# STUDIES ON SORTING NETWORKS AND EXPANDERS

A Thesis Presented to

The Faculty of the

School of Electrical Engineering and Computer Science

and Fritz J. and Dolores H. Russ

College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirement for the Degree

Master of Science

Hang Xie

August, 1998

# Acknowledgment

I would like to express my special thanks to my advisor, Dr. David W. Juedes, for the insights he has provided to accomplish this thesis and the guidance given to me. Without this, it is impossible for me to accomplish this thesis.

I would like to thank Dr. Dennis Irwin, Dr. Gene Kaufman, and Dr. Liming Cai for agreeing to be members of my thesis committee and giving valuable suggestions. I would also like to thank Mr. Dana Carroll for taking time to review my thesis. Finally, I would like to thank my wife and family for the help, support, and encouragement.

# Contents

# List of Figures

# Chapter 1

# Introduction

Over the past few decades, sorting has attracted a great deal of attention [Lei92, CLR90] in computer science research. Sorting is a theoretically interesting problem with a great deal of practical significance. In almost any type of computing system, sorting is one of the most common activities performed. Therefore, it is desirable to design sorting algorithms that are as fast as possible. For sequential algorithms, the speed is measured with the total number of steps [CLR90]. For parallel algorithms, speed translates to the depth of the computation, i.e., the number of parallel steps.

When examining the complexity of sorting, it is advantageous to restrict our attention to *comparison sorts*, i.e., those sorting algorithms that only use comparisons to determine the sorted order of a list of members. Much is known about comparison sorts on sequential machines. In early work, Ford and Johnson [FJ59] introduced the decision-tree model approach to studying comparison sorts. Building on this work, Knuth [Knu73] gave the following well-known information-theoretic lower bound on the complexity of comparison sorts.

**Theorem 1.1** *In the worst case, any comparison sort must perform at least* $\Omega(n \log n)$ *comparisons.*

Since simple algorithms such as heap sort and merge sort achieve this information-theoretic lower bound, little work remains on sequential comparison sorts. For parallel sorting, however, some work remains.

## 1.1 Sorting Networks

The most simple model of parallel sorting is the *sorting network*. These networks are comparison networks that always sort their input. A sorting network is comprised solely of wires and comparators. A comparator is a device with two inputs, $x$ and $y$, and two outputs, $x'$ and $y'$, that computes the functions (Figure 1.1)

$$x' = \min(x, y) \text{ and } y' = \max(x, y).$$

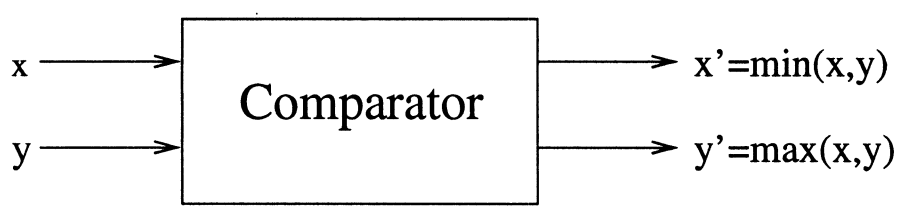


Figure 1.1: Comparator

In sorting networks, $n$ inputs are presented on $n$ wires and at each successive level of the network, at most $n/2$ disjoint pairs of wires are put into comparators. After each level the $n$ wires carry the original elements in some permuted order. The *depth* of a network is just the number of levels.

Just as with sequential comparison sorts, sorting networks obey an information theoretic lower bound on the total number of comparisons. This lower bound translates into a lower bound on the depth of sorting networks.

**Corollary 1.1** *The depth of any sorting network that sorts $n$ numbers must be at least $\Omega(\log n)$.*

**Proof**: Given a sorting network $S$, let the depth of $S$ be $d(n)$, and let the total number of comparators in the network be $t(n)$. In each level of the network, there are at most $n/2$ pairs of comparators. Therefore, it follows that $d(n) \geq t(n)/\frac{n}{2}$. Since $t(n) = \Omega(n \log n)$, it follows that $d(n) = \Omega(\log n)$. □

The existence of efficient sorting networks is well-known. Batcher described a sorting network of depth $1/2 \log n(\log n + 1)$ for sorting $n$ elements [Bat68]. However, Batcher's network is $\log n$ times the information-theoretic lower bound. The question of whether or not there exist $O(\log n)$ depth sorting networks remained open for some time. This question was resolved by Ajtai, Komlós and Szemerédi [AKS83b, AKS83a]. Their construction and proof are of some intricacy. Moreover, the numerical constant factor in the depth bound is enormous (estimated to be $2^{100}$). The AKS algorithm is based on a probabilistic algorithm that becomes deterministic through the use of *expander graphs.* (In related work, Reif and Valiant gave a randomized algorithm that sorts on an $n$ node network and with high probabilities in $O(\log n)$ steps [RV83, RV87].)

There are several current research effort on sorting networks. One effort seeks to find practical sorting networks for special problems with well-known sorting networks such as Batcher's network [AHB93, AHB94]. The another effort seeks to prove the

existence of sorting networks with certain complexity [AKS83a, Pat90]. In this thesis, we study the explicit construction of the latter approach. The constant factor for the depth we achieve is still too enormous to be practical. However, compared to the original construction by Ajtai, Komlós and Szemerédi, the constant decreases greatly.

The AKS algorithm has been studied from many aspects. Bilardi and Preparata [BP85] studied the performance of the original AKS algorithm in the synchronous VLSI model of computation. Later, Paterson [Pat90] considerably simplified the original AKS algorithm. Paterson's work provides a simplification of their construction which allows a more accessible proof. Paterson's algorithm also reduces the constant factor greatly for an explicit construction. Moreover, Paterson proved that for a probabilistic construction, there is a sorting network with depth under $6100 \log n$.

Even though the constant factor provided by Paterson is considerably reduced compared with the original AKS network, it is still impractically large. After a discussion with Paterson, Ajtai, Komlós, and Szemerédi [AKS92] conjectured that the only way to improve the performance of their sorting network was to construct a new kind of halver. However, they did not give an explicit construction in the paper. Cole and O'Dunlaing [CO86] analyzed Paterson's algorithm and gave some small ideas for improvement. However, these improvements cannot influence the depth of the network for the probabilistic construction. Whether or not there exists a practical $O(\log n)$ depth sorting network remains a major open question. (Inceipi and Sedgewick [IS87] conjectured that an approach based on shellsort might lead to such a network.)

## 1.2 Contributions

In this thesis, we analyze Paterson's algorithm in detail. In an attempt to make the algorithm more easily understood, we give the precise definitions for $\epsilon$-halvers and separators, detailed explanations on some conclusions in Paterson's paper, and the pseudocode for Paterson's algorithm.

In Paterson's algorithm, the critical component is a network that correctly partitions a list into left and right sides of equal size. To do this, Paterson uses an idea of Ajtai, Komlós, and Szemerédi [AKS83a]. The idea is to make an approximate partition of elements, which can be achieved in only a constant number of comparator levels, and then introduce some error-recovery structure into the sorting scheme. In Paterson's algorithm, the approximate partition is implemented with *separators.*

A *separator* partitions a set of elements into four parts. In theory, the two extreme parts generated by the separator include most of the elements that are wrongly routed in a procedure that will be described later. A separator can be constructed with an $\epsilon$-halver. (See Chapter 3 and Chapter 4 for the precise definitions of separators and $\epsilon$-halvers.) The correctness of the network is demonstrated by proving an invariant which bounds the proportion, within each bag, of elements with any particular degree of displacement from their proper positions.

The depth of Paterson's algorithm is dependent on the depth and parameters of the separator, which is determined by the algorithm that implements it and the depth of an $\epsilon$-halver. An $\epsilon$-halver can be constructed through expanders using constant depth.

There are two possible approaches to improve the efficiency of Paterson's algorithms. The first approach is to construct more efficient halvers by using better expanders. Another approach is to find a more efficient construction of separators.

A common way to prove that a particular graph is a good expander is to examine the second-largest eigenvalue of its adjacency matrix [Tan84]. The greatest possible separation between the first and second-largest eigenvalues in a graph was achieved in explicit constructions [LPS86] with graphs called *Ramanujan graphs* (These graphs are asymptotically optimal in making the second-eigenvalue as small as possible). Here, we use Ramanujan graphs to construct small $\epsilon$-halvers.

According to an idea in [CO86], we analyze a new kind of separator and give its depth. This separator can achieve better separation without increasing its depth.

Finally, we use all our results to analyze the depth of Paterson's algorithm. Through our construction, we achieve sorting networks with a depth less than 947512 $\log n$ using the original separator and a depth less than 765544 $\log n$ using the new kind of separator.

## 1.3 Structure of this Thesis

This thesis examines the explicit construction of Paterson's algorithm. Chapter 2 introduces expanders and the major research results in the area. There we adopt a more uniform definition of expanders than the one in the original paper of Ajtai, Komlós, and Szemerédi. This guarantees consistency in our analysis.

In Chapter 3, we analyze the construction of $\epsilon$-halvers through matching in regular graphs. There we give the precise definition of an $\epsilon$-halver. In order to use Ramanujan graphs to construct $\epsilon$-halvers, we introduce a very simple method to construct a balanced regular expander. The second largest eigenvalue of a graph is then used to analyze the expansion of a graph. We analyze the performances of Kahale's result [Kah93a] and Tanner's result [Tan84] and show that Kahale's result is only effective

for very small subsets of the vertices. We adopt Tanner's result and calculate the depth of the associated $\epsilon$-halver.

In Chapter 4, we give the precise definition of a separator and analyze the algorithm to implement it [Pat90]. We then analyze an improved construction of a separator and compute its new parameter values.

In Chapter 5, Paterson's algorithm is described in detail. We present the pseudocode for the algorithm, which is an attempt to make the algorithm more easily understood. Based on the results obtained in Chapter 3, we compute the depth of Paterson's construction. As an application of the new kind of halver, we choose a set of new parameters which satisfy the constraints of the algorithm and make the depth using the explicit construction of Paterson's algorithm.

In Chapter 6, we conclude this thesis with possible research directions.

# Chapter 2

# Expander Graphs and Related Constructions

In this section, we introduce the notion of an *expander graph* and give several related constructions. Expander graphs are widely used in computer science in areas ranging from parallel computation [AKS87] to information theory and cryptography [Spi95, SS96]. (See [Kla84] for more applications of expander graphs.) Here we are particularly interested in the applications of expander graphs to parallel computation. In particular, we will use expander graphs in the construction of $\epsilon$-halvers in chapter 3.

Roughly speaking, an *expander graph* is a graph in which every set of vertices has an unusually large number of neighbors. In Section 2.1, we give the precise definition of expanders and some research results in graph expansion. In Section 2.2, we survey research results concerning explicit constructions of expander graphs.

## 2.1 Expander Graphs

**Definition 2.1** *Let $G = (V, E)$ be a graph. The number of edges in graph $G$ incident with vertex $v$ is called the degree of $v$ (in $G$).*

**Definition 2.2** *Given a graph $G = (V, E)$, the set $\Gamma(S)$ is defined to be the set of neighbors of $S$, i.e., $\Gamma(S) = \{z | \exists x \in S, (z, x) \in E\}$.*

The above definitions allow us to precisely define expander graphs.

**Definition 2.3** *A graph $G(V, E)$ is an expander graph with parameters $(\lambda, \alpha, \mu)$ if*

*(i) for any set $A \subseteq V$ such that $|A| \leq \alpha|V|$, we have $|\Gamma(A)| \geq \lambda|A|$, and*

*(ii) the degree of any vertex is $\leq \mu$.*

*The parameter $\lambda$ is the expansion of the graph.*

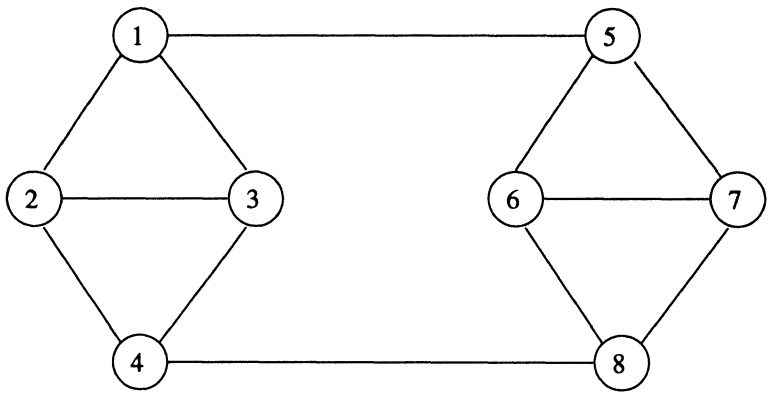


Figure 2.1: A (3/2,1/2,3) Expander.

