ON MERGING NETWORKS

A Technical Report By

Levy Tamir
Litman Ami

Faculty of computer science, Technion, Haifa, Israel
{levyt,litman @ cs.technion.ac.il}

Abstract

Many algorithms for oblivious merging have been invented. However, it is a common phe-
nomena that radically different algorithms produce identical networks. In order to prevent
a duplication of effort it is desired to tell when different algorithms produce identical net-
works. Our work studies this question and out main result is a criteria which shows that
all published merging networks belong to a family of networks which is a generalization
of Batcher’s odd/even network. A significant advantage of our criteria is that it does not
require a complete understanding of the technique in question; in fact, a very superficial
understanding of the algorithm suffices to establish that a merging technique produces
generalized Batcher merging network.

Our criteria is as follows: A network has the At Most One Path Property, or is AMOP,
if it has at most one path from every input to every output. A comparator of a network is
degenerate if its incoming edges can be named $e'$ and $e''$ s.t. under any valid input the value
transmitted on $e'$ is smaller or equal to the value transmitted on $e''$. Our main theorem
states the following characterization of the Batcher merging networks. A merging network
is a Batcher merging network iff it is AMOP, it has no degenerate comparators and its width
is a power of two. We survey several published merging techniques and use this criteria to
show that all these techniques produce Batcher merging networks. Additional contributions
of this work are presented in the introduction.
# Table of Contents

Table of Contents  ii  

1 Introduction  1  

2 Isomorphism of networks  5  

3 Keys and Edges  9  

4 The bypass transformation  12  
   4.1 Bypass charted by a vector  14  
   4.2 Splitting of a network  16  
      4.2.1 Producing half merging networks  17  
      4.2.2 Pruning a network  18  
   4.3 Minors of merging networks  19  

5 The Batcher merging networks  22  

6 Congruent functions  28  

7 The Input Cone  31  

8 The i.m.f. of a split  35  

9 Degenerate comparators for half merging.  39  

10 Characterization of the Batcher merging networks  42  

11 oblivious algorithms  44
Chapter 1

Introduction

Many ingenious algorithms for oblivious merging have been invented. However, it is a common phenomena that radically different algorithms produce identical networks. For many applications the networks themselves are of importance and their properties are investigated. In order to prevent a duplication of effort it is desired to tell when different algorithms produce identical networks. Our work studies this question and our main result is a criteria which shows that all published merging networks belong to a family of networks which is a natural generalization of Batcher’s odd/even network [1]. A significant advantage of our criteria is that it does not require a complete understanding of the technique in question; in fact, a very superficial understanding of the algorithm suffices to establish that a merging technique produces generalized Batcher merging network.

We do not know of any other work on the general question of isomorphism of arbitrary merging networks. There are some works on specific merging networks. Bilardi [6] have shown that the Bitonic merging network which was invented by Batcher [1] is isomorphic to the Balanced merging network invented by Dowd, Perl, Rudolph and Sacks. Dalpiaz and Rizzi [17] shown that the bitonic merging network is isomorphic to a merging network presented by Leighton [13, pp 623].

Batcher, in his seminal paper [1], introduced the idea of recursive construction of merging networks; moreover, the depth of his networks is much smaller than previous ones and in fact is minimal. Batcher actually presented two networks. The first is his odd/even network described shortly and the second, which is based on his concept of bitonic sequences, is discussed later in Chapter 14.

Batcher’s odd/even merging technique works as follows: Each of the input sequences is partitioned into its even part and its odd part. The even part of one sequence is merged with the even part of the other sequence recursively and similarly, the odd part of one sequence is recursively merged with the odd part of the other
sequence. As Batcher has shown [1], the two resulting sequences can be combined into a single sorted sequence by a network of depth one. Moreover, there are exactly two such networks of depth one but one of them has a degenerate comparator (defined shortly) and we insist on using the other network.

A slight variant of this method, mentioned by Leighton [13, pp 623], recursively merges the even part of each input sequence with the odd part of the other sequence. Again the two resulting sequences can be combined by a network of depth one. In this variant there exists only one such depth one network. We refer to the family of networks produced by allowing each of the above two variants in each step of the recursive algorithm as the Batcher merging networks. By our terminology, a Batcher merging network merges two sequences of the same length which is a power of two. All published merging networks we encountered are members of this family.

As said, in this work we provide an easy criteria to establish that a merging network is a Batcher merging network. To this end, a network has the At Most One Path Property, or is AMOP, if it has at most one path from every input to every output. A comparator of a network is degenerate if its incoming edges can be named $e'$ and $e''$ s.t. under any valid input the value transmitted on $e'$ is smaller or equal to the value transmitted on $e''$. A merging network is non-degenerate if it has no degenerate comparators. Our criteria is a conjunction of the following requirements: AMOP, non-degenerate and a width which is a power of two. Our main theorem is that a merging network is a Batcher merging network iff it satisfies the above criteria.

