Space-Time Trade-offs.*

Chethan Kamath

03.07.2017

1 Motivation

An important question in the study of computation is how to best use the registers in a CPU. In most cases, the amount of registers available is not sufficient to hold all the data pertaining to a computation. Even though an increase in the number of registers could lead to a decrease in the computation time, it is costly to include more registers in the CPU. Therefore it is interesting to study how space can be traded off for time in computations. But how does one even approach such a question? It turns out that the key is to come up with the right abstraction: for studying space-time trade-offs, one relies on *pebbling games on directed acyclic graphs* (DAGs). Bounds on the complexity of the pebbling games translate, roughly, to bounds on space and time used in computation (for certain models of computation).

Fast Fourier Transform. In this lecture we cover the space-time trade-off for the Fast Fourier Transform (FFT) algorithm. We will see that the savings offered by the FFT algorithm ($O(n \cdot \log\, n)$, compared to $O(n^2)$ for the naïve discrete Fourier transform (DFT) algorithm), can be lost if sufficient amount of storage is not available. To be precise, the FFT can be computed in $O(n \cdot \log\, n)$ time only if $\Omega(n/\log\, n)$ space is used concurrently; if only $o(n/\log\, n)$ space is used then the time required grows at a rate of $\omega(n/\log\, n)$.

Overview. First, in **Section 2**, we discuss the pebbling game and, in particular, focus on the relevant definitions; as a concrete example, we demonstrate various pebbling strategies for the perfect binary tree of depth d, denoted $BT^{(d)}$. Next, in **Section 3**, we refresh the definitions of the DFT, discuss the FFT and discuss, in detail, the properties of the 2^d-point FFT graph, denoted $F^{(d)}$. In **Sections** 4 and 5, we establish upper and lower bounds on the pebbling complexities of $F^{(d)}$. We conclude with some remarks in **Section 6**.

*The write-up is a summary of a lecture given as part of the course *Pearls of Computer Science* at IDC, Herzliya. Most of the material, in particular the figures, in the write-up is from [Sav98] and [SS78].

2 Pebbling

The pebbling game on DAGs is used to abstract computation using straight-line programs[1], which can be viewed as computation carried out on a DAG. Roughly speaking, vertices correspond to gates (depending on the model e.g., Boolean gates AND, OR or NOT, or algebraic gates + or ×) and edges correspond to wires that bring input to the gates. A pebble placed on a vertex indicates that the value associated with that vertex resides in the register. The rules of the game are the following:

Rule 1. (Initialization) A pebble can be placed on an input vertex (i.e., a source) at any time.

Rule 2. (Computation) A pebble can be placed on (or moved to) any non-input vertex iff its parents[2] all carry pebbles.

Rule 3. (Deletion) A pebble can be removed at any time.

The *goal* of the pebbling game is to pebble each output vertex (i.e., a sink) at least once. Intuitively, Rule 1 captures reading of an input in to a register; Rule 2 captures the computation of a value associated with a vertex; and Rule 3 captures erasure of value from the register. The goal captures computation of the output.

Pebbling sequences. A (pebbling) sequence for a graph $G = (\mathcal{V}, \mathcal{E})$ is an execution of rules of the pebbling game on it. It is denoted by $P := \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_T$, where $P_0 = \emptyset$ and $P_i \subseteq V$, $1 \leq i \leq T$ are pebbling *configurations*. The sequence is *valid* if configuration $\mathcal{P}_{i+1}$ was derived from $\mathcal{P}_i$, for each $1 \leq i < T$, by applying one of the pebbling rules. The complexity measures for sequences that we are interested are:

1. Time complexity: $\mathsf{T}_G(P) = T$
2. Space complexity: $\mathsf{S}_G(P) = \max_{0 \leq i \leq T} |\mathcal{P}_i|$
3. Space-time complexity: $\mathsf{ST}_G(P) = \mathsf{S}_G(P) \cdot \mathsf{T}_G(P)$

Given a complexity measure C for a sequence P, the corresponding complexity measure for the graph $\mathsf{C}(G)$ is the minimum over all the possible valid sequences; i.e.,

$$\mathsf{C}(G) = \min_P(\mathsf{C}_G(P)).$$

Since there is a rough correspondence between a straight-line program and its underlying graph, lower bounds on pebbling complexities translates to lower bounds on the resource used in the computation: e.g., a lower bounds on the space-complexity of the graph gives a rough estimate on the number of registers that are required for the corresponding computation. Therefore, for discussing trade-off properties of, say, FFT, the semantics of the nodes in the straight-line program is irrelevant — it suffices to know the *structure* of the graph that underlies it.