It is relatively easy to construct some expander graphs. As an example, consider Figure 2.1. The graph in Figure 2.1 is a $(3/2, 1/2, 3)$ expander. To see this, select any subset $A$ of $V$ with less than $1/2$ of the vertices. In every case, $\Gamma(A)$ will have at least $3/2$ as many vertices as $A$.

As in the above example, many expander graphs will have the property that the degree of each vertex will exactly be $k$. Such graphs are *$k$-regular*.

The expansion properties of a graph are strongly related to the eigenvalues of its adjacency matrix. Currently, the best known efficient algorithm to determine lower bounds on the expansion of a graph relies on analysis of its second largest eigenvalue [AM84, Alo86]. (In a related work, Buck [Buc86] used the doubly-stochastic matrices and Kahale [Kah91, Kah93b, Kah93a, Kah95] used various techniques related to the second largest eigenvalue to analyze the expansion properties of graphs.) Here we recall the appropriate definitions and give several important properties that relate the expansion properties of graphs and their eigenvalues. Many of these properties appear in [vLW92]. We refer the readers to [HK71, Gan59] for the related content on linear algebra.

**Definition 2.4** *Let $G$ be a graph of $n$ vertices. The adjacency matrix of $G$ is the $n$-by-$n$ matrix, $A_G$, with entries $a_{i,j}(i = 1, 2, ..., n; j = 1, 2, ...n)$, such that*

$$a_{i,j} = \begin{cases} 1, & \text{if } (i,j) \text{ is an edge of } G \\ 0, & \text{otherwise.} \end{cases}$$

Let $A$ be an $n \times n$ matrix. Let $x \in R^n$ be a non-zero vector. If there exists a $\lambda$ such that $Ax = \lambda x$, then $x$ is an eigenvector of $A$ and $\lambda$ is an eigenvalue of $A$. The eigenvalues of $G$ is defined to be the eigenvalues of the matrix $A_G$.

For example, consider the adjacency matrix $A_G$ of $G$,

$$A_G = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

In this case, the eigenvalues of $A_G$ are $\{3, \sqrt{5}, 1, -1, -1, -1, -1, -\sqrt{5}\}$.

Since the adjacency matrix $A_G$ is always symmetric, it is *diagonalizable* (i.e., it has $n$ linearly independent eigenvectors) and all of its eigenvalues are real [HK71].

We will now restrict our discussion to graphs that are $k$-regular. For these graphs, $k$ is an eigenvalue since it corresponds to the eigenvector with a 1 in each entry. Furthermore, $k$ must be the largest eigenvalue for the graph. To see this, consider the product of $A_G$ and any vector. The largest value in the product vector is at most $k$ times the largest value in the vector. Therefore, its associated eigenvalue is bounded below by $k$. Similarly, $-k$ is an eigenvalue if and only if the graph is bipartite. (To see this, break the graph $G$ into two groups $V_1$ and $V_2$ and construct an eigenvector with 1's for the positions associated with $V_1$ and $-1$'s for the positions associated with $V_2$. The eigenvalue for this vector is $-k$. On the other hand, notice that if $-k$ is an eigenvector then we can partition $V$ into two sets to verify that $G$ is bipartite.)

By examining the adjacency matrix $A_G$ of a graph $G$ using methods from linear algebra, we are able to study the expansion properties of the graph. For every subset $X$ of $V$, let $f$ be the characteristic vector of $X$, i.e,

$$f(v) = \begin{cases} 1 & \text{if } v \in X \\ 0 & \text{otherwise.} \end{cases}$$

It is not difficult to show that $||A_G f||^2 \geq k^2|X|^2/|\Gamma(X)|$ [Kah93a]. This shows that to get a lower bound on $\Gamma(X)$, it is sufficient to bound $||A_G f||$ from above. If we use the well-known inequalities in norm space, we can get the expansion property of a graph.

Tanner [Tan84] proved the following result.

**Theorem 2.1** *Given an n-vertex k-regular graph $G(V, E)$, every subset $X$ of $V$ has*

*the property that*

$$|\Gamma(X)| \geq \frac{k^2|X|}{\lambda^2 + (k^2 - \lambda^2)|X|/n},$$

*where $\lambda$ is the second largest eigenvalue.*

Kahale [Kah93a] improved this result as follows.

**Theorem 2.2** *If $G = (V, E)$ is $k$ -regular, then for any subset $X$ of $V$, we have*

$$\frac{|\Gamma(X)|}{|X|} \geq \frac{k^2 + (k^2 - \lambda^2)^2/\lambda^2 k}{\lambda^2 + k^2(k^2 - \lambda^2)|X|/(\lambda^2 n)}.$$

In order to get a high rate of expansion, we want to find graphs whose second largest eigenvalue, $\lambda$, is as small as possible.

## 2.2 Explicit Constructions

It is known that random regular graphs are good expanders [AS92]. For examples for any $\beta < k - 1$, there exists a constant $\alpha$ such that, with high probability, all the subsets of a random $k$-regular graph of size at most $\alpha n$ have expansion at least $\beta$.

However, the explicit construction of expanders is much more difficult. The first explicit construction of an infinite family of expanders was discovered by Margulis [Mar73]. He gave an explicit construction of expander graphs with a linear number of edges.

Margulis constructed bipartite 5-regular graphs which are expanders. We describe in the following the construction method. Let $m$ be any positive integer, and $Z_m$ denote the integer field mod $m$ ($Z_m = \{0, 1, 2, 3, ..., m-1\}$). We define $A_m = Z_m \times Z_m$. Thus, the elements of $A_m$ are pairs $(u, v)$. Let $f_i$ for $i = 0, 1, ..., 4$, be functions on

$A_m$ defined by

$$\begin{aligned} f_0(u,v) &= (u,v), \\ f_1(u,v) &= (u+1,v), \\ f_2(u,v) &= (u,v+1), \\ f_3(u,v) &= (u,u+v), \text{ and} \\ f_4(u,v) &= (-u,v), \end{aligned}$$

using the addition modulo $m$. A graph $G(M)$ is defined as the bipartite graph where each node $(u,v)$ is connected to the output nodes $f_0(u,v), f_1(u,v), f_2(u,v), f_3(u,v)$ and $f_4(u,v)$. The proof that these graphs are expanders is based on some non-trivial results in group representation theory. Although Margulis was able to prove that the expansion constant was greater than 1, he was not able to bound it strictly away from 1. In 1981, after slightly modifying Margulis's construction, Gabber and Galil [GG81] constructed similar graphs and gave an estimate of their expansion coefficient using techniques from harmonic analysis. Their result was later improved by Buck using techniques from linear algebra [Buc86].

Pippenger [Pip93] points out that one can obtain a family of good expander graphs by exponentiating the expander graphs constructed by Gabber and Galil. In [GG81], Gabber and Galil construct an infinity family of 5-regular expander graphs. By their result, we know that all of the eigenvalues of these graphs other than 5 must be bounded by some number $\lambda < 5$. Consider the graphs obtained by exponentiating the adjacency matrices of these graphs $t$ times. They will have degree $5^t$, and all eigenvalues but the largest will have absolute value at most $\lambda^t$. This construction yields a family of good expander graphs. These graphs will have multiple edges and

self-loops, but this is irrelevant in our applications. (We could remove the multiple edges and self-loops without adversely affecting the quality of the expanders.)

Alon and Boppana [Alo86] showed the following.

**Theorem 2.3** *For any sequence $G_{n,k}$ of $k$-regular graphs on $n$ vertices,*

$$\liminf_{n\to\infty} \lambda(G_{n,k}) \geq 2\sqrt{k-1},$$

*where $\lambda(G_{n,k})$ is the second largest eigenvalue of $G_{n,k}$.*

Therefore, even for sparse sets, the best expansion coefficient we can obtain by applying Tanner's result is $k^2/(4(k-1)) \approx k/4$. This bound is achieved by graphs for which $\lambda \leq 2\sqrt{k-1}$. These graphs, called *Ramanujan graphs*, have been explicitly constructed in [LPS86, LPS88].

**Definition 2.5** *A Ramanujan graph is a connected $k$-regular graph whose eigenvalues $\neq k$ are at most $2\sqrt{k-1}$ in absolute value.*

For example, consider Figure 2.1. This 8 vertex $(3/2, 1/2, 3)$-expander is Ramanujan since its second largest eigenvalue, $\sqrt{5}$, is less that $2\sqrt{2}$.

Ramanujan graphs are relatively easy to construct. Here we present a construction given in [LPS86, LPS88]. The construction builds a class of non-bipartite Ramanujan graphs $Y_{p,q}$ on $q+1$ vertices, each of degree $p+1$, where $p$ and $q$ are primes congruent to 1 modulo 4.

The construction proceeds as follows. Let $p$ and $q$ be primes congruent to 1 modulo 4. From elementary number theory, it is known that there exist $p+1$ unique vectors $a = (a_0, a_1, a_2, a_3)$ such that $\|a\|^2 = a_0^2 + a_1^2 + a_2^2 + a_3^2 = p$, where $a_0$ is an odd positive

integer, and $a_1$, $a_2$, and $a_3$ are even integers. Let $\gamma_a$ be the matrix

$$\gamma_a = \begin{pmatrix} a_0 + ia_1 & a_2 + ia_3 \\ -a_2 + ia_3 & a_0 - ia_1 \end{pmatrix},$$

where $i$ is an integer such that $i^2 \equiv -1(\mathrm{mod}q)$. The $p+1$ matrices $\gamma_a$ act on the elements $z \in P^1(F_q) = \{0, 1, ..., q-1, \infty\}$ in the following fashion.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} z = \frac{az+b}{cz+d}(\mathrm{mod}q)$$

Define a graph $Y_{p,q}$ with $n = q+1$ vertices by associating with each vertex a value $z \in P^1(F_q)$ and by joining $z$ to each $\gamma_a(z)$. We call the graph $Y_{p,q}$ LPS graphs. The proof that these graphs are Ramanujan graphs relies on very sophisticated mathematical techniques [LPS86, LPS88].

Margulis [Mar88] independently proved the same result. In a related work, Morgenstern [Mor95, Mor94] constructed explicit Ramanujan graphs of degree $s+1$, where $s$ is any prime. The recently published books by Lubotzky give a mathematical background on the construction of Ramanujan graphs [Lub95a, Lub95b].

# Chapter 3

# The Construction of $\epsilon$ -halvers

A major component of known logarithmic depth sorting networks (such as the AKS algorithm and Paterson's algorithm) is the $\epsilon$-halver. An $\epsilon$-halver for $2n$ elements is a comparator network that partitions the $2n$ inputs into left and right blocks each of size $n$. Every $\epsilon$-halver has the property that less than $\epsilon k$ of the $k$ smallest elements are placed in the right block and less than $\epsilon k$ of the $k$ largest elements are placed in the left block. By using expander graphs, we can construct an $\epsilon$-halver in constant depth (depending on $\epsilon$).

Here we show how to build $\epsilon$-halvers from expander graphs. In Section 3.1, we explain the standard approach and give some necessary definitions, algorithms and related analysis.

In Section 3.2, we also introduce the Hungarian algorithm which can be used to obtain an 1-factor of a bipartite graph if it exists. In Section 3.4, we approach a simple method to construct balanced bipartite expanders from unbalanced expanders. We compare Tanner's and Kahale's results about expansion and demonstrate that when used to find the lower bound of the degree of expander graphs which can be used to implement the $\epsilon$-halver, Kahale's result can not achieve better results than Tanner's.

We derive a lower bound on the depth of a halver and give the algorithms to construct the halver explicitly.