We survey several published merging techniques and use this criteria to show that all these techniques produce Batcher merging networks.

Each of the three ingredients of our criteria is mandatory for the above characterization. Consider the AMOP property. A counter-example is a network that merges two sorted sequences $\langle a_0, a_1, a_2, a_3 \rangle$ and $\langle b_0, b_1, b_2, b_3 \rangle$ as follows. A comparator $c'$ sorts the pair $(a_0, b_0)$ and a comparator $c''$ sorts the pair $(a_2, b_1)$. The four keys emerging from these comparators together with the rest of the input keys are now sorted by an arbitrary sorting network. The resulting network is a merging network which may have degenerate comparators. Clearly, all the degenerate comparators can be omitted without disturbing the functionality of the network (This is discussed in Lemma 4.1.2). The comparators $c'$ and $c''$ are not degenerate and are not omitted. The resulting merging network has no degenerate comparators. It is not hard to see that no Batcher merging network has comparators $c'$ and $c''$ as above.

Next, consider the property of having no degenerate comparators. A counter-example of an AMOP merging network which is not a Batcher merging network is depicted in Figure 2.1.

A natural question is whether the AMOP property can be replaced by the condition of having minimal depth. This question is answered negatively in Chapter 16, but a similar question regarding the stronger functionality of sorting bitonic sequences
(defined in Chapter 14) is answered positively. In fact, we show that for any \( n = 2^k \), there is a unique bitonic sorter of width \( n \) and of minimal depth. This statement does not hold when \( n \) is not a power of two. Moreover, for such general \( n \), the minimal depth of a bitonic sorter of width \( n \) is not a monotonic function of \( n \).

Since most networks are described indirectly via an oblivious algorithm, this work studies the concept of an oblivious algorithm. We present several oblivious models of computation and emphasize their differences and their relative computational power. We present a model which we feel is more natural than the generally accepted one used in literature. Our model is stronger; yet, it maintains the famous “0 − 1 principle” and all its known generalizations.

This work presents and uses some generalizations of the known “0-1 principle” [10, pp 224]. Instead of addressing specific functionalities such as sorting [10] or merging [11] we consider a more general context. For a given functionality, we search for a small set \( V \) (called a conclusive set) of valid input vectors s.t. any network has this functionality w.r.t. \( V \) iff it has this functionality w.r.t. all its valid input vectors. Such conclusive sets simplify the design and analysis of oblivious algorithms of all mentioned computations models, among them networks of comparators. For the functionality of merging two sequences of length \( n \) each, we present a conclusive set having \( n + 1 \) vectors. This set is much smaller than a conclusive set of \( 0 − 1 \) vectors since, as we show, such a set of \( 0 − 1 \) vectors has at least \( (n + 1)^2 - 2 \) vectors.

We also introduce two new merging techniques. The first called “zipper sorting” is based on a certain way to quantify the amount of “unsortedness” of a pair of sorted sequences and on a network of depth one which halves this quantity. This technique produces minimal depth merging networks which, not surprisingly, are Batcher merging networks.

The second technique, called “tri-section”, is based on partitioning, by a depth one network, the input of a merging network into three sequences, such that every element in one sequence is less than or equal to every element in the next sequence. This technique enables us to build minimal depth merging network which produce some of the output keys faster then the depth of the entire network. Namely, for any \( k \), significantly smaller than \( n \), we construct a network of minimal depth which produces the \( k \) lowest and \( k \) highest keys in a delay of \( \log(k) + 1 \) comparators. This technique also gives rise to minimal depth merging networks which have no recursive structure and are clearly not Batcher merging networks.

The outline of this work is as follows: Chapter 2 sets the preliminary definitions of our work, what is a network, and when are two networks identical. Chapter 3 studies the key values that are transmitted on the edges of a network under certain scenarios. Chapter 4 presents a technique for deriving networks from larger ones. Chapter 5 elaborates on the Batcher merging networks, establishes some of their properties and computes the number of such non-isomorphic networks. Chapters 6 introduces
the notion of input matching function of a merging network and characterizes input matching functions of Batcher merging networks. Chapters 7, 8 and 9 study special subnetworks of Batcher merging networks which are needed to establish the above criteria in Chapter 10. Chapter 11 discusses several oblivious models of computation. Chapter 12 surveys several published oblivious merging algorithms and shows them to produce Batcher merging networks. Chapter 13 presents useful generalizations of the “0-1 principle”. Chapter 14 establishes the uniqueness of minimal depth bitonic sorters. Chapters 15 presents the “zipper sorting” technique and Chapter 16 presents the “tri-section” technique.
Chapter 2

Isomorphism of networks

This chapter formalizes the concept of a network of comparators and defines when two such networks are identical (isomorphic). It shows that any two networks that perform the same comparisons for a certain input are isomorphic.