[1] A straight-line program is a restricted model of computation in which loops and conditional statements are not allowed — c.f., [Sav98, **Section 2.2**] for the formal definition. It covers, for example, logic circuits as well as algebraic circuits.

[2] For a graph $G = (\mathcal{V}, \mathcal{E})$, the parents of a vertex $v \in \mathcal{V}$ is the set $\{u : u \in \mathcal{V}, \exists (u, v) \in \mathcal{E}\}$.

2.1 Example: The Perfect Binary Tree

The perfect binary tree of depth[3] d is denoted by $BT^{(d)}$. Its vertices are arranged in $d+1$ layers, with layer i, $0 \leq i \leq d$, comprising of 2^{d-i} vertices (e.g., see **Figure 1**) — therefore, the number of vertices is $|\mathcal{V}| = \sum_{i=0}^{d} 2^i = 2^{d+1} - 1$. The lone vertex that resides in layer d is called the *root*, whereas the vertices at layer 0 are the *leaves*. Three pebbling strategies for $BT^{(d)}$ along with their complexity is discussed below; the choice of $BT^{(d)}$ for the demonstration is down, mainly, to two reasons: a) it serves as a clean, non-trivial example; and b) the arguments for the FFT graph $F^{(d)}$ are built on top of those for $BT^{(d)}$ (c.f., for example **Theorems** 3 and 4).

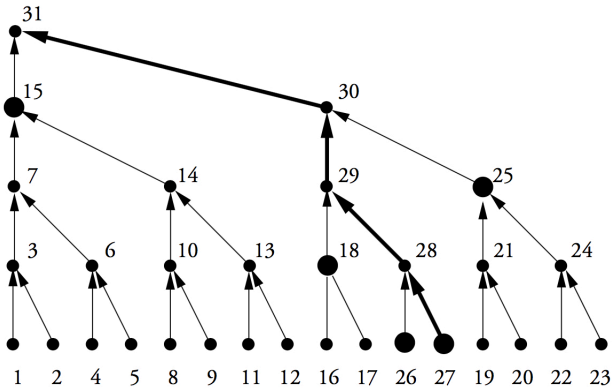


Figure 1: A perfect binary tree of depth 4 ($BT^{(4)}$). The vertices are arranged in five layers (0 through 4) with 16 (leaves), 8, 4, 2 and 1 (root) vertices respectively. The critical point for a particular sequence for BT^4 has also been highlighted: the leaf vertex involved is $u^* = 26$, and the "siblings" of the vertices on the critical path are $\{15, 18, 25, 27\}$. ([Sav98, **Figure 10.2**])

1. The naïve strategy (P_1) is to pebble layer-by-layer starting from the leaves and ending with the root, *without* removing any pebble (see **Figure 2**.(a)). The space-complexity of the strategy is $\mathsf{S}_{BT^{(d)}}(P_1) = |\mathcal{V}| = 2^{d+1} - 1$, which also turns out to be its time-complexity.

2. The "breadth-first" strategy (P_2) also involves placing pebbles layer-by-layer, but pebbles are removed as soon as they are of no use (see **Figure 2**.(b)). To be more precise, at any point in the sequence at most two layers carry pebbles, and pebbles on the lower layer is used to pebble their children in the higher layer and subsequently removed. To be precise, the steps are:

   (a.) Pebble layer 0 from left to right

   (b.) Repeat for each $1 \leq i \leq d$: for each vertex on layer i, move pebble from the left parent on to the vertex and remove the pebble on the right parent.

[3]The depth of a graph is the length of its longest path.

The number of pebbles used is the most at the end of Step (a.), and thus the space-complexity of the sequence is $\mathsf{S}_{BT^{(d)}}(P_2) = 2^d$. The number of moves is

$$2^d + 2 \cdot 2^{d-1} + 2 \cdot 2^{d-2} + \cdots + 2 \cdot 1 = 2 \cdot (2^d + 2^{d-1} + 2^{d-2} + \cdots + 1) - 2^d$$

and therefore $\mathsf{T}_{BT^{(d)}}(P_2) = 2 \cdot (2^{d+1} - 1) - 2^d$.

3. The "depth-first" strategy (P_3) results in the pebble-optimal sequence for $BT^{(d)}$ (see **Figure 2**.(c)). The strategy is recursive in depth: to pebble a particular vertex v

   (a.) pebble its left parent (recursively)

   (b.) pebble its right parent (recursively)

   (c.) move the pebble on the left parent on to v

   (d.) remove the pebble on the right parent.