## 3.1 $\epsilon$-halvers

**Definition 3.1** *Given a set of $m$ elements $S = \{a_i | 1 \leq i \leq m\}$, such that $a_1 \leq a_2 \leq ... \leq a_{m-1} \leq a_m$ and a permutation of the elements of $S$, $\Pi_S = a_{\pi_1}, a_{\pi_2}, ..., a_{\pi_m}$, such that whenever $a_{\pi_i} = a_{\pi_j}$ and $i < j$, then $\pi_i < \pi_j$, set $\pi_L = \{\pi_1, \pi_2, ..., \pi_{m/2}\}$ and $\pi_R = \{\pi_{m/2+1}, \pi_{m/2+2}, ..., \pi_m\}$.*

*$\Pi_S$ is $\epsilon$-halved if for $1 \leq k \leq m/2$,*

$$|\{x | 1 \leq x \leq k, x \in \pi_R\}| \leq \epsilon k$$

*and*

$$|\{x | m - k + 1 \leq x \leq m, x \in \pi_L\}| \leq \epsilon k.$$

*A procedure is called an $\epsilon$-halver (of $m$ elements) if it accepts $S$ as input, and produces an $\epsilon$-halved permutation of $S$ as output.*

In Ajtai, Komlós and Szemerédi's original paper [AKS83a], the authors described a way to use expanders to construct $\epsilon$- halvers. Their construction is based on *bipartite* $k$-regular graphs. Here we recall the definition.

**Definition 3.2** *A graph $G = (V, E)$ is bipartite if the set of vertices $V$ can be partitioned into two sets $V_1$ and $V_2$ such that for each $\{u, v\} \in E$, $u \in V_1$ and $v \in V_2$.*

Such a graph is denoted by $G = ([V_1, V_2], E)$, where $[V_1, V_2]$ is the partition and $E$ is the set of edges.

Later, we will build comparator networks by finding *1-factors* in $k$-regular bipartite graphs. We first recall the definition.

**Definition 3.3** *Let $G = (V, E)$ be a graph. A 1-factor(or perfect matching) of $G$ is a set $S$ of disjoint edges that span the graph such that for every vertex $u \in V$, there is exactly one edge in $S$ that has $u$ as an endpoint.*

(Think of $S$ as a subgraph. A subgraph of a graph *spans* $G$ if all the vertices of the graph are also in the subgraph.)

Expander graphs can be used to construct $\epsilon$-halvers directly. To see this, let $G = ([V_1, V_2], E)$ be a $((1-\epsilon')/\epsilon', \epsilon'/2, c)$-expander with $2n$ vertices and decompose $G$ into $c$ different 1-factors. We will use these-1 factors to build an $\epsilon'$-halver.

Since each edge in each 1-factor connects a vertex from $V_1$ to a vertex from $V_2$, we can use these edges as comparators in a natural way. Number the vertices in the graph $G$ so that the vertices in $V_1$ are numbered from 1 to $n$ and the vertices in $V_2$ are numbered from $n+1$ to $2n$. Build a comparison network in stages, with each stage corresponding to a 1-factor. For each edge $(i, j)$ in a 1-factor, add a comparator that compares the elements at position $i$ and $j$, and places the smaller of the two at position $i$ and the larger at position $j$. If we do this for each 1-factor, we get a comparator network of depth $c$.

**Algorithm 3.1** *Let $G = ([V_1, V_2], E)$ be an $((1-\epsilon')/\epsilon', \epsilon'/2, c)$ expander with $2n$ vertices, where $V_1$ and $V_2$ each has $n$ vertices. Associate $V_1$ with the lower half of the $n$ registers, and $V_2$ with the upper half. Decompose $G = ([V_1, V_2], E)$ into $c$ different 1-factors $F_1, ..., F_c$.*

*for $i = 1, ..., c$*

*for all edges* $e \in F_i$ *do in parallel*

*Compare elements at the ends of* $e$

*and interchange them if they are out of sequence*

*endfor*

*endfor*

**Theorem 3.1** *The above comparator network is an* $\epsilon'$*-halver.*

**Proof**: Consider the output at position $j > n$ in the network. Since $j > n$, the value at position $j$ is larger than all the output values of the neighbors $i$ of $j$ in $G$. To see this, notice that $j$ position is in the second half of the inputs. Therefore, the value at position $j$ is always the larger of the two in a comparison between $j$ and one of its neighbors. Therefore, the final value at position $j$ is the maximum value attained at position $j$, and these values are monotonically increasing. Since $j$'s neighbors are in the first half of the input, their values are likewise monotonically decreasing. Since the values of elements of at $i$ and $j$ are compared at some point, this chain of inequalities gives us that the final value at position $j$ must be larger than the final value at position $i$.

Without loss of generality, we assume that the set of the input elements to the comparator is $\{1, 2, .., n-1, n, n+1, ..2n\}$. To see that the above network is an $\epsilon'$-halver, let $Z$ be the set of first $k$ values, that is $\{1, 2, ..., k\}$ (where $k$ ranges from 1 to $n$), and assume that more than $\epsilon' \cdot k$ of the values of $Z$ end up in the second half of the output. Let $S$ be the set of positions of these elements. Obviously, all the elements whose positions are in $S$ must be less than $k+1$.

Assume without loss of generality that $\epsilon' k < |S| \leq \epsilon' n$. (If $|S| > \epsilon' n$, pick $\epsilon' n > \epsilon' k$ elements from $S$.) Since $G$ is an expander and $\epsilon' k < |S| \leq \frac{\epsilon'}{2} 2n = \epsilon' n$, we have that $|\Gamma(S)| \geq (1-\epsilon')/\epsilon' |S| > (1-\epsilon')k$ by the defining property of the $((1-\epsilon')/\epsilon', \epsilon'/2, c)$-expander graph used to implement the $\epsilon'$-halving algorithm. According to the algorithm, for any element $x$ whose position is in $\Gamma(S)$, there must exist an element $y$ whose position is in $S$, such that $x < y$. This means that any element whose position is in $\Gamma(S)$ is less than $k+1$. It follows that the total number of elements less than $k+1$ is greater than $|S| + |\Gamma(S)| = (1-\epsilon')k + \epsilon' k = k$. However, There are at most $k$ elements less than $k+1$ (*i.e* $1, 2, ..., k$). This is a contradiction. □

## 3.2 Matching

The problem of determining whether or not there exists a 1-factor in a graph is NP-complete [Smi93]. However, for bipartite graphs, we can determine the existence of 1-factors in polynomial time. Lovasz and Plummer [LP86], gives an algorithm to find the 1-factors (or a perfect matching) of a bipartite graph.

**Definition 3.4** *A set of edges $M$ in a graph $G$ is a matching if no two edges have a vertex in common.*

*A vertex $v$ is said to be matched by $M$ if some edge of $M$ is incident on $v$.*

*A matching is perfect if it matches all vertices of $V(G)$. Among all matchings of a graph, any one with the maximum cardinality is a maximum matching.*

A path in a graph $G$ is an alternating sequence of vertices and edges. Here we use various notations of paths to compute matchings. Let $M$ be a matching. The following notation is used in Lovasz and Plummer's explanation.

**Definition 3.5** *A path $P$ is said to be an $M$-alternating path if the edges of $P$ are alternately in and not in $M$. If an $M$-alternating path $P$ begins and ends with vertices not in the matching $M$, then we call $P$ an $M$-augmenting path.*

If given an $M$-augmenting path $P$, we can construct a larger matching $M'$ which covers all the vertices of $M$ and the two extra vertices at the beginning and ending of $P$.

**Theorem 3.2** *Let $M$ be a matching in a graph $G$. Then $M$ is a maximum matching if and only if there exists no augmenting path in $G$ relative to $M$.*

**Proof**: Assume that there exists an $M$-augmenting path $P$. By the definition, such a path has $n$ edges in the matching $M$ and $n+1$ edges outside of the matching. By removing the $n$ edges from $P$ in the matching, and adding the $n+1$ other edges from $P$ to $M$, we get a matching $M'$ that has more edges than the original. Therefore, $M$ is not a maximum matching.

If $M$ is not the maximum matching, then there exists another matching $M'$, $|M'| > |M|$. Let $H$ be the graph with all the vertices and edges incident with $M$ or $M'$. Since each vertex is at most incident with 1 edge in $M$ or 1 edge in $M'$, the degrees of all the vertices in $H$ are 1 or 2. Each strongly connected component[1] in $H$ must be even cycles or alternating paths. Since $|M'| > |M|$, there are more edges belonging to $M'$ in $H$ than belonging to $M$. Therefore, at least a strongly connected component is an alternating path, and in the path there are more edges belonging to $M'$ than to $M$. This means that the beginning and ending of the path must be in $M'$ and not in $M$. We get an $M$-augmenting path. □

[1]A strongly connected component is a maximal set of vertices $U \subseteq V$ such that for every pair of vertices $u$ and $v$ in $U$, vertices $u$ and $v$ are reachable from each other.

**Theorem 3.3** *Given a bipartite graph* $G = ([V_1, V_2], E)$, *there exists a matching which covers all vertices in* $V_1$ *if and only if for all sets* $S \subseteq V_1$, $|\Gamma(S)| \geq |S|$.

**Proof**: If there exists a matching $M$ which covers all the vertices in $V_1$, then for all sets $S \subseteq V_1$, all the vertices in $S$ are matched. All the neighbors of $S$ must be in $V_2$ and not in $V_1$. Therefore, $|\Gamma(S)| \geq |S|$.

Assume that for any $S \subseteq V_1$, $|\Gamma(S)| \geq |S|$ and that there is no matching which covers all the vertices of $V_1$. Assume $M^*$ is a maximum matching in $G$ . Let $u$ be a vertex in $V_1$ which is not matched by $M^*$. Construct an $M^*$-alternating path on $G$ that begins with $u$. (Notice that this is possible since $|\Gamma(S)| \geq |S|$). Let $Z$ be the set of vertices along this path. From the previous theorem, we know that $u$ is the only vertex in $Z$ that is not matched by $M^*$. Let $S = Z \cap V_1$ and let $T = Z \cap V_2$. All the vertices in $S - \{u\}$ are matched with the vertices in $T$ by $M^*$. Thus, we have that $|T| = |S| - 1$.

Now, if $|\Gamma(S)| > |T|$ then there exists an augmenting path; namely, start at $u$ and follow the original path until you hit a member of $S$ that is connected to an element outside of $T$. This path must be an $M^*$-augmenting path because it begins and ends with vertices that are not matched. This is a contradiction. It follows that $|\Gamma(S)| = |T|$. (Notice that $T$ is the subset of the set of the neighbors of $S$; that is $|\Gamma(S)| \geq T$.)

So, $|\Gamma(S)| = |S| - 1$. This is a contradiction with $|\Gamma(S)| \geq |S|$. □

**Corollary 3.1** *If* $k > 0$ *then there exists a perfect matching in every* $k$*-regular bipartite graph* $G = ([V_1, V_2], E)$.

**Proof**: Assume that $G = ([V_1, V_2], E)$ is a $k$-regular bipartite graph. Then we have that $|V_1| = |V_2|$.

It is always that

$$\sum_{x \in S} degree(x) \leq \sum_{x \in \Gamma(S)} degree(x).$$

Let $S \subseteq V_1$. Since $G$ is a $k$-regular graph, we have the case that

$$\sum_{x \in S} degree(x) = |S| \cdot k \leq \sum_{x \in \Gamma(S)} k = |\Gamma(S)| \cdot k.$$

It follows that $|S| \leq |\Gamma(S)|$. In this case, Theorem 3.4 applies. Therefore, $G$ has a matching which covers all vertices in $V_1$. Considering $|V_1| = |V_2|$, such a matching will also cover all vertices in $V_2$. □

Given a bipartite graph $G = ([V_1, V_2], E)$, the following algorithm will find a perfect matching if one exists.

**Algorithm 3.2 ([Edm65])** *Hungarian Algorithm for bipartite matching:*

1. *Start with any matching $M$;*
2. *If $M$ matches all the vertices in $V_1$, stop; Otherwise pick $u \in V_1$, such that $u$ isn't matched by $M$. Let $S = \{u\}, T = \emptyset$;*
3. *If $|\Gamma(S)| = T$, there is no perfect matching, stop; Otherwise pick $y \in \Gamma(S) - T$;*
4. *If $y$ is matched by $M$ , assume $yz \in M$, then let $S \leftarrow S \cup \{z\}, T \leftarrow T \cup \{y\}$, goto 3; Otherwise, pick augmenting path from $u$ to $y$, $P(u, y)$, let $M \leftarrow M \cup E(P) - M \cap E(P)$, goto 1.*

The algorithm creates an initial matching and successively augments this matching by using augmenting paths.

