s(n)$ and hence cannot accept x, whereas by case 2c, M accepts x. Finally, if T_j runs for time greater than $2^{S(n)} \geq 2^{c_j s(n)+\lceil \log_2 n \rceil}$, then T_j must have repeated some configuration and hence cannot accept x, whereas by case 2d, M accepts x. □

As a consequence of Theorem 3.8, there are arbitrarily complex languages among the recursively enumerable languages.

3.2.2. The Deterministic Time Hierarchy Theorem

Because of the need to diagonalize over all multitape machines with some fixed number of tapes, the deterministic time hierarchy theorem is not quite as tight as the deterministic space hierarchy theorem. In Lemma 4.8 we will show how to simulate a multitape deterministic Turing machine running in time $T(n)$ by a one-tape deterministic Turing machine running in time $O((T(n))^2)$. To get the tightest known deterministic time hierarchy theorem, we will need a better time bound, which is achieved at the expense of extra worktapes:

Lemma 3.9 (Hennie and Stearns [17]): For any k, if L is a language accepted by a k-worktape deterministic Turing machine in time $T(n)$, then L is accepted by a 2-worktape deterministic Turing machine in time $O(T(n) \log T(n))$.

Open Problem 3.10: Simulate k tapes by any constant number of tapes with a blowup in time that is $o(\log T(n))$.

Definition 3.11: $T(n)$ is *(deterministic) time constructible* if and only if there is a deterministic Turing machine that, on all inputs x, runs for exactly $T(|x|)$ steps and then halts.

Theorem 3.12 is the deterministic time hierarchy theorem.

Theorem 3.12 (Hartmanis and Stearns [15]): For any function $t(n) \geq n$, and any time constructible function $T(n) = \omega(t(n) \log t(n))$, [2]

$$\mathrm{DTIME}(T(n)) - \mathrm{DTIME}(t(n)) \neq \emptyset.$$

[2] As in Theorem 3.8, $T(n) = \omega(t(n) \log t(n))$ can be replaced by the weaker condition

$$\lim_{n \to \infty} \inf \frac{t(n) \log t(n)}{T(n)} = 0.$$

Proof: The proof is similar to that of Theorem 3.8, but uses Lemma 3.9 in order to simulate T_j (which may have arbitrarily many worktapes) by M (which has a fixed number of worktapes). □

A typical consequence of the deterministic time hierarchy theorem is that DTIME(n) $\subset$ DTIME(n^2). This is because $n^2 = \omega(n \log n)$. However, that theorem is not tight enough to have as an immediate consequence that DTIME(n) $\subset$ DTIME($n \log n$).

Exercise 3.13: Consider what might further complicate the hierarchy theorems for nondeterministic Turing machines. Hint: It is not a problem for alternating Turing machines.

3.3. Exercises

1. Complete the details of Theorem 3.1.

2. Prove Theorem 3.3.

   (Hint: Convert the alternating Turing machine into one in which at most one input symbol is read on any path of the computation tree.)

3. Prove that, if $S(n)$ and $T(n)$ are space constructible, then so are $S(n)+T(n)$, $S(n)T(n)$, and $(S(n))^{T(n)}$.

4. Prove Theorem 3.12. You may assume the result in Lemma 3.9.

5. Do Exercise 3.13.