   To pebble $BT^{(d)}$, the above steps are carried out on the root. The time-complexity of the sequence is captured by the recursion $T(d) = 2 \cdot T(d-1) + 2$, with $T(0) = 1$. Hence $\mathsf{T}_{BT^{(d)}}(P_3) = 2^{d+1}$. As for the space-complexity, we show next using induction (on depth) that the number of pebbles in play is at most $(d + 1)$. It is clear that P_3 pebbles $BT^{(0)}$ using a single pebble (the base case); let's assume that P_3 pebbles $BT^{(d-1)}$ using d pebbles (the induction hypothesis). Note that $BT^{(d)}$ comprises of two copies of $BT^{(d-1)}$. For the root of $BT^{(d)}$, Step (a) requires only d pebbles (by the induction hypothesis); while the right sub-tree is pebbled in Step (b) (using at most d pebbles), only the root of the left sub-tree carries a pebble and therefore the number of pebbles in play is at most $d + 1$.

Note that although P_1 is naïve in terms of the number of pebbles used, it is optimal in terms of the time-complexity (as at each vertex in $BT^{(d)}$ must carry a pebble at least once at some point during the pebbling). Next, we show that it takes at least $(d + 1)$ pebbles to pebble $BT^{(d)}$ — i.e., P_3 is pebble-optimal. The proof traces back to [PH70], and serves a warm-up to lower-bounds for the FFT graphs,.

Theorem 1 ([PH70])**.** $\mathcal{S}(BT^{(d)}) = d + 1$.

Proof. Consider the paths in $BT^{(d)}$. Initially, no paths carry pebbles; in the end *all* the paths end up carrying a pebble (as the root carries one). Now, consider the latest point in time in the sequence when a path was pebble-free, denoted the *critical point* (see **Figure 1**). It must be the case that it is one of the leaves, denoted u^*, that is pebbled in this move leading to the (critical) path from u^* to the root being "closed". Since the rest of the paths carry a pebble at the critical point, at least d pebbles must be present on the tree, one per sibling of the vertices on the critical path (i.e., one on each layer). Counting also the pebble placed on u^*, it follows that $\mathsf{S}(BT^{(d)}) \geq d + 1$; taking into account the upper bound due to depth-first pebbling completes the proof. □

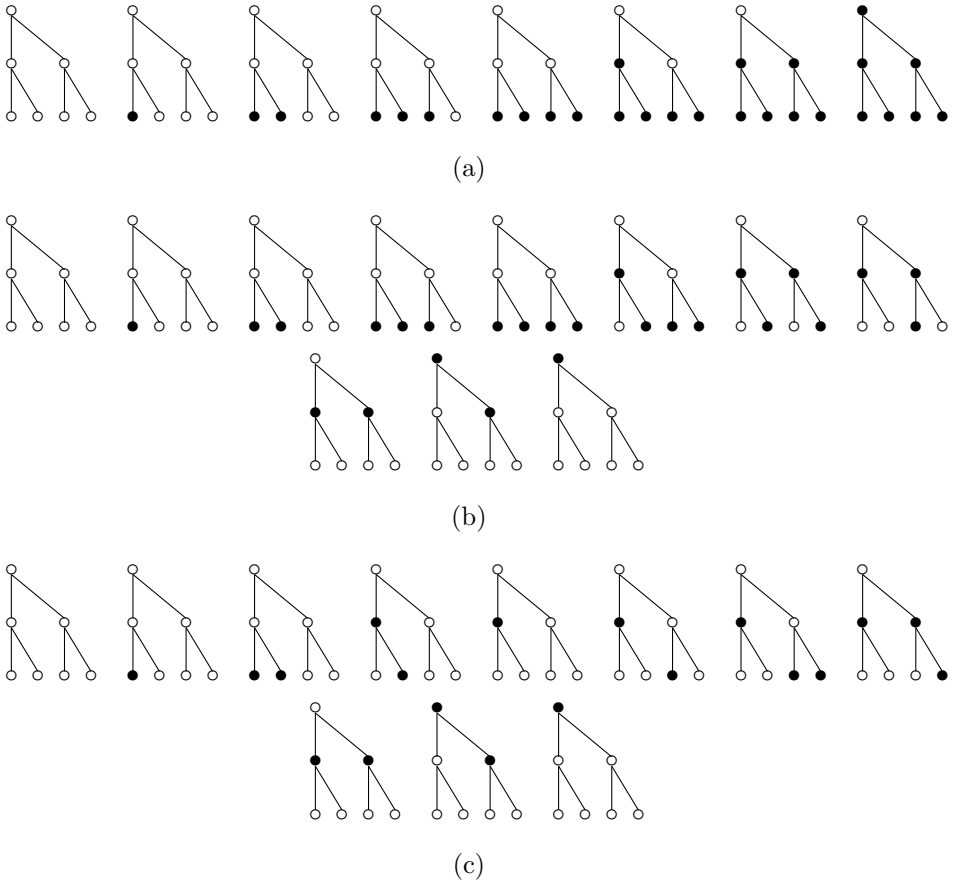


Figure 2: The pebbling strategies for $BT^{(2)}$: naïve pebbling (a), breadth-first pebbling (b) and depth-first pebbling (c).