The matching which covers all the vertices in $V_1$ is maximal if it exists. From Theorem 3.3, if such a matching exists, we can construct a larger matching from an initial matching $M$ with an augmenting path beginning from any vertex $u$ in $V_1$ which is not matched by $M$.

In the algorithm, the sets $S$ and $T$ are subsets of $V_1$ and $V_2$, respectively. All the vertices in $S$ and $T$ are reachable from $u$ through an $M$-alternating path. If $\Gamma(S) = T$, there is no vertex in $V_2 - T$ which is reachable from $u$ through an $M$-alternating path. To see this, assume that there exists a vertex $v \in V_2 - T$ which is reachable through a $M$-alternating path $P$ from $u$. Since $u \in S$, the path contains some vertices of $V_1$. Pick the nearest vertex of $V_2$ to $u$ which is not in $T$ in path $P$ and let it be $v_0$. Obviously $v_0 \in \Gamma(S)$, which means $|\Gamma(S)| > T$. This is a contradiction.

If $\Gamma(S) = T$, it is impossible to find an $M$-augmenting path from $u$. This means that the graph has no such matching. If we can find a vertex $y \in \Gamma(S) - T$ and $y$ is not matched by $M$, then $P(u, y)$ is an $M$-augmenting path. Therefore, we can obtain a larger matching.

Since in each iteration from step 1 to step 4, we obtain a larger matching with at least one more vertex from $V_1$, at most $|V_1|$ iterations are needed. In addition, since $y \in V_2$, at most $|V_2|$ iterations are needed to verify if an augmenting path exists. Thus, the algorithm will finish in $O(|V_1||V_2|)$ steps.

## 3.3 Explicit Construction

From Algorithm 3.1 and its proof, it can be seen that the construction of an $\epsilon$-halver involves finding a bipartite expander graph $G = ([V_1, V_2], E)$ with parameters $((1 - \epsilon')/\epsilon', \epsilon'/2, c)$ such that $c$ is as small as possible. (The sizes of the sets $V_1$

and $V_2$ must be equal.) As mentioned in Chapter 2, we can construct non-bipartite Ramanujan graphs which have very good expansion properties. Here, we describe a way to construct balanced bipartite graphs which have the same expansion properties as those non-bipartite graphs. In a related work, Spielman and Sipser [SS94, SS96] used edge-vertex incidence graphs to obtain unbalanced bipartite expander graphs which they later used to construct computationally efficient error-correcting codes. Here we take the following approach to get balanced bipartite graphs.

**Algorithm 3.3** *Given a graph with n-vertices,* $G = (V, E)$, $V = \{v_1, v_2, ..., v_n\}$, *construct a* $2n$*-vertex graph* $G_b = ([U, W], E_b)$ *with vertices* $U = \{u_1, u_2, ..., u_n\}$, *and* $W = \{w_1, w_2, ..., w_n\}$. *The edges of* $G_b$ *satisfy that, for positive integers* $i \leq n$, $j \leq n$,

$$(v_i, v_j) \in E \Leftrightarrow (u_i, w_j) \in E_b.$$

**Lemma 3.1** *The bipartite graph* $G_b$ *has the same expansion coefficient as* $G$. *That is, if* $G = (V, E)$ *is an* $(\lambda, \alpha, c)$ *expander, then* $G_b = ([U, W], E_b)$ *is an* $(\lambda, \frac{\alpha}{2}, c)$ *expander.*

**Proof**: Notice that graph $G_b$ is bipartite since the endpoints of each edge lie in $U$ and $W$ respectively and $U \cap W = \emptyset$.

Assume $S_u$ is a subset of $U$ of size $m \leq \frac{\alpha}{2}2n = \alpha n$, $S_u = \{u_{s_1}, u_{s_2}, ..., u_{s_m}\}$. Since $G$ is an expander, we know that $\forall S_v \subseteq V$, with $|S_v| = m \leq \alpha n$, $|\Gamma(S_v)| \geq \lambda|S_v|$. Let $S_v = \{v_{s_1}, v_{s_2}, ..., v_{s_m}\}$. Its neighbors are $\Gamma(S_v) = \{v_{n_1}, v_{n_2}, ..., v_{n_\gamma}\}$, where $|\Gamma(S_v)| \geq \lambda|S_v|$. For any vertex $v_{n_j} \in \Gamma(S_v) \subseteq V$ $(j = 1, 2, ..., \gamma)$, there exists a vertex $v_{s_i} \in S_v \subseteq V$, such that $(v_{s_i}, v_{n_j}) \in E$.

Consider the vertex set $W_n = \{w_{n_1}, w_{n_2}, ..., w_{n_\gamma}\}$. For any vertex $w_{n_j}$ $(j = 1, 2, ..., \gamma)$, there exists $u_{s_i} (1 \leq i \leq m)$ such that $(u_{s_i}, w_{n_j}) \in E_b$. Thus $W_n \subseteq \Gamma(S_u)$.

We obtain $|\Gamma(S_u)| \geq |W_n| \geq \lambda|S_v| = \lambda|S_u|$. Similarly, we can prove that for any subset of $W$ of size $\leq \alpha n$ $S_w$, $|\Gamma(S_w)| \geq \lambda|S_w|$. Finally, it is clear from the construction that the degree of any vertex of $G_b$ is $\leq c$. □

From Lemma 3.3, we see that we can construct an $\epsilon$-halver by constructing a non-bipartite expander. First we construct $((1-\epsilon)/\epsilon, \epsilon, c)$ expander with $c$ as small as possible. Since Ramanujan graphs are good expanders, we construct $((1-\epsilon)/\epsilon, \epsilon, c)$ expanders with Ramanujan graphs.

For our purpose, first, we need to find a graph, for any subset $X$, $|X| = \epsilon n$, the number of the neighbor of $\Gamma(X) \geq (1-\epsilon)n$. From Tanner's and Kahale's results, we get

$$\frac{|\Gamma(X)|}{n} \geq \frac{k^2\frac{|X|}{n}}{\lambda^2 + (k^2-\lambda^2)\frac{|X|}{n}} = \phi_1(k,\lambda),$$

and

$$\frac{|\Gamma(X)|}{n} \geq \frac{\epsilon(k^2 + (k^2-\lambda^2)^2/(\lambda^2 k))}{\lambda^2 + k^2(k^2-\lambda^2)|X|/(\lambda^2 n)} = \phi_2(k,\lambda),$$

respectively. Let $g(k) = \max_\lambda(\phi_1(k,\lambda), \phi_2(k,\lambda))$.

If we find a integer $K$ such that for all $k \geq K$, $g(k) \geq (1-\epsilon)$, then we can construct an expander with degree $K$ which satisfies that for any subset $X$, $|X| = \epsilon n$, the number of the neighbors of $X$ is $|\Gamma(X)| \geq (1-\epsilon)n$. We can verify that for any given $k$, $\phi_1(k,\lambda)$ and $\phi_2(k,\lambda)$ are decreasing functions of $\lambda$. On the other hand, we can only construct a graph with $\lambda \geq \lambda_{rk} = 2\sqrt{k-1}$. We have for any given $k$,

$$\phi_1(k,\lambda_{rk}) \geq \phi_1(k,\lambda) \quad \text{for } k > \lambda \geq \lambda_{rk}$$

and

$$\phi_2(k,\lambda_{rk}) \geq \phi_2(k,\lambda) \quad \text{for } k > \lambda \geq \lambda_{rk}.$$

We obtain,

$$g(k) = \max(\phi_1(k, \lambda_{rk}), \phi_2(k, \lambda_{rk}))$$

We first analyze the function $g(k)$. Let $\alpha = k^2/\lambda^2$ and plug it into $\phi_1(k, \lambda)$ and $\phi_2(k, \lambda)$ respectively,

$$\phi_1(k, \lambda) = \frac{\epsilon\alpha}{1 + (\alpha - 1)\epsilon}$$

and

$$\phi_2(k, \lambda) = \frac{\alpha + (\alpha - 1)^2/k}{1 + \alpha(\alpha - 1)\epsilon}\epsilon,$$

Therefore,

$$\phi_1(k, \lambda) - \phi_2(k, \lambda) = \frac{\epsilon(\alpha - 1)^2(\alpha\epsilon - (\alpha - 1)\epsilon/k - 1/k)}{(1 + (\alpha - 1)\epsilon)(1 + \alpha(\alpha - 1)\epsilon)}.$$

Considering $\alpha > 1$,if $\alpha\epsilon - \frac{1}{k} - \frac{\alpha-1}{k}\epsilon \geq 0$, then above formula is $\geq 0$. This formula is equal to $\alpha\epsilon(k - 1) \geq 1 - \epsilon$. Considering that the two functions are decreasing, the maximum must occur when $\lambda = 2\sqrt{k - 1}$. If

$$\frac{k^2\epsilon}{4} \geq (1 - \epsilon),$$

or

$$k \geq 2\sqrt{\frac{1 - \epsilon}{\epsilon}},$$

then $\phi_1(k, \lambda_{rk}) \geq \phi_2(k, \lambda_{rk})$. Otherwise, $\phi_1(k, \lambda_{rk}) < \phi_2(k, \lambda_{rk})$.

Therefore,

$$g(k) = \begin{cases} \phi_1(k, \lambda_{rk}), & \text{when } k \geq 2\sqrt{\frac{1-\epsilon}{\epsilon}}; \\ \phi_2(k, \lambda_{rk}), & \text{when } k < 2\sqrt{\frac{1-\epsilon}{\epsilon}}. \end{cases}$$

It is not difficult to verify that if $k < 2\sqrt{\frac{1-\epsilon}{\epsilon}}$, we cannot guarantee that the graph satisfies the conditions to be an $\epsilon$ -halver. So, the minimum $K$ must be greater than this value. It follows that for $k \geq K$, we have $\phi_1 \geq \phi_2$. This means that by using

Kahale's result we cannot get a better result than Tanner's. Therefore, Kahale's result is only effective for very small subsets $|X|$.

We arrive at the following conclusions.

**Lemma 3.2** *Let*

$$K = \frac{2(1-\epsilon)(1-\epsilon+\sqrt{1-2\epsilon})}{\epsilon^2}.$$

*For every k-regular LPS graph, if $k \geq K$ then every subset $X$ of size $\epsilon n$ has at least $(1-\epsilon)n$ neighbors.*

**Proof**: From Tanner's formula, we hope that

$$\frac{|\Gamma(X)|}{n} \geq \frac{k^2}{\lambda^2 + (k^2-\lambda^2)\frac{|X|}{n}} = \frac{k^2\epsilon}{\lambda^2 + (k^2-\lambda^2)\epsilon} \geq (1-\epsilon).$$

This is equal to

$$(1-\epsilon)^2\lambda^2 \leq \epsilon^2 k^2.$$

we already know that for LPS graphs, $\lambda \leq 2\sqrt{k-1}$. For LPS graph, it is enough if

$$4(1-\epsilon)^2(k-1) \leq \epsilon^2 k^2$$

or

$$\epsilon^2 k^2 - 4(1-\epsilon)^2 k + 4(1-\epsilon)^2 \geq 0.$$

Considering that $k > 0$, we need

$$k \geq \frac{2(1-\epsilon)(1-\epsilon+\sqrt{1-2\epsilon})}{\epsilon^2}.$$

The conditions will suffice. □

**Lemma 3.3** *If $k \geq K$, k-regular LPS graph is an $(\frac{1-\epsilon}{\epsilon}, \epsilon, k)$ expander, where $K$ is the same constant as Lemma 3.2.*

**Proof**: Assume that $S$ is a subset of vertices, where $|S| \leq \epsilon n$. From Tanner's result,

$$|\Gamma(S)| \geq \frac{k^2|S|}{\lambda^2 + (k^2 - \lambda^2)\frac{|S|}{n}} = \frac{k^2\epsilon}{\lambda^2 + (k^2 - \lambda^2)\epsilon\frac{|S|}{\epsilon n}} \frac{|S|}{\epsilon}.$$

Notice that $|S|/(\epsilon|n|) \leq 1$, and

$$\Gamma(S) \geq \frac{k^2|S|}{\lambda^2 + (k^2 - \lambda^2)\epsilon} \frac{|S|}{\epsilon} = \frac{1-\epsilon}{\epsilon}|S|.$$

Thus, we obtain the conclusion. □

From number theory, for any integer $m \geq 13$, we can find a prime congruent to 1 mod 4 such that $m \leq q \leq 2m$.

