Time-Space and Size-Space Tradeoffs for Oblivious Computations

by

David Allen Carlson

B.A., University of Connecticut, 1977

Thesis

Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of
Computer Science at Brown University

June, 1981

Abstract of

Time-Space and Size-Space Tradeoffs for Oblivious Computations

by

David Allen Carlson

Ph.D., Brown University, June 1981

A major area of interest in computational complexity is the development of lower bounds on the simultaneous resource requirements of an algorithm implementing a particular problem. Important resources that can be used to determine the cost of computing are an algorithm's execution time, storage space needs, and size. Significant relationships can occur between these parameters, such as tradeoffs, for which it is impossible to achieve two resource values simultaneously.

This dissertation investigates the types of tradeoffs which relate the resources of time, space, and size for computations executed in a data-independent manner. We associate a directed acyclic graph with an algorithm, and simulate the allocation of storage space by placing "pebbles" on vertices of the graph. Using this technique, we are able to demonstrate time-space tradeoffs for a number of data-independent computations. For n-degree polynomial multiplication implemented via Fast Fourier transforms, we show that the product of time and space must grow as $\Omega(n^2 \log n)$. Thus, in order to achieve time $O(n \cdot \log n)$, a sacrifice in space of $\Omega(n)$ must be made. We develop a graph model for the evaluation of a non-linear recursive function, and exhibit pebbling strategies optimal with respect to either time or space. We also

discover that, in certain cases, near optimal time and space can be achieved simultaneously.

Another topic which we examine is the consequences of restricting the resource of space too severely. One such consequence is time which grows as superpolynomial in the size of the computation's corresponding graph. We demonstrate that this type of behavior is associated with a particular data-independent sorting algorithm. We also examine the range of space in which superpolynomial time can occur, and show that the lower limit on space can grow as any slowly increasing function of the graph's size. In addition to this, we show that when space is limited, it is impossible to construct graphs of optimal size for the problems of oblivious merging and pattern matching.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my advisor, Dr. John E. Savage, for his support and guidance during my stay at Brown University. Professor Savage directly influenced many of the ideas in this dissertation, and also helped to shape my entire perspective on research. The time and effort which he expended on my behalf is sincerely appreciated.

I am also grateful to the readers, Drs. Steven P. Reiss and Robert Sedgewick, for their interest and suggestions, and to the computer science faculty, staff, and graduate students at Brown for providing an intellectually stimulating environment.

Finally, I recognize my wife Louise, whose understanding and encouragement contributed greatly to the completion of this dissertation.

Financial support for this research was received from National Science Foundation grant MCS-20023.

TABLE OF CONTENTS

NOTATIONS

O $f(n) = O(g(n))$ iff there are constants $M > 0$ and $N > 0$ such that for all $n \geq N$, $f(n) \leq M \cdot g(n)$.

Ω $f(n) = \Omega(g(n))$ iff there are constants $M > 0$ and $N > 0$ such that for all $n \geq N$, $f(n) \geq M \cdot g(n)$.

Θ $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

o $f(n) = o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.

ω $f(n) = \omega(g(n))$ if $\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$.

log logarithm base 2.

ln logarithm base e.

$\lfloor\ \rfloor$ $\lfloor x \rfloor$ is the greatest integer less than or equal to x.

$\lceil\ \rceil$ $\lceil x \rceil$ is the least integer greater than or equal to x.

CHAPTER 1

INTRODUCTION

One of the fundamental issues in theoretical computer science is the determination of bounds on the cost of performing a computational problem. The cost of computing is usually measured in terms of one or more computer resources required by an algorithm implementing the problem. The *execution time* of the algorithm is perhaps the most frequently used measure, but other quantities such as *storage space* and *algorithm size* also reflect, in part, the cost associated with a computation. Investigating the resource requirements of a problem can often lead to more efficient algorithms, and can provide a greater understanding of what makes a problem difficult to compute.

Computational complexity is the area of computer science which is concerned with establishing lower bounds on the resources required by a particular problem. Such lower bounds are usually expressed as a function of the number of inputs to the problem. Complexity results have traditionally been stated in the form of bounds on the individual measures of time, space, and size. See Aho, Hopcroft and Ullman [1] or Borodin and Munro [6] for surveys of such results. However, recent research [15, 18, 23, 34, 36, 37, 39, 38, 43, 44] indicates that significant relationships can occur between these quantities, and the study of these relationships can contribute to a more complete understanding of a problem's computational costs. For instance, there may be a *tradeoff* between two resources, so that in order to reduce the use of one, a corresponding increase must be made in the other. Knowledge of such tradeoffs can be quite useful when selecting an algorithm for a problem, since it can indicate the values of different resources which are simultaneously achievable, and the consequences of restricting one resource too severely.

This dissertation investigates the types of tradeoffs which can occur between the measures of execution time, storage space, and algorithm size for various computational problems. The algorithms which we consider are restricted in the sense that they are *straight-line*, i.e. they perform a sequence of operations in an oblivious manner, independent of their input values. Our analysis technique is to associate a directed acyclic graph with a straight-line algorithm, and then play the well-known *"pebble game"* on the graph to determine the time and space requirements of the algorithm (algorithm size is measured by the number of vertices in the graph). Using this technique, we are able to derive tradeoffs between time and space for specific algorithms such as polynomial multiplication implemented via Fast Fourier transforms, stack-based implementations of non-linear recursion, and oblivious sorting based on bitonic merging. In addition to this, we examine the negative effects of restricting space too severely. For certain graphs, we consider the range of space for which time grows as *superpolynomial* (asymptotically faster than any polynomial) in the size of the graph. We also show that it is impossible to construct optimal sized graphs for the problems of oblivious merging and pattern matching when space is limited.

1.1. Basic Concepts.

Straight-line algorithms are useful in the computation of problems for which it appears that any solution proceeds in a sequence of data-independent steps. An example of such a problem is polynomial multiplication. It involves the computation of a set of *functions*, each of which corresponds to a coefficient of the product polynomial and depends on a subset of the coefficients of the input polynomials.

In order to provide lower bounds on the resource requirements of a straight-line algorithm, we must provide a precise formulation of the computational process. A straight-line algorithm consists of a sequence of assignment statements, each of which is able to access a (possibly infinite) set $\{R_i\}$ of temporary registers. Each assignment statement stores a value into an available register. This value is obtained either from an input set, from a set of available constants, or as the result of performing an operation whose arguments are values previously stored into registers. The input set is assumed to be stored in a read-only memory which does not contribute to the space requirement of the algorithm. Operations are drawn from a fixed set (usually addition, multiplication, subtraction, and division), and accept only a bounded number of arguments. Thus, a straight-line algorithm is a sequence of instructions, each of the form:

(1): $R_i \leftarrow x$ (x is an input or constant)

or

(2): $R_i \leftarrow \rho(R_j, \ldots, R_z)$ (ρ is a legal operator).

In (2), it can be seen that the value stored into R_i depends on the values which are contained in $R_j, \ldots, R_z$ at the beginning of the instruction. Such an algorithm is said to compute a set of functions F if for each function f in F, there is a step in the algorithm which, for any input x, assigns the value f(x) to a register. The important resources associated with a straight-line algorithm are its time and space requirements, which are measured by the number of assignment statements in the algorithm and the number of registers used by the algorithm, respectively.

Any straight-line algorithm which computes a set of functions can be represented as a directed acyclic graph *(dag)*. A dag G = (V,E) is defined to have vertex set V and edge set E. If (u,v) is an edge in E directed from u to v, then u

is called a *predecessor* of v and v is a *successor* of u. The *in-degree* and *out-degree* of a vertex in V are the number of its predecessors and successors, respectively. Vertices of the graph with in-degree zero are called *inputs*, while *outputs* of the graph are vertices with out-degree zero. The *depth* of a vertex in G is the length of the longest path directed into the vertex from any input, and the *depth* of G is the maximum of the depths of its vertices.

In the graphical representation of a straight-line algorithm, input vertices of the graph correspond to instructions in the algorithm where an input variable is assigned to a register, and output vertices correspond to the values of the functions which are being computed. Non-input vertices of the graph represent operations which assign values to registers, and edges of the graph denote argument assignments for these operations.

1.2. The Pebble Game.

In this dissertation, we use the "pebble game" on directed acyclic graphs to provide an abstraction that simulates the allocation of registers when evaluating a set of functions in a straight-line algorithm. In the game, pebbles are placed on vertices of the graph according to the following rules:

(1) Input vertices can be pebbled at any time.

(2) Pebbles can be removed from the graph at any time.

(3) A pebble can be placed on a non-input vertex x if and only if all vertices with edges directed into x have a pebble. This placement can be accomplished by moving a pebble from a predecessor of x directly onto x.

The goal of the game is to place a pebble on each output vertex, starting from a graph empty of pebbles.

When a straight-line algorithm is represented by a graph, placing a pebble on a vertex according to rules 1 and 3 corresponds to performing an operation and storing the result in an available register. Removing a pebble from the graph (rule 2) corresponds to freeing a register for future use. The resources of time and space associated with a straight-line algorithm can be modeled accurately by T, the number of pebble placements or *moves* made on the graph in order to reach all output vertices; and S, the maximum number of pebbles used at any time during the pebbling process. Note that T represents the number of assignment statements executed, and S represents the number of temporary registers required by the algorithm. Other quantities which may have importance when the pebble game is used to simulate a straight-line algorithm include the *size* or number of vertices in the graph to be pebbled, and the *space requirement* of the graph, which is the minimum number of pebbles necessary to reach all outputs (if less space is used, then there is an output which cannot be reached).

The pebble game has been used as an analysis technique in many areas of theoretical computer science, ranging from the study of abstract machines and languages to the examination of the resource requirements of certain algebraic problems. Paterson and Hewitt [28] introduced the game, and by considering the space requirements of a complete binary tree, were able to show that recursive program schemata are strictly more powerful than non-recursive ones. Cook [9] analyzed the space requirements of graphs referred to as pyramids, and applied his results to derive the existence of a problem solvable in time polynomial in n, the length of the input set, but whose space requirements grow faster than any polynomial in log n.

The space requirements of general graphs were studied by Hopcroft, Paul, and Valiant [17], who demonstrated that any graph on N vertices can be pebbled with space $O(\frac{N}{\log N})$. The coefficient on this bound depends on the maximum in-degree of the vertices in the graph. From this fact, they were able to prove that space is strictly more powerful than time on deterministic multi-tape Turing Machines. Paul, Tarjan, and Celoni [29] found that the above bound on space is tight in that there exist graphs on N vertices for which any pebbling strategy requires space $\Omega(\frac{N}{\log N})$. They also gave a procedure for pebbling an arbitrary graph using space $O(\frac{N}{\log N})$.

It is obvious that any graph on N vertices can be pebbled in N moves using N pebbles. However, since space equal to the size of the graph being pebbled is not always available, it is of interest to know the number of moves required when space is restricted. In general, certain vertices of the graph will have to be repebbled, which may lead to meaningful time-space tradeoffs. The relationship between time and space has been the topic of a large amount of research concerning the pebble game.

The pebble game has been useful in studying the simultaneous time and space requirements of straight-line algorithms for certain algebraic problems. Grigoryev [15] has derived lower bounds on the product of time and space for the problems of boolean matrix multiplication and polynomial multiplication. His bounds are of the form $T \cdot S = \Omega(n^2)$ for polynomials of degree n, and $T \cdot S = \Omega(n^3)$ for n by n matrices. Savage and Swamy [36, 37, 38] obtained the result $T \cdot S = \Omega(n^2)$ for the n-input Fast Fourier Transform (FFT), n-bit integer multiplication, and n-input oblivious sorting. In the case of performing a Fast Fourier Transform, they also showed that for space S in the range $[\log n, \frac{n}{\log n}]$,

there exists a pebbling strategy using time T for which $T \cdot S = O(n^2)$. Thus, the bound $T \cdot S = \Omega(n^2)$ can be achieved for space in this range. The sorting result indicated above was obtained independently by Tompa [44], who also extended the FFT result to apply to more general problems such as polynomial and matrix multiplication, the Discrete Fourier Transform, and oblivious merging. His technique was to establish time-space tradeoffs for graphs associated with any straight-line algorithm solving a given problem. For example, associated with any computation of an n-input Discrete Fourier Transform is a graph called an n-hyperconcentrator, for which Tompa derived the lower bound $T \cdot S = \Omega(n^2)$. Ja'Ja' [18] has also studied algebraic problems, and has derived a lower bound of $\Omega(p^4)$ on the time-space product for inversion of p by p matrices along with a lower bound for computing the product of several non-square matrices.

Another problem for which the pebble game has been used is linear recursion. Here, the number of pebbles models the stack space available to a stack-based implementation of a linear recursive program. Paterson and Hewitt [28] studied the problem of restricted space in this context, and showed that for fixed space S, $T = \Omega(N^{1+\frac{1}{S-1}})$. Chandra [8] exhibited good but non-optimal algorithms for evaluating linear recursive programs using small, medium, and large amounts of stack space. Savage and Swamy [39] were able to derive the full class of optimal stack-based algorithms, and showed that significant reductions in space can be achieved at the expense of an increase by only a constant factor in time.

When available space is restricted in the pebble game, it has been shown that time superpolynomial in the size of the graph may be necessary. Paul and Tarjan [30] have exhibited graphs on N vertices for which the reduction in space by a constant factor causes the pebbling time to expand from linear to

superpolynomial in the size of the graph. Here, the largest range of space for which superpolynomial pebbling time occurs is from $\omega((\log N)^2)$ to $o(\frac{N}{\log N})$. Van Emde Boas and van Leeuwen [5] have constructed a family of graphs for which exponential time is necessary when the minimum space (which is $\Theta(N^{1/3})$) is used, and for which time becomes polynomial when one extra pebble is used. Lingas [24] has a similar result except that pebbling time becomes provably polynomial when two extra pebbles are available. Reischuk [34] and Lengauer and Tarjan [23] have examined the time-space tradeoff for arbitrary N vertex graphs for which space is primarily $\Omega(\frac{N}{\log N})$, and have obtained fairly tight bounds on the time required in the worst case. The best bounds were derived by Lengauer and Tarjan, who show for space in the range $\Theta[\frac{N}{\log N}, N]$, that any graph can be pebbled in time $T = 2^{2^{o(\frac{N}{S})}}$ using S pebbles, while there exist graphs for which any pebbling strategy requires $T = 2^{2^{\Omega(\frac{N}{S})}}$. Loui [26] has subsequently obtained the same upper bound using a more elegant argument.

Another problem related to the pebble game is the complexity of pebbling: given as inputs an arbitrary graph G and a natural number S, determine if G can be pebbled using space S (or simply determine the minimum space requirement of G). This problem models the register allocation strategies which a compiler might employ on expressions that cannot be simplified. Sethi [40] initially has shown that this problem is NP-hard, and its exact complexity was determined by Gilbert, Lengauer, and Tarjan [14], and Loui [25], who established that the problem is polynomial space complete.

1.3. An Example of a Time-Space Tradeoff.

In order to illustrate the types of relationships which can occur between resources of time and space in the pebble game, we introduce two examples.

First, consider a complete binary tree with depth d, 2^d leaves, and $2^{d+1}-1$ vertices. Here, leaves of the tree serve as inputs, and the root is the unique output of the tree. A complete binary tree with depth 3 is shown in Figure 1. The space requirement of a complete binary tree was analyzed by Paterson and Hewitt [28] as shown below.

Proposition 1.1 (Paterson and Hewitt [28]): Let B be a complete binary tree with depth d, 2^d leaves, and size $N = 2^{d+1}-1$. Then the space necessary to reach the root of B is $d+1 = \log (N+1)$ pebbles.

> ***Proof:*** Consider the point in time when a pebble placement is made which closes the last open path from an input to the output of the tree. At least d pebbles must be on the tree at this point in time in order to "block" all other paths between inputs and the output, and one extra pebble is needed to close the last open path (this is done by pebbling the input of the path). Thus, d+1 pebbles are required. □

It is easily seen that a complete binary tree of depth d can be pebbled in time equal to its size when d+1 pebbles are available. Thus, there is no time-space tradeoff in this case; there is a pebbling strategy which achieves both optimal space and optimal time.

A graph which resembles the complete binary tree is the pyramidal graph, introduced by Cook [9]. The pyramidal graph of depth d has d inputs and $d(d+1)/2$ vertices, and is shown in Figure 2. Through an argument similar to that in Proposition 1.1, the space requirement of a pyramidal graph on d inputs

is d pebbles. Again, there is a pebbling strategy which pebbles the output vertex in optimal time when d pebbles are available.

To provide an example of a time-space tradeoff, we consider the graph G with 4 inputs and 4 outputs shown in Figure 3. In Chapter 2, we will see that G is the graph associated with the straight-line computation of a Fast Fourier Transform on 4 inputs.

Proposition 1.2 (Savage and Swamy [36]): The minimum space necessary to pebble G is 3 pebbles.

Proof: Associated with each output of G is a complete binary tree of depth 2 whose leaves are inputs of the graph. By Proposition 1.1, pebbling an output of G requires 3 pebbles. □

Proposition 1.3 (Savage and Swamy [36]): For the graph G, when at most 3 pebbles are available, at least 16 moves are necessary to reach all outputs. When 5 pebbles are available, the outputs can be pebbled using 12 moves.

Proof: Since G has size 12, at least 12 moves are necessary in any pebbling strategy.

Consider the case when S = 3, and let O^* be the first output of G to be pebbled. At the point in time just before O^* is pebbled, pebbles must be on the predecessors of O^*, and at most one input of G can have a pebble. To pebble O^*, we must move a pebble from either the input holding a pebble or one of the predecessors of O^*. But, both this vertex and the three uncovered inputs must be repebbled at some point later in the pebbling process in order to reach the remaining outputs. Thus, at least 4 extra pebble placements are required in order to reach all outputs of G, so when

$S = 3, T \geq 16$.

When 5 pebbles are available, G can be pebbled in 12 moves using the strategy which proceeds from level to level in the graph. First, pebbles are placed on the 4 inputs of G. The fifth available pebble is then placed on a successor to an input, which allows other successors of inputs to be pebbled. Outputs of G are pebbled in the same manner. □

The above argument shows us that the graph G exhibits a time-space tradeoff, since the values of $S = 3$ and $T = 12$ cannot be simultaneously achieved. In order to decrease time, space must be increased, and in order to decrease space, time must necessarily increase.

1.4. Topics Discussed.

This dissertation analyzes a number of different tradeoffs which can occur between resources in the pebble game abstraction. In Chapter 2, we study time-space tradeoffs for pebbling the graph associated with back-to-back applications of a Fast Fourier transform. The main result derived is a lower bound of the form $T \cdot S = \Omega(n^2 \log n)$ for an n-input back-to-back FFT graph. Since n-degree polynomial multiplication can be implemented using this type of structure, the above lower bound shows us that space must be on the order of n to achieve time on the order of $n \cdot \log n$. We can also observe that such an implementation is non-optimal with respect to the product of time and space, since the result of $T \cdot S = \Omega(n^2)$ due to Grigoryev [15] and Tompa [44] is achieved by the standard "high-school" algorithm (it uses constant space and n^2 time). Another application for the back-to-back FFT graph which we discuss is a permutation graph. For the back-to-back FFT graph, non-intersecting paths connecting inputs with associated outputs can be found for any permutation mapping inputs to outputs. Thus, the bound $T \cdot S = \Omega(n^2 \log n)$ holds for n-input

permutation graphs implemented in this manner.

Chapter 3 introduces the concept of a size-space tradeoff in the pebble game. We demonstrate that for some graphs, if the minimal space needed for pebbling is restricted, then the graphs cannot have optimal size. Problems studied from this point of view include oblivious merging and pattern matching. The main result stated is that if available space is restricted to grow asymptotically as $o(\log n)$, then a merging network with n-inputs must have size growing as $\omega(n \cdot \log n)$. Since merging networks of size $\Theta(n \cdot \log n)$ can easily be constructed, it follows that optimal sized merging networks do not exist when the space restriction in effect is too severe. We also investigate the relationship between size and space for another type of graph called a superconcentrator.

Chapter 4 analyzes the types of time-space tradeoffs associated with a pebble game model for stack-based implementations of non-linear recursion. This extends the work of Paterson and Hewitt [28], Chandra [8], and Savage and Swamy [39], who considered implementations of linear recursion. Pebbling strategies optimal with respect to either time or space are presented, and the types of behavior associated with minimum space are explored. While pebbling time growing on the order of the path length of an associated tree may be necessary when minimum space is used, we demonstrate that time proportional to the size of the tree can be achieved in certain situations. Knowledge of this kind can be helpful when space is an important consideration in an implementation of non-linear recursion.

Chapter 5 treats the topic of "extreme" time-space tradeoffs in the pebble game. It is demonstrated that n-input oblivious sorting based on a bitonic merge procedure exhibits an extreme time-space tradeoff in that space restricted to be on the order of $n^{1-\epsilon}$ implies superpolynomial time. Tompa [43]

has independently obtained this result and a similar one for computing the transitive closure of a boolean matrix using successive squaring. In addition to this, we present graph families for which superpolynomial pebbling time is associated with a large range of space. The upper bounds in these ranges are shown to compare favorably with the best known bounds due to Lengauer and Tarjan [23]. We also examine the lower limit on space, and construct graphs with N vertices which require superpolynomial pebbling time for space in the range: $\Theta[f(N), (\frac{\sqrt{N}}{f(N)})^{1-\epsilon}]$. Here, f(N) is a function of N which grows asymptotically as $\omega(1)$. From this, we conclude that the lower limit on space associated with superpolynomial pebbling time can be reduced to grow as any slowly increasing function of the graph's size.

The tradeoffs outlined above and discussed in this dissertation have importance in a number of different contexts. A lower bound on the product of time and space for a particular algorithm is a useful tool which can be utilized to quickly determine the sacrifice which must be made in order to reduce the use of a critical resource. Extreme tradeoffs give examples of resource restrictions which should be avoided when possible, and tradeoffs between size and space point to a new topic that warrants further investigation. While the results presented here may not have a large impact on practical computing, they do contribute to a more complete understanding of the complexity of certain computational problems.