3 Fast Fourier Transform: a Refresher

Let $n = 2^d$. The n-point discrete Fourier transform (DFT) $F_d : \mathcal{R}^n \to \mathcal{R}^n$ maps n-tuples $(a_0, \ldots, a_{n-1})$ over $\mathcal{R}$ to n-tuples $(f_0, \ldots, f_{n-1})$ over $\mathcal{R}$, where $f_i = p(\omega^i)$ and $p(x) = \sum_{i=0}^{n-1} a_i x^i$, ω being the n^{th} primitive root of unity. Naïve computation of DFT through polynomial multiplication takes $O(n^2)$ operations, n operations per polynomial evaluation. The fast Fourier transform (FFT) exploits the following decomposition of $p(x)$:

$$\begin{aligned} p(x) &= a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \\ &= (a_0 + a_2 x^2 + \cdots + a_{n-2} x^{n-2}) + x(a_1 + a_3 x^2 + \cdots + a_{n-1} x^{n-2}) \\ &= p_e(x^2) + x p_o(x^2). \end{aligned}$$

In particular, $p(\omega^i) = p_e(\omega^{2i}) + \omega^i p_o(\omega^{2i})$. Since ω^2 is the $(n/2)^{th}$ principal root of unity, we have decomposed the computation of n-point FFT to the computation of two $(n/2)$-point FFTs, a ring multiplication and a ring addition operation. Thus, the time-complexity of n-point FFT is captured by the recurrence $T(n) = 2T(n/2) + O(n)$, and hence $T(n) = O(n \cdot \log n)$.

The FFT graph. The straight-line programs that compute 2-point FFT (the butterfly graph) and 16-point FFT are shown in **Figure 3**. In general, the straight-line program that computes n-point FFT, has $n \cdot (d+1)$ nodes which are arranged in $d+1$ layers (i.e., depth is d) with n nodes per layer. Therefore, we focus on the underlying graph, denoted $F^{(d)}$, with nodes corresponding to vertices and wires to edges. $F^{(d)}$ has sever interesting structural properties, of which a couple are highlighted below.

(1.) Owing to the recursive definition of the FFT algorithm, the graph $F^{(d)}$ is highly recursive. In particular, it can be viewed as composed of copies of smaller FFT graphs. For example, in **Figure 3**.(b), the graph $F^{(4)}$ is shown decomposed into eight copies of $F^{(1)}$ on top of two copies of F^8 (boxed). In general, for $0 \leq j \leq d$, $F^{(d)}$ can be viewed as decomposed into

   (a) 2^{d-j} copies of $F^{(j)}$ denoted $\mathcal{A}^{(j)} := A_0^{(j)}, \ldots, A_{2^{d-j}-1}^{(j)}$ *above*; and

   (b) 2^j copies of $F^{(d-j)}$ denoted $\mathcal{B}^{(d-j)} := B_0^{(d-j)}, \ldots, B_{2^j-1}^{(d-j)}$ *below*

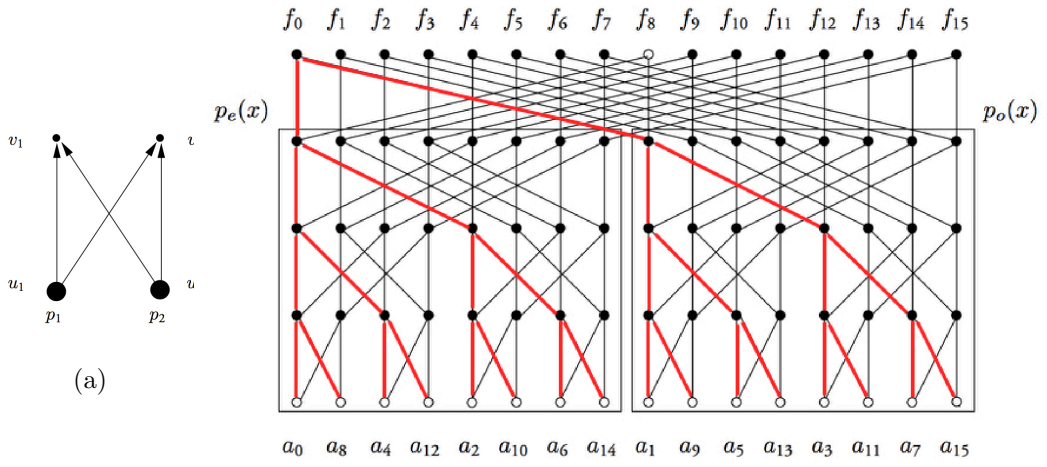
   (see **Figure 4** for a schematic, and **Figure 5** for an example).