A comparator is a combinational device that sorts two keys. Namely, it has two incoming edges and it receives a key from each one of them. It has two outgoing edges of distinct types; a min edge and a max edge. A comparator transmits the minimal key on the min edge and the maximal key on the max edge. A network of comparators is an acyclic network of these devices. These networks are useful for performing operation on keys such as merging or sorting.

Our concept of a network encompasses both the structure of the network and the manner in which the network is used to process keys. The structure of the network is manifested by its underlying graph which is a directed acyclic graph having three types of vertices and three types of edges. The type of a vertex is determined by its indegree and its outdegree, as follows:

1. An input vertex has no incoming edges and one outgoing edge.
2. An output vertex has one incoming edge and no outgoing edges.
3. An internal vertex (a comparator) has two incoming edges and two outgoing edges.

The three types of edges concern the functionality of a comparator. Out of the two edges which exit a comparator, one is of type min and carries the minimal key and the other is of type max and carries the maximal key. Edges of the third type are those which exit an input vertex rather than a comparator.
As said, our concept of a network encompass also the manner in which the network is used to process keys. We use labels to specify how to apply a given arrangement of keys to the network and how to assemble the resulting keys. Our labels are symbols of the form $\hat{\alpha}_i$ where $\alpha$ range on the lower case Latin letters and $i \in \mathbb{N}$. For example, the symbols $\hat{a}_0, \hat{b}_3, \hat{o}_7$ are labels. We refer to the set of these labels as $\mathcal{L}$. Labels are assigned to the input edges (edges which exit an input vertex) and output edges (edges which enter an output vertex).

Consider for example a merging network. Such a network merges two sorted sequences of the same given length $n$. In this context, we refer to the two input sequences of such a network by $\vec{a} = \langle a_0, a_1 \ldots a_{n-1} \rangle$ and $\vec{b} = \langle b_0, b_1 \ldots b_{n-1} \rangle$ and to the output sequence as $\vec{o} = \langle o_0, o_1 \ldots o_{2n-1} \rangle$. The input edges are labelled by $\{\hat{a}_i, \hat{b}_i | i \in [0, n)\}$ and the output edges are labelled by $\{\hat{o}_i, | i \in [0, 2n)\}$. The two sequences $\vec{a}$ and $\vec{b}$ enter the network according to those labels; that is, the key $a_i$ enter the edge labelled $\hat{a}_i$ and the key $b_i$ enter the edge labelled $\hat{b}_i$. Similarly, the output labels denote how to assemble the resulting keys into a single sequence. In all the networks studied in this work the labels on the input edges are non-repeating; that is, distinct input edges have distinct labels; the same holds for the output labels. This enables us to name the input/output edges by their labels and we usually do so.

The width of a network is the number of its input edges. Clearly, this equals the number of its output edges. The depth of an edge or a vertex $x$ is the number of comparators along the longest path from any input vertex to $x$. The depth of a network is the depth of its deepest edge. Note that a network may have no comparators at all and in this case its depth is zero.

In our drawings of networks the type of an edge is depicted by the form of its arrowhead; namely a hollow arrowhead depicts a min edge, a solid arrowhead depicts a max edge and an open arrowhead depicts an untyped (input) edge; input and output vertices are omitted and the labels of the corresponding input and output edges are written instead of them. Fig 2.1 is a drawing of two merging networks of width four and of depth two.

The keys that the networks process are members of an infinite ordered set called $\mathcal{K}$. The order type of $\mathcal{K}$ and the identity of its members are usually irrelevant to our studies. However, it is sometimes convenient to assume that $\mathcal{K}$ is the set of the natural numbers. Informally, a vector (of keys) is an arrangement of keys having a certain structure. For example, a vector composed of a pair of sequences of the same given length can be an input of a given merging network; a vector composed of a single sequence is the output of this network. Formally, a vector is a function $v : D \rightarrow \mathcal{K}$ where $D$ is a finite subset of $\mathcal{L}$ (the set of labels). The width of the vector is the size of $D$. When $D$ is the set of (labels of) input edges of a network $N$, we say that $v$ conforms to $N$ or that $v$ is an input vector of $N$. Only in this case it is meaningful to apply $v$ to $N$. This is done by transmitting the key $v(e)$ on each
input edge \( e \). (Recall that we name input/output edges by their labels.) Note that the fact that \( v \) is an input vector of a certain network implies nothing on the value of the keys in \( v \); it only specify how these keys are structured. We usually refer to a vector in an indirect manner; for example, when \( \vec{a} \) and \( \vec{b} \) are two sequences of the same length, \( n \), we denote by \( v \triangleq (\vec{a}, \vec{b}) \) the vector composed of these two sequences. Such a vector is called a \textit{bisequenced} vector. Formally, it is a function as above whose domain is \( D = \{ \hat{a}_i, \hat{b}_i | i \in [0,n) \} \) and is defined by \( v(\hat{a}_i) = a_i \) and \( v(\hat{b}_i) = b_i \). When an input vector \( v \) is applied to a network \( N \) the network produces an output vector which is formally a function from the set of output labels into \( \mathcal{K} \). We denote this output vector by \( T^N(v) \) and refer to \( T^N \) as the \textit{input/output transformation} of \( N \).