CHAPTER 2

THE BACK-TO-BACK FFT GRAPH

This chapter examines the graph associated with back-to-back applications of a Fast Fourier Transform algorithm, and uses the pebble game to derive lower bounds on its simultaneous time and space requirements. The main result stated is a lower bound on the product of time and space for a back-to-back FFT graph. Specifically, $T \cdot S = \Omega(n^2 \log n)$ for an n-input back-to-back FFT graph. This lower bound is derived by considering the number of moves required to pebble a subset of outputs with size proportional to the space available. A back-to-back FFT graph can be used to implement a straight-line algorithm for polynomial multiplication. It also forms a permutation graph, since vertex-disjoint paths joining inputs with associated outputs can be found for any permutation mapping inputs to outputs. For these applications, we discuss the implications of the above lower bound.

The Fourier Transform of a sequence $a = (a_0, \ldots, a_{n-1})$ over a ring S is a sequence $f(a) = (f_0, \ldots, f_{n-1})$. The function f_i is the value of the polynomial

$$p(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

at v^i, where v is a principal nth root of unity. The FFT algorithm is derived by splitting the polynomial $p(x)$ into its odd and even terms. To form a Fourier transform of size n, we recursively form two Fourier transforms of size $n/2$ each on the odd and even terms, and then combine the results. This yields an algorithm in which the operations are executed in an oblivious, or data-independent manner. As our canonical representation of the graph associated with an FFT algorithm, we will use the definition presented in Savage and Swamy [36].

Definition: Let $n=2^d$, $d \geq 0$. The directed graph $FFT[d] = (V[d], E[d], O[d])$ on n inputs has vertex set $V[d]$, edge set $E[d]$, and output vertex set $O[d]$. It is defined recursively in terms of two disjoint graphs

$FFT_0[d-1] = (V_0[d-1], E_0[d-1], O_0[d-1])$ and

$FFT_1[d-1] = (V_1[d-1], E_1[d-1], O_1[d-1])$

on $n/2$ inputs each as follows. Let

$O[d] = \{x_0^d, \ldots, x_{2^d-1}^d\}$

be a set of vertices disjoint from $FFT_0[d-1]$ and $FFT_1[d-1]$. Then

$V[d] = V_0[d-1] \cup V_1[d-1] \cup O[d]$

$E[d] = E_0[d-1] \cup E_1[d-1] \cup \Delta[d]$

where

$$\Delta[d] = \bigcup_{i=0}^{2^{d-1}-1} \delta_i[d]$$

$\delta_i[d] = \{(x_{i,0}^{d-1}, x_i^d), (x_{i,0}^{d-1}, x_{i+2^{d-1}}^d), (x_{i,1}^{d-1}, x_i^d), (x_{i,1}^{d-1}, x_{i+2^{d-1}}^d)\}$

Here, (v,w) denotes an edge directed from v to w, and

$FFT[0] = \{V[0], \phi, V[0]\}$ where $V[0] = \{x_0^0\}$ and ϕ is the empty set.

An FFT graph on n=4 inputs with labeled vertices is shown in Figure 3.

Definition: A back-to-back FFT graph on $n=2^d$ inputs consists of two disjoint FFT graphs, FFT[0] and FFT[1], each on $n=2^d$ inputs, where the outputs of FFT[0] are connected to the inputs of FFT[1] by directed edges which realize some bijection.

An example of a back-to-back FFT graph on n=8 inputs is shown in Figure 4, where the connection between the two FFT subgraphs is formed by the identity permutation.

2.1. A Lower Bound on the Time-Space Product.

In order to derive a lower bound on the simultaneous time and space requirements for pebbling back-to-back FFTs, we must first establish some basic properties concerning the FFT graph itself.

Definition (Valiant [47]): An [f(S),n,m]-grate is defined to be a directed acyclic graph G with n inputs and m outputs which has the property that removing any S vertices from G leaves at least f(S) of the n·m input-output pairs connected.

Savage and Swamy [36] obtained the lower bound $T \geq \left\lfloor \frac{n}{S} \right\rfloor \cdot (n-S)$ for pebbling an n-input FFT graph with S pebbles. In their lower bound derivation, they noticed that if there are S pebbles on the FFT graph, then at most n·S input-output pairs are blocked. Thus, the FFT graph forms an [n(n−S),n,n]-grate. Tompa [44] has derived lower bounds on the pebbling time for grates when at most S pebbles are available, and in particular shows that $T \cdot S = \Omega(n^2)$ for the FFT graph. This result differs by only a constant factor from the result obtained by Savage and Swamy. It is dependent on the following lemma, which will be used in future results concerning the FFT graph.

Lemma 2.1 (Tompa [44]): For an FFT graph on $n=2^d$ inputs, pebbling any set of 2S outputs, when at most S pebbles are available and initially on the graph ($S \leq n/2$), requires pebbling at least $n/2$ inputs.

At this point, we introduce the concept of a rotation graph.

Definition (Pippenger and Valiant [31]): An n-rotation graph is a dag on n inputs $\{x_i\}$ and n outputs $\{y_i\}$ $(0 \leq i < n)$ such that for each rotation $R_j(x_i) = y_{(i+j) \bmod n}$ $(0 \leq j < n)$, there exist vertex-disjoint paths joining inputs with their associated outputs.

The following lemma shows that an FFT graph forms a rotation graph.

Lemma 2.2: The FFT graph on $n=2^d$ inputs is an n-rotation graph.

Proof: By induction on d.

Basis: d=0. The FFT graph on 1 input trivially implements the set of 1−rotations.

Induction: Assume that the FFT graph on 2^{d-1} inputs forms a 2^{d-1}−rotation graph. To show that the FFT graph on $n=2^d$ inputs is an n-rotation graph, consider an arbitrary rotation $R_j(x_i) = y_{(i+j) \bmod n}$. By the definition given in Savage and Swamy [36], the FFT graph on $n=2^d$ inputs consists of two disjoint FFT graphs on 2^{d-1} inputs each, where the ith output of each subgraph is connected to outputs i and $(i+\frac{n}{2}) \bmod n$ of the original graph.

By induction, we can implement the rotation

$R_{j \bmod n/2}(x_{i \bmod n/2}) = y_{(i \bmod n/2 + j \bmod n/2) \bmod n/2} = y_{(i+j) \bmod n/2}$

on the two distinct FFT subgraphs on 2^{d-1} inputs each. This gives us vertex-disjoint paths from inputs to vertices at level d−1 in the FFT graph. For $0 \leq i < n/2$, input i connects with vertex $(i+j) \bmod n/2$, and for $n/2 \leq i < n$, input i connects with vertex $(i+j) \bmod n/2 + n/2$. But, by the definition of the FFT graph, vertices $(i+j) \bmod n/2$ and $(i+j) \bmod n/2 + n/2$ at level d−1 connect with output vertices $(i+j) \bmod n/2$ and $(i+j) \bmod n/2 + n/2$ at level d. Since n is a power of two, $(i+j) \bmod n$ is

either

$(i+j) \bmod n/2$ (when $0 \leq (i+j) \bmod n < n/2$) or

$(i+j) \bmod n/2 + n/2$ (when $n/2 \leq (i+j) \bmod n < n$), and thus we are able to find edges from level $d-1$ to level d which allow us to implement the original rotation. □

In some sense, the FFT graph forms an optimal rotation graph, since any rotation graph which has outdegree two must possess at least $2n \cdot \log n$ edges (Valiant [45], and Lamagna and Savage [22]). This is related to a more general result for rotation graphs which states that their size must grow on the order of $n \cdot \log n$. The following lemma extends this result to account for deleted or "blocked" vertices in a rotation graph.

Lemma 2.3: Let G be an n-rotation graph with bounded in-degree. If S vertices are deleted from G, then the augmented rotation graph G^* must have size of at least $(n-S) \cdot \log(n-S)$.

Proof: Without loss of generality, we can assume that G has indegree 2, since a graph with bounded indegree can be converted to one with indegree 2 with at most a constant factor increase in size (see Savage [35]). Since at most S vertices have been deleted from the graph G, for every rotation, there must exist sets of $n-S$ inputs and $n-S$ outputs in G^* which are connected by vertex-disjoint paths. Let these paths be denoted by $P_{0,i}, \ldots, P_{n-S-1,i}$ for rotation i $(0 \leq i < n)$. Paralleling the argument Pippenger and Valiant [31] and Lamagna and Savage [22] employ, we consider the sum:

$$D(G^*) = \sum_{0 \leq i < n} \sum_{0 \leq j < n-S} \sum_{v \in P_{j,i}} 1.$$

For each rotation, the paths are vertex-disjoint, thus

$$\sum_{0\le j<n-S} \sum_{v\in P_{j,i}} 1 \le \text{size}(G^*) \quad \text{(for all i)}$$

so,

$D(G^*) \le n\cdot\text{size}(G^*)$.

The path $P_{j,i}$ joins some input x_k with output $y_{(k+i) \bmod n}$. Considering all possible rotations, there will be a set of paths emanating from the kth input, which we denote by T_k. We can rewrite the sum $D(G^*)$ as:

$$D(G^*) = \sum_{0\le k<n} \sum_{v\in T_k} 1.$$

If input k has paths leading to s_k outputs, then the set of paths T_k will contain a binary tree with s_k leaves, and

$$\sum_{v\in T_k} 1 \ge \{\text{internal path length of a tree with } s_k \text{ leaves}\}$$

$$\ge s_k \log s_k.$$

Thus,

$$D(G^*) \ge \sum_{0\le k<n} s_k \log s_k.$$

Now, there are n rotations and n−S paths for each rotation, so

$$\sum_{0\le k<n} s_k \ge n\cdot(n-S).$$

By Jensen's inequality (see Feller [11]), we have

$$\frac{1}{n} \sum_{0\le k<n} s_k \log s_k \ge \left(\frac{1}{n} \sum_{0\le k<n} s_k\right)\cdot\log\left(\frac{1}{n} \sum_{0\le k<n} s_k\right)$$

hence

$$D(G^*) \ge n\cdot(n-S)\cdot\log(n-S)$$

and it follows that

$$\text{size}(G^*) \ge (n-S)\cdot\log(n-S). \quad \square$$

We are now in a position to prove a lower bound on the pebbling time for back-to-back FFT graphs when S pebbles are available.

Theorem 2.1: The number of moves necessary to pebble the outputs of a back-to-back FFT graph on $n=2^d$ inputs, when $S \leq n/2$ pebbles are used, satisfies:

$$T \geq \left\lfloor \frac{n}{2S} \right\rfloor \cdot (\frac{n}{2} - S) \cdot \log(\frac{n}{2} - S).$$

Proof: A back-to-back FFT graph consists of the two disjoint FFT graphs FFT[0] and FFT[1], where FFT[0] contains the inputs and FFT[1] contains the outputs of the back-to-back FFT graph. By Lemma 2.1, when $S \leq n/2$, pebbling a set of 2S output vertices of the entire graph requires pebbling at least $n/2$ inputs of FFT[1], which in turn, are outputs of FFT[0]. To analyze the work entailed in pebbling these $n/2$ outputs of FFT[0], we observe that at most S pebbles are on FFT[0], and at most $n/2$ of its outputs do not have to be pebbled. By Lemma 2.2 and the above observation, FFT[0] forms an n-rotation graph where at most $S + n/2$ of its vertices are "blocked". Thus, we can apply Lemma 2.3 to show that the subgraph of FFT[0] which must be pebbled contains at least $(n - S - \frac{n}{2}) \cdot \log(n - S - \frac{n}{2})$ vertices. This process is repeated for every block of 2S outputs of FFT[1] pebbled, hence

$$T \geq \left\lfloor \frac{n}{2S} \right\rfloor \cdot (\frac{n}{2} - S) \cdot \log(\frac{n}{2} - S). \quad \square$$

Corollary 2.1: The time-space product for pebbling back-to-back FFTs grows as $T \cdot S = \Omega(n^2 \log n)$.

Proof: Consider the cases $S > n/4$ and $S \leq n/4$. $\square$

From this lower bound, we can conclude that the optimal pebbling time of $\Theta(n \cdot \log n)$ is achievable only when space is $\Omega(n)$. This is in marked contrast to a single FFT graph, which achieves optimal pebbling time for a much larger range of space [36]. Also, it is obvious that the lower bound shown is the best possible,

since the pebbling strategy which operates by pebbling the back-to-back FFT graph "level by level" (n+1 pebbles are necessary for an n-input graph) achieves the time-space product $T \cdot S = O(n^2 \log n)$ (and is optimal with respect to time). However, it would be interesting to exhibit a pebbling strategy which is optimal in that $T = O(\frac{n^2 \log n}{S})$ for some large range of space. Pippenger (see Tompa [42]) has come the closest to meeting this bound by developing a pebbling strategy similar to that used in Savage and Swamy [36], which pebbles a back-to-back FFT graph on n inputs with S pebbles in time $T = O(\frac{(n \cdot \log n)^2}{S})$.

2.2. Applications for Back-to-Back FFTs.

A well known method for speeding up the multiplication of two polynomials, each of degree n, is to employ the Fast Fourier Transform. This reduces the time complexity from $O(n^2)$ for the straightforward algorithm to $O(n \cdot \log n)$. The basic idea is to perform back-to-back transformations on the polynomials. The first transformation is from a representation of the polynomials by their coefficients to a representation by their values at points (nth roots of unity). In this environment, multiplication can be done in O(n) operations versus the $O(n^2)$ operation count of the standard algorithm. We then transform back to the original representation via interpolation, which yields the desired product. In each case, the transformation is done by computing a Fourier Transform, which can be done in $O(n \cdot \log n)$ steps by the FFT algorithm. Thus, we can perform polynomial multiplication in time $O(n \cdot \log n)$ by computing "back-to-back" FFTs, or analogously, "pebbling" the graph associated with a back-to-back FFT algorithm.

Corollary 2.2: The time-space product for multiplying two n-degree polynomials grows as $\Omega(n^2 \log n)$.

Proof: Let p_1 and p_2 be two n-degree polynomials. When p_1 and p_2 are considered to be 2n-degree polynomials whose n higher order coefficients are all zero, then their product can be computed by taking 2n-input Fourier transforms of each polynomial, computing the pairwise product of the transforms, and then inverting (see Aho, Hopcroft, and Ullman [1]). Inversion can be accomplished using another Fourier transform, so that when the FFT algorithm is employed, the graph associated with n-degree polynomial multiplication contains a back-to-back FFT graph on 2n inputs. By Corollary 2.1, $T \cdot S = \Omega(n^2 \log n)$. □

An important observation concerning polynomial multiplication implemented via FFTs is that algorithms (pebbling strategies) optimal with respect to time exist only when space grows asymptotically on the order of the number of inputs to the problem (the degree of the polynomials being multiplied). As space is restricted, the savings offered by performing a fast transform decrease, and essentially evaporates as space approaches its minimum value. When minimum space is used in an FFT implementation, time on the order of n^2 is required. It is worthwhile to note that this time bound is achieved by the standard "high-school" algorithm, which uses substantially less space.

We saw earlier that an FFT graph on $n=2^d$ inputs implements the set of n-rotations. It is also true that a back-to-back FFT graph constructed with a certain connection pattern implements the set of n-permutations (i.e. for any permutation between inputs and outputs, we can find vertex-disjoint paths joining inputs with their associated outputs). Permutation graphs have

applications in the design of communication switching networks [3, 4, 19, 48]. These networks have equal numbers of inputs and outputs, and allow any set of input-output pairs forming a permutation to communicate simultaneously via distinct connections.

Beizer [3] showed that the graph P[d] of Figure 5 is a permutation graph. It has $n=2^d$ inputs and outputs and is constructed recursively from two disjoint subgraphs $P_0[d-1]$ and $P_1[d-1]$ which are permutation graphs on $n/2$ inputs each. This construction is symmetric about a horizontal axis, and it is easily shown that both the top and bottom halves are FFT graphs. The bottom half is actually a "reversed" FFT graph (inputs and outputs switch roles). But, as noted in Stone [41], in this case, the bottom half can be regarded as an FFT graph which produces its outputs "scrambled" by the bit reversal permutation. Thus, P[d] forms a back-to-back FFT graph where the connection is made with the bit reversal permutation.

Corollary 2.3: The time-space product for pebbling P[d], a permutation graph graph on $n = 2^d$ inputs, grows as $\Theta(n^2 \log n)$.

Proof: Immediate from Corollary 2.1 and the remarks following it. □

The best known lower bound on the simultaneous time and space requirements for a general permutation graph on n inputs is $T \cdot S = \Omega(n^2)$. This follows from the lower bound given in Tompa [44] for graphs possessing concentration properties, because permutation graphs possess these properties. We have seen that back-to-back FFT graphs, with a suitable connection pattern, form a permutation graph, and come close to meeting this lower bound. However, graphs with concentration properties are known to have a less restrictive size requirement, so it would be of interest to either construct

permutation graphs which meet the above lower bound for some range of space, or demonstrate that they have an asymptotically stronger tradeoff than concentrators.

CHAPTER 3

SIZE-SPACE TRADEOFFS

In this chapter, we examine the effects that restricted space has on the resource of graph size in the pebble game. Tradeoffs which relate both size and space and depth and space are obtained for graphs modeling problems such as oblivious merging and pattern matching. Thus, for certain problems, it is impossible to achieve given values of size and space (or depth and space) simultaneously. This extends the relationships which can occur between resources in a straight-line computation, since previous research has dealt primarily with tradeoffs between time and space. The results we derive also have applications to the register allocation problem for arithmetic expressions.

3.1. Restricted Space in Binary Trees.

In order to derive tradeoffs relating size and space, we will first study the depth and path length requirements of a binary tree with n leaves when the space available to pebble the root of the tree is restricted. The binary trees we deal with are extended binary trees (Knuth [20]), where □ represents an external vertex, ○ represents an internal vertex, and L and R denote the left and right subtrees respectively.

The space requirement of a binary tree is defined inductively by:

$S(\square) = 0$;

$S(_L\overset{\bigcirc}{}_R) =$ **if** $S(L)=S(R)$ **then** $S(L)+1$ **else** $\max(S(L),S(R))$;
(see Ershov [10] or Nakata [27] or Redziejowski [33]).

For an example of a binary tree with each vertex labeled with its space requirement, see Figure 6. The space requirement of an extended binary tree is

also given by the height of the largest complete subtree which can be embedded in the tree [10, 27, 33], so that any binary tree T with n external vertices can be pebbled with space $S(T) \leq \lfloor \log n \rfloor$.

The internal (external) path length of an extended binary tree is the sum over all internal (external) vertices of the length of the path joining the vertex with the root of the tree. We can begin by obtaining expressions for the minimal path length (internal and external) of a binary tree with n external vertices which has a fixed space requirement of S pebbles. It is equivalent to minimize either internal or external path length (see Knuth [20]), so we choose to minimize internal path length by "packing" vertices as close as possible to the root of the tree. Thus, we place as many vertices as we can into each level of the binary tree.

Theorem 3.1: For a binary tree T which requires space S to pebble its root, and has minimal internal path length (over all trees with space requirement S), we have:

If k is the depth of the tree, and $k > S$, then

(1) the total *number of external vertices* in T is given by

$$EN(k,S) = \sum_{0 \leq m \leq S} \binom{k}{m}$$

(2) the *internal path length* of T is given by

$$IPL(k,S) = (k-2) \cdot \sum_{0 \leq m \leq S} \binom{k}{m} - \binom{k}{S+1} + 2.$$

Proof: Since the tree T can be pebbled in space S, the largest complete binary tree which can be embedded in T is of depth S, and has 2^S external vertices. Thus, if the number of external vertices in T is no more than 2^S, then T has minimal internal path length when it is a complete binary tree. However, if T has more than 2^S external vertices, then the problem of

minimizing path length for T can be decomposed into finding subtrees of minimal path length, one of which can be pebbled with space S, and the other which can be pebbled with space S−1 (this is the "best" we can do, since if both subtrees required space S, then the root of the original tree could not be pebbled with S pebbles).

If $k > S$ is the depth of T, then we can derive an expression for the *number of external vertices* at level j ($0 \leq j < k$), denoted here by E(j,S). Since the number of external vertices at level j in the tree with depth $k > S$ is the sum of the external vertices at level j−1 in the left and right subtrees, the quantity E(j,S) can be expressed by the recurrence:

$$E(j,S) = \begin{cases} E(j-1,S) + E(j-1,S-1) & \text{for } j \geq S \\ 0 & \text{for } 0 \leq j < S \end{cases}$$

With the initial condition that $E(j,1) = 1$ for $j \geq 1$, we obtain*

$$E(j,S) = \binom{j-1}{S-1}$$

(the binomial coefficient uniquely satisfies the above conditions). The value of E(0,S) is given by $\delta_{S,0}$.

From the above expression, we can calculate the *total number of vertices at level j* ($0 \leq j \leq k$), denoted by N(j,S). The number of vertices at level j in a complete binary tree is 2^j, and we can extend our tree T to be a complete binary tree by "planting" complete subtrees at external vertices. Thus, the number of vertices at level j in the original tree is 2^j minus the contribution which external vertices at lower levels make to a complete binary tree, i.e.