(2.) The sub-graph rooted at an output vertex (i.e., the sub-graph consisting of all vertices from which that output vertex is reachable) is a perfect binary tree $BT^{(d)}$ as highlighted in red in **Figure 3**.

We will refer frequently to the representation of $F^{(d)}$ in Item (1.) and the observation in Item (2.) in order to explain pebbling strategies for $F^{(d)}$.

4 Upper Bounds

In this section, we show that $F^{(d)}$ has ST-complexity of $O(n^2)$. Intuitively, this involves establishing that the best pebbling strategy given S pebbles is to use,



(a)

(b)

Figure 3: (a). The butterfly graph ($F^{(1)}$). (b) The 16-point FFT graph $F^{(4)}$. It consists of 5 layers, with the input vertices at layer 0 ($a_0, a_8, \ldots, a_{15}$, obtained through the bit-reversal permutation) and the output vertices at layer 4 ($f_0, \ldots, f_{15}$). The edges between layers $i-1$ and i, for $1 \leq i \leq 4$, are each associated with the value $\omega^{2^{4-i}}$; a vertex in layer i with parent vertices $v_{i-1,j}$ and $v_{i-1,k}$ (in that order), for $0 \leq i, j <$, represents the computation $v_{i-1,j} + \omega^{2^{4-i}} v_{i-1,k}$ (a basic $F^{(1)}$ operation). The graph $F^{(4)}$ is shown decomposed into eight copies of $F^{(1)}$ on top of two copies of F^8 (boxed). The perfect binary tree $BT^{(d)}$ rooted at f_0 is highlighted in red. ([Sav98, **Figures 11.8, 6.7**])

appropriately, a "hybrid" of the breadth-first pebbling P_2 (which is move-efficient) and the depth-first pebbling P_3 (which is pebble-optimal).

Theorem 2 ([SS78, Theorem 1])**.** *The FFT graph $F^{(d)}$ can be pebbled in T moves with S pebbles where*

$$T \leq \begin{cases} 2n^2/2^j + (j+1)n & \textbf{\textit{if}}\ S \geq d + 2^j - j, 1 \leq j \leq d-1 \\ n \cdot (d+1) & \textbf{\textit{otherwise if}}\ S = 2^d + 1. \end{cases} \tag{1}$$

Proof sketch. First, we observe that the breadth-first pebbling P_2 can be easily adapted into a breadth-first pebbling for $F^{(d)}$, which uses at most $(n+1)$ pebbles and $n \cdot (d+1)$ moves[4]:

$$\mathsf{S}_{F^{(j)}}(P_2) = n+1 \ \textit{and}\ \mathsf{T}_{F^{(j)}}(P_2) = n \cdot (d+1).$$

This establishes the second inequality in (1); to establish the first inequality, sup-

[4]There seems to be a mistake in [SS78] with regard to this claim: it is impossible to pebble in $n(d+1)$ moves, as claimed, without *not* dropping pebbles (and there are only 2^d+1 pebbles available). It takes $3nd/2+n$ moves to pebble $F^{(d)}$ using 2^d+1 pebbles in the manner described above. Nevertheless, that does not make any (asymptotic) difference in the analysis.