The following algorithm will construct an $\epsilon$-halver on $n$ elements explicitly.

**Algorithm 3.4** *1. Find the minimum $K$ for an $\epsilon$-halver.*

2. *Pick the minimum prime $p$ congruent to 1 mod 4($\geq K - 1$). Let $k = p + 1$.*

3. *Find the minimum prime $q$ congruent to 1 mod 4, such that $q \geq n/2$.*

4. *Construct a $k$-regular Ramanujan graph with $q$ vertices.*

5. *Construct a balanced bipartite $k$-regular graph $G_B$ with $2q$ vertices.*

6. *Find the one factor of the graph using Hungarian algorithm; remove the 1-factor from the graph to get a $k-1$ regular graph; recursively find the remaining $k-1$ 1-factors.*

**Theorem 3.4** *We can construct an $\epsilon$-halver with depth of $K$, where*

$$K = \frac{2(1-\epsilon)(1-\epsilon+\sqrt{1-2\epsilon})}{\epsilon^2}.$$

**Proof**: From Lemma 3.2, Lemma 3.3, and Algorithm 3.4, we obtain the conclusion. □

If $\epsilon = 1/72$ we can get $k \geq 20164$. Since that $p$ must be prime congruent 1 mod 4, we choose $p = 20173$. Therefore, we can construct a $1/72$ -halver in less than 20174 steps.

# Chapter 4

# Separators

The replacement of the nearsort in the original AKS algorithm by a *separator* was Paterson's major improvement to the algorithm. In this chapter, we examine Paterson's separators. Section 4.1 is devoted to describing the basic definitions and algorithms for separators. In Section 4.2, we analyze a new kind of separator that has the potential to achieve better separation. In Section 4.3, we provide the parameters of that separator.

## 4.1 $(\lambda, \epsilon, \epsilon_0)$- separators

**Definition 4.1** *Given a set of* $m$ *elements* $S = \{a_i | 1 \leq i \leq m\}$ *such that* $a_1 \leq a_2 \leq ... \leq a_{m-1} \leq a_m$ *and a permutation of the elements of* $S$, $\Pi_S = a_{\pi_1}, a_{\pi_2}, ..., a_{\pi_m}$ *such that whenever* $a_{\pi_i} = a_{\pi_j}$ *and* $i < j$, *then* $\pi_i < \pi_j$. *Define* $\pi_L = \{\pi_1, \pi_2, ..., \pi_{m/2}\}$ , $\pi_{FL} = \{\pi_1, \pi_2, ..., \pi_{\lambda m/2}\}$, $\pi_R = \{\pi_{m/2+1}, \pi_{m/2+2}, ..., \pi_m\}$, *and* $\pi_{FR} = \{\pi_{m-\lambda m/2+1}, \pi_{m-\lambda m/2+2}, ..., \pi_m\}$.

*The permutation* $\Pi_S$ *is* $(\lambda, \epsilon, \epsilon_0)$*-separated if for* $1 \leq k \leq m/2$,

$$|\{x | 1 \leq x \leq k, x \in \pi_R\}| \leq \epsilon_0 k$$

*and*

$$|\{x|m-k+1 \leq x \leq m, x \in \pi_L\}| \leq \epsilon_0 k$$

*and for* $1 \leq k \leq \lambda m/2$,

$$|\{x|1 \leq x \leq k, x \notin \pi_{FL}\}| \leq \epsilon k$$

*and*

$$|\{x|m-k+1 \leq x \leq m, x \notin \pi_{FR}\}| \leq \epsilon k.$$

*A procedure is called a* $(\lambda, \epsilon, \epsilon_0)$*-separator (of* $m$ *elements) if it accepts a permutation of* $S$ *as input, and produces an* $(\lambda, \epsilon, \epsilon_0)$*-separated permutation of* $S$ *as output.*

This formal definition above requires a bit of explanation. Very briefly, a $(\lambda, \epsilon, \epsilon_0)$-**separator** on $m$ elements partitions its input values into two main parts $L$ ("left") and $R$ ("right") of size $m/2$, and two subparts, $FL$ ("far left") and $FR$ ("far right") of size $\lambda m/2$. The set $L$ satisfies the condition that the number of elements from the set of the $k$ smallest input values which are not in $L$ is less than $\epsilon_0 k$. Similarly, the set $R$ satisfies the condition that the number of elements from the set of the $k$ largest input values which are not in $R$ is less than $\epsilon_0 k$. The set $FL$ satisfies the condition that the number of element from the set of $k$ smallest input values which are not in $FL$ is less than $\epsilon k$. The same holds for $FR$ with respect to the largest input values.

The following algorithm provides a $(\lambda, \epsilon, \epsilon_0)$-separator(Figure 4.1).

**Algorithm 4.1** *We build a* $(2^{-p+1}, p\epsilon_0, \epsilon_0)$ *separator in stages as follows. At stage 0, we apply an* $\epsilon_0$*-halver on the* $m$ *inputs. In stage* $i$*, we have* $2^i$ *blocks of size* $m/2^i$*, and we apply an* $\epsilon_0$ *-halver to each block. At the end of the* $(p-1)$*th stage, the* $p$

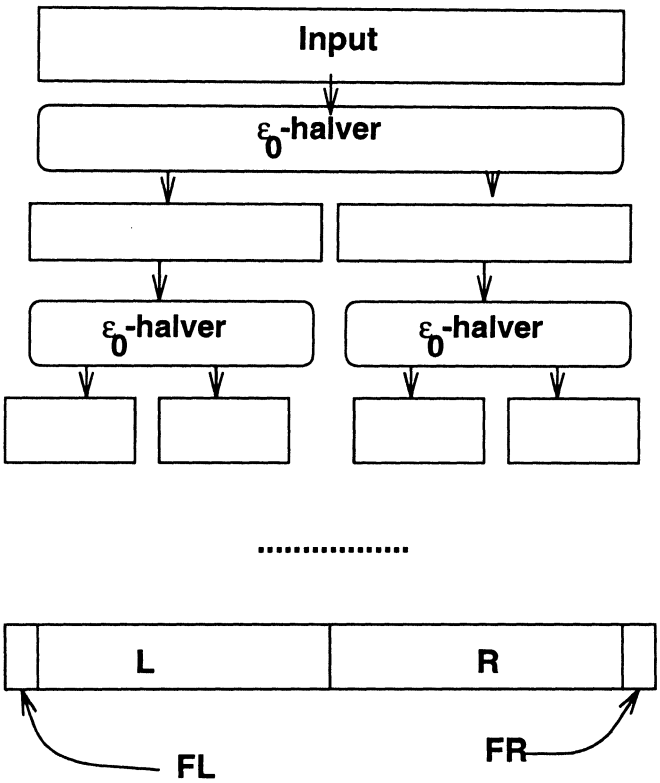


Figure 4.1: A $(\lambda, \epsilon, \epsilon_0)$-Separator

*levels of halvers produce a sequence of* $2^p$ *blocks, each of size* $m/2^p$. *The resulting network yields a* $(2^{-p+1}, p\epsilon_0, \epsilon_0)$*-separator since each block has size* $m/2^p = \lambda \cdot m/2$. *The extreme blocks are taken for* $FL$ *and* $FR$, *while the left and right halves of the sequence form* $L$ *and* $R$, *respectively.*

For example, if $p = 4$ and $\epsilon_0 = 1/72$, then we get a $(\lambda = 1/8, \epsilon = 1/18, \epsilon_0 = 1/72)$ separator. The correctness of the algorithm can be seen through the following analysis. The first stage uses an $\epsilon_0$-halver on the $m$ inputs, which produces partitions $L$ and $R$ with the desired properties. Subsequent stages refine the partitions $L$ and $R$. In stage $i$, the left most block contains $m/2^{i-1}$ elements. Of the $k(k < m/2^{i-1})$ smallest elements, at most $i \cdot \epsilon_0 \cdot k$ of those elements are not in the left most block. By

applying an $\epsilon_0$ halver to this block, we get that at most $i \cdot \epsilon_0 \cdot k + \epsilon_0 \cdot k = (i+1) \cdot \epsilon_0 \cdot k$ of the $k$ smallest elements are not in the new left most block. This means that at the $p^{th}$ level at most $p\epsilon_0 k$ elements are not in $FL$. A similar analysis apples to the right half.

## 4.2 Construction of the New Separator

In this section, we describe a new kind of separator that has been mentioned in [Pat90] and [CO86]. Here we find the general parameters for these separators.

In our original definition of the separator, it is assured that for any $k$, $k \leq m/2$, the number of elements from the set of $k$ leftmost input values which are not in $FL$ or $L$ is less than $\epsilon_0 k$, and similarly for elements from the right half of the ordering which end up not in $FR$ or $R$. In fact, we will only need to assure that the above constants hold for $k = m/2$ since we don't care about the relative position among these elements in Paterson's algorithm. From this observation, we present a method that produces a better separator. The separator is constructed as follows. First, we perform $p-1$ stages of $\epsilon_0$-halver operations that yields $2^{p-1}$ segments. Then, we combine the middle two segments and perform an $\epsilon_0$-halver operations on the resulting set. We also perform an $\epsilon_0$ halver on each of the extreme segments.

The following is the accurate description of the algorithm(Figure 4.2).

**Algorithm 4.2** *(1) Apply an $\epsilon_0$-halver to the $m$ inputs. This produces a left set $L_1$ and right set $R_1$, each size of $m/2$.*

*(2) Apply an $\epsilon_0$ -halver on the sets $L_1$ and $R_1$, which partitions the sets $L_1$ and $R_1$ into four parts, $FL_2$, $CL_2$, $CR_2$ and $FR_2$, respectively.*

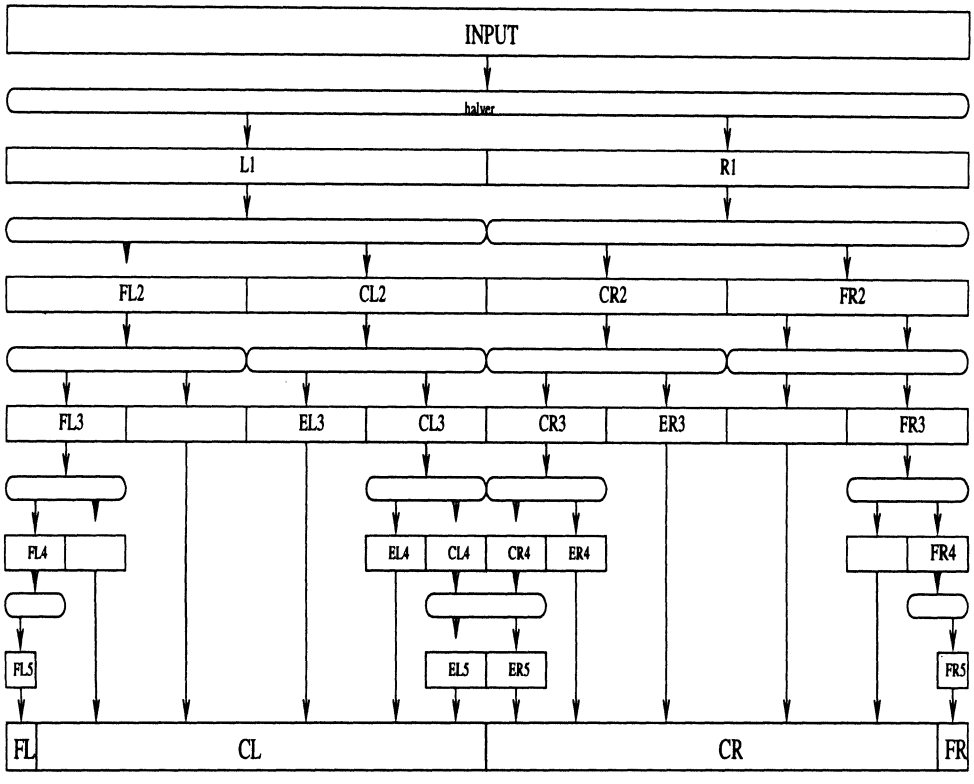


Figure 4.2: Improved Separator

*(3) Apply an $\epsilon_0$- halver on $FL_{k-1}$ and $CL_{k-1}$, respectively, for $k = 3, ..., p-1$. Let the extreme left set generated from $FL_{k-1}$ ($FR_{k-1}$) be $FL_k$ ($FR_k$). Similarly, let $EL_k$ ($CR_k$)and $CL_k$ ($ER_k$) be the left and right sets generated from $CL_{k-1}$ ($CR_{k-1}$).*

*(4) Combine the sets $CL_{p-1}$ and $CR_{p-1}$ and apply an $\epsilon_0$- halver on the set. We get the left set $EL_p$ and right set $ER_p$. Also, apply an $\epsilon_0$-halver to both $FL_{p-1}$ and $FR_{p-1}$. Let the extreme left part from $FL_{p-1}$ be labeled as $FL_p$ and let the extreme right part from $FR_{p-1}$ be labeled as $FR_p$.*

## 4.3 Parameter of the New Separator

We use the natural location of an element as the location of the element if the elements are sorted. We define $L$ parts to be the combination of the $CL$ and $FL$ parts and $R$ to be the combination of the $CR$ and $FR$ parts.