---

*We define $\binom{m}{k} = 0$ for $m < k$.

$$N(j,S) = 2^j - \{2\cdot E(j-1,S) + 2^2\cdot E(j-2,S) + \cdots + 2^{j-1}\cdot E(1,S)\}$$

$$= 2^j - \sum_{1\le i\le j-1} \binom{j-1-i}{S-1}\cdot 2^i$$

$$= 2^j - \sum_{1\le i\le j-1} \binom{i-1}{S-1}\cdot 2^{j-i}$$

$$= 2^j - \sum_{1\le i\le j-1} \binom{i-1}{S-1} \sum_m \binom{j-i}{m}$$

$$= 2^j - \sum_m \sum_{0\le i\le j-2} \binom{i}{S-1}\binom{j-i-1}{m}$$

$$= 2^j - \sum_m \{ \sum_{0\le i\le j-1} \binom{i}{S-1}\binom{j-i-1}{m} - \binom{j-1}{S-1}\binom{0}{m}\}$$

$$= 2^j - \sum_{m\ge 0} \sum_{0\le i\le j-1} \binom{i}{S-1}\binom{j-i-1}{m} + \binom{j-1}{S-1}$$

$$= 2^j - \sum_{m\ge 0} \binom{j}{m+S} + \binom{j-1}{S-1} \quad \text{[Knuth vol. 1 pg. 58 eq. (25)]}$$

$$= 2^j - \sum_{m\ge S} \binom{j}{m} + \binom{j-1}{S-1}$$

$$= 2^j - \{ \sum_{m\ge 0} \binom{j}{m} - \sum_{0\le m<S} \binom{j}{m}\} + \binom{j-1}{S-1}$$

$$= \sum_{0\le m<S} \binom{j}{m} + \binom{j-1}{S-1}.$$

From this, we can calculate the total number of external vertices, which is the number of vertices at level k plus the sum of the external vertices at level j ($0 \le j < k$). If EN(k,S) is the number of external vertices in T, then

$$EN(k,S) = N(k,S) + \delta_{S,0} + \sum_{1\le j<k} E(j,S)$$

$$= \sum_{0\le m<S} \binom{k}{m} + \binom{k-1}{S-1} + \delta_{S,0} + \sum_{1\le j<k} \binom{j-1}{S-1}$$

$$= \sum_{0\le m<S} \binom{k}{m} + \binom{k-1}{S-1} + \delta_{S,0} + \binom{k-1}{S} \quad \text{[Knuth vol. 1 pg. 54 eq. (11)]}$$

$$= \sum_{0\le m<S} \binom{k}{m} + \delta_{S,0} + \binom{k}{S}$$

$$= \sum_{0\le m\le S} \binom{k}{m}$$

(note that $\delta_{S,0}$ is absorbed into the above sum).

Knowing the number of external vertices at level j, we can easily calculate the external path length of the tree to be:

$$\begin{aligned}
EPL(k,S) &= k\cdot N(k,S) + \sum_{0\le j<k} j\cdot E(j,S) \\
&= k\cdot\{\sum_{0\le m<S}\binom{k}{m} + \binom{k-1}{S-1}\} + \sum_{0\le j<k} j\cdot\binom{j-1}{S-1} \\
&= k\cdot\sum_{0\le m<S}\binom{k}{m} + \sum_{0\le j\le k} j\cdot\binom{j-1}{S-1} \\
&= k\cdot\sum_{0\le m<S}\binom{k}{m} + \sum_{0\le j\le k} S\cdot\binom{j}{S} \\
&= k\cdot\sum_{0\le m<S}\binom{k}{m} + S\cdot\binom{k+1}{S+1} \\
&= k\cdot\sum_{0\le m\le S}\binom{k}{m} - k\cdot\binom{k}{S} + S\cdot\binom{k+1}{S+1} \\
&= k\cdot\sum_{0\le m\le S}\binom{k}{m} + \left(\frac{S(k+1)-k(S+1)}{S+1}\right)\binom{k}{S} \\
&= k\cdot\sum_{0\le m\le S}\binom{k}{m} + \left(\frac{S-k}{S+1}\right)\binom{k}{S} \\
&= k\cdot\sum_{0\le m\le S}\binom{k}{m} - \binom{k}{S+1}
\end{aligned}$$

and the internal path length is given by:

$$\begin{aligned}
IPL(k,S) &= EPL(k,S) - 2\cdot\{\text{number internal vertices in T}\} \\
&= EPL(k,S) - 2\cdot\{EN(k,S) - 1\} \\
&= EPL(k,S) - 2\cdot\{\sum_{0\le m\le S}\binom{k}{m} - 1\} \\
&= (k-2)\cdot\sum_{0\le m\le S}\binom{k}{m} - \binom{k}{S+1} + 2.
\end{aligned}$$

Note that the above formulas hold even for $k = S$ (we have a complete binary tree in this case). □

Theorem 3.1 considered the binary tree with space restriction S and minimal path length which had the maximum number of vertices at each level. However, trees with minimal path length may not necessarily have a "complete"

bottom level, so the following corollary considers this case in deriving a relationship between the internal path length, the depth, and the number of external vertices in a binary tree with minimal path length.

Corollary 3.1: For a binary tree T which requires space S to pebble its root and has minimal internal path length, if the tree has depth $k > S$, then

$$IPL(k,S) = \Omega(k \cdot EN(k,S)).$$

Proof: Here, we assume that we may not have packed the maximum number of vertices into level k, so that level k has N(k,S) vertices (all external), and level k−1 has $N(k-1,S) - \frac{1}{2}N(k,S)$ external vertices (since the number of internal vertices at level k−1 is half of the number of vertices at level k). Thus,

$$EN(k,S) = N(k,S) + N(k-1,S) - \frac{1}{2}N(k,S) + \delta_{S,0} + \sum_{1 \le j < k-1} E(j,S)$$

$$= \frac{1}{2}N(k,S) + \sum_{0 \le m \le S} \binom{k-1}{m}$$

(from the expression derived for EN(k,S) in Theorem 3.1).

From this, we can derive a lower bound on the external path length of T as follows:

$$EPL(k,S) = k \cdot N(k,S) + (k-1)\{N(k-1,S) - \frac{1}{2}N(k,S)\} + \sum_{0 \le j < k-1} j \cdot E(j,S)$$

$$= \frac{k+1}{2}N(k,S) + (k-1) \cdot N(k-1,S) + \sum_{0 \le j < k-1} j \cdot E(j,S)$$

$$= \frac{k+1}{2}N(k,S) + (k-1) \cdot \sum_{0 \le m < S} \binom{k-1}{m} + S \cdot \binom{k}{S+1}$$

(from the expression derived for EPL(k,S) in Theorem 3.1)

$$= \frac{k+1}{2}N(k,S) + (k-1) \cdot \sum_{0 \le m < S} \binom{k-1}{m} + \frac{S \cdot k}{S+1}\binom{k-1}{S}.$$

Now,

$$\frac{S \cdot k}{S+1} \geq \frac{k}{2} \text{ for } S \geq 1$$

so,

$$EPL(k,S) \geq \frac{k+1}{2} N(k,S) + (k-1) \cdot \sum_{0 \leq m < S} \binom{k-1}{m} + \frac{k}{2} \binom{k}{S}$$

$$\geq \frac{k-1}{2} \{ \frac{1}{2} N(k,S) + \sum_{0 \leq m < S} \binom{k-1}{m} + \binom{k-1}{S} \}$$

$$= \frac{k-1}{2} EN(k,S).$$

Thus,

$$IPL(k,S) = EPL(k,S) - 2 \cdot EN(k,S) + 2$$

$$\geq \frac{(k-5)}{2} EN(k,S).$$

$$= \Omega(k \cdot EN(k,S)). \quad \square$$

From the above Corollary, we see that depth and minimal path length are closely related. If we can show that restricted space causes the depth of the tree to grow non-optimally, then the same premise will be true for the tree's path length. We can apply the formulas of Theorem 3.1 and Corollary 3.1 when the binary trees we are dealing with have a space restriction of S(n), and have at least $n \geq 2^{S(n)}$ external vertices. Here, the available space may be some function of the number of external vertices in the tree. For example, if $S(n) = o(\log n)$, then $2^{S(n)} = o(n)$, so we must be able to find an integer M such that any binary tree with $n \geq M$ external vertices and a space restriction of S(n) satisfies Corollary 3.1. The following theorem gives an asymptotic bound on depth when space grows as $o(\log n)$.

Theorem 3.2: For binary trees with n external vertices, if the space available to pebble the root grows as $S(n) = o(\log n)$, then the depth of the tree grows as $\omega(\log n)$.

Proof: by contradiction.

Let T be a binary tree with n external vertices, with the space restriction S(n), and assume that the depth of T grows as $\Theta(\log n)$. The tree T must have depth of at least k, where k is the depth of a binary tree constructed to have minimal path length with space restriction S(n). From Theorem 3.1 and Corollary 3.1, we have

$$\sum_{0\le m\le S(n)} \binom{k-1}{m} \le n \le \sum_{0\le m\le S(n)} \binom{k}{m}.$$

Note that there exists an integer M such that $S(n) \le k/2$ for all $n \ge M$, since depth $k \ge \log n$ and $S(n) = o(\log n)$. Thus, for $n \ge M$, the above expression can be bounded by

$$\binom{k-1}{S(n)} \le \sum_{0\le m\le S(n)} \binom{k-1}{m} \le \sum_{0\le m\le S(n)} \binom{k}{m} \le S(n)\cdot\binom{k}{S(n)}.$$

We can approximate the binomial coefficient $\binom{k}{S(n)}$ by

$$\left(\frac{k}{2\pi S(n)(k-S(n))}\right)^{1/2}\cdot\exp\{k\cdot H(\frac{S(n)}{k})\}$$

where

$$H(\frac{S(n)}{k}) = -\frac{S(n)}{k}\ln(\frac{S(n)}{k}) - (1-\frac{S(n)}{k})\cdot\ln(1-\frac{S(n)}{k})$$

(see Gallager [13]). Since $S(n) = o(\log n)$ and $k = \Omega(\log n)$, the function $H(\frac{S(n)}{k}) \to 0$ as $n \to \infty$ (note that if $x \to 0$, then $x\cdot\log x \to 0$).

If $k = \lambda\cdot\log n$ for some constant λ, then

$$\exp\{k\cdot H(\frac{S(n)}{k})\} = \exp\{\lambda\cdot\log n\cdot H(\frac{S(n)}{k})\}$$

$$= n^{\lambda\cdot H(\frac{S(n)}{k})} = n^{o(1)}.$$

But, this is the dominant term in the approximation for n, the number of external vertices in the tree. Thus, we have a contradiction, and k, the depth of the tree, must grow as $\omega(\log n)$. From Corollary 3.1, we can also

conclude that if available space grows as o(log n), then path length must grow as $\omega(n \cdot \log n)$. □

The above theorem has a wide range of applications, since many problems in computer science have the concept of a binary tree in their underlying representations. For one, the evaluation of an arithmetic expression on n terms can be modeled by a binary tree, where pebbling a vertex of the tree corresponds to placing the value of a sub-expression into a register. If available space is restricted to o(log n) for some reason, then the binary tree associated with the expression must be "skewed" to some degree, and has non-optimal depth.

3.2. Graphs for Merging and Pattern Matching.

The main result of this section is the demonstration of a size-space tradeoff for rotation graphs, which were defined in Section 2.1. By a size-space tradeoff, we mean that when space is restricted too severely, optimal sized rotation graphs cannot be constructed. Thus, certain values of size and space cannot be realized simultaneously.

***Theorem** 3.3:* If the space available to pebble a rotation graph on n inputs with bounded indegree grows as o(log n), then the size of the graph must grow as $\omega(n \cdot \log n)$.

Proof: Let G be a rotation graph on n inputs, and let size(G) denote the size of G. Without loss of generality, we can assume that G has indegree 2, since a graph with bounded indegree can be converted to one with indegree 2 with at most a constant factor increase in size (see Savage [35]). We can parallel the argument in Pippenger and Valiant [31] and Lamagna and Savage [22], as follows. Consider the sum

$$D(G) = \sum_{0 \le i < n} \sum_{0 \le j < n} \sum_{v \in P_{j,i}} 1.$$

Here, $P_{0,i}, \ldots, P_{n-1,i}$ are n vertex-disjoint paths from inputs to outputs associated with the ith rotation. Since the paths are vertex-disjoint, it is easily seen that

$$D(G) \le n \cdot size(G).$$

Changing the order of summation in D(G) yields:

$$D(G) = \sum_{0 \le j < n} \{ \sum_{0 \le i < n} \sum_{v \in P_{j,i}} 1 \}.$$

But, the inner sum is at least the internal path length of a binary tree rooted at input j with n external vertices. Since the space available to pebble this binary tree is S = o(log n), Theorem 3.2 gives us

$$D(G) \ge n \cdot \omega(n \cdot \log n).$$

Thus,

$$size(G) = \omega(n \cdot \log n). \quad \square$$

Since rotation graphs of size $\Theta(n \cdot \log n)$ can easily be constructed (an FFT graph for example), Theorem 3.3 shows us that for a suitable restriction on space, optimal sized rotation graphs cannot be constructed. An equivalent statement is that the value of space o(log n) and the value of size $\Theta(n \cdot \log n)$ cannot be achieved simultaneously. The theorem can also be modified to graphs on n inputs which implement a set of m assignments ($m \le n$), where each input is assigned to m different outputs by the m assignments. Such graphs include displacement graphs, which form the heart of oblivious algorithms for the problems of merging and pattern matching (see Pippenger and Valiant [31] and Valiant [45]). Thus, a result analogous to Theorem 3.3 holds for these problems: optimal sized networks cannot be constructed if enough space is not available.

Corollary 3.2: If available space is o(log n), then the graph associated with a straight-line algorithm for merging two sorted lists, each of length n, must have size $\omega(n \cdot \log n)$.

Proof: By Pippenger and Valiant [31], the graph associated with merging two sorted lists, each of length n, forms an $n/2$-rotation graph. Applying Theorem 3.3 yields the desired result. □

Corollary 3.3: If available space is o(log n), then the graph associated with a straight-line algorithm for matching a pattern of length r in a string of length $n-r$ must have size $\omega(n \cdot \log n)$.

Proof: By Valiant [45], the graph associated with such an algorithm forms an $(n/3,2n/3)$-displacement graph. Displacement graphs are defined in the above reference, and resemble rotation graphs in that they implement a set of vertex-disjoint paths for each displacement. By an argument similar to that in Theorem 3.3, when space is o(log n), the size of an $(n/3,2n/3)$-displacement graph must be $\omega(n \cdot \log n)$. □

3.3. Superconcentrators.

Another class of graphs which are interesting to study from the viewpoint of restricted space are superconcentrators. A superconcentrator with m inputs and m outputs is defined to be a dag such that for any $i \leq m$, for any set of i inputs and any set of i outputs, there is a set of vertex-disjoint paths from inputs to outputs. Since each output of a superconcentrator depends on all of its inputs, by Theorem 3.2 (assuming indegree of two), if space is chosen as o(log n), then the depth of the graph grows as $\omega(\log n)$. This is non-optimal, since back-to-back FFTs implement a superconcentrator [46] and have depth $\Theta(\log n)$.

For superconcentrators, we can only conjecture the analogue of Theorem 3.3, i.e. if space is restricted to o(log n), then size must be $\omega(n)$ (this is non-optimal in that Valiant [45], Pippenger [32], and Gabber and Galil [12] have all shown the existence of linear-sized superconcentrators). However, when only two pebbles are available, we can show that a superconcentrator G on n inputs, with indegree and outdegree of two, must have size $\Omega(n^2)$.

In the previous discussion on the space requirements for binary trees, we assumed that external vertices could be pebbled at no cost, while here we will assume that they have cost one. Since each output of G can be pebbled with two pebbles, the binary tree rooted at an output must have a chain-like structure, as shown in Figure 7. Notice that only external vertices can be inputs to the chain (they may be repeated), since otherwise we violate the ability to reach the root (output) of the tree with two pebbles. Vertices along the chain can have outdegree two, i.e. they can "spawn" another chain; such vertices will be referred to as splitting vertices. If we consider a binary chain off of which other chains are spawned, we will be able to locate the lowest splitting vertex in the chain, as shown in Figure 8 (all other vertices in the chain below it have outdegree one). External vertices can be inputs to a number of chains in the set.

Consider the case when this "set" of chains connects n input vertices with m output vertices (each output is the root of a binary tree which must depend on all inputs). If L(n,m) denotes the number of internal vertices in this set of chains, then we can derive a lower bound on the size of a set of chains as follows:

Lemma 3.1: If L(n,m) is the number of internal vertices in a set of chains which connects n inputs with m outputs in a superconcentrator that can be pebbled with two pebbles, then

$$L(n,m) \geq n \cdot m - m.$$

Proof: First, we can derive a recurrence on L(n,m) by splitting the set of chains into two sets of chains at a lowest splitting vertex. When $m > 1$, there must be a lowest splitting vertex which splits the m outputs in the set into two disjoint classes of outputs whose sizes are both greater than zero and sum to m. Since an internal vertex cannot have two internal vertices (outputs of two chains) as predecessors, the internal vertices which appear above a splitting vertex will also be split into two disjoint classes. If r inputs appear in the original set of chains only at or below the lowest splitting vertex, then the splitting vertex is the output of a chain with r external vertices and outdegree one, which has at least $r-1$ internal vertices. Also, paths connecting these inputs to any of the m outputs must pass through the splitting vertex. At least $n-1$ inputs must appear in each of the two sets of chains derived, since otherwise, at least two appear only at or below the splitting vertex, and by the above comment, there will not be vertex-disjoint paths joining them with any two of the m outputs in the original set of chains. This contradicts superconcentration properties. The splitting vertex can be regarded as the point of entry of the remaining input into the two sets of chains derived. Thus, when $m > 1$ we have:

$$L(n,m) \geq \min_{r,m_1,m_2} \{r-1 + L(n,m_1) + L(n,m_2)\}.$$

$$\text{where } r \geq 1 \text{ and } m_1 + m_2 = m.$$

To prove the Lemma, we use induction on m.

Basis: m=1. The chain with one output has size

$$L(n,1) \geq n - 1.$$

Induction: Assume, for $k < m$, that

$$L(n,k) \geq n \cdot k - k.$$

From the above recurrence, minimizing over r, we have

$$L(n,m) \geq L(n,m_1) + L(n,m_2) \quad (m_1 + m_2 = m).$$

Both m_1 and m_2 must be less than m, since otherwise we contradict the assumption that we have a chain with a splitting vertex. Thus, we can apply the inductive hypothesis, which yields:

$$L(n,m) \geq n \cdot m_1 - m_1 + n \cdot m_2 - m_2$$

$$= n \cdot m - m. \quad \square$$

Theorem 3.4: Given a superconcentrator G on n inputs with indegree and outdegree of two, for which all outputs of G can be pebbled with two pebbles. Then the size of G is $\Omega(n^2)$.

Proof: The superconcentrator G can be decomposed into k disjoint subgraphs $\{G_1, \ldots, G_k\}$ of the structure discussed in Lemma 3.1, each with m_j outputs ($1 \leq j \leq k$). From the result obtained in Lemma 3.1, we obtain an expression for the size of G:

$$\text{size}(G) = \sum_{1 \leq j \leq k} \text{size}(G_j)$$

$$\geq \sum_{1 \leq j \leq k} \{n \cdot m_j - m_j\} \quad (\text{where } \sum_{1 \leq j \leq k} m_j = n)$$

$$\geq n \cdot \sum_{1 \leq j \leq k} m_j - \sum_{1 \leq j \leq k} m_j$$

$$= n^2 - n$$

$$= \Omega(n^2). \quad \square$$

CHAPTER 4

TRADEOFFS FOR NON-LINEAR RECURSION

Functions implemented in a computer system can often be written and understood in simple terms using a recursive definition. In this type of definition, invocation of the function with a specific argument may lead to one or more invocations of the function with modified arguments. A schema for such recursion is given as:

$$F(x) := \textbf{if } p(x) \textbf{ then } h(x) \textbf{ else } g(x,F(f_1(x)), \ldots, F(f_k(x))) \textbf{ fi;}$$

Here, F is the function being implemented, p is a unary predicate, h is a unary function, $f_1, \ldots, f_k$ are unary functions which modify the argument to F, and g is a k+1-ary function.

Many programming languages allow the definition of recursive functions. The standard implementation of such a function uses a stack, and in the worst case, the size of the stack grows linearly with the number of recursive calls made. In certain situations, it may be desirable to reduce the space requirements of such a stack-based implementation at the expense of a possible increase in computation time. This chapter presents a graph model for the sequence of steps involved in evaluating a recursively defined function, and uses the pebble game to analyze both time and space efficient implementations of recursion.

Paterson and Hewitt [28], Chandra [8], and Savage and Swamy [39] have used the pebble game to analyze the time and space requirements of linear recursion, where a procedural call can activate at most one other procedural call. Here, we will deal with non-linear recursion, in which more than one procedural call may be activated. Specifically, we will consider the recursive schema:

$F(x)$:= **if** $p(x)$ **then** $h(x)$ **else** $g(x,F(f_1(x)), F(f_2(x)))$ **fi**;

The interpretation of this schema is that, with input a, if p(a) is TRUE, then F(a) = h(a). Otherwise, we must evaluate F with arguments $f_1(a)$ and $f_2(a)$, and F(a) is given by $g(a, F(f_1(a)), F(f_2(a)))$. An example of a recursive definition is for a procedure which converts an arithmetic expression written in polish prefix notation to one written in infix notation. Such a procedure can be defined as:

INF(expr) := **if** (expr = var) **then** expr **else** INF(expr1) op INF(expr2) **fi**;

Here, the argument expression is split into two subexpressions when it is not a simple variable, and the subexpressions are each converted from polish to infix. This example does not reflect the full generality of the schema presented for non-linear recursion, since the function involked when the predicate is not true does not depend on the original argument.