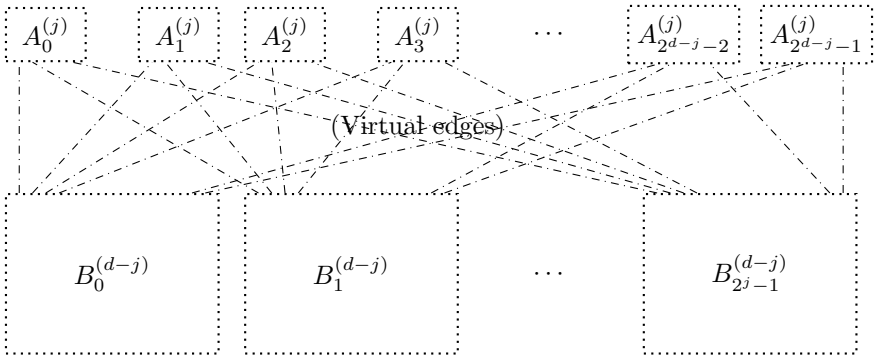


Figure 4: Schematic diagram showing the decomposition of $F^{(d)}$ into $\mathcal{A}^{(j)} := A_0^{(j)}, \ldots, A_{2^{d-j}-1}^{(j)}$ and $\mathcal{B}^{(d-j)} = B_0^{(d-j)}, \ldots, B_{2^j-1}^{(d-j)}$. The virtual edges between $\mathcal{A}^{(j)}$ and $\mathcal{B}^{(d-j)}$ are determined as follows: the m^{th} input to $A_\ell^{(j)}$ is the ℓ^{th} output of $B_m^{(d-j)}$. The output edges, on the other hand, are determined as follows: the ℓ^{th} output of $F^{(d)}$ is the r^{th} output of $A_s^{(j)}$ for unique $r, s \geq 0 : i = r \cdot 2^{d-j} + s, s < 2^{d-j}$.

pose that only

$$S = \underbrace{2^j}_{\text{pebbles for } P_2} + \underbrace{(d-j)}_{\text{pebbles for } P_3}$$

pebbles, for $1 \leq j \leq d-1$, are available. We view the graph $F^{(d)}$ as decomposed into $\mathcal{A}^{(j)}$ and $\mathcal{B}^{(d-j)}$ and, on a high level, use a strategy that is a hybrid of the breadth-first and depth-first pebbling strategies. The hybrid pebbling uses the $(d-j)$ pebbles to pebble, depth-first, the perfect binary tree $BT^{(d-j)}$ rooted at each input vertex of $A_\ell^{(j)}$, and then uses the 2^j pebbles to pebble $A_\ell^{(j)}$ breadth-first. The steps are detailed below.

Repeat for each $A_\ell^{(j)} \in \mathcal{A}^{(j)}$:

Step 1. Repeat for each input vertex $v_m \in A_\ell^{(j)}$:

Step 1.a Pebble, *depth-first*, the binary tree $BT^{(d-j)}$ contained in $B_m^{(d-j)}$ and rooted at v_m (using $(d-j)$ pebbles)

Step 2. Pebble, *breadth-first*, the output vertices of $A_\ell^{(j)}$ (using 2^j pebbles)

The total number of moves is

$$\begin{aligned}
&|\mathcal{A}^{(j)}| \cdot \Big((\#\text{ input vertices in } A_\ell^{(j)} \cdot \#\text{moves in Step 1.a}) + \#\text{moves in Step 2}) \Big) \\
&\quad = |\mathcal{A}^{(j)}| \cdot \Big((\#\text{ input vertices in } A_\ell^{(j)} \cdot \mathsf{T}_{BT^{(d-j)}}(P_3) + \mathsf{T}_{F^{(j)}}(P_2)) \Big) \\
&\quad = 2^{d-j} \cdot \big((2^j \cdot (2^{d-j+1} - 1)) + (j+1) \cdot 2^j \big) \\
&\quad = 2n^2/2^j + (j+1) \cdot n.
\end{aligned}$$

That completes the proof. □

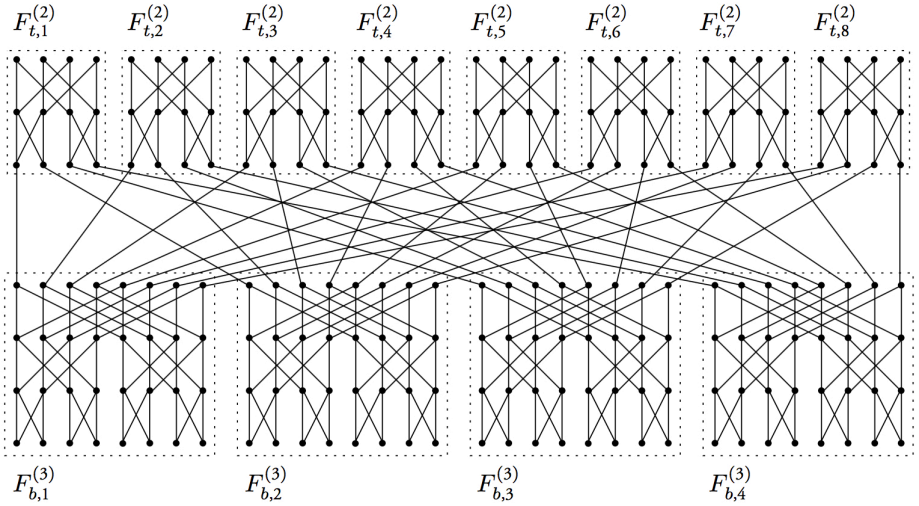


Figure 5: The graph $F^{(5)}$ viewed as composed of eight copies of $F^{(2)}$ FFT graphs ($\mathcal{A} := A^{(2)}_0, \ldots, A^{(2)}_7$) and four copies of the $F^{(3)}$ graph ($\mathcal{B} := B^{(3)}_0, \ldots, B^{(3)}_3$). Note that the vertices in the bottom layer of $\mathcal{A}$ and in the top layer of $\mathcal{B}$ are, in fact, the *same*, and hence the vertices from $\mathcal{B}$ to $\mathcal{A}$ are *virtual*. The reason for their introduction is aesthetic, and to simplify exposition. ([Sav98, **Figure 10.8**])