To get the parameters of the separators, we try to find an upper bound on the number of elements in $L$ whose natural locations are in $R$.

Let the number of elements in $L_1$ whose natural location are in $R$ be $l_1$, $l_1 = \epsilon_{l1} m/2 \leq \epsilon_0 m/2$. In the set $FL_2$, the number of elements whose natural location are on $R$ is $f_{l2} = \epsilon_{l2} l_1 = \epsilon_{l2}\epsilon_{l1} m/2$.

The number of such elements in $CL_2$ is

$$c_{l2} = (1 - \epsilon_{l2})\epsilon_{l1} m/2.$$

Partition $CL_{i-1}$ into two sets $EL_i$ and $CL_i$. The number of such elements in $EL_i$ is $e_{li} = \epsilon_{li} c_{l(i-1)} \leq \epsilon_0 c_{l(i-1)}$ and the number of such elements in $CL_i$ is

$$c_{li} = (1 - \epsilon_{li}) c_{l(i-1)} = (\prod_{j=2}^{i} (1 - \epsilon_{lj}))\epsilon_{l1} m/2$$

for $i = 3, ..., p-1$.

Similarly, we label the extreme left set in $R$ by $CR_{p-1}$. Let the number of elements in $CR_{p-1}$ whose natural locations are in $L$ as $c_{r(p-1)}$.

When we combine $CL_{p-1}$ and $CR_{p-1}$ and apply an $\epsilon_0$ -halver on this set, we get two sets $EL_p$ and $ER_p$. Let the number of elements in $EL_p$ whose natural locations are on $R$ be $e_{lp}$, and let $e_{rp}$ be the number of elements in $ER_p$ whose natural location are in $L$.

Obviously the number of elements in $L$ whose natural locations are in $R$ is

$$l = l_1 - c_{l(p-1)} + e_{lp} = \epsilon_{l1} m/2 - (\prod_{j=2}^{p-1}(1-\epsilon_{lj}))\epsilon_{l1} m/2 + e_{lp}.$$

The value $l$ is our measure of separation. We now try to bound the value of $l$.

**Lemma 4.1** *The value of $l$ reaches its maximum value only if $c_{l(p-1)} = c_{r(p-1)}$. In this case, $e_{lp} \leq \epsilon_0 m/2^{p-1}$.*

**Proof**: We give the proof by contradiction. Assume that $c_{l(p-1)} > c_{r(p-1)}$ and that $l$ is maximal. In the set $CL_{p-1} \cup CR_{p-1}$, the number of elements whose natural locations are in $L$ is $m/2^{p-1} - c_{l(p-1)} + c_{r(p-1)} < m/2^{p-1}$. After applying the $\epsilon_0$-halver, at most $\epsilon_0(m/2^{p-1} - c_{l(p-1)} + c_{r(p-1)})$ of such elements enter $R$. Therefore,

$$\begin{aligned} e_{lp} &\leq \epsilon_0(m/2^{p-1} - c_{l(p-1)} + c_{r(p-1)}) + (c_{l(p-1)} - c_{r(p-1)}) \\ &= \epsilon_0 m/2^{p-1} + (1-\epsilon_0)(c_{l(p-1)} - c_{r(p-1)}). \end{aligned}$$

Let $x = c_{l(p-1)} - c_{r(p-1)}$. Then, we get

$$\begin{aligned} l &= l_1 - c_{l(p-1)} + e_{lp} \leq l_1 - c_{l(p-1)} + \epsilon_0 m/2^{p-1} + (1-\epsilon_0)x \\ &= l_1 - c_{r(p-1)} + \epsilon_0 m/2^{p-1} - \epsilon_0 x < l_1 - c_{r(p-1)} + \epsilon_0 m/2^{p-1}. \end{aligned}$$

So, if we let $c_{l(p-1)} = c_{r(p-1)}$, then we get a new value of $l$ that is larger than the original. This is a contradiction to our assumption that $l$ is maximal. Therefore, we get a maximum value of $l$ when $c_{r(p-1)} \geq c_{l(p-1)}$. Moreover, we obtain $e_{lp} \leq \epsilon_0 m/2^{p-1}$.□

**Theorem 4.1** *The algorithm 4.2 produces an $(2^{-p+1}, p\epsilon_0, \epsilon)$ -separator, where*

$$\epsilon = \epsilon_0(1-(1-\epsilon_0)^{p-2}) + \epsilon_0/2^{p-2}.$$

**Proof**: Algorithm 4.2 produces a $(2^{-p+1}, \epsilon', \epsilon)$ - separator. As in section 4.1, we get that $\epsilon' = p\epsilon_0$.

From Lemma 4.1, we have that

$$l \leq \epsilon_{l1} m/2(1 - \prod_{i=2}^{p-1}(1 - \epsilon_{li})) + \epsilon_{lp} m/2^{p-1} \leq \epsilon_0 m/2(1 - (1 - \epsilon_0)^{p-2}) + \epsilon_0 m/2^{p-1}.$$

since $\epsilon_{li} \leq \epsilon_0$ for $i = 1, ..., p$.

If we let $\epsilon = \epsilon_0(1 - (1 - \epsilon_0)^{p-2}) + \epsilon_0/2^{p-2}$, then we get a $(2^{-p+1}, p\epsilon_0, \epsilon)$ separator.

□

In the case when $p = 4$ and $\epsilon_0 = 1/72$, we get that $\epsilon$ is $1439/373248 (\approx= 0.00386) < 1/259 < 1/72$. Therefore, we produced a $(1/8, 1/18, 0.00386)$ separator, while the original algorithm produced only a $(1/8, 1/18, 1/72)$-separator. Notice that both separators have the same depth, yet the latter one produces a better separation.

In the next step, we will use the new separator to find parameters which make that all the constraints of Paterson's algorithm hold and allow a larger $\epsilon_0$. It is well-known that a $\epsilon_0$-halver can be constructed in less depth if $\epsilon_0$ is larger.

# Chapter 5

# Paterson's Algorithm

As mentioned earlier, the AKS algorithm [AKS83b, AKS83a] is a constructive proof for the existence of $O(\log n)$ depth sorting networks. The idea behind the AKS algorithm is to take an approximate partition of elements, which can be achieved in only a constant number of comparator levels, and to introduce some error-recovery structures into the sorting scheme. The construction and proof are of some intricacy and the numerical constant in the depth bound is enormous.

In this chapter, we present a sorting algorithm developed by Paterson[Pat90] which improves upon the original AKS algorithm. Paterson's algorithm is a simplification of the basic construction which allows for a more accessible proof. Paterson's algorithm greatly reduces the constant in the $O(\log n)$ bound on the depth of the sorting network. Using a probabilistic construction, Paterson proved that there is a sorting network with depth under $6100 \log n$.

In the last part of the chapter, we derive the complexity of Paterson algorithm.

## 5.1 A Tree-sort Metaphor

Cole and O'Dunlaing [CO86] described Paterson's algorithm through a tree-sort metaphor. Let the input be of size $n$. We sort by using a binary tree $T$ with $n$ leaves and depth $\log n$. During the progression of the algorithm, internal nodes of the sorting tree, contain sets of elements. And at the end of the algorithm , each leaf contains a single element. A left-to-right traversal of the leaves of $T$ will yield the elements in ascending order.

Paterson's algorithm begins with all elements in a set at the root of the tree $T$. The algorithm works by redistributing the elements down the tree until each element is attached to the correct leaf of $T$. The algorithm proceeds in $\log n$ parallel stages. At each stage the elements are split correctly into two parts. In each step in a stage, an approximate partition of elements in each node of the tree is taken, which can be achieved in only a constant number of comparator levels. Meanwhile, some error-recovery structures are introduced into the sorting scheme.

In the basic scheme, sorting with error-recovery is performed by partitioning a bag of elements at each node into four parts. These parts include the main left and right "halves", which are sent down to the child's bags, and two small fragments from the extreme ends of the partitioning process. These two small fragments are intended to include most of the elements that were wrongly routed from higher in the tree. These fragments are returned to the parent bag above.

At any one time, the sizes of the bags at any one level in the sorting tree are equal. The size of bags increases in geometric progression with the depth of level below the root. As time progresses in the sort, the size of each bag is reduced, again in a geometric progression.

The correctness of the network is demonstrated by proving an invariant which bounds the proportion, within each bag, of elements with any particular degree of displacement from their proper positions. If the number of such elements is $< 1$, then there is no such element in that bag.

## 5.2 The Parameters and Constraints

In the tree sort, each node in the tree contains a set of elements that we call a **bag**. The capacity of such a bag is the maximum number of elements that it can hold. The capacity of a bag at level $d$ will be $rA^d$ for some constant $A$ and a value $r$ that will decrease during the construction. Here, we will assume that the level of the root is 0. Therefore, the capacity of the root at time 0 is $r$.

At successive steps in the tree sort, either all the even levels or all the odd levels of the tree will be empty. Let $b$ be the capacity of a bag at level $i$. During a step of the sort, we will move $\lambda b$ elements from the bag to its parent, and we will move $(1-\lambda)/2 \cdot b$ elements to each of the bag's children.

If a bag with capacity $b$ is empty at stage $i$, then its parent has capacity $b/A$ and its children have capacity $bA$(Figure 5.1). Therefore, this bag receives

$$\nu b = 2\lambda bA + (1-\lambda)b/(2A).$$

elements in the next stage. We will use $\nu b$ as the capacity of this bag at the next stage. Since we desire that the capacities diminish during successive stages of the sort, we need that

$$\nu = 2\lambda A + (1-\lambda)/(2A) < 1. \tag{5.1}$$

For example, if $A = 3$ and $\lambda = 1/8$, then $\nu = 43/48$.

At the beginning of the algorithm, the capacity of the root is $cn$, and therefore the capacity of the each bag at level $d$ will be $cnA^d$. The capacity of each bag after stage $t$ is $\nu$ times the capacity of the bag after stage $t-1$. Therefore, the capacity of each bag at level $d$ after stage $t$ will be $cn\nu^t A^d$.

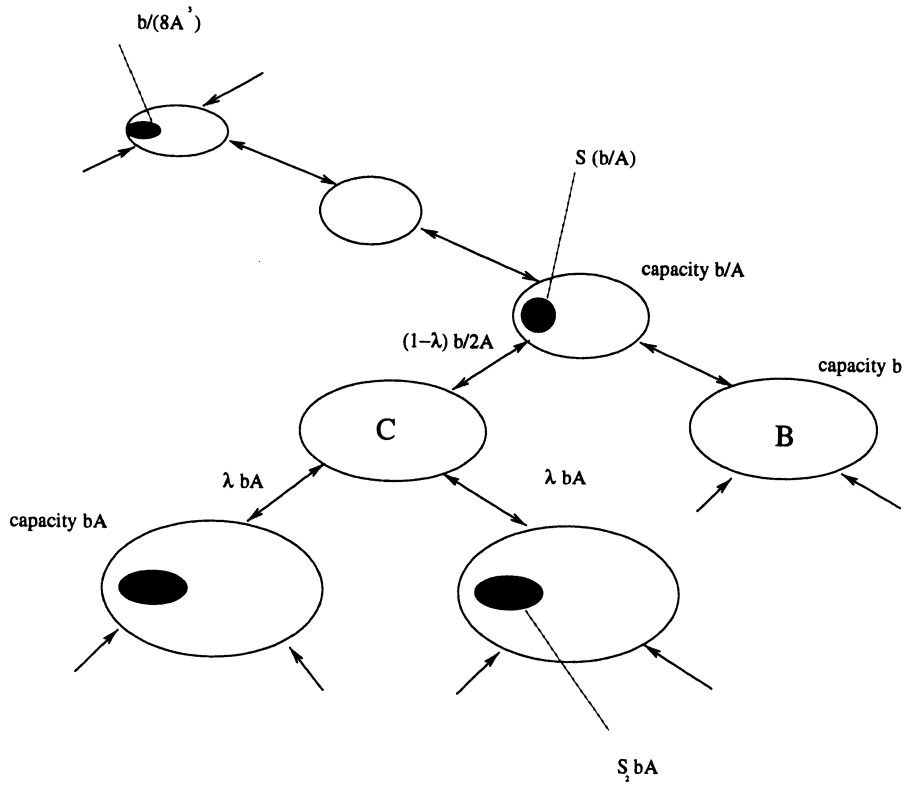


Figure 5.1: Paterson's Algorithm

There naturally corresponds an interval within the sorted order of the elements for each bag. The root bag corresponds to the whole ordered set, and its left and right child bags corresponds to the left and right halves of the ordering, respectively. At each stage, a $(\lambda, \epsilon, \epsilon_0)$ -separator is applied to each bag. We move the $FL$ and $FR$ parts to parent, and we send the $CL$ and $CR$ parts to the left and right child, respectively. Since the separation may contain errors, a bag may contain some ele-

ments whose natural locations are not in the natural interval of the bag. We call such elements **strangers**.