In the above schema, to compute F(a), we must have available the values $F(f_1(a))$ and $F(f_2(a))$. Also, $f_1(a)$ and $f_2(a)$ are computed from the argument a. From these functional dependencies, we can form a graph model as follows:

Definition: The NLR-graph $G[a] = (V[a],I[a],O[a],E[a])$ associated with the evaluation of a non-linear recursive function F at argument a has vertex set V[a], input vertex $I[a] = \{a\}$, output vertex $O[a] = \{F(a)\}$, and edge set E[a]. It is defined recursively in terms of two disjoint NLR-subgraphs $G[f_1(a)] = (V[f_1(a)],I[f_1(a)],O[f_1(a)],E[f_1(a)])$ and $G[f_2(a)] = (V[f_2(a)],I[f_2(a)],O[f_2(a)],E[f_2(a)])$ as follows: Let a and F(a) be two vertices disjoint from $G[f_1(a)]$ and $G[f_2(a)]$. If p(a) is true, then $V[a] = \{a, F(a)\}$ and $E[a] = \{(a, F(a))\}$. Otherwise,

$V[a] = \{a, F(a)\} \cup V[f_1(a)] \cup V[f_2(a)]$ and

$E[a] = \{(a, F(a)), (a, I[f_1(a)]), (a, I[f_2(a)]),(O[f_1(a)], F(a)), (O[f_2(a)], F(a))\}$

$\cup E[f_1(a)] \cup E[f_2(a)]$

A diagram of such an NLR–graph is given in Figure 9. It is easily seen that associated with an NLR–graph of a recursive evaluation is an extended binary tree (see Section 3.1) which has a number of vertices (internal and external) equal to the number of recursive calls made and whose root is the output of the graph. External vertices of the tree represent the value of the function at arguments for which the predicate p is true. For example, associated with the NLR–graph shown in Figure 9 is a complete binary tree of depth 3. Also, the evaluation of a recursive function can be abstracted by playing the pebble game on the corresponding NLR–graph. Here, pebbles can be thought of as representing elements of the stack which are used to store the values of intermediate results, so that the number of pebbles used to reach the output of an NLR–graph accurately models the stack space requirements of a specific implementation of non-linear recursion.

4.1. Pebbling in Optimal Time.

In order to evaluate a recursively defined function, an amount of time proportional to the number of recursive calls made is necessary. This relates to the fact that each vertex of the corresponding NLR–graph must be pebbled at least once. However, some sacrifice may have to be made in terms of space to realize optimal pebbling time for an NLR–graph. To analyze this sacrifice, we introduce the concept of a ladder graph.

Definition: A ladder graph $L = (V,E)$ with $2n$ vertices has vertex set

$$V = \{s_1, \ldots, s_n, t_1, \ldots, t_n\}$$

and edge set

$$E = \{(s_i, s_{i+1}), (t_i, t_{i+1}) \mid 1 \leq i < n\} \cup \{(s_i, t_{n+1-i}) \mid 1 \leq i \leq n\}.$$

Ladder graphs serve to model the evaluation process in a linear recursive function (see Paterson and Hewitt [28], Chandra [8], or Savage and Swamy [39]). A ladder graph with ten vertices is shown in Figure 10.

Lemma 4.1: For a ladder graph with 2N vertices, N pebbles are necessary and sufficient to pebble the ladder graph in optimal time.

Proof: It is obvious that a ladder graph with 2N vertices can be pebbled in optimal time with N pebbles; we simply cover each vertex of the input path and then advance the pebble on the last vertex of the input path down the output path. Now, assume that less than N pebbles are available and consider the point in time during the pebbling process when the last vertex of the input path is first pebbled. At this point, none of the vertices in the output path have been reached, and at least one of the vertices in the input path, say vertex s_i, is not covered by a pebble. To reach the output of the ladder graph, a pebble must be advanced along the output path, and at some point vertex s_i must be repebbled, since otherwise we cannot advance past vertex t_{i-1} in the output path. Thus, the ladder graph cannot be pebbled in optimal time if less than N pebbles are available. □

The following two propositions analyze the space requirements of an optimal time pebbling strategy for an arbitrary NLR-graph. Necessary and sufficient conditions on space are given.

Proposition 4.1: For an NLR-graph whose associated binary tree has depth d, at least d+1 pebbles are required to pebble the NLR-graph in optimal time.

Proof: Since the binary tree associated with the NLR-graph has depth d, the NLR-graph must have embedded in it a ladder graph with 2(d+1) vertices. By Lemma 4.1, if less than d+1 pebbles are available, some vertex

of the embedded ladder graph must be repebbled in the course of pebbling its output vertex. Hence, the same statement holds for the NLR-graph: if less than d+1 pebbles are available, pebbling time is non-optimal. □

Proposition 4.2: For an NLR-graph whose associated binary tree has depth d, the NLR-graph can be pebbled in optimal time if 2d+1 pebbles are available.

Proof: By induction on d.

Basis: d=0. An NLR-graph whose associated binary tree has depth 0 can be pebbled with one pebble.

Induction: Assume that an NLR-graph whose associated binary tree has depth c < d can be pebbled in optimal time with 2c+1 pebbles. Consider an NLR-graph whose associated binary tree has depth d. By definition, it contains two disjoint NLR-subgraphs, each with an associated binary tree of depth c < d (one must have depth d-1 in order for the original NLR-graph to have associated depth d). To pebble the original NLR-graph in optimal time with 2d+1 pebbles, we first place a pebble on its input. While this pebble is held in place, we then pebble the two disjoint NLR-subgraphs one after the other, leaving pebbles on their outputs. By induction, this can be done in optimal time, since at least 2(d-1)+1 pebbles are available to pebble each NLR-subgraph. With pebbles on the outputs of each NLR-subgraph and the input of the original NLR-graph, the output of the NLR-graph is easily reached. Optimal time is achieved since at no time during the pebbling process did a vertex of the NLR-graph have to be repebbled. □

For an NLR-graph having associated depth d, anywhere between d+1 and 2d+1 pebbles may be required to achieve optimal pebbling time. Only d+1

pebbles are necessary for a ladder graph, but the following proposition shows that an NLR-graph whose associated binary tree is complete requires space 2d+1. Thus, Propositions 4.1 and 4.2 are the strongest statements we can make, and demonstrate that in order to realize optimal computation time, stack space must grow in proportion to the longest sequence of recursive calls made in a function evaluation.

Proposition 4.3: For an NLR-graph whose associated binary tree is complete and has depth d, 2d+1 pebbles are required to pebble the NLR-graph in optimal time.

> ***Proof:*** Consider the point in time when the last pebble-free path from an external vertex to the root of the binary tree is closed by placing a pebble on the external vertex. Just before this path is closed, there must be at least d pebbles on the tree (Proposition 1.1). Also, since the binary tree is complete, the pebble-free path is the output path in a ladder graph on d+1 elements. By Lemma 4.1, if all d+1 vertices of the ladder graph's input path do not hold pebbles, then some vertex on the input path must be repebbled in order to reach the output. Thus, 2d+1 pebbles are required to pebble the NLR-graph in optimal time. □

4.2. Pebbling with Minimum Space.

It is fairly straightforward to design a stack-based algorithm for evaluating recursively defined functions which operates in optimal time, and whose stack size grows in proportion to the depth of the associated binary tree. This type of algorithm is efficient with respect to space when the binary tree is balanced to some degree; in this case the stack size grows in proportion to the logarithm of the number of recursive calls made. However, in the worst case, the associated

binary tree will have depth growing in proportion to the tree's size (linear recursion for example), so the stack space of such an implementation will grow in proportion to the number of recursive calls made. It may be desirable to have a more space efficient implementation when such an extra storage requirement is prohibitive.

The following proposition derives an expression for the space requirement of an NLR–graph in terms of the space requirement of its associated binary tree (defined in Section 3.1). In this respect, it resembles a result due to Gurari and Ibarra [16], and shows that any recursively defined function can be evaluated with space proportional to the logarithm of the number of recursive calls made. However, it also gives an upper bound on the pebbling time associated with minimum space.

Proposition 4.3: For an NLR–graph whose associated binary tree has a space requirement of $S > 1$ pebbles, $S+1$ pebbles are necessary and sufficient to pebble the NLR–graph. Pebbling time proportional to the path length of the associated binary tree can always be achieved when $S+1$ pebbles are available.

Proof: To show that $S+1$ pebbles are necessary to reach the output of the NLR–graph, first note that pebbling its output is equivalent to pebbling the root of its associated binary tree. Consider the point in time when the last pebble-free path from an external vertex to the root of the binary tree is closed by placing a pebble on the external vertex. When this path is closed, there must be at least S pebbles on the tree; otherwise we could reach its root with less than S pebbles. Since $S > 1$, the binary tree is non-trivial, and we have not just pebbled its root. Hence, in order to reach the root, further pebble advancements are necessary which keep all paths from external vertices to the root blocked. If an extra pebble is not available, then an

advance cannot be made, since making such an advance requires pebbling a non-trivial chain which originates at the input to the NLR-graph and terminates at the vertex to be pebbled in the associated binary tree. Thus, S+1 pebbles are required to reach the output of the NLR-graph.

To show that S+1 pebbles are sufficient to reach the output of the NLR-graph, we exhibit a pebbling strategy which uses S+1 pebbles. Consider a pebbling strategy for reaching the root of the associated binary tree, and devote S of the available pebbles to this task. Before making each pebble placement on a vertex of the binary tree, we use the extra pebble at our disposal to pebble the chain beginning at the input to the NLR-graph and ending at the vertex of the NLR-graph which is "paired" with the vertex of the tree being pebbled. Note that this chain has length equal to the depth of the vertex being pebbled in the binary tree. Repeating this process for each pebble placement made in reaching the root of the tree yields a pebbling strategy which uses space S+1 (minimum space) and operates in time equal to the sum of the internal path length and the size of the associated binary tree. □

When minimum space is used to pebble an NLR-graph, time proportional to the path length of its associated binary tree may be necessary in the case of linear recursion (see Savage and Swamy [39]). This type of behavior can also be observed for strictly non-linear recursion, when the associated binary tree has a certain "bad" structure (see Figure 7 for such a binary tree). The main contribution here is to show that time proportional to path length does not always occur with minimum space implementations. In fact, time proportional to the size of the NLR-graph is achievable in certain cases.

To further study the pebbling time associated with minimum space, we introduce the Fibonaccian schema:

$$F(x) := \textbf{if } p(x) \textbf{ then } h(x) \textbf{ else } g(x, F(f(x)), F(f(f(x)))) \textbf{ fi};$$

If $p(f^{k-1}(a)) = p(f^k(a)) = \text{TRUE}$, then the NLR-graph associated with the evaluation of F at argument a will contain FT[k], the Fibonaccian tree of order k defined in Knuth [21]. We will refer to such an NLR-graph here as FNLR[k], the Fibonaccian NLR-graph of order k. Just as the Fibonacci tree of order k is comprised of Fibonacci trees of order k−1 and k−2, FNLR[k] contains the subgraphs FNLR[k−1] and FNLR[k−2]. An FNLR-graph of order 6 is shown in Figure 11.

By Proposition 4.3, the space requirement of an FNLR-graph depends on the space requirement of its associated Fibonacci tree, which is analyzed as follows.

Lemma 4.2: The Fibonacci tree FT[k] has a space requirement of $\lceil (k-1)/2 \rceil + 1$ pebbles.

Proof: By induction on k.

Basis: k=0 and k=1. FT[0] and FT[1] are each the trivial tree given by an external vertex. As in Section 3.3, we will assume each requires one pebble.

Induction: Assume that FT[k−1] and FT[k−2] have space requirements of $\lceil (k-2)/2 \rceil + 1$ and $\lceil (k-3)/2 \rceil + 1$ pebbles respectively. As in Section 3.1, the space requirement of FT[k] is given by the space requirements of its subtrees:

if S(FT[k−1]) = S(FT[k−2]) **then** S(FT[k−1])+1

else max{S(FT[k−1]),S(FT[k−2])};

When k is even, $S(FT[k-1]) = S(FT[k-2])$, so

$S(FT[k]) = \lceil (k-2)/2 \rceil + 2 = \lceil (k-1)/2 \rceil + 1.$

Otherwise,

$S(FT[k]) = \max\{\lceil (k-2)/2 \rceil + 1, \lceil (k-3)/2 \rceil + 1\}$

$= \lceil (k-2)/2 \rceil + 1 = \lceil (k-1)/2 \rceil + 1.$

Thus, in both cases, FT[k] has a space requirement of $\lceil (k-1)/2 \rceil + 1$ pebbles. □

Another property possessed by FNLR-graphs is that they can be pebbled in optimal time when available space is about double the minimum space requirement. This is easily shown with an argument similar to that in Proposition 4.2. Thus, a time-space product on the order of being optimal is achievable for FNLR-graphs. The following theorem demonstrates that this is also the case when minimum space is used to pebble an FNLR-graph.

Theorem 4.1: With minimum space, the Fibonaccian NLR-graph which has N vertices can be pebbled in time

$O(N) + O(N^{.7202} \log N) = O(N).$

Proof: Consider the task of pebbling FNLR[k] with $\lceil (k-1)/2 \rceil + 2$ pebbles which, by Lemma 4.2 and Proposition 4.3, is its minimum space requirement. Our pebbling strategy will proceed in what we will refer to as an e-order method: to pebble the output of FNLR[k], we first pebble the output of FNLR[k-1], hold a pebble there, and then pebble the output of FNLR[k-2]. Also, when enough pebbles are available, we will pebble FNLR-subgraphs in optimal time. Our claim is that there exists a pebbling strategy for FNLR[k] which uses minimum space of $\lceil (k-1)/2 \rceil + 2$ pebbles, and pebbles all FNLR-subgraphs whose output vertices have height $\lfloor (k-1)/2 \rfloor$ in the embedded Fibonacci tree in optimal time. The ***height*** of a

vertex in a binary tree is the number of edges in the unique path joining it to the root of the tree. FNLR–subgraphs in FNLR[k] whose output vertices have height $\lfloor (k-1)/2 \rfloor$ will be referred to here as good FNLR–subgraphs. The proof of the claim is by induction on k.

Basis: k=1 and k=2. The output vertices of FNLR[1] and FNLR[2] have height 0, and each graph can be pebbled in optimal time when minimum space is available.

Induction: Assume that there exist pebbling strategies for FNLR[k−1] and FNLR[k−2], each of which uses minimum space, is an e-order method, and pebbles all good FNLR–subgraphs in FNLR[k−1] and FNLR[k−2], respectively, in optimal time.

Case 1: k odd. Here, the space requirement of FNLR[k] equals the space requirement of FNLR[k−1] and is given by $\lceil (k-1)/2 \rceil + 2$, while FNLR[k−2] has a space requirement of $\lceil (k-1)/2 \rceil + 1$. To pebble FNLR[k], we first pebble FNLR[k−1] using the inductive strategy which pebbles all good FNLR–subgraphs in FNLR[k−1] in optimal time. Leaving a pebble on the output of FNLR[k−1], we then pebble FNLR[k−2] using the inductive strategy which pebbles all good FNLR–subgraphs in FNLR[k−2] in optimal time. Note that when necessary, a pebbling of the input of FNLR[k−1] or FNLR[k−2] is achieved simply by first pebbling the input of FNLR[k]. Thus, FNLR–subgraphs whose outputs have height $\lfloor (k-2)/2 \rfloor$ and $\lfloor (k-3)/2 \rfloor$ in FNLR[k−1] and FNLR[k−2], respectively, were pebbled in optimal time. But, since k is odd, these FNLR–subgraphs have height $\lfloor (k-1)/2 \rfloor$ in FNLR[k], and we have produced a pebbling strategy with the desired properties.

Case 2: k even. In this case, the space requirement of FNLR[k] is $\lceil (k-1)/2 \rceil + 2$ pebbles, while the space requirements of FNLR[k−1] and

FNLR[k−2] are identical and are given by $\lceil (k-1)/2 \rceil + 1$ pebbles. To pebble FNLR[k], we begin by pebbling FNLR[k−1]. By induction, there exists a strategy which uses $\lceil (k-1)/2 \rceil + 1$ pebbles and pebbles all good FNLR−subgraphs in FNLR[k−1] in optimal time. With the extra pebble at our disposal, we can modify this strategy so that any FNLR−subgraph in FNLR[k−1] whose output has height $\lfloor (k-2)/2 \rfloor - 1$ is pebbled in optimal time. This is done by simply inserting a sequence of moves into the inductive strategy which places the extra pebble on the input to such an FNLR−subgraph before pebbling its two good FNLR−subgraphs. Since the inductive strategy is an e-order method, and the two good FNLR−subgraphs are pebbled in optimal time, it follows that the FNLR−subgraph whose output has height $\lfloor (k-2)/2 \rfloor - 1$ is pebbled in optimal time by the modified strategy (the modification also preserves the e-order property). After FNLR[k−1] has been pebbled using this modified strategy, we hold a pebble on its output and pebble FNLR[k−2] using the inductive strategy which requires $\lceil (k-1)/2 \rceil + 1$ pebbles and pebbles all good FNLR−subgraphs in FNLR[k−2] in optimal time. Thus, we have a pebbling strategy for FNLR[k] which is an e-order method and pebbles all FNLR−subgraphs whose outputs have height $\lfloor (k-2)/2 \rfloor - 1$ and $\lfloor (k-3)/2 \rfloor$ in FNLR[k−1] and FNLR[k−2], respectively, in optimal time. Since k is even, these FNLR−subgraphs have height $\lfloor (k-1)/2 \rfloor$ in FNLR[k]. The claim follows.

To prove the theorem, consider the number of extra moves made on vertices of FNLR[k] when minimum space is used. By the previous argument, there exists a pebbling strategy for which no extra moves are made on good FNLR−subgraphs. Also, no extra moves are made on the associated Fibonacci tree (output vertices of FNLR[k]). Hence, the only vertices in FNLR[k] on which extra moves are made are inputs of an

FNLR-subgraph whose output vertex has height (the length of the path joining it to the output of FNLR[k]) less than $\lfloor (k-1)/2 \rfloor$. The number of extra moves made on such a vertex is at most the number of good FNLR-subgraphs which can be reached from it, since the vertex is pebbled once when its "paired" output is pebbled, and possibly once each time the input of a reachable good FNLR-subgraph is pebbled (the input of a good FNLR-subgraph is pebbled only once). It is easily seen that there can be at most $2^{\lfloor (k-1)/2 \rfloor - j}$ good FNLR-subgraphs reachable from a vertex with associated height $0 \leq j < \lfloor (k-1)/2 \rfloor$, and there are at most 2^j such vertices. Thus, we can form the following upper bound on the number of extra moves necessary when pebbling FNLR[k] with $\lceil (k-1)/2 \rceil + 2$ pebbles, denoted here by E(k):

$$E(k) \leq \sum_{0 \leq j < \lfloor (k-1)/2 \rfloor} 2^j \cdot 2^{\lfloor (k-1)/2 \rfloor - j}$$

$$= \lfloor (k-1)/2 \rfloor \cdot 2^{\lfloor (k-1)/2 \rfloor}$$

$$\leq \frac{k}{2} \cdot 2^{k/2}.$$

Now, the number of vertices in FNLR[k] is given by:

N = 2·{number of vertices in the Fibonacci tree of order k}.
The Fibonacci tree of order k has F_{k+1} external vertices, where F_{k+1} is the k+1st Fibonacci number (Knuth vol. 1 pg. 78 eq.(2)), so

$$N = 2(2 \cdot F_{k+1} - 1)$$

$$\geq \frac{4\varphi^{k+1}}{\sqrt{5}} - 3 \quad \text{[Knuth vol. 1 pg. 82 eq.(15)]}.$$

Here, $\varphi = \frac{1}{2}(1 + \sqrt{5})$.

It follows that

$$k \leq \log_{\varphi}(N+3) + c = O(\log_2 N)$$

and

$$2^{k/2} \leq 2^{\frac{1}{2}\{\log_\varphi(N+3) + c\}} = O(2^{\frac{1}{2}\log_\varphi N})$$

$$= O(2^{.7202 \cdot \log_2 N}) = O(N^{.7202}).$$

Thus, the time necessary to pebble FNLR[k] is given by

$$N + E(k) = N + O(N^{.7202}\log_2 N) = O(N). \quad \square$$

While the result of Theorem 4.1 was established only for Fibonaccian NLR–graphs, it also holds true for any NLR–graph whose associated binary tree is balanced in the sense that the height of the left subtree of every vertex never differs by more than ±1 from the height of its right subtree (see Knuth [21]). This follows since Fibonaccian NLR–graphs typify the worst case behavior of an NLR–graph with such a structure. Thus, near optimal pebbling time can be achieved using minimum space for a fairly large class of NLR–graphs. However, minimum space implementations perform badly with respect to time in the worst case, so it would be of use to investigate the relationship between time and space in more detail. For instance, it may be possible to reduce space significantly from the upper bound given in Proposition 4.2, at the expense of an modest increase in computation time.

CHAPTER 5

EXTREME TIME-SPACE TRADEOFFS

Up to this point, we have examined a number of different relationships that can occur between resources in a straight-line algorithm. In this chapter, we consider another such relationship between the resources of time and space. We exhibit graphs which possess an "extreme" time-space tradeoff in the pebble game model of a straight-line algorithm. This type of tradeoff is characterized by pebbling time which is superpolynomial in the size (also in the number of inputs) of the graphs, when space is restricted too severely. By studying these graphs, we will be able to demonstrate that oblivious sorting based on bitonic merging is a "natural" problem which exhibits this type of tradeoff. Tompa [43] has independently obtained this result and a similar one for computing the transitive closure of a boolean matrix using successive squaring. The results we state also exhibit infinite families of constructible graphs where superpolynomial pebbling time is associated with large ranges of space, and we show that the lower limit on space can grow as any slowly increasing function of the graph's size.

The straight-line algorithms for sorting and merging which we consider are implemented using the operations Min(x,y) and Max(x,y). Here, x and y are elements of the list to be operated on, which is drawn from some total ordering. An example of such an algorithm is Batcher's bitonic merge-sort [2], where sorting is performed by merging a bitonic sequence made up of two previously sorted subsequences.

5.1. An Extreme Tradeoff for Bitonic Sorting.

In this section, we obtain an extreme time-space tradeoff for oblivious sorting based on bitonic merging by deriving such a tradeoff for its associated graph family. The graphs which we construct are built from subgraphs which themselves are hard to pebble, in that placing pebbles on a relatively small set of outputs requires pebbling a relatively large set of inputs. One example of such a subgraph which we have already studied is the FFT graph (see Section 2.1). We can construct the graph S[d], which is central to our results, from FFT graphs in the following way.