5 Lower Bounds

In this section, we complement the upper bound in the previous section with an (almost) tight lower bound; that is, we show that $\mathsf{ST}(F^{(d)}) = \Omega(n^2)$. Intuitively, we show that the hybrid pebbling discussed in the previous section *is* the best one can hope for. We warm up with the bound for the extreme case of $S = d + 1$.

Theorem 3 ([SS78, Theorem 2])**.** *At least $T \geq n(n-d)/2 + n - 1$ moves are necessary to pebble $F^{(d)}$ with $S = d + 1$ pebbles.*

Proof sketch. The proof builds on that of **Theorem 1**. We view $F^{(d)}$ as decomposed into the n output vertices and two copies of $F^{(d-1)}$ denoted $B^{(d-1)}_0, B^{(d-1)}_1$ (e.g., see **Figure 3**). Consider the *first* critical point of a pebbling sequence (where the notion of critical point is same as in **Theorem 1**). By an argument similar to that in the proof of **Theorem 1**, exactly one of the $(d+1)$ pebbles lies on level $d-1$ in either $B^{(d-1)}_0$ or $B^{(d-1)}_1$ — without loss of generality, let it lie in $B^{(d-1)}_0$. This particular pebble is useful *only* in pebbling the two output vertices that it is the parent of and is, after that, useless. Therefore, before the next critical point it must be that, for $B^{(d-1)}_0$, either:

a. a single pebble is brought to level $(d-1)$ using $2^d - 1$ moves; or

b. d pebbles are brought to levels $0, \ldots, d-2$ with two pebbles at level 0 — this requires $1 + \sum_{j=0}^{d-2}(2^{j+1} - 1) = 2^d - d$ moves.

As this needs to be repeated at least 2^{d-1} times, the total number of moves is at

least

$$(2^d - 1) + 2^{d-1} \cdot (2^d - d) = n(n-d)/2 + n - 1,$$

where $(2^d - 1)$ is the number of moves made till the first critical point. □

Next, we state and prove the full theorem. Although it builds on **Theorem 3**, the proof is quite fine-grained and as a result very technical.

Theorem 4 ([SS78, Theorem 3]). *The number of moves necessary to pebble $F^{(d)}$ using $S < n/2 + 1$ pebbles satisfies*

$$T \geq n \cdot (j+2) + n(n-S)/2^{j+1} \tag{2}$$

where $j \leq d-1$ is determined by

$$2^{j-1} + d - (j-1) < S \leq 2^j + (d-j). \tag{3}$$

Proof sketch. As in the proof of **Theorem 3**, we decompose the graph into $\mathcal{A}^{(j)}$ and $\mathcal{B}^{(d-j)}$, and show that pebbling $\mathcal{A}^{(j)}$ breadth-first (once) and pebbling $\mathcal{B}^{(d-j)}$ depth-first (multiple times) is the optimal strategy. Hence, the upper bound is tight up to constant factors. We establish the lower bound for $\mathcal{A}^{(j)}$ and $\mathcal{B}^{(d-j)}$ separately (and this explains the two terms in (2)).

A. The lower-bound for $\mathcal{A}^{(j)}$ is straightforward: we use the fact that each vertex must be visited at least once. Therefore, the number of moves involved is at least

$$\underbrace{2^{d-j}}_{|\mathcal{A}^{(j)}|} \cdot \underbrace{(2^j \cdot (j+2))}_{|A_\ell^{(j)}|}$$

and the number of pebbles in play are at most 2^j. This establishes the first term of (2).

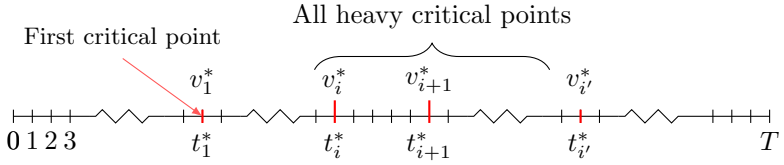


Figure 6: The timeline of a pebbling for $F^{(d)}$. Critical points are marked in red; heavy critical points *with respect to* t_i^* are marked by the longer red bar. t_1^* is the first critical point; $t_{i'}^*$ is the first non-heavy critical point with respect to t_i^*.