**Definition 5.1** *The strangeness of element is defined iteratively as follows.*

1. *All elements in the root bag have strangeness zero.*

2. *If an element has non-zero strangeness, then its strangeness is decreased by one if the element is sent up to the parent bag.*

3. *If an elements strangeness is zero and it is sent down to the correct child bag, its strangeness remains zero. Otherwise, its strangeness increases by one.*

**Definition 5.2** *For any bag $B$ and integer $j > 0$, define $S_j(B)$ to be the number of elements currently in $B$ with strangeness $j$ or more divided by the capacity of $B$.*

$S_j(B)$ is the proportion of the number of elements currently in $B$ with strangeness $j$ or more. To assure the correctness of our construction, we maintain that

$$S_j(B) < \mu\delta^{j-1} \text{ for all } B \text{ and for all } j \geq 1. \tag{5.2}$$

In our example, we will let $\mu = 1/36$ and $\delta = 1/40$.

For $j > 1$, the elements of strangeness $j$ or more in a bag $B$ come from two sources.

(1) The elements of strangeness $j + 1$ or more from the children of $B$.

(2) The elements of strangeness $j - 1$ or more which are wrongly sent down by the parent of $B$.

The number of such elements from (1) is less than or equal to the number of elements of strangeness $j+1$ or more in the two children of $B$, which is bounded by $2 \cdot bA \cdot \mu\delta^j$. Similarly, the number of elements in the parent of $B$ with strangeness $j-1$ or more is bounded by $b/A \cdot \mu\delta^{j-2}$.

We need to ensure that the $FL$ and $FR$ parts of the separators (which have size $\lambda/2$ of the capacity) are sufficiently large to accommodate the elements of positive strangeness so that at most a proportion $\epsilon$ of these elements are sent down. Since the number of these elements of positive strangeness is $\mu\delta^{1-1}$ of capacity, we need that

$$\mu \leq \lambda/2. \tag{5.3}$$

For example, if $\lambda = 1/8$, then a choice of $\mu = 1/36 \leq 1/(8 \times 2)$ suffices.

The natural location of any stranger in a bag should belong to the $FR$ or $FL$. According to the property of separator, at most a proportion $\epsilon$ of such elements in $FL$ are not in $FL$. The same holes for $FR$. Then the number of elements of strangeness $j-1$ or more wrongly sent down by the parent is $< \epsilon \cdot (b/A) \cdot \mu\delta^{j-2}$. So, we have that

$$S'_j(B)\nu b < 2bA\mu\delta^j + \epsilon(b/A)\mu\delta^{j-2} \text{ for } j > 1. \tag{5.4}$$

To ensure that both the above formula and $S'_j(B) < \mu\delta^{j-1}$ hold, we choose

$$2A^2\delta^2 + \epsilon \leq \nu A\delta. \tag{5.5}$$

For example, if $\epsilon = 1/18$, $\nu = 43/48$, $A = 3$, and $\delta = 1/40$, then this choice suffices since $2 \cdot 3^2/40^2 + 1/18 < 43/640$.

The elements of strangeness one or more are composed of the following.

(1) The import of elements of strangeness two or more from $B$'s two children. There are at most $2bA\mu\delta$ such elements.

(2) The elements of strangeness 1 or more from $B$'s parent $B'$. There are at most $S_1(B') < \mu b/A$ such elements.

(3) The elements whose strangeness is 0 in $B$'s father but is 1 in $B$.

The third part of these elements comes from the elements whose strangeness are 0 in $B$'s sibling $C$, which complicates matters. Let the set of elements with strangeness 0 in $C$ be denoted by $V$. The natural location for $V$ may be the whole content of the subtree rooted at $C$ together with 1/2 the contents of $B$'s parent, 1/8 of their great-grandparent, and so on.

In the subtree rooted at $C$, the bag of a daughter of $C$ may contain up to $S_2bA$ elements from outside $V$. A bag two levels further down the tree may contain $S_4bA^3$ elements outside of $V$. From condition (5.5), $2A^2\delta^2 < 2A^2\delta^2 + \epsilon \leq \nu A\delta$, we can get $2A\delta \leq \nu < 1$. Noticing that $S_j < \mu\delta^{j-1}$, we have that the total number of elements whose natural location are outside $V$ 's area in levels lower than $B$ is at most

$$\begin{aligned} 2S_2bA + 8S_4bA^3 + 32S_6bA^5 + ... &< 2\mu b\delta A + 8\mu b\delta^3A^3 + 32\mu b\delta^5A^5 + ... \\ &= 2\mu\delta bA\sum_{i=0}^{\infty}(4\delta^2A^2)^i = 2\mu\delta bA/(1-4\delta^2A^2). \end{aligned}$$

The area above $B$'s parent approximate to $V$ may in fact contain no elements of $V$. We have chosen $A > 1$, and therefore $1/(2A) < 1$. The total space here amounts to at most

$$b/(8A^3) + b/(32A^5) + ... = 1/(8A^3)\sum_{i=0}^{\infty}1/(2A)^{2i} = b/(8A^3 - 2A).$$

If all these elements came from the elements in $B$'s father whose natural location are in $B$, then, in $B$'s father, there will be equal number of extra elements whose natural

location are in $V$. The number of elements in $B$'s father whose natural location are $V$ is half of its capacity plus the above two term. After separating, at least $(1-\epsilon_0)b/2A$ such elements are put into $C$. Therefore, at most $< b/(2A) + 2\mu\delta bA/(1-4\delta^2A^2) + b/(8A^3-2A) - (1-\epsilon_0)b/(2A) = 2\mu\delta bA/(1-4\delta^2A^2) + b/(8A^3-2A) + \epsilon_0 b/(2A)$ elements enter $B$. These elements have strangeness 1 in $B$. Thus we have,

$$S_1'(B)\nu < 2\mu\delta A + 2\mu\delta A/(1-4\delta^2A^2) + 1/(8A^3-2A) + \mu/A + \epsilon_0/(2A).$$

Since we require $S_1'(B) < \mu$, our parameters must satisfy

$$2\mu\delta A^2 + 2\mu\delta A^2/(1-4\delta^2A^2) + 1/(8A^2-2) + \mu + \epsilon_0/2 \leq \nu\mu A. \tag{5.6}$$

If $\epsilon_0 = 1/72$, $\mu = 1/36$, $A = 3$, $\nu = 43/48$, and $\delta = 1/40$, then these choices suffice, since $1/80 + 5/391 + 1/70 + 1/36 + 1/444 < 43/576$.

## 5.3 On the Boundary

### 5.3.1 The Bottom Level

During the execution of Paterson's algorithm, certain bags will not be filled to capacity. To ensure that most bags are filled to the capacity, Paterson's algorithm arranges the bags so that all partially filled are in the same level- the bottom level. So, all bags below the current bottom level are empty and all the bags above this level will be empty or filled to capacity.

To ensure that the bottom level is the only partially filled level, Paterson's algorithm behaves as follows. At the bottom level, each separator sends up to its parent the normal number of elements($\lambda b$) if it contains that many elements. An equal number of the remaining elements are sent to each of the bags children. To do this, we

add *virtual elements* to the separator. These virtual elements always lose or win in a comparison, regardless of the other element. Paterson's algorithm adds two equal sized sets of virtual elements to fill the bag to capacity. The virtual elements in the first set always lose in a comparison; the virtual elements in the second set always win. If any virtual elements reach $FL$ or $FR$, these elements are replaced by real values from $CL$ or $CR$, respectively.

### 5.3.2 Cold Storage and the Root

For Paterson's algorithm to operate cleanly, the root of the tree must act as an interior node. To ensure this, Paterson's algorithm keeps an additional node above the root. This node acts as the parent of the root and is referred to as cold storage. The node differs from the others in the sense that no comparisons are needed during each stage for this node.

The capacity of the cold storage can be thought of as the capacity of an infinite tree above the root. Since the elements in the infinite tree above the root will be feed by only the root, these elements will only be partially filled. Since the capacity of the root is $r$ during the first stage, the parent of the root may contain at most $1/2$ of capacity during an even stage since it will not receive any elements from a sibling of the root. Similarly, the great grand parent of the root, whose capacity is $r/A^3$, may contain at most $1/8$ of its capacity. Since $1/(2A) < 1$, the capacity of the cold storage at the even stages is therefore

$$r/(2A) + r/(8A^3) + ... = r/2A \sum_{i=0}^{\infty} 1/(4A^2)^i = 2Ar/(4A^2 - 1) \text{ at even stages} \quad (5.7)$$

During the odd stages, the grand parent of the root, whose capacity is $r/A^2$, may contain at most $1/4$ of its capacity. Therefore, the capacity of the cold storage at the

odd stages is

$$r/(4A^2) + r/(16A^4) + ... = r/(4A^2) \sum_{i=0}^{\infty} 1/(4A^2)^i = r/(4A^2 - 1) \text{ at odd stages} \quad (5.8)$$

When Paterson's algorithm begins, the root has capacity $r_0$ and cold storage has capacity $r_0/(4A^2 - 1)$. Since they are the only non-empty bags, we have that

$$r_0 + r_0/(4A^2 - 1) = n$$

or that $r_0 = n/(1 - 1/(4A^2))$ and the cold storage has capacity $n/(4A^2)$.

## 5.4 Wrapping Everything Up

As Paterson's algorithm proceeds, the capacity of each bags gets smaller. Moreover, the elements progressively down the tree.

Paterson's algorithm completely sorts the $n$ elements when there are no strangers in the bottom level. Since the number of strangers in a bag of capacity $b$ is $< b\mu$, $\mu = 1/36$, and the bottom level is at depth $\log_2 n$, there are no strangers in the bottom level when

$$cnv^t A^{\log_2 n} \cdot \mu < 1.$$

This occurs when

$$t \geq (\log_2(2A))(\log_2 n)/\log_2(48/43) - O(1).$$

Therefore, Paterson's algorithm completes after $t < 17 \log_2 n$ stages.

## 5.5 Pseudocode of Paterson's Algorithm

We consider a binary tree of depth $\log m$ and $m$ leaves. At the end of the algorithm, all the elements will reside on the leaves and on each leaf, there is only one element.

Left-to-right traversal of these leaves yields the elements in ascending order.

Each node of the tree has a "bag". During the algorithm, the memory locations needed for each bag will decrease with time. At the end of the algorithm, only one memory location is needed for each leaf.