Definition: The graph S[d] on $n=2^d$ inputs contains d "levels" of subgraphs, where level j consists of 2^{d-j} disjoint FFT graphs on 2^j inputs each. These levels are connected by associating outputs at level j with inputs at level j+1 ($0 \leq j \leq d-1$).

The graph S[d] is shown in Figure 12. An equivalent definition for S[d] can be given in terms of two inductive subgraphs, $S_0[d-1]$ and $S_1[d-1]$ (disjoint from each other), whose outputs are associated with the inputs of an FFT graph on 2^d inputs.

The fact that an FFT graph is relatively hard to pebble leads to an extreme time-space tradeoff for the graph S[d].

Lemma 5.1: Pebbling a set of 2S outputs of the graph S[d] requires at least $2^{(d-j-2)(d-j-3)/2}$ moves.

Proof: By induction on d.

Basis: d=j+3. When at most S pebbles are available and $S < 2^j$, pebbling a set of 2S outputs of S[j+3] requires pebbling 2S outputs of an FFT graph on

2^{j+3} inputs, which by Lemma 2.1 requires pebbling at least 2^{j+2} inputs of the FFT graph. Thus,

$T \geq 2^{j+2} \geq 2^{(j+3-j-2)(j+3-j-3)/2} = 2^0 = 1.$

Induction: Assume pebbling a set of 2S outputs of S[d−1] for $d-1 \geq j+3$ takes time

$T \geq 2^{(d-j-3)(d-j-4)/2}$

when $S < 2^j$ pebbles are available. Then, since $d-1 > j$, pebbling a set of 2S outputs of S[d] requires pebbling at least 2^{d-1} inputs to the FFT graph at level d, which are in turn outputs of two distinct copies of S[d−1]. Hence, we must pebble Z_1 outputs of one copy of S[d−1] and Z_2 outputs of the other copy of S[d−1], where $Z_1 + Z_2 \geq 2^{d-1}$. The Z_1 outputs of one copy of S[d−1] can be split into at least $\left\lfloor \frac{Z_1}{2S} \right\rfloor$ groups of 2S outputs each. This can also be done for the Z_2 outputs of the other copy of S[d−1]. By our inductive hypothesis, we form the lower bound:

$$T \geq \left\lfloor \frac{Z_1}{2S} \right\rfloor \cdot 2^{(d-j-3)(d-j-4)/2} + \left\lfloor \frac{Z_2}{2S} \right\rfloor \cdot 2^{(d-j-3)(d-j-4)/2}$$

$$\geq \left(\frac{Z_1}{2S} + \frac{Z_2}{2S} - 2\right) \cdot 2^{(d-j-3)(d-j-4)/2}$$

$$\geq \left(\frac{2^{d-1} - 4S}{2S}\right) \cdot 2^{(d-j-3)(d-j-4)/2}.$$

Since S was chosen so that $S < 2^j$, we have $2^m - 4S > 2^{m-1}$ for $m \geq j+3$. Thus,

$$T \geq \left(\frac{2^{d-2}}{2S}\right) \cdot 2^{(d-j-3)(d-j-4)/2}$$

$$\geq 2^{d-j-3} 2^{(d-j-3)(d-j-4)/2}$$

$$= 2^{(d-j-2)(d-j-3)/2}. \quad \square$$

Theorem 5.1: The graph S[d], on $n=2^d$ inputs, requires pebbling time

$$T \geq 2^{\frac{1}{2}(\log(\frac{n}{8S}))^2},$$

when $S < n/8$ pebbles are available. Pebbling time is superpolynomial in n for space restricted to be on the order of $n^{1-\varepsilon}$, where $0 < \varepsilon < 1$ is a fixed real constant.

Proof: Assume that space S is chosen in the range $2^{j-1} \leq S < 2^j$ where $j+3 \leq d$. Pebbling the entire set of outputs of S[d] requires pebbling sets of 2S outputs, thus by Lemma 5.1,

$$T \geq \left\lfloor \frac{2^d}{2S} \right\rfloor \cdot 2^{(d-j-2)(d-j-3)/2}$$

$$\geq 2^{d-j-2} 2^{(d-j-2)(d-j-3)/2}$$

$$= 2^{(d-j-1)(d-j-2)/2}$$

$$\geq 2^{\frac{1}{2}(\log(\frac{n}{8S}))^2}.$$

Choosing $S = n^{1-\varepsilon}$ obviously implies that T is superpolynomial in n, the number of inputs to the graph. □

A statement equivalent to Theorem 5.1 holds even when n, the number of inputs to the graph is not a power of two. In this case, the graphs considered are formed from an FFT graph on n inputs, and inductive subgraphs on $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ inputs each. Thus, we can relate the result of Theorem 5.1 to the problem of oblivious sorting based on a bitonic merge procedure as follows.

Corollary 5.1: For n input oblivious sorting performed by a bitonic merge on two previously sorted subsequences, time is superpolynomial in n when space is restricted to be on the order of $n^{1-\varepsilon}$ ($0 < \varepsilon < 1$ is a fixed real constant).

Proof: Without loss of generality, we consider the case when n is a power of two. When the vertices of the FFT graph are given the right interpretation, the graph implements a bitonic merging network (see Batcher [2]). Thus, the graph S[d] can be used to implement an $n=2^d$ oblivious sorting algorithm based on a recursive bitonic merge. Associated with this algorithm is the extreme time-space tradeoff of Theorem 5.1. □

Tompa [43] has derived the above result independently for oblivious sorting based on merging independent of the merge subprocedure used. His argument is basically the same as that used in Theorem 5.1, where graphs associated with a merge subprocedure possess a grate-like property (see Section 2.1) which forces many repebblings of their inputs when space is restricted.

5.2. Large Ranges of Space.

Previous results concerning extreme time-space tradeoffs have exhibited families of graphs for which superpolynomial pebbling time is associated with a large range of space. This work has been motivated primarily from a theoretical standpoint, with the underlying assumption that the graphs studied typify the worst-case behavior of a straight-line algorithm. In this section, we relate Theorem 5.1 to these results, and extend the range of space over which superpolynomial pebbling time is observed.

In order to establish this relationship, we must express the statement of Theorem 5.1 in terms of the size of the graph S[d]. Level j of S[d] is composed of 2^{d-j} FFT graphs on 2^j inputs each, so the size of S[d], denoted by N, satisfies:

$$N = \sum_{1 \le j \le d} (j+1) \cdot 2^d = \Theta(n(\log n)^2)$$

and

$$\log N = \Theta(\log n).$$

Thus, superpolynomial pebbling time occurs for space in the range:

$$\Theta[(\log N)^2, (\frac{N}{(\log N)^2})^{1-\varepsilon}].$$

Note here that the minimum space necessary to pebble S[d] is proportional to its depth, which is $\Theta((\log N)^2)$.

We can extend the range of space where superpolynomial pebbling time occurs for the graph S[d] by considering more "compact" graphs. These graphs will have asymptotically smaller size, along with the same pebbling behavior as S[d].

Definition: The graph G[d] on $n=2^d$ inputs contains d "levels", where level j consists of the subgraph SG[j] on 2^j inputs and outputs. These levels are connected by associating the outputs of SG[j] at level j with the left hand 2^j inputs of SG[j+1] ($0 \le j \le d-1$).

The graph G[d] is shown in Figure 13. To generate an extreme time-space tradeoff for G[d], we must choose the subgraphs SG[j] so that pebbling a set of outputs (with S pebbles on the subgraph) requires repebbling many "left-hand side" inputs of SG[j]. One type of graph which possesses this property is a superconcentrator, defined in Section 3.3. If SG[j] is a 2^j–superconcentrator, then pebbling any set of S+1 outputs requires repebbling at least $2^j - S$ inputs of SG[j] when S pebbles are available (see Tompa [44]). Thus, at least $2^{j-1} - S$ inputs of SG[j] on the left hand side must be repebbled when $S \le 2^{j-1}$. Using an argument almost identical to that in Theorem 5.1, we can show that G[d] requires superpolynomial pebbling time when space is on the order of $n^{1-\varepsilon}$.

Valiant [46] has shown that back-to-back FFT graphs form a superconcentrator (this fact also follows from Theorem 2.2). Thus, when SG[j] is chosen to be a back-to-back FFT graph on 2^j inputs, we have the following:

Proposition 5.1: For the graph G[d] constructed from back-to-back FFT subgraphs, superpolynomial pebbling time occurs for space in the range:

$$\Theta[(\log N)^2, (\frac{N}{\log N})^{1-\varepsilon}].$$

Proof: The size of G[d], when subgraphs are chosen in this manner, satisfies:

$$N = \sum_{1 \le j \le d} (2j+2) \cdot 2^j = \Theta(d \cdot 2^d) = \Theta(n \cdot \log n)$$

where n is the number of inputs to G[d]. Thus, $\log N = \Theta(\log n)$, and time grows as $2^{\Omega((\log N)^2)}$ for space in the range $\Theta[(\log N)^2, (\frac{N}{\log N})^{1-\varepsilon}]$. □

As Valiant [45], Pippenger [32], and Gabber and Galil [12] have demonstrated, we can also choose the subgraph SG[j] to be a superconcentrator with size linear in the number of inputs. This gives us:

Proposition 5.2: For the graph G[d] constructed from linear superconcentrators, superpolynomial pebbling time occurs for space in the range:

$$\Theta[(\log N)^2, N^{1-\varepsilon}].$$

Proof: The size of G[d], in this case, is given by:

$$N = \sum_{1 \le j \le d} c \cdot 2^j = \Theta(2^d) = \Theta(n)$$

so

$N^{1-\varepsilon} = \Theta(n^{1-\varepsilon})$. □

5.3. Lower Limits on Space.

The upper bounds on space given in Propositions 5.1 and 5.2 compare favorably with the bounds established in Lengauer and Tarjan [23], which were discussed in Section 1.2. However, it is also of theoretical interest to investigate the lower limits on space which can be associated with superpolynomial pebbling time, since this may point to very large ranges of space over which pebbling time is superpolynomial. Research concerning extreme time-space tradeoffs has considered a variety of different graphs families, all of whose members have a minimum space requirement growing at least as fast as the logarithm of the graph's size. This section improves upon this lower limit to show the existence of graph families exhibiting extreme tradeoffs whose minimum space requirement grows as $\omega(1)$, any slowly increasing function of the graph's size. Much of this work originally was reported by Carlson and Savage [7], and a summary of how these results compare with previous research is given in Figure 14.

We are now in a position to describe a family of directed acyclic graphs on N vertices whose main property is that any pebbling strategy using less than a predetermined amount of space requires a number of moves which is superpolynomial in the size of the graph. Basic to all graphs discussed in this paper is the pyramidal graph introduced in Section 1.2 (see Figure 2). A pyramidal graph P_k on k inputs has $k(k+1)/2$ vertices, and requires k pebbles to pebble the output. Also, the output can be pebbled using k pebbles without repebbling any vertex.

The family of graphs H_k^c $\{k=1,2,...\}$ is defined inductively in the variable k for any prechosen integral constant $c \geq 1$, where c is the number of outputs of all graphs in the family. The basis graph H_1^c is simply one input with fan-out c to its

c outputs. In Figure 15, H_1^2 is diagramed as part of H_2^2. The graph H_k^c is composed of the subgraph H_{k-1}^c flanked on both sides by c copies of the pyramidal graph P_{k-1}. These subgraphs are connected to c spines (output paths) which lead to the outputs of H_k^c. A spine is comprised of k+1 sections of vertices, where each section consists of 3c vertices with edges directed in from all outputs of the subgraphs (H_k^2 is shown in Figure 16). A precise definition of the graph H_k^c is given as follows.

Definition: The directed graph $H_k^c = (V[H_k^c], E[H_k^c], O[H_k^c])$ with c outputs has vertex set $V[H_k^c]$, edge set $E[H_k^c]$, and output vertex set $O[H_k^c] = \{x_k^1, \ldots, x_k^c\}$. It is defined recursively in terms of the graph $H_{k-1}^c = (V[H_{k-1}^c], E[H_{k-1}^c], O[H_{k-1}^c])$ with c outputs as follows. Let P_{k-1}^i $(1 \leq i \leq 2c)$ be 2c distinct copies of a pyramidal graph on k−1 inputs whose output vertices are denoted by p^i. Let S be a set of vertices disjoint from H_{k-1}^c and P_{k-1}^i, where

$$S = \bigcup_{j=1}^{c} S^j \text{ and } S^j = \bigcup_{m=1}^{k+1} \{v_{m,1}^j, \ldots, v_{m,3c}^j\}.$$

Here, S^j denotes a spine composed of k+1 sections. Then,

$$V[H_k^c] = V[H_{k-1}^c] \cup S \cup \bigcup_{i=1}^{2c} V[P_{k-1}^i]$$

$$O[H_k^c] = \{v_{k+1,3c}^1, \ldots, v_{k+1,3c}^c\}$$

and

$$E[H_k^c] = E[H_{k-1}^c] \cup \bigcup_{i=1}^{2c} E[P_{k-1}^i] \cup \Delta$$

where

$$\Delta = \bigcup_{j=1}^{c} \bigcup_{m=1}^{k+1} \{(v_{m,q}^j, v_{m,q+1}^j) \mid 1 \leq q < 3c\} \cup \bigcup_{j=1}^{c} \bigcup_{m=1}^{k} (v_{m,3c}^j, v_{m+1,1}^j)$$

$$\cup \bigcup_{j=1}^{c} \bigcup_{m=1}^{k+1} \{(p^q, v_{m,q}^j), (x_{k-1}^q, v_{m,q+c}^j), (p^{q+c}, v_{m,q+2c}^j) \mid 1 \leq q \leq c\}.$$

Here, (v,w) denotes an edge directed from v to w. The basis graph H_1^c is defined to have input vertex x, output set $\{x_1^1, \ldots, x_1^c\}$, and edge set $\{(x, x_1^q) \mid 1 \leq q \leq c\}$.

Important characteristics of the graph family $\{H_k^c\}$ include the size of the graphs and the minimum space needed to pebble them.

Proposition 5.3: The graph H_k^c has the following properties:

(a) $\text{size}(H_k^c) = \Theta(c^2k^2 + c \cdot k^3)$.

(b) the number of inputs of H_k^c, $I(H_k^c) = \Theta(c \cdot k^2)$.

(c) the minimum space necessary to pebble all outputs of H_k^c is $S_{min}(H_k^c) = k$.

Proof:

(a) By the construction of H_k^c, its size satisfies the recurrence:

$$\begin{aligned}\text{size}(H_k^c) &= 2c \cdot \text{size}(P_{k-1}) + 3c^2(k+1) + \text{size}(H_{k-1}^c) \\ &= \Theta(c^2k^2 + c \cdot k^3).\end{aligned}$$

(b) The number of inputs of H_k^c satisfies the recurrence:

$$\begin{aligned}I(H_k^c) &= 2c(k-1) + I(H_{k-1}^c) \\ &= \Theta(c \cdot k^2).\end{aligned}$$

(c) By induction on k:

Basis: H_1^c requires one pebble to reach any output and hence $S_{min}(H_1^c) = 1$.

Induction: Assume that $k-1$ pebbles are necessary and sufficient to pebble the outputs of H_{k-1}^c in consecutive order. To pebble the outputs of H_k^c, we proceed one at a time, moving a pebble down the spine of the associated output. Since an optimal pebbling strategy has one pebble marching down the spine, we have $k-1$ pebbles available to pebble the subgraphs P_{k-1} and H_{k-1}^c. Note that pebbling pyramidal graphs leaves H_{k-1}^c entirely devoid of pebbles, but we always have $k-1$ pebbles at our disposal to repebble it. Thus, by induction, we are able to pebble any output of H_k^c with k pebbles, so we can pebble all of its outputs in consecutive order with k pebbles.

Note that if less than k pebbles are available, it is impossible to reach any output of H_k^c. □

At this point, we also note that with only $S_{min}(H_k^c)$ pebbles available, the graph H_k^c is very difficult to pebble, since each time an output is pebbled, the subgraph H_{k-1}^c must be repebbled in its entirety k times. This leads to a recurrence of the form:

$$T(H_k^c) \geq c \cdot k \cdot T(H_{k-1}^c) \geq \cdots \geq c^k k!$$

We are interested in the behavior of the graph family H_k^c with restricted extra space, and the following theorem establishes that superpolynomial time (in the size of the graph) is required when extra space is no more than c−1, for any fixed constant c.

Theorem 5.2: To pebble all outputs of the graph H_k^c, starting from an initial configuration of no more than c−1 pebbles on the graph, and using available space of no more than $S_{min}(H_k^c)$ + c−1 at any time, requires a number of moves $T(H_k^c, c-1) \geq k!$.

Proof: By induction on k:

Basis: k=1.

To pebble H_1^c with no more than c−1 pebbles initially on the graph requires at least 2 moves (the best case is when the input vertex and c−2 output vertices are initially covered). Thus $T(H_1^c, c-1) \geq 2 \geq 1$ for any c.

Induction: Assume true for k−1.

If we start with an initial configuration of no more than c−1 pebbles on H_k^c, then at least one spine is devoid of a pebble, and therefore must be pebbled in order to pebble all outputs of H_k^c. By the construction of H_k^c, pebbling a set of spines entails pebbling a series of k+1 sections. For each section,

except possibly the first, we find that one pebble is devoted to the spine being pebbled and a pyramidal graph P_{k-1} must be pebbled in order to advance through the section, because we cannot hold pebbles on all pyramidal outputs. Thus, when pebbling the section, at some point we are faced with pebbling the subgraph H^c_{k-1} with no more than $c-1$ pebbles initially on it and with available space of no more than $S_{min}(H^c_{k-1}) + c-1$. This process is repeated at least k times, so by induction, we can form the recurrence:

$T(H^c_k, c-1) \geq k \cdot T(H^c_{k-1}, c-1) \geq \cdots \geq k!$ □

Thus, it can be seen that extra space $c-1$ is accompanied by a superpolynomial blow-up in pebbling time. We can also show that increasing space past a certain level results in pebbling time polynomial in the size of the graph H^c_k.

Theorem 5.3: The graph H^c_k can be pebbled in $\Theta(c^2k^2 + c \cdot k^3)$ moves if the space available is at least $S_{min}(H^c_k) + c \cdot k = (c+1)k$.

Proof: To pebble H^c_k with $S_{min}(H^c_k) + c \cdot k$ pebbles, we first pebble the c outputs of H^c_{k-1} using $S_{min}(H^c_{k-1}) + c(k-1)$ pebbles. Each time a pebble is placed on an output of H^c_{k-1}, we use one of the $c+1$ remaining pebbles to cover it. When this process is finished, all the outputs of H^c_{k-1} are covered and we have $c(k-1) + k$ pebbles left to work with. Assuming that $k \geq 3$, we have at least $2c + k$ pebbles at this point, so we can pebble the subgraphs P_{k-1}, leaving pebbles on their output vertices. With the k pebbles remaining we pebble the spines and outputs of H^c_k. Thus, for $k \geq 3$, we have the recurrence:

$T(H^c_k, c \cdot k) \leq T(H^c_{k-1}, c \cdot k) + 2c \cdot T(P_{k-1}) + 3c^2(k+1)$.

The graph H^c_3 can easily be pebbled in time $T(H^c_3, 3c) = size(H^c_3) + d$, where d

is some constant. Telescoping the above recurrence yields:

$T(H_k^c, c \cdot k) \leq \Theta(c^2k^2 + c \cdot k^3)$. □

For the graph families $\{H_k^c\}$, we can think of each constant $c \geq 1$ as generating an associated family. Diagonalizing over this "grid" of graphs yields some interesting results.

Theorem 5.4: There exists a family of graphs $\{H_k\}$ such that if the amount of available space lies in the interval

$$\Theta[\log N, \frac{\sqrt{N}}{\log N}]$$

then the number of moves necessary to pebble H_k is superpolynomial in N, the size of the graph H_k.

Proof: For the graph families $\{H_k^c\}$ defined earlier, for a fixed value of k (k=1,2,...) we choose $c=\exp(\alpha k)$. By Proposition 5.3(a), we have $N = \Theta(c^2k^2 + c \cdot k^3)$ so that $k = \Theta(\log N)$ and $c = \Theta(\frac{\sqrt{N}}{\log N})$. By Theorem 5.2, when space is in the range

$\Theta[\log N, \frac{\sqrt{N}}{\log N}]$

we obtain

$T(H_k) \geq k! = \Theta((\log N)^{\lambda \log N}) = \Theta(N^{\lambda \log\log N})$

for some constant $\lambda > 0$, which is superpolynomial in N. □

The result of Theorem 5.4 extends the lower limit of space associated with superpolynomial pebbling time to be on the order of the logarithm of the graph's size. However, to further analyze the minimum space requirements associated with the graph family $\{H_k^c\}$, we must provide a more general lower bound on pebbling time with restricted extra space.

Theorem 5.5: To pebble H_k^c, starting with an initial configuration of no more than e ($0 \leq e \leq c$) pebbles on the graph, using a total of no more than $S_{min}(H_k^c) + e$ pebbles, requires a number of moves satisfying $T(H_k^c, e) \geq (\frac{c-e}{e+1})^k$.

Proof: First, we discuss some properties of an optimal pebbling strategy for H_k^c when at most e extra pebbles are available:

(a) An optimal strategy has a set of spines pebbled together, with one pebble moving down each spine until the output is reached. Each spine requires at most one pebble, and it is obviously non-optimal to remove this pebble before the associated output has been reached.

(b) An optimal strategy exists in which the outputs of the subgraphs H_{k-1}^c and P_{k-1} are pebbled one at a time, driving one or more "platoons" of pebbles down a set of spines in unison, where a platoon consists of a set of pebbles at the same level on a set of spines. If more than one platoon is active at a time, then we will see below that they can be considered to be one large platoon for the purposes of our analysis.

