Space Complexity: Short Guide

Vitaly Parnas

March 23, 2018

I briefly introduce the notion of Space Complexity[1] in the scope of other complexity classes, and then proceed to discuss the relevant complexity classes PSPACE, NPSPACE, L, NL, as well as PSPACE and NL completeness.

Helpful prerequisites: Asymptotic notation, set-builder notation, Turing machines, computational complexity.

Space Complexity

The space complexity of machine M is defined by the number of cells scanned by M. A Space($f(n)$) problem constitutes one computable in $f(n)$ tape cells by a deterministic Turing Machine. Similarly, a NSpace($f(n)$) problem yields a computation in $f(n)$ tape cells by a nondeterministic Turing Machine.

Intuitively, space, unlike time, can be reused. The fundamental factor behind the power of space and the theorems that follow, lies in this very notion of reusable space. A machine M cannot belong to a space complexity class of greater measure than time, since M cannot explore more tape cells than the number of time units for which it executes. On the contrary, a problem often requires a small fraction of space in comparison to time. For example, a function featuring multiple recursive calls only consumes the space of one, since each recursive call executes sequentially, reusing the space of the previous.

Savitch's Theorem

Probably the most important discovery in space complexity is the Savitch's theorem, and for that reason I'll present the proof in considerable detail.

Theorem. $NSpace(f(n)) \subseteq Space(f^2(n))$

Proof. Let N be the NTM computing a problem in $f(n)$ space. We want to construct a deterministic machine M to compute the same problem in space $f^2(n)$.

First, how do we simulate a nondeterministic computation deterministically in general, while remaining problem oblivious? We must explore N at the machine configuration level. Specifically, we determine if we can reach an accepting configuration from the initial. Because a nondeterministic execution visually resembles a tree, we cannot simply store one configuration at a time on M's tape without loosing track of our location and what branches we've already explored. Consequently, to anticipate the worst case, we could store all the N configurations on the M working tape with states Q and the tape alphabet Γ, but this amounts to

$$|Q|f(n)|\Gamma|^{f(n)} = O(f(n))2^{O(f(n))} = 2^{O(f(n))}$$

configurations and an exponential amount of space unacceptable for our purposes.

To simplify convention, let's define the configuration reachability problem by the function $canyield(c_1, c_2, t)$, which returns true if we can reach configuration c_2 from c_1 in t steps or less. We want to solve $canyield(c_{start}, c_{accept}, 2^{df(n)})$, choosing d appropriately to insure a sufficient upper bound on the number of configurations.

Provided that a sequential solution has failed, let's attempt a divide and conquer solution:

```
*canyield*$(c_1, c_2, t)$
    if $t = 1$:
        if $c_1 = c_2$, or $c_2$ reachable in one step:
            accept.
    else reject.
    if $t > 1$:
        for each configuration $c_m$ of $N$:
            canyield$(c_1, c_m, t/2)$
            canyield$(c_m, c_2, t/2)$
            if both accept: accept
    reject
```

We still examine a hopelessly exponential number of total configurations with two recursions per all possible configurations per level. However, the number of recursive *levels*, which concerns us most in space complexity by virtue of reusing space, results in

$$\begin{aligned} f(t) &= f(t/2) + O(1) = O(\log t) \\ &= O(\log 2^{O(f(n))}) = O(f(n)) \end{aligned}$$

And since we require $O(f(n))$ space per call to check for terminating conditions, M uses $O(f^2(n))$ space overall.[1] □

PSPACE and NPSPACE

Analogous to the time complexity classes P and NP, PSPACE represents problems solvable in polynomial (n^k) space on a deterministic TM, and NPSPACE those solvable polynomially on a nondeterministic TM. However, in light of Savitch's theorem, PSPACE = NPSPACE, an incredible result demonstrating the resolution of any NPSPACE problem by a deterministic TM with equal space efficiency (to a difference of only a polynomial factor.) It follows that

$$\begin{aligned} &P \subseteq PSPACE \\ &\text{and } NP \subseteq NPSPACE = PSPACE \\ &\Rightarrow NP \subseteq PSPACE \end{aligned}$$

reemphasizing the power of space with respect to time.

PSPACE Completeness and TQBF

Similar to time complexity class completeness, language B is PSPACE-complete if 1) $B \in PSPACE$ and 2) for all $A \in PSPACE$, A polynomially reduces to B (B is PSPACE-hard). And similarly to the SAT problem polynomially reduced from all other NP problems by the Cook-Levin theorem[2], the TQBF problem serves this baseline role in establishing PSPACE completeness.

$TQBF$ = set of all truly fully quantified boolean formulas.

A TQBF formula Φ in *prenex normal form* incorporates all quantifiers prior to the clauses as in the following example:

$$\Phi = \forall x \exists y [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$$

Theorem. *TQBF is PSPACE complete.*

Proof. TQBF $\in$ PSPACE:

We demonstrate this by a function f that receives a formula Φ as input and iterates through all variables one by one, generating two recursive calls: one with the modified formula setting the variable to true, and the other set to false. Depending on the type of quantifier ($\forall$ or $\exists$), the function asserts appropriately. If no variables are left, the function simply evaluates the remaining boolean equation.

Provided that f receives input of length n, evaluating one variable per recursion call, it generates $O(n)$ recursion levels, each using maximum $O(n)$ space to store the formula or evaluate the base condition. The overall computation thus runs in PSPACE.