**Algorithm 5.1** *Paterson(set L, int m)*

{

$r = m(1 - 1/(4A^2))$;

*Associate* $m/(1 - 1/(4A^2))$ *elements to the memory locations of root of the tree;*

*Associate* $m/(4A^2)$*elements to memory locations of the cold storage;*

*int i=0;*

*do*

{

*if(i % 2==0)*

{

*for all the nodes in even level of the tree in parallel*

*apply a separator to the nodes;*

*if the node is root*

*send elements of memory locations* $FL$ *and* $FR$ *to cold storage*

*else*

*send the elements of memory locations of* $FL$ *and* $FR$ *of a node to its parents;*

*send the elements of memory locations of* $CL$ *to its left child;*

*send the elements of memory locations of $CR$ to its right child;*

*i=1;*

}

*else*

{

*for all the nodes in odd level of the tree in parallel;*

*apply a separator to the nodes.*

*send the elements of memory locations of $FL$ and $FR$ of a node to its parents;*

*send the elements of memory locations of $CL$ to its left child;*

*send the elements of memory locations of $CR$ to its right child;*

*send $(1-\lambda)r/(2A)$ elements from cold storage to root;*

*i=0;*

}

$r = r * \nu$;

}*while*$(rA^{\log_2 n} \cdot \mu > 1)$;

}

## 5.6 The Complexity of Paterson's Algorithm

As we saw earlier, a $(1/8, 1/18, 1/72)$-separator can be constructed using 1/72-halver in a depth 4 networks. With this separator, $n$ elements can be sorted in less $17 \log n$ stages, where each stage uses a separator. Since a 1/72-halver can be constructed explicitly(using Ramanujan graphs) in depth 20174, the depth of Paterson's sorting network is $< 4 \times 17 \times 20174 \times \log n = 1371832 \log n (\approx 2^{20} \log n)$.

We can improve the performance of Paterson's algorithm as follows. If we choose $A = 3$, $p = 4$, $\lambda = 1/8$, $\nu = 43/48$, $\epsilon_0 = 1/60$, and $\epsilon = 1/15$, and if we pick $\delta = 1/30$ and $\mu = 1/20$, then it is easily verified that these new parameters meet constraints given in the previous sections.

To construct a 1/60- halver, we need $k > 13923$. The next prime congruent to 1 modulo 4 which is larger than $k$ is $p = 13933$. It follows that the depth of Paterson's algorithm is $< 4 \times 17 \times 13934 \log n = 947512 \log n$.

## 5.7 A Set of New Parameters for Paterson's Algorithm

Using our new separator, we can choose the following parameters to construct more efficient sorting networks: $A = 3$, $p = 4$, $\lambda = 1/8$, $\nu = 43/48$, and $\epsilon_0 = 1/54$, $\epsilon = 2/27$, if we pick $\delta = 1/27$ and $\mu = 1/18$, it is easily verified that the parameters meet constraints. To get an 1/54 halver, we need $k > 11234$. If we pick $p = 11257$. The depth of the algorithm $< 4 \times 17 \times 11258 \log n = 765544 \log n$.

# Chapter 6

# Conclusion and Possible Future Work

This thesis studies construction of sorting network with Ramanujan graphs. We adopted a effective method to construct a halver from a Ramanujan graph, and gave an explicit construction of Paterson's algorithm. Since the Ramanujan graphs are the best explicit expanders known, this result likely gives the best known depth of the explicit construction of sorting networks.

Paterson's major improvement to the AKS algorithm was the replacement of the nearsort in AKS algorithm with a separator. With this approach, Paterson produced a sorting networks whose depth is $17 \log_2 n$ times the depth of a separator. As in the AKS algorithm, the major cost of calculation in Paterson algorithm is the $\epsilon$-halver. So, it will be very difficult to improve Paterson's result greatly without changing the construction method of separator. Paterson showed that if a new kind of halver can be constructed, the efficiency of the algorithm will improve considerably. However, it does not seem easy to construct such a new separator explicitly. Ajta and Paterson [AKS92] discussed this approach. However, the construction will be difficult. Another idea of improvement recommended by Paterson [Pat90] is to send

the elements of strangeness 2 or more up two levels at once. However, the formulas for the constraints in the algorithm become very complicated.

Ramanujan graphs are very good expanders. Using Tanner's result, we can produce an $\epsilon$-halver of depth $O(1/\epsilon^2)$. Kahale provided an improvement to Tanner's result( Theorem 2.2). In section 2, we analyzed the two results and show that Kahale's result cannot be used to decrease the depth of an $\epsilon$-halver.

In this thesis, we designed a new kind of separator according to the idea of [CO86]. In Paterson's original paper, this improvement is not analyzed. Here we analyzed this kind of separator and determined its parameters. As an application, we chose a set of parameters such that when the new separator is used, Paterson's algorithm will sort the elements in less depth than the original. Unfortunately, the constant behind the big $O$ in our construction is still too large for practical use.

# Bibliography

[AHB93] M. Z. Al-Hajery and K. E. Batcher. On the bit-level complexity of bitonic sorting networks. In *Proceedings of the 1993 International Conference on Parallel Processing*, pages 209–213, 1993.

[AHB94] M. Z. Al-Hajery and K. E. Batcher. On the role of $k$-bits bitonic sorting network in multicast routing. In *Proceedings of the 6th Symposium on Parallel and Distributed Processing*, pages 706–714, Los Alamitos, CA, USA, October 1994. IEEE Computer Society Press.

[AKS83a] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 1–9, Boston, Massachusetts, 25–27 April 1983.

[AKS83b] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in LOGSPACE. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 132–140, 1987. .

[AKS92] M. Ajtai, J. Komlós, and E. Szemerédi. Halvers and expanders. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 686–692, Pittsburgh, PN, October 1992. IEEE Computer Society Press.

[Alo86] Noga Alon. Eigenvalues and expanders. *Combinatoria*, 6(2), 1986.

[AM84] Noga Alon and V. D. Milman. Eigenvalues, expanders and superconcentrators (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 320–322, Singer Island, Florida, 24–26 October 1984. IEEE.

[AS92] Noga Alon and Joel H. Spencer. **The Probabilistic Method**. John Wiley Sons, New York, 1992.

[Bat68] K. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computer Conference*, volume 32, pages 307–314, Reston, VA, 1968. AFIPS Press.

[BP85] G. Bilardi and F. P. Preparata. VLSI optimality of the AKS sorting network. *Information Processing Letters*, 20(2):55–59, February 1985.

[Buc86] Marshall W. Buck. Expanders and diffusers. *SIAM Journal of Algebraic Discrete Methods*, 7(2):282–304, April 1986.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. **Introduction to Algorithms**. MIT Press, 1990.

[CO86] Richard Cole and Colm O'Dunlaing. **note on the aks sorting network.** Ultracomputer Note 109, New York University, September 1986. Also Computer Science Technical Report 243. No publication.

[Edm65] J. Edmonds. Maximum matching and a polyhedron with $(0, 1)$ vertices. *J. Res. Nat. Bur. Standards Sect. B*, 8:67–72, 1965.

[FJ59] Lestor R. Ford and Selmer M. Johnson. A tournament problem. *The American Mathematical Monthly*, 66:387–389, 1959.

[Gan59] F. R. Gantmacher. **Applications of the Theorey of Matrices**. interscience Publishers, Inc., NY, 1959.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[HK71] Kenneth Hoffman and Ray Kunze. **Linear Algebra**. Prience-Hall , Inc, NJ, 1971.

[IS87] Janet Incerpi and Robert Sedgewick. Practical variations of shellsort. *Information Processing Letters*, 26:37–43, September 1987.

[Kah91] N. Kahale. Better expansion for ramanujan graphs. In IEEE, editor, *Proceedings of the 32rd Annual Symposium on Foundations of Computer Science*, pages 398–404, San Juan, Porto Rico, October 1991. IEEE Computer Society Press.

[Kah93a] N. Kahale. **Expander graphs**. Ph.D . thesis, MIT Laboratory for Computer Science, September 1993.

[Kah93b] Nabil Kahale. On the second eigenvalue and linear expansion of regular graphs. In *Proceedings of the 33rd IEEE symposium on Foundations of Computer Science*, pages 296–303, 1993.

[Kah95] Nabil Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM*, 42(5):1091–1106, September 1995.

[Kla84] M. Klawe. Limititation on explicit constructions of expanding graphs. *SIAM Journal of Computing*, 13:156–166, 1984.

[Knu73] Donald E. Knuth. Sorting and searching. In **The Art of Computer Programming**. Addison-Wesley, 1973.

[Lei92] F.T. Leighton. **Introduction to Parallel Algorithms and Architectures: Arrays Trees Hypercubes**. Morgan Kaufmann, 1992.

[LP86] L. Lovasz and M. D. Plummer. **Matching Theory**. Annals of Discrete Mathematics 29. Elsevier Science Publishers B.V. The Netherlands, 1986.

[LPS86] A. Lubotzky, R. Phillips, and P. Sarnak. Explicit expanders and the Ramanujan conjectures. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 240–246, Berkeley, California, 28–30 May 1986.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *COMBINAT: Combinatorica*, 8(3):261–277, 1988.

[Lub95a] Alexander Lubotzky. Cayley graphs: Eigenvalues, expanders and random walks. In **Surveys in Combinatorics, 1993, Walker (Ed.), London**

**Mathematical Society Lecture Note Series 187**. Cambridge University Press, 1995.

[Lub95b] Alexander Lubotzky. **Discrete Groups, Expanding Graphs and Invariant Measures**. Progress in Mathematics, volume 125. Birkhauser Verlag, 1995.

[Mar73] G. A. Margulis. Explicit constructions of concentrators. *Problems of Information Transmission*, 9(4):325–332, October 1973.

[Mar88] G. A. Margulis. Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24(1):39–46, October 1988.

[Mor94] M. Morgenstern. Existance and explict constructions of $q + 1$ regular ramanujan graphs for every prime power $q$. *Journal of Combinatorial Theory, Series B*, 62:44–62, 1994.

[Mor95] M. Morgenstern. Natural bounded concentrators. *Combinatorica*, 15(1):111–122, 1995.

[Pat90] M. S. Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5(1):75–92, 1990.

[Pip93] Nicholas Pippenger. Self -routing superconcentrators. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 355 –361, 1993.

[RV83] John H. Reif and Leslie G. Valient. A logarithmic time sort for linear size networks. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 10–20, Boston, Massachusetts, 25–27 April 1983.

[RV87] John H. Reif and Leslie G. Valiant. A logarithmic time sort for linear size networks. *Journal of the Association for Computing Machinery*, 34(1):60–76, January 1987.

[Smi93] Justin R. Smith. **The Design and Analysis of Parallel Algorithms**. Oxford University Press, New York, 1993.

[Spi95] Daniel Alan Spielman. **Computationally Efficient Error-Correcting Codes and Holographic Proofs**. Ph.D. thesis, Department of Mathematics, MIT, June 1995.

[SS94] M. Sipser and D. A. Spielman. Expander codes. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 556–576, November 1994.

[SS96] M. Sipser and D.A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42, 1996.

[Tan84] R. Michael Tanner. Explicit construction of concentrators from generalized $n$-gons. *SIAM Journal of Algebrac Discrete mathematics*, 5(3):287–293, September 1984.

[vLW92] J.H. van Lint and R.M. Wilson. **A Course in Combinatorics**. Cambridge University Press,New York, 1992.

# Abstract

This thesis examines the explicit construction of Paterson's algorithm. In an attempt to make the algorithm more easily understood, we give the precise definitions for $\epsilon$-halvers and separators, detailed explanations about some conclusions in Paterson's paper, and the pseudocode for Paterson's algorithm.

The idea in Paterson's algorithm is to take an approximate partition of elements, which can be achieved in only a constant number of comparator levels, and then introduce some error-recovery structure into the sorting scheme. In Paterson's algorithm, the approximate partition is implemented with *separators*. The depth of the Paterson's algorithm is dependent on the depth and parameters of the separator, which is determined by the algorithm that implements it and the depth of an $\epsilon$-halver. An $\epsilon$-halver can be constructed through expanders using constant depth. We introduces expanders and the major research results in the area.

Ramanujan graphs, which are asymptotically optimal in making the second- eigenvalue as small as possible, are used to construct $\epsilon$-halves. We analyze the construction of $\epsilon$-halvers through matching in regular graphs. We give a very simple method to construct a balanced regular expander. The second largest eigenvalue of a graph is then used to analyze the expansion of a graph. We analyze the performances of Kahale's result and Tanner's result and point out that the Kahale's result is only

effective for very small subsets of the vertices. We adopt the Tanner's result and calculate the depth of the associated $\epsilon$-halver.

We analyze a new kind of separator and give its depth. This separator can achieve better separation without increasing its depth.

Finally, we use all our results to analyze the depth of Paterson's algorithm. Through our construction, we achieve sorting networks in less than depth of 947512 $\log n$ with the original separator and depth less than 765544 $\log n$ with the new kind of separator.