Let r_i be the number of pebbles in the ith platoon where $0 \leq i \leq p$. (Note that $1 \leq r_i \leq e+1$ for all i). Each time a platoon of size r_i is advanced across a section of several spines, except possibility the first section, the pebbling of pyramidal graphs forces the repebbling of the subgraph H_{k-1}^c with an initial configuration of no more than $e-r_i+1$ pebbles using no more than $e-r_i+1$ extra pebbles at any time (since it is non-optimal to remove pebbles associated with the platoon).

If two of more platoons are on the spines simultaneously, we can treat them as one platoon as follows. At the last point in time before the lead platoon crosses from the first section into the second, let r_0^* be the number of

pebbles on the board and consider them to constitute one new platoon. Before the leading pebbles in this platoon have left the end of the second section, by the above argument H^c_{k-1} will have been repebbled from an initial configuration of $e-r_0^*+1$ pebbles using at most this many extra pebbles. The next platoon will begin with pebbles that enter the first section as the first platoon moves on. It follows that

$$T(H^c_k,e) \geq T(H^c_{k-1},e-r_0^*+1) + \cdots + T(H^c_{k-1},e-r_p^*+1)$$

where

$$\sum_{0\leq i\leq p} r_i^* \geq c-e$$

since at most e of the c outputs initially carry pebbles. It should be clear that the above expression holds when each spine has two or more sections. However, the result of Theorem 5.4 depends on the fact that the graph has k sections.

We claim that $T(H^c_k,e) \geq (\frac{c-e}{e+1})^k$. The proof is by induction on k.

Basis: k=1.

$$T(H^c_1,e) \geq c-e \geq \frac{c-e}{e+1}.$$

Induction: Assume true for k−1. In general,

$$T(H^c_k,e) \geq m_0 \cdot T(H^c_{k-1},0) + \cdots + m_e \cdot T(H^c_{k-1},e)$$

where m_i is the number of terms in which $e-r_j^*+1 = i$. Then

$$T(H^c_k,e) \geq (m_0 + \cdots + m_e) \cdot T(H^c_{k-1},e)$$

since

$$T(H^c_{k-1},e) \leq T(H^c_{k-1},q) \text{ for } q \leq e.$$

Now,

$$(e+1)\cdot m_0 + e\cdot m_1 + \cdots + m_e = \sum_{0\leq i\leq p} r_i^* \geq c-e$$

and since the left-hand side is no more than $(e+1)\cdot(m_0 + \cdots + m_e)$, we have:

$$T(H_k^c,e) \geq (\frac{c-e}{e+1})\cdot T(H_{k-1}^c,e)$$

$$\geq (\frac{c-e}{e+1})^k \quad \text{by induction on k.} \ \square$$

Corollary 5.2: Any strategy for pebbling the graph H_k^c with $S \leq c^{1-\varepsilon}-1+k$ pebbles requires at least $(c^\varepsilon - 1)^k$ moves, which is superpolynomial in the size of the graph for any constant c. Here, ε is a real constant chosen in the interval (0,1).

With the above result in mind, we can again use a diagonalization procedure to produce graph families with extreme time-space exchanges.

Theorem 5.6: There exist graph families $\{H_k\}$ on N vertices with the property that the number of moves necessary to pebble a member of the family $\{H_k\}$ is superpolynomial in N for space in the interval:

$$\Theta[\sqrt{\log N}, (\sqrt{\frac{N}{\log N}})^{1-\varepsilon}].$$

Proof: For the graph families $\{H_k^c\}$ previously discussed, for fixed values of k, we find a graph in the grid with $c = \exp(\alpha k^2)$. This gives us

$$k = \Theta(\sqrt{\log N}), \quad c = \Theta(\sqrt{\frac{N}{\log N}}).$$

Hence, for space in the range

$$\Theta[\sqrt{\log N}, (\sqrt{\frac{N}{\log N}})^{1-\varepsilon}]$$

we have

$$T(H_k) \geq (c^\varepsilon-1)^k = \Theta((\sqrt{\frac{N}{\log N}})^{\varepsilon\sqrt{\log N}})$$

which is superpolynomial in N for fixed ε. $\square$

This result is not as strong as that shown in Theorem 5.4, however we can diagonalize in a different manner to yield results concerning the minimum space requirements for graph families with extreme time-space exchanges.

Theorem 5.7: There exist graph families $\{H_k\}$ on N vertices with the property that the number of moves necessary to pebble a member of the family $\{H_k\}$ is superpolynomial in N for space in the interval:

$$\Theta[\omega(1), (\frac{\sqrt{N}}{\omega(1)})^{1-\varepsilon}]$$

and the minimum space necessary to pebble H_k is $S_{min}(H_k) = \omega(1)$. Here, $\omega(1)$ is any slowly increasing function of N.

Proof: Earlier, we saw that $N = \Theta(c^2k^2 + c \cdot k^3)$ for the family of graphs $\{H_k^c\}$. Choosing $k = f(N)$ $(k=1,2,...)$ where $f(N)$ is sufficiently small forces $c = \Theta(\frac{\sqrt{N}}{f(N)})$. Thus, for space in the interval

$\Theta[f(N), (\frac{\sqrt{N}}{f(N)})^{1-\varepsilon}]$

we have

$T \geq \Theta((\frac{\sqrt{N}}{f(N)})^{\varepsilon f(N)})$

by Corollary 5.2.

We can conclude that the number moves required is superpolynomial in N when $f(N)$ is $\omega(1)$. □

CONCLUSIONS

Investigations of the simultaneous resource requirements of a computational problem can lead to new and meaningful insights concerning its complexity, and can provide more efficient implementations when the use of a critical resource must be reduced. This dissertation has explored tradeoffs which relate the resources of execution time, storage space, and algorithm size for various problems. The analysis tool we have used was the pebble game, played on directed acyclic graphs. With it, we were able to simulate the allocation of registers which takes place in a straight-line algorithm.

In Chapter 2, we examined the graph associated with back-to-back applications of a Fast Fourier transform, and derived a lower bound of the form $T \cdot S = \Omega(n^2 \log n)$ for such a graph containing n inputs. The back-to-back FFT graph has applications to "fast" polynomial multiplication and to the construction of permutation graphs. When n-degree polynomial multiplication is implemented with this type of structure, we are able to conclude that space on the order of n is necessary to achieve time on the order of $n \cdot \log n$.

Chapter 3 introduced the concept of a tradeoff between algorithm size and storage space. We considered the problems of oblivious merging and pattern matching, and showed that it is impossible to construct graphical representations of these problems having optimal size when the space restriction in effect is too severe. Specifically, if available space is restricted to grow asymptotically as $o(\log n)$, then merging and pattern matching networks must have size growing asymptotically as $\omega(n \cdot \log n)$. We also investigated the tradeoff between size and space for graphs possessing concentration properties.

In Chapter 4, we discussed the relationship between time and space for stack-based implementations of a non-linear recursive function. We presented a

graph model for the computational process involved, and gave pebbling strategies optimal with respect to either time or space. We also demonstrated that, in certain cases, near optimal time can be achieved by minimum space implementations.

Chapter 5 explored "extreme" tradeoffs between time and space in the pebble game, which are characterized by superpolynomial pebbling time when there is a certain space restriction. We derived an extreme time-space tradeoff for an oblivious sorting algorithm based on bitonic merging, and also considered the range of space which can be associated with superpolynomial pebbling time. For certain graph families possessing an extreme tradeoff, we demonstrated that the lower limit of space can be reduced to grow as any slowly increasing function of the graph's size.

Open Problems.

In closing, we present a list of questions that this dissertation has left as unresolved:

(1) For the back-to-back FFT graph, is there a pebbling strategy which achieves the lower bound of Corollary 2.1 over a large range of space?

(2) Do permutation graphs have an asymptotically stronger tradeoff (in terms of the product of time and space) than superconcentrators?

(3) Can Theorem 3.4 be extended to provide a more general tradeoff between size and space for superconcentrators?

(4) Are there other problems or graph families which exhibit size-space tradeoffs?

(5) For non-linear recursion, is there a more general time-space tradeoff resembling that of Savage and Swamy [39], which allows space to be reduced significantly at the expense of a modest increase in time?

(6) Are there problems for which all solutions exhibit an extreme time-space tradeoff?

(7) Can the range of space presented in Theorem 5.7 be extended in either direction? Is the lower bound on space of $\omega(1)$ the best possible?

BIBLIOGRAPHY

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley (1974).

2. K. E. Batcher, "Sorting Networks and Their Applications," *AFIPS Proc. Spring Joint Comput. Conf.* **32** pp. 307-314 (1968).

3. B. Beizer, "The Analysis and Synthesis of Signal Switching Networks," *Proc. Symp. on Math. Theory of Automata*, pp. 113-135 (1962).

4. V. E. Benes, "," in *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, N.Y. (1965).

5. P. van Emde Boas and J. van Leeuwen, "Move Rules and Trade-Offs in the Pebble Game," Report RUU-CS-78-4, Vakgroep Informatica, U. Utrecht, Utrecht, Netherlands (April/August 1978).

6. A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier, New York, N.Y. (1975).

7. D. Carlson and J. E. Savage, "Graph Pebbling with Many Free Pebbles Can be Difficult," *12th Ann. ACM Symp. on Th. Computing*, pp. 326-332 (April 28-30, 1980).

8. A. K. Chandra, "Efficient Compilation of Linear Recursive Programs," *Proc. 14th Ann. Symp. on Switching and Aut. Th.*, pp. 16-25 (Oct. 15-17, 1973).

9. S. A. Cook, "An Observation on Time-Storage Tradeoff," *Proc. 5th Ann. Symp. Th. Comp.*, pp. 29-33 (May 1973).

10. A. P. Ershov, "On Programming Arithmetic Operations," *CACM* **1** pp. 3-6 (1958).

11. W. Feller, *An Introduction to Probability Theory and its Applications*, John Wiley and Sons, New York, NY. (1966).

12. O. Gabber and Z. Galil, "Explicit Construction of Linear Sized Concentrators and Superconcentrators," *Procs. 20th Ann. Symp. on Foundations of Comp. Sci.*, pp. 364-370 (Oct. 29-31, 1979).

13. R. G. Gallager, *Information Theory and Reliable Communication*, John Wiley and Sons, Inc. (1968).

14. J. R. Gilbert, T. Lengauer, and R. E. Tarjan, "The Pebbling Problem is Complete in Polynomial Space," *Procs. 11th Ann. Symp. Th. Comp.*, pp. 237-248 (April 1979).

15. D. Yu. Grigoryev, "An Application of Separability and Independence Notions for Proving Lower Bounds of Circuit Complexity," *Notes of Scientific Seminars, Steklov Math. Inst.* **60** pp. 35-48 (1976).

16. E. M. Gurari and O. H. Ibarra, "On the Space Complexity of Recursive Algorithms," *Information Processing Letters* **8**(5) pp. 267-271 (June 11, 1979).

17. J. E. Hopcroft, W. J. Paul, and L. G. Valiant, "On Time Versus Space," *JACM* **24**(2) pp. 332-337 (April 1977).

18. J. Ja'Ja', "Time-Space Tradeoffs for Some Algebraic Problems," *12th Ann. ACM Symp. on Th. Computing*, pp. 339-350 (April 28-30, 1980).

19. A. E. Joel, "On Permutation Switching Networks," *Bell Syst. Tech. J.* **47** pp. 813-822 (1968).

20. D. E. Knuth, *The Art of Computer Programming - Fundamental Algorithms*, Addison-Wesley, Reading, Mass. (1973).

21. D.E. Knuth, *The Art of Computer Programming - Sorting and Searching*, Addison-Wesley, Reading, Mass. (1973).

22. E. A. Lamagna and J. E. Savage, "Combinational Complexity of Some Monotone Functions," *Procs. 15th Ann. Symp. Switching and Aut. Th.*, pp. 140-144 (October 1974).

23. T. Lengauer and R. E. Tarjan, "Upper and Lower Bounds on Time-Space Tradeoffs," *Proc. Eleventh Ann. Symp. Th. Comp.*, pp. 262-277 (April 1979).

24. A. Lingas, "A PSPACE- complete Problem Related to a Pebble Game," pp. 300-321 in *Automata Languages and Programming*, ed. C. Boehm, Springer Lecture Notes in Computer Science (1978).

25. M. C. Loui, "Minimun Register Allocation is Complete in Polynomial Space," Report TM-128, Lab. for Comp. Sci., M.I.T., Cambridge, MA (March 1979).

26. M. C. Loui, *A Note on the Pebble Game*, Lab for Computer Science, MIT (February 1980).

27. I. Nakata, "On Compiling Algorithms for Arithmetic Expressions," *CACM* **10** pp. 492-494 (1967).

28. M. S. Paterson and C. E. Hewitt, "Comparative Schematology," *Proc. Proj. MAC Conf. on Concurrent Systems and Parallel Computation*, pp. 119-127 (June 1970).

29. W. J. Paul, R. E. Tarjan, and J. R. Celoni, "Space Bounds for a Game on Graphs," *Math. Sys. Th.* **10** pp. 239-251 (1977).

30. W. J. Paul and R. E. Tarjan, "Time-Space Trade-Offs in a Pebble Game," *Acta Informatica* **10** pp. 111-115 (1978).

31. N. Pippenger and L. G. Valiant, "Shifting Graphs and Their Properties," *J. Assoc. Comp. Mach.* **23**(3) pp. 423-432 (July 1976).

32. N. Pippenger, "Superconcentrators," *SIAM J. Comput.* **6**(2) pp. 298-304 (June 1977).

33. R. R. Redziejowski, "On Arithmetic Expressions and Trees," *CACM* **12** pp. 81-84 (1969).

34. R. Reischuk, "Improved Bounds on the Problem of Time-Space Trade-off in the Pebble Game (Preliminary Version)," *The 19th Ann. Symp. Foundations Comp. Sci.*, pp. 84-91 (Oct. 1978).

35. J. E. Savage, *The Complexity of Computing*, John Wiley and Sons (1976).

36. J. E. Savage and S. Swamy, "Space-Time Tradeoffs on the FFT Algorithm," *IEEE Trans. on Info. Th.* **IT-24** pp. 563-568 (Sept. 1978).

37. J. E. Savage and S. Swamy, "Space-Time Tradeoffs for Oblivious Sorting and Integer Multiplication," Report No. 37, Program in Computer Science, Brown University, Providence, RI (August 1978).

38. J. E. Savage and S. Swamy, "Space-Time Tradeoffs for Oblivious Integer Multiplication," pp. 498-504 in *Lecture Notes in Computer Science*, ed. H. A. Maurer, Springer-Verlag, Berlin, Heidelberg, New York (July, 1979).

39. J. E. Savage and S. Swamy, "Space-time Tradeoffs for Linear Recursion," *The 6th Ann. ACM Symp. Princ. Prog. Langs.*, pp. 135-142 (Jan. 1979).

40. R. Sethi, "Complete Register Allocation Problems," *SIAM J. Comp.* 4(3) pp. 226-248 (Sept. 1975).

41. H. S. Stone, "Parallel Processing With the Perfect Shuffle," *IEEE Trans. Comput.* **C-20** pp. 153-161 (February 1971).

42. M. Tompa, "Time-Space Tradeoffs for Straight-Line and Branching Programs," Technical Report No. 122/78, Dept. Comp. Sci., U. Toronto, Toronto (July 1978).

43. M. Tompa, "Two Familiar Transitive Closure Algorithms which Admit no Polynomial Time, Sublinear Space Implementations," *12th Ann. ACM Symp. on Th. Computing*, pp. 333-338 (April 28-30, 1980).

44. M. Tompa, "Time-Space Tradeoffs for Computing Functions, Using Connectivity Properties of Their Circuits," *Proc. 10th Ann. ACM Symp. Th. Comp.*, pp. 196-204 (May 1978).

45. Leslie G. Valiant, *On Non-Linear Lower Bounds in Computational Complexity*, Univ. of Leeds, Leeds, U.K. ().

46. L. G. Valiant, "Graph-Theoretic Properties in Computational Complexity," *J. Comp. Sys. Scs.* **13** pp. 278-285 (1976).

47. L. G. Valiant, "Graph-Theoretic Arguments in Low-Level Complexity," *Mathematical Foundations of Computer Science* **53** pp. 162-176 Lecture Notes in Computer Science, Springer Verlag, (1977).

48. A. Waksman, "A Permutation Network," *JACM* **15** pp. 159-163 (1968).

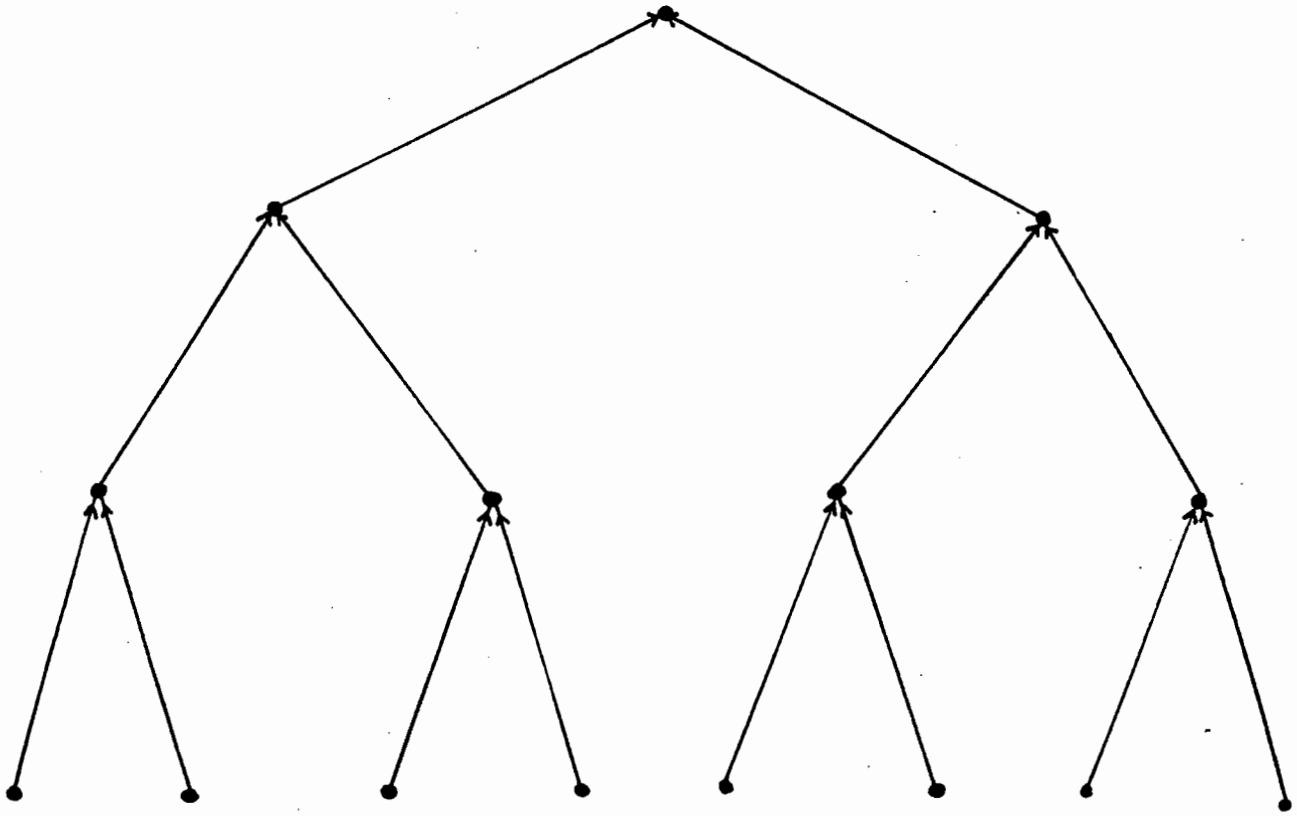


Figure 1: A complete binary tree of depth three.

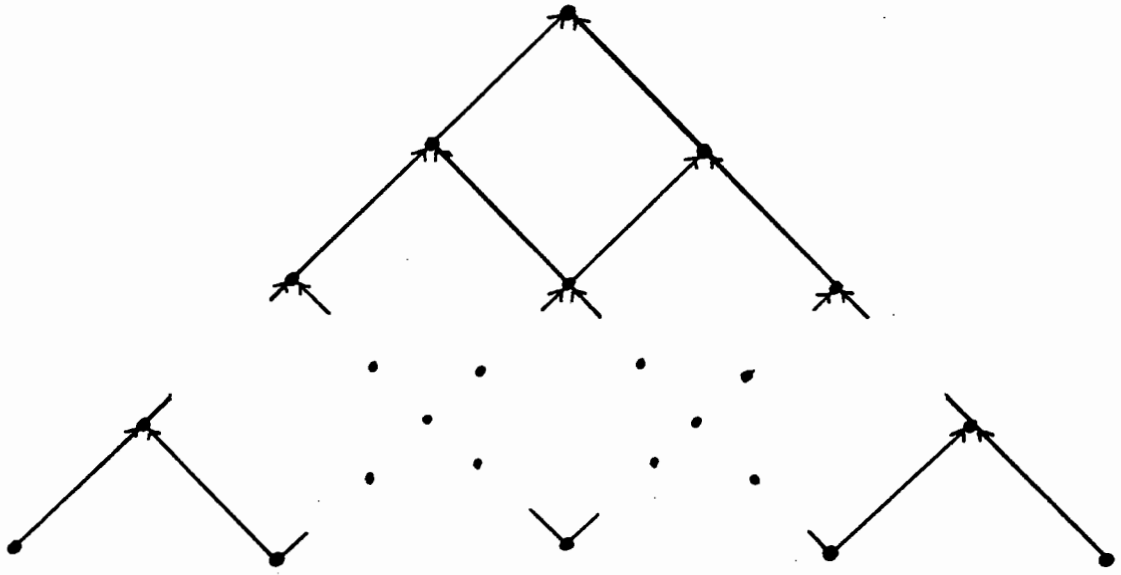


Figure 2: A pyramid graph.