B. Bounding the number of moves in $\mathcal{B}^{(d-j)}$ is non-trivial as the sub-trees are pebbled (depth-first) multiple times. The argument is a fine-grained version of that in **Theorem 3**, and is summarised below.

(i.) Let's look at the time-line of a pebbling sequence for $F^{(d)}$ (see **Figure 6**). Let's examine the snapshot at a critical time t_i^*: let the path closed be for an output vertex v_i^*, and let the path pass through $A_{\ell^*}^{(j)}$

and $B_{m^*}^{(d-j)}$, for some $0 \leq \ell^* < 2^{d-j}$ and $0 \leq m^* < 2^j$. Assume that at this point of time (t_i^*), there are a pebbles on $\mathcal{A}^{(j)}$ and b pebbles on $\mathcal{B}^{(d-j)} \setminus \{B_{m^*}^{(d-j)}\}$ (i.e., on the bottom graphs except the one involving the critical path). Let R denote $a+b$. It follows that

$$S = \underbrace{a+b}_{R} + \underbrace{(d-j+1)}_{\#\text{pebbles on } B_{m^*}^{(d-j)}} \stackrel{(3)}{\Longrightarrow} 2^{j-1} < R < 2^j. \tag{4}$$

(ii.) The first step is to show that the vertex v_i^* is "heavy" at time t_i^*, where an v_i^* is deemed heavy (at time t_i^*) if at least $\lceil a/2 \rceil$ of the 2^j of its paths from the input vertices in $A_{\ell^*}^{(j)}$ are blocked by pebbles in the pebbling configuration $\mathcal{P}_{t_i^*}$. This makes crucial use of the inequality in (4). (c.f., [SS78, Lemmas 3 and 4])

(iii.) Next, we show that there are at most 2^{j+1} heavy output nodes after v_i^* *at the critical point* t_i^*; let's denote the first non-heavy output node at t_i^* by $v_{i'}^*$, $i' > i$. This is shown by establishing that around 2^{j+1} trees in $\mathcal{B}^{(d-j)}$ are *empty* — therefore, a lot of pebbles (around $2^d - S$) are moved on $\mathcal{B}^{(d-j)}$ in between the critical points for v_i^* and $v_{i'}^*$. (c.f., [SS78, Lemma 5])

(iv.) As there must be at least $2^d/2^{j+1}$ snapshots of the above type for the whole sequence, (ii.) and (iii.), together, imply that at least $(2^d - S) \cdot 2^{d-(j+1)}$ moves are made overall.

This establishes the second term of (2).

□

6 Epilogue

A. The lower-bound technique we saw is quite general (see [Gri76], for example) and can be applied to other problems that have straight-line programs. A couple of examples are given below.s

1. For multiplication of two $n \times n$ matrices $(S+1) \cdot T \geq n^3/4$; since the naïve multiplication algorithm takes $T = O(n^3)$ and $O(1)$ space, it follows that $(S+1) \cdot T = \Theta(n^3)$.
2. For multiplication of two n bit integers, $(S+1) \cdot T \geq n^2/64$; however, no matching upper bound is known — the best bound is $(S+1) \cdot T = O((n \log n)^2)$.

B. The pebbling paradigm is quite powerful: it allows establishing similar bounds for other models of computation. For example, it is possible to capture parallel computation if the pebbling moves are allowed to be made in parallel (i.e., Rules 2 qne 3 can be carried out in parallel). It is possible to model reversible computation [Ben89] by adding a constraint to Rule 3: *A pebble can be removed only if all its parents carry pebbles.*

References

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989. (Cited on page 11.)

[Gri76] D. Grigoriev. An application of separability and independence notions for proving lower bounds of circuit complexity. *Notes of the Scientific Seminar Leningrad branch of the Steklov Institute*, 60:38–48, 1976. (Cited on page 11.)

[PH70] Michael S. Paterson and Carl E. Hewitt. Record of the project mac conference on concurrent systems and parallel computation. chapter Comparative Schematology, pages 119–127. ACM, New York, NY, USA, 1970. (Cited on page 4.)

[Sav98] John E. Savage. *Models of computation - exploring the power of computing.* Addison-Wesley, 1998. (Cited on pages 1, 2, 3, 7 and 9.)

[SS78] J. Savage and S. Swamy. Space-time trade-offs on the fft algorithm. *IEEE Transactions on Information Theory*, 24(5):563–568, September 1978. (Cited on pages 1, 7, 9, 10 and 11.)