TQBF is PSPACE-hard:

Let A be a PSPACE problem decidable by machine M in polynomial space. We'll construct a polynomial reduction from A to TQBF as follows, such that M accepts input w iff formula Φ is true.

[1] The *canyield* algorithm also demonstrates that a general reachability problem consisting of a graph with n vertices is deterministically solvable in $O(\log^2 n)$ space.

Similar to the Cook-Levin and the Savitch's theorem, we explore M at the configuration level, and construct Φ to represent a successful path from the start to the accept configuration of M. Specifically, let $\Phi_{c_1,c_2,t}$ be true iff M can reach configuration c_2 from c_1 in under t steps. Φ then becomes $\Phi_{c_{start},c_{accept},2^{df(n)}}$, with constant d properly set to represent a sufficient total number of configurations in M.

c_i actually encodes an entire M configuration consisting of $O(n^k)$ variables, in accordance with PSPACE constraints.

Our base case involves $t = 1$, in which case the respective formula represents either a check for c_1 and c_2 equality or a single-step transition.

For $t > 1$, using divide and conquer, we can recursively construct the formula as follows:

$$\Phi_{c_1,c_2,t} = \exists c_m[\Phi_{c_1,c_m,t/2} \wedge \Phi_{c_m,c_2,t/2}]$$

This reduction would work if not for the space doubling in size at each recursion level. After $\log t = \log(2^{O(f(n))}) = O(f(n))$ recursion levels, Φ will occupy $O(f(n))2^{O(f(n))} = 2^{O(f(n))}$ space, an unacceptable exponential amount.

A clever formula redesign leads to the following:

$$\Phi_{c_1,c_2,t} = \exists c_m \forall (c_3, c_4) \in \{(c_1, c_m), (c_m, c_2)\}[\Phi_{c_3,c_4,t/2}]$$

This reduces Φ from two to only one recursive subformula, at the expense of a constant size formula increase. With $\log t = O(f(n))$ recursion levels, and $O(f(n))$ space consumed per level, the resulting formula size results in $O(f^2(n))$.

The reduction trivially works and requires only polynomial time to generate the top-level formula. $\square$

L and NL

L and NL represent logarithmic space bound classes. As we must still read the entire input, we require a two-tape Turing Machine model containing 1) a read-only input tape that always remains on the portion with input, and 2) a work tape with usual properties and the only tape contributing to space constraints.

Class L is decidable in log space on a deterministic TM, and class NL similarly on a nondeterministic TM.

Our definition of a TM configuration also changes to incorporate the two tapes. The total number of tape configurations for machine M with input w of length n and space $f(n)$ work tape becomes

$$|Q|nf(n)|\Gamma|^{f(n)} = n2^{O(f(n))} = 2^{O(f(n))} \text{ if } f(n) \geq \log n$$

NL-Completeness and PATH

A language B is NL-complete if 1) $B \in NL$ and 2) for all $A \in NL$, A reduces to B in *log space* ($A \leq_L B$), or rather, B is NL-hard. The log-space reducibility is particularly important, as the reduction complexity cannot exceed the complexity of the problem we're reducing to in order for A to actually use the reduction as the problem decider.

Another condition for the log-space reducibility is a 3-tape TM model: machine M computing the function $f(w)$ with 1) a read-only input tape, 2) a write-only output tape with a forward-only tape head, and 3) a read/write work tape of $O(\log n)$ symbols.

The same reduction implication implies: $A \leq_L B$ and $B \in L \Rightarrow A \in L$, although involving a more complex proof, omitted here.

Let us define the *PATH* problem as a set of all directed graphs G and two vertices s and t, such that G contains a path from s to t.

Theorem. *PATH is NL complete.*

Proof. PATH $\in$ NL:

Nondeterministically proceed from one vertex to the next, storing only the current vertex on the work tape and the number of vertices traversed. If we reach t, accept. If we traverse more than $|V|$ vertices, reject.

For all $A \in NL, A \leq_L PATH$:

We'll construct $< G, s, t >$ such that graph G contains a path from s to t if and only if a nondeterministic TM N computing A can reach the accepting configuration c_{accept} from the initial

c_{start}. For each configuration of N, create a vertex, and generate an edge $(c_1, c_2) \in E$ if and only if $c_2 \in \delta(c_1)$, or rather, if configuration c_1 can yield c_2.

The reduction works as N accepts iff some accepting branch exists iff exists a path from c_{start} to c_{accept}. We also satisfied the log-space constraint, as the work tape stores only $O(1)$ configurations at a time, each consisting of state, pointers to the two tape locations, and the two respective tape symbols, all manageable in $O(\log n)$ space. □

Corrolary. $NL \subseteq P$

Proof. Suppose $A \in NL$. Then

$$\begin{aligned}
&A \leq_L PATH, \text{ as we've just demonstrated.}\\
\Rightarrow &A \leq_P PATH \text{ (log space requires only polynomial time)}\\
\Rightarrow &A \in P, \text{ since } PATH \in P. \text{ Ex: SSSP algorithm.}\\
\Rightarrow &NL \subseteq P
\end{aligned}$$

□

References

[1] Michael Sipser. *Introduction to the Theory of Computation.* Course Technology, Boston, MA, third edition, 2013.

[2] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.