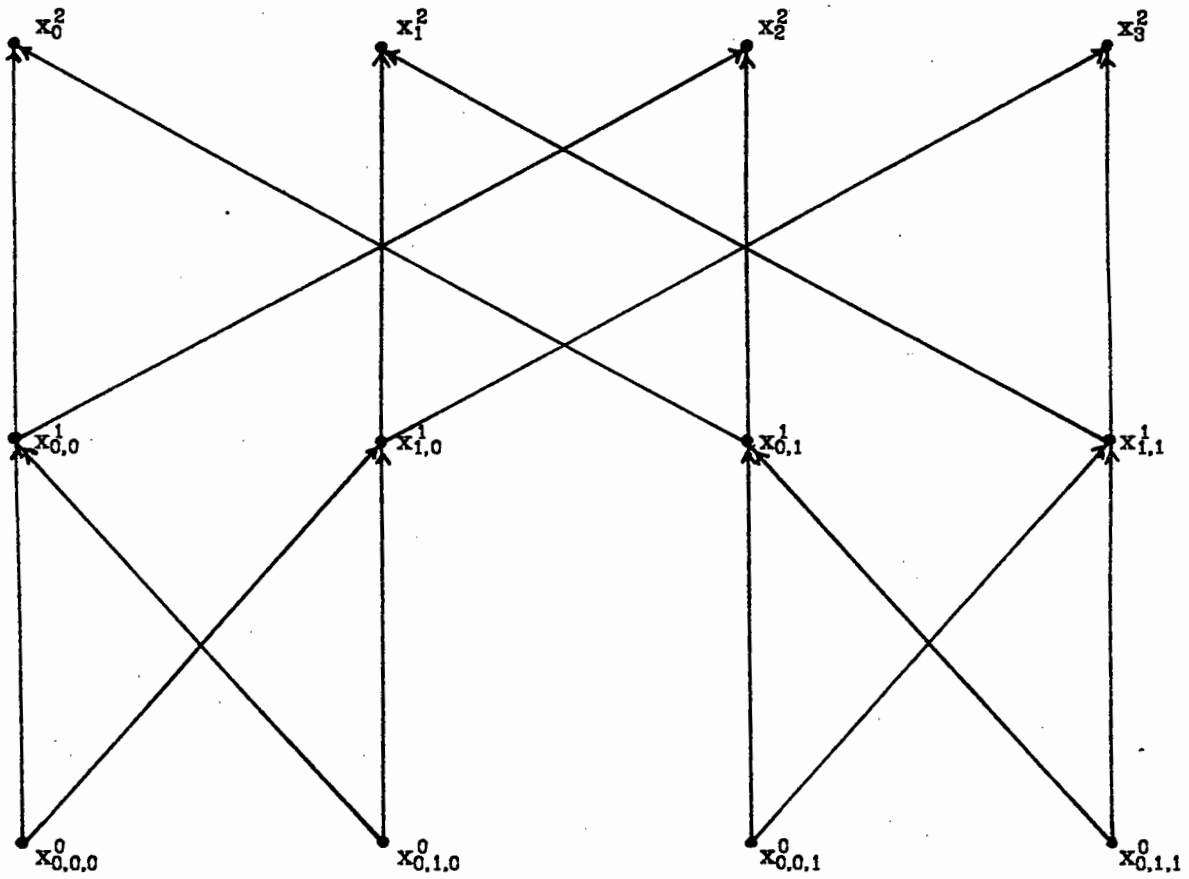


Figure 3: The graph G on four inputs (also FFT[d] for d=2).

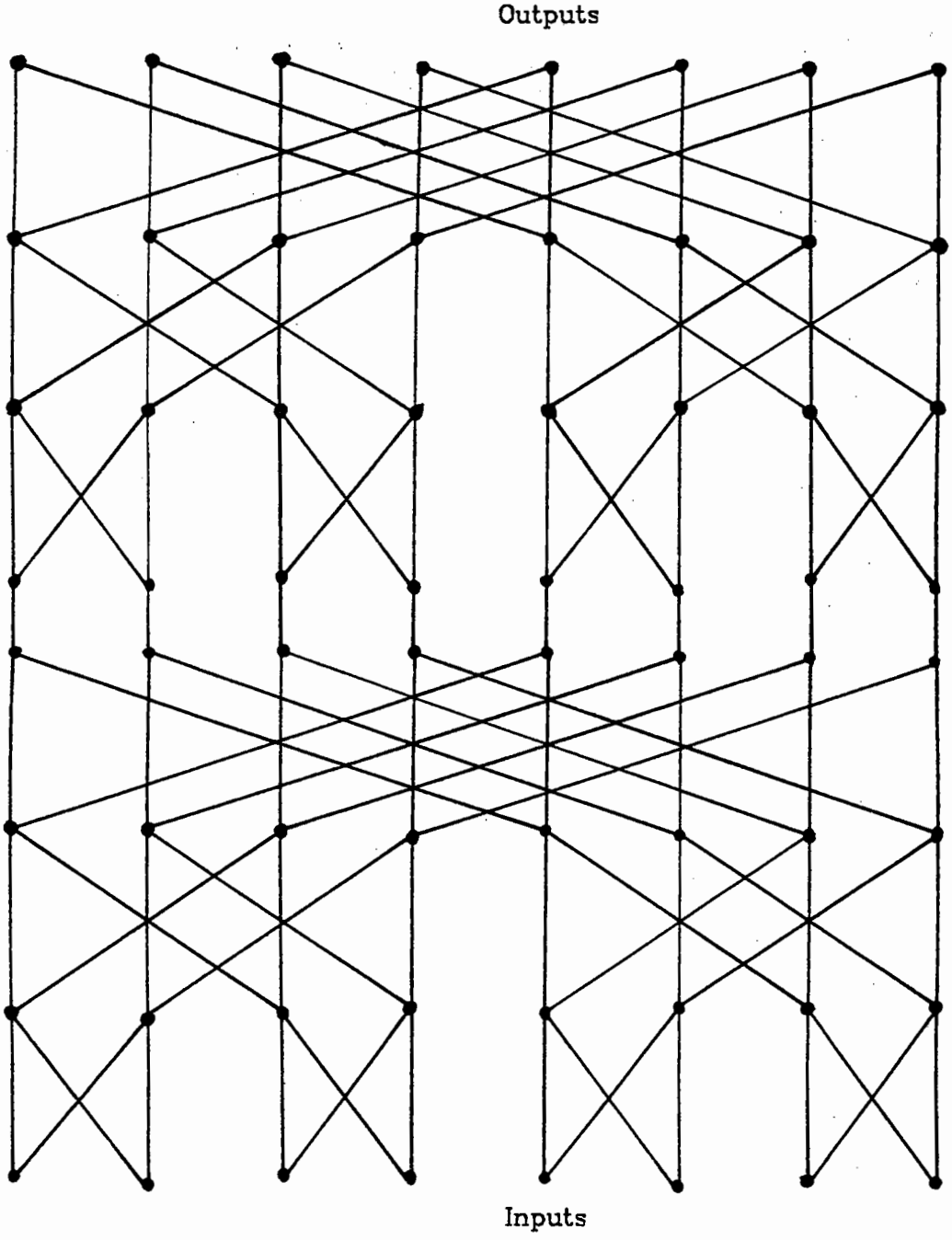


Figure 4: The back-to-back FFT graph on n=8 inputs.

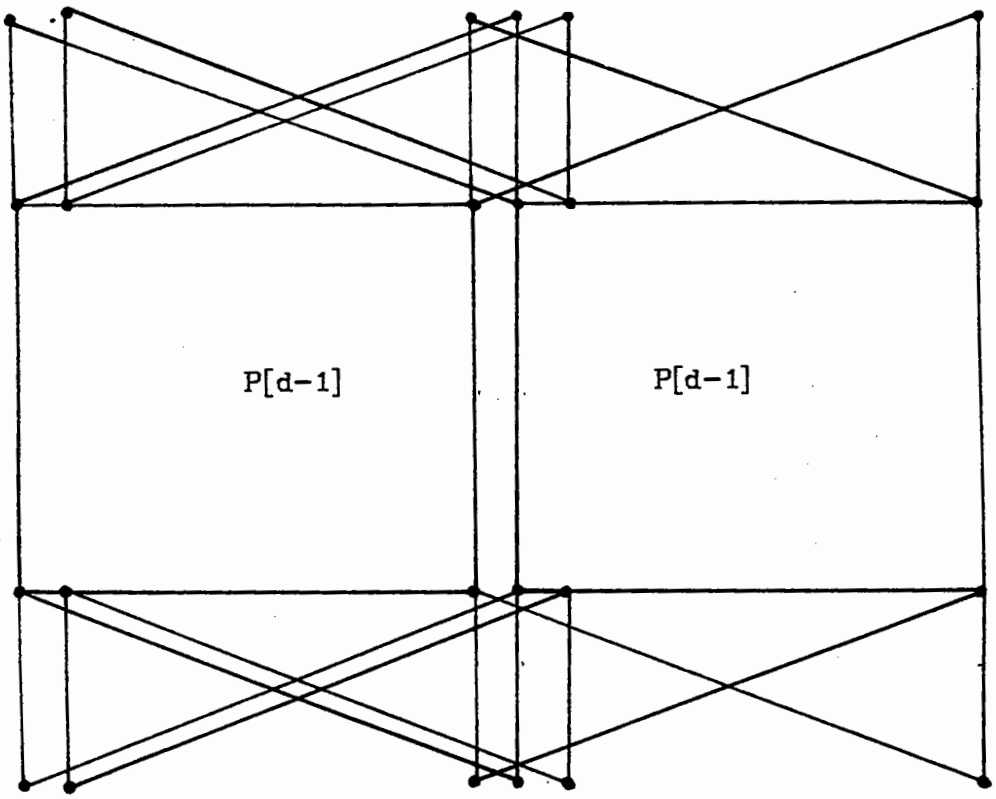


Figure 5: The graph P[d].

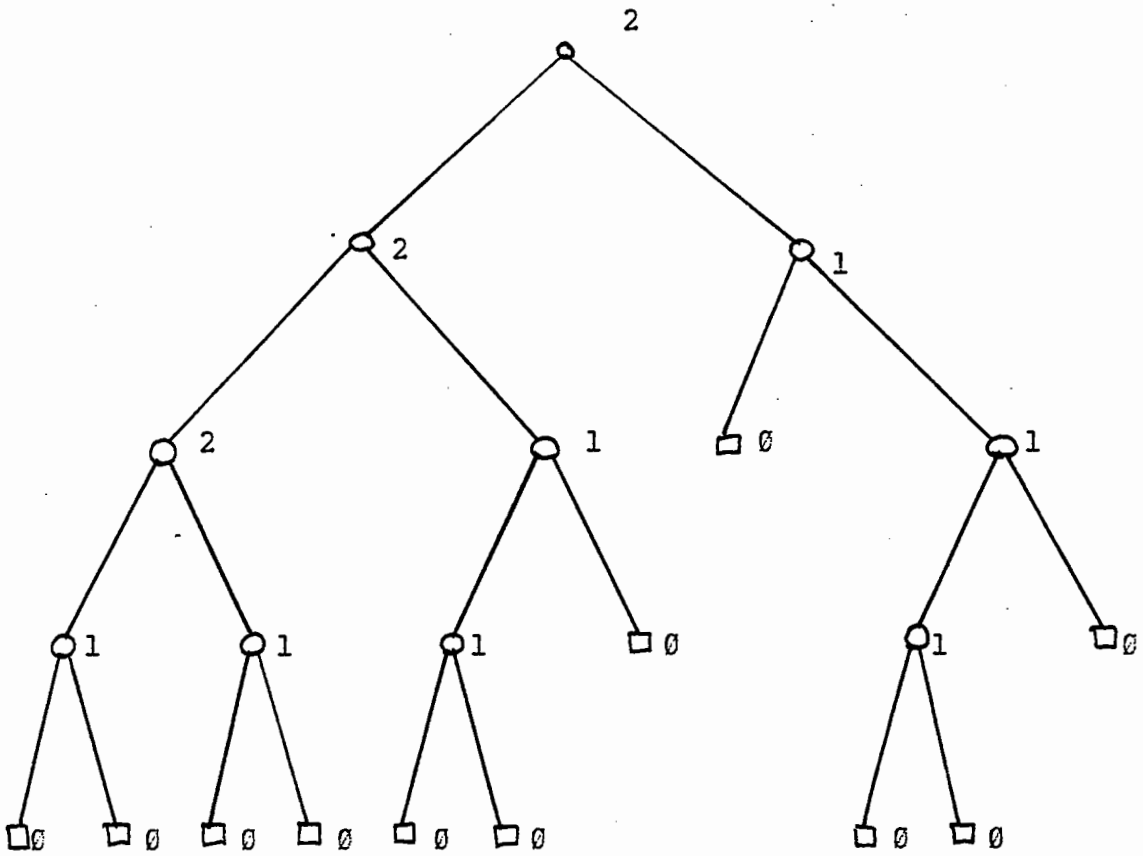


Figure 6: The space requirements of an extended binary tree.

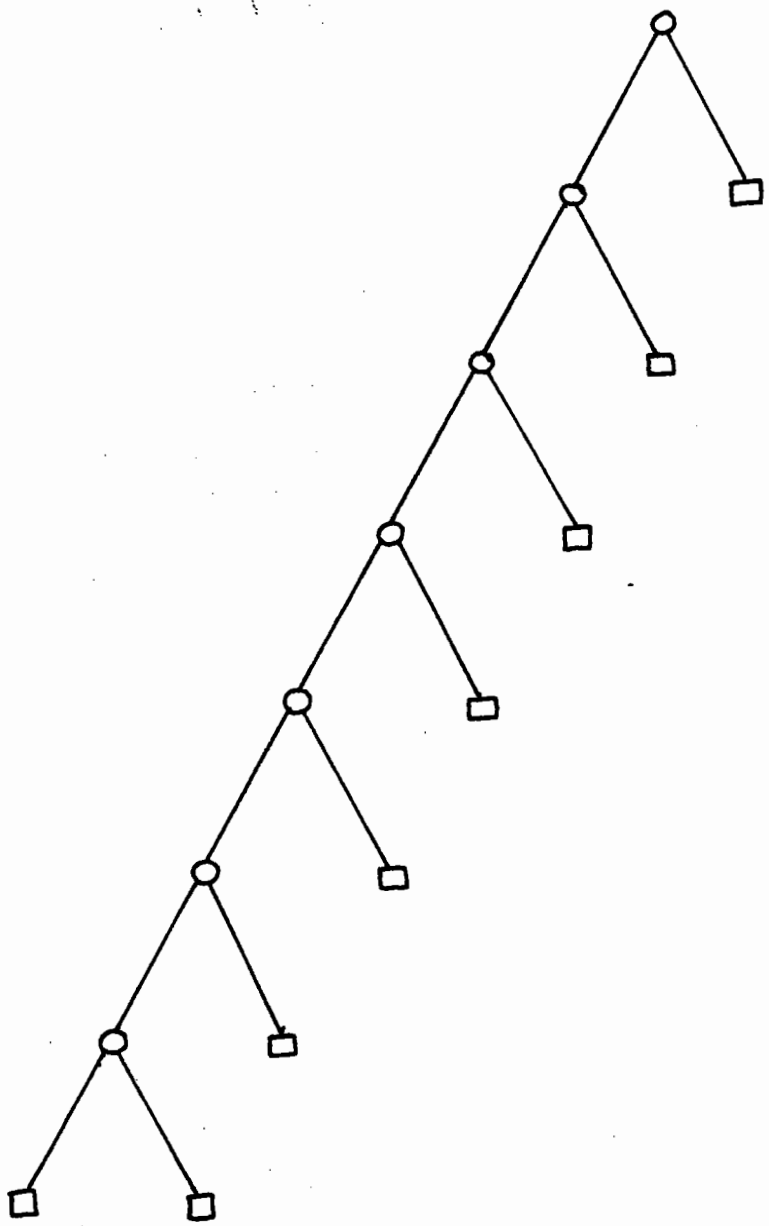


Figure 7: An extended binary tree with space restriction two.

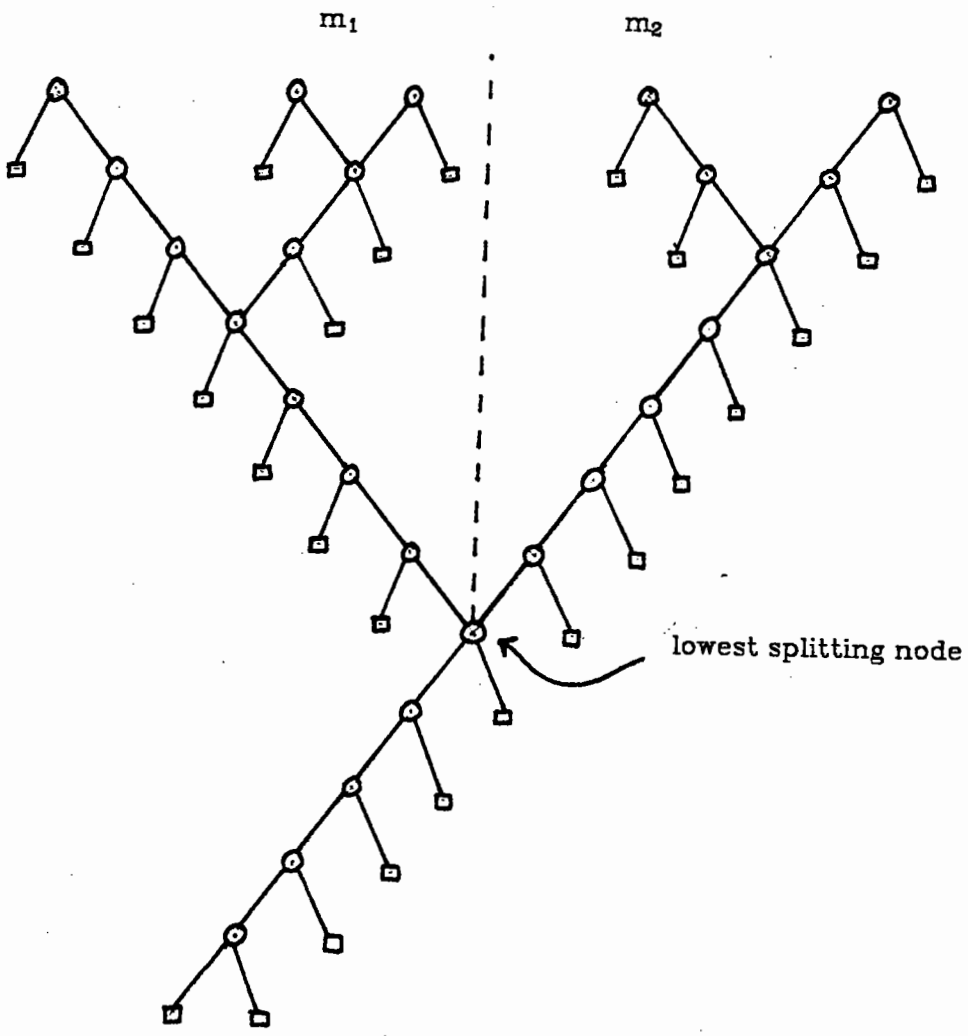


Figure 8: A set of chains.

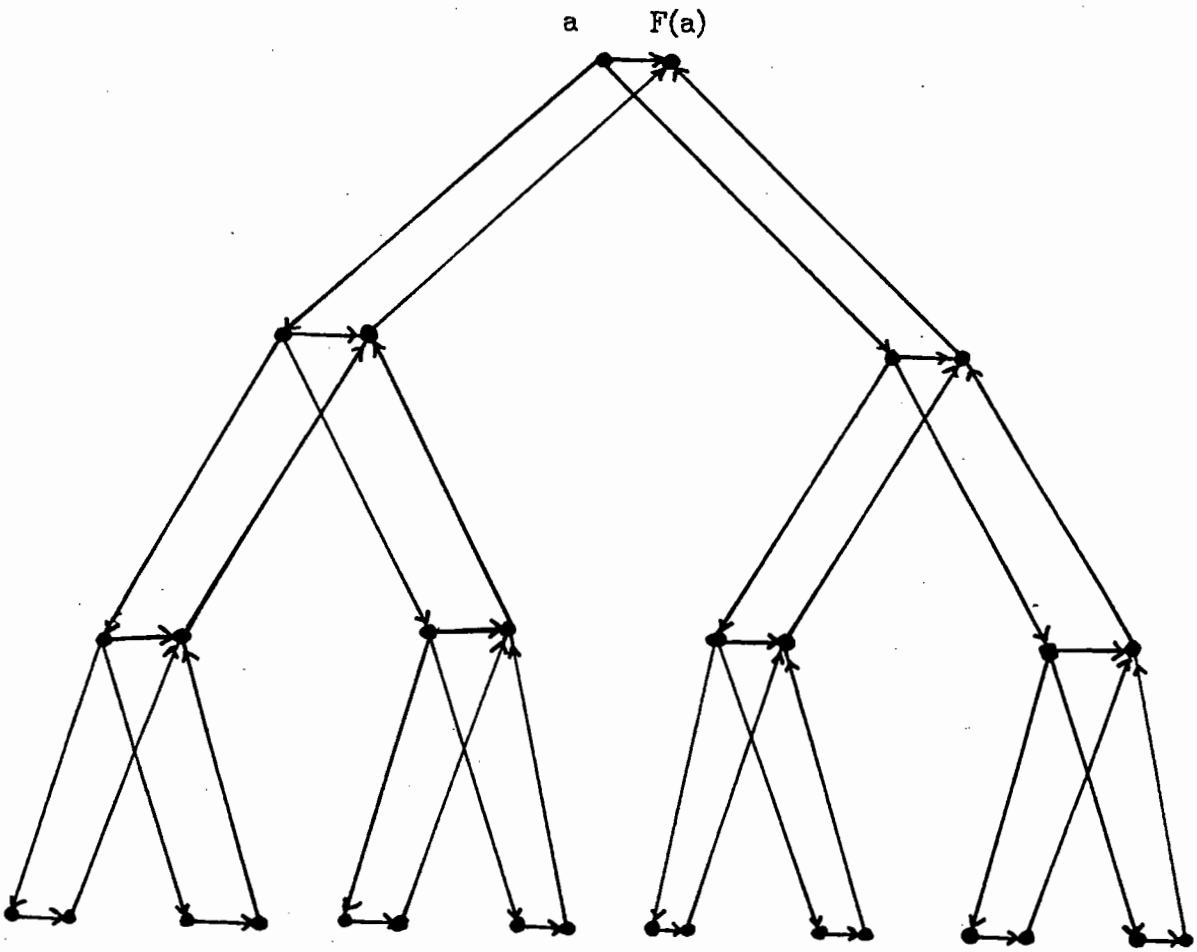


Figure 9: An NLR-graph.

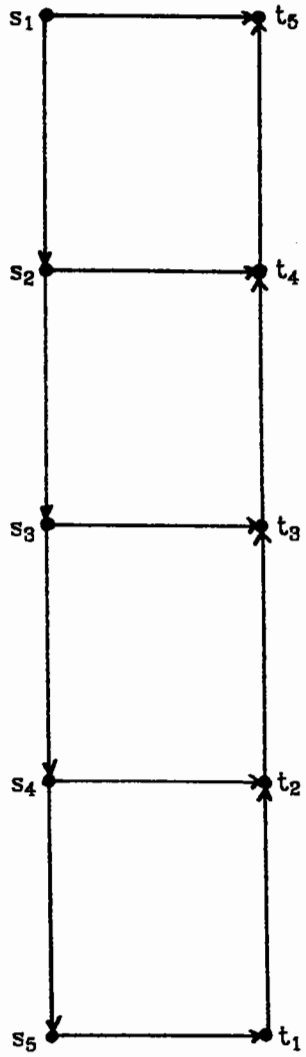


Figure 10: A ladder graph on five elements.

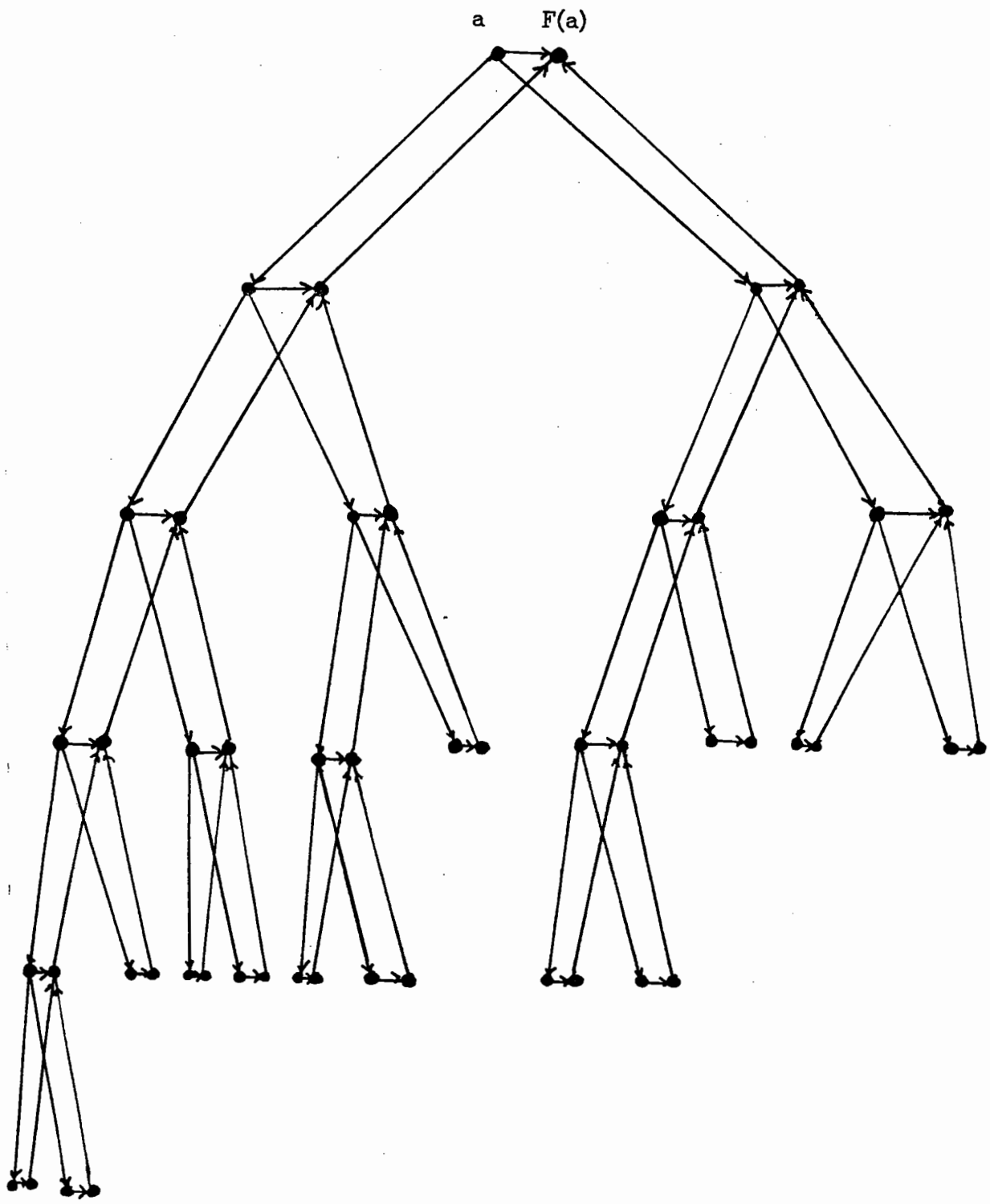


Figure 11: An FNLR-graph of order six.

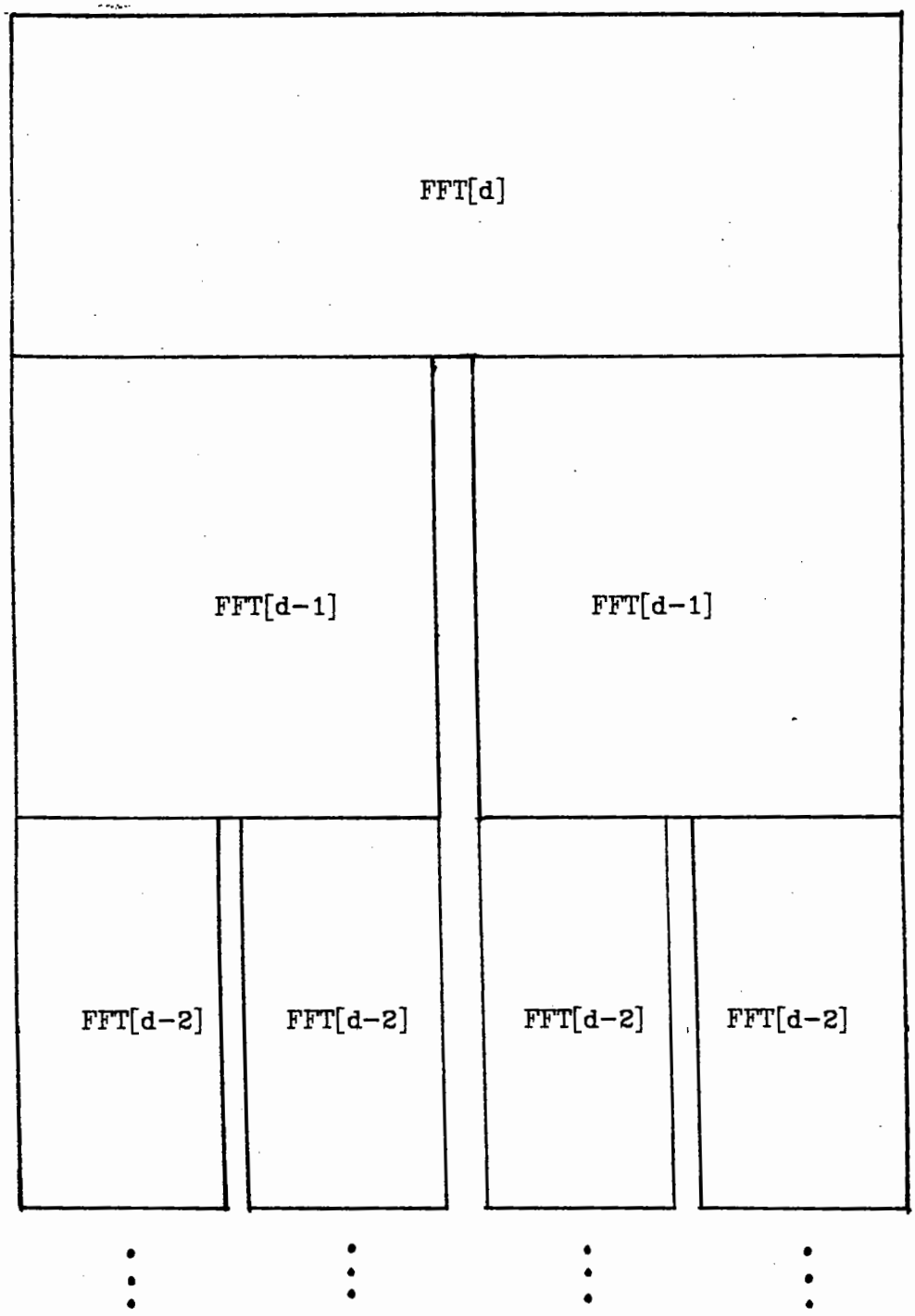


Figure 12: The graph S[d].

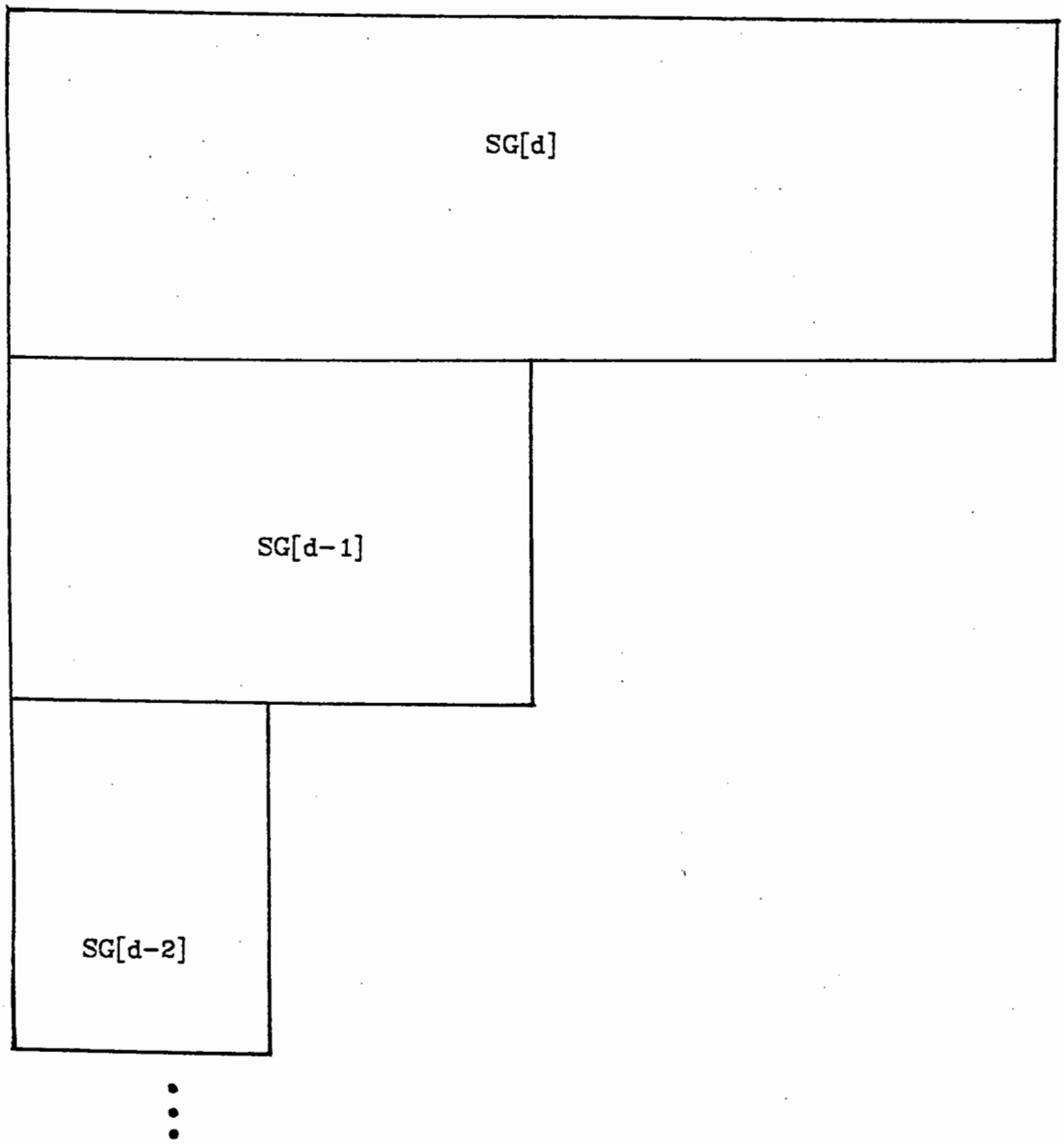
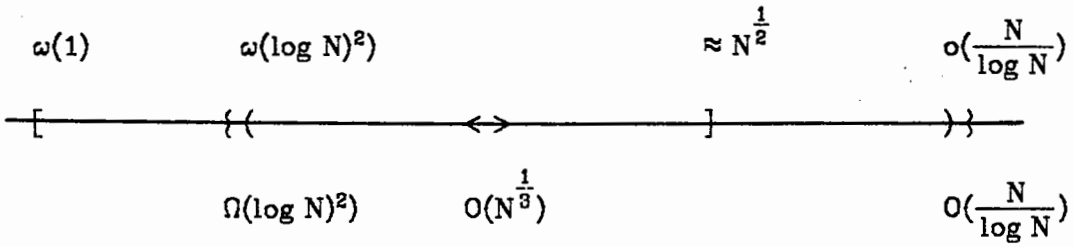


Figure 13: The graph G[d].



[]: Carlson, Savage

{}: Lengauer, Tarjan

(): Paul, Tarjan

<>: van Emde Boas, van Leeuwen; Lingas

Figure 14: Results concerning extreme time-space tradeoffs.

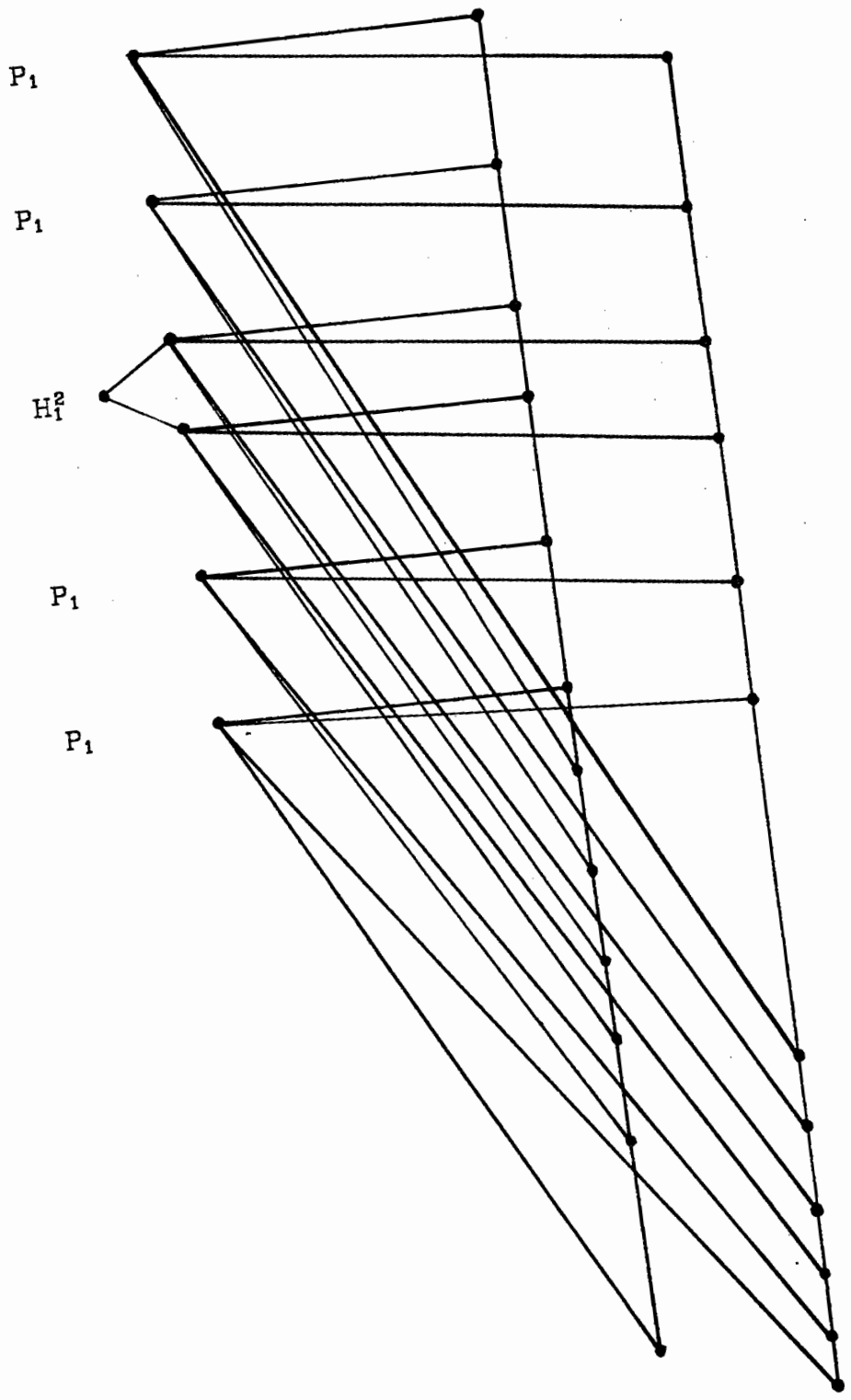


Figure 15: The graph H_2^2, which has two outputs.

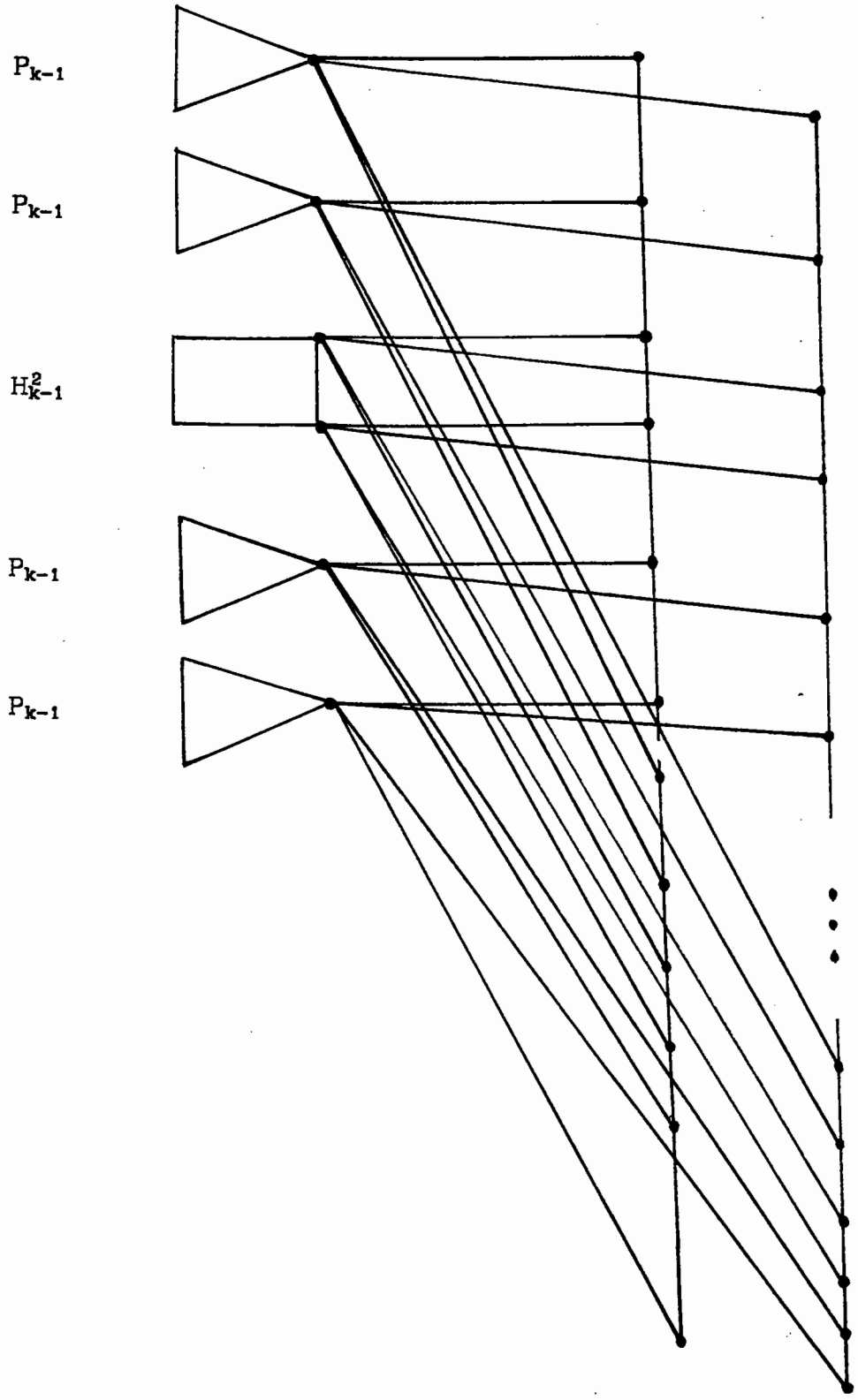


Figure 16: The graph H^2_k has two spines, each with k+1 sections.