Following mathematical logic, the concept of an isomorphism of networks encompass all aspects of a network that are relevant to our studies. Namely, an isomorphism, \( \pi \), of a network \( N_1 \) onto a network \( N_2 \) is a one-to-one mapping of the vertices and edges of \( N_1 \) onto the vertices and edges of \( N_2 \). This mapping is required, of course, to preserve the connectivity of vertices and edges; i.e. if an edge \( e \) enters (exits) the vertex \( x \) in \( N_1 \), then \( \pi(e) \) enters (exits) the vertex \( \pi(x) \) in \( N_2 \); this mapping is also required to preserve the labels of the input/output edges and the \textit{min/max} type of the edges. Usually, we do not distinguish between isomorphic networks and consider them identical. Clearly, a network can be drawn in many different ways. We do not study drawing of networks and so isomorphism does not need to preserve this aspect of a network.

A vector is \textit{non-repeating} if it is one-to-one. Let \( v \) be a non-repeating input vector of a network \( N \). Under \( v \), a key \( k \) of \( v \) traverses a path in \( N \). In each comparator along this path, the key \( k \) \textit{encounters} another key. The sequence of keys that \( k \) encounters is denoted \( \vec{s}^N(v,k) \). The next lemma shows that if those sequences of some input vector are equal in two networks then these networks are isomorphic. For \( N, v \) and

Figure 2.1: Two merging network of width four and of depth two. The network (b) has one degenerate comparator.
k as above, let \( \text{act}(N, v, k) \triangleq \langle k, l_0, s^N(v, k), l_\infty \rangle \) where \( l_0 \) and \( l_\infty \) are the labels of the input edge and output edge that \( k \) traverses. The syndrome of \( v \) in \( N \) is the set \( \text{synd}(N, v) \triangleq \{ \text{act}(N, v, k) | k \text{ appears in } v \} \). A syndrome of \( N \) is a set \( \text{synd}(N, v) \), for some non-repeating input vector \( v \) of \( N \). Note that, in the above definitions, there is no requirement on the values of the keys of \( v \) except of the non-repeating requirement; for example, in the case of a merging network, the two sequences in question are not required to be sorted.

**Theorem 2.0.1.** Two networks are isomorphic iff they have a common syndrome.

**Proof.** The Left to right direction is immediate. Consider the other direction and let \( S \) be a common syndrome of two networks \( N_1 \) and \( N_2 \). Let \( S = \text{synd}(N_1, v) \). Clearly, a syndrome encodes the input vector that generates it; hence, \( S = \text{synd}(N_2, v) \).

We show, by induction on the number of comparators in \( N_1 \), that \( N_1 \) and \( N_2 \) are isomorphic. The case where \( N_1 \) has no comparators is trivial so assume \( N_1 \) has at least one comparator. In this case, \( N_1 \) has a comparator \( c_1 \) of depth one. ( The two incoming edges of \( c_1 \) are input edges.)

Let the edges \( d_1' \) and \( d_1'' \) enter the comparator \( c_1 \), let \( l' \) and \( l'' \) be their labels, let \( k' \) and \( k'' \) be the two keys traversing those edges and assume, without loss of generality, that \( k'' > k' \). Let \( e_1' \) and \( e_1'' \) be the min and max edges exiting \( c_1 \), respectively. The fact that \( \text{synd}(N_1, v) = \text{synd}(N_2, v) \) imply that the same scenario happens in \( N_2 \).

Namely, two input edges, \( d_2' \) and \( d_2'' \), labelled by \( l' \) and \( l'' \), enter a comparator \( c_2 \) and (under \( v \)) carry the keys \( k' \) and \( k'' \), respectively; let \( e_2' \) and \( e_2'' \) be the min and max edges exiting \( c_2 \). Let \( \tilde{N}_1 \) be the network generated from \( N_1 \) by:

1. Removing \( c_1, d_1' \) and \( d_1'' \).
2. Assigning to \( e_1' \) and \( e_1'' \) the labels \( l' \) and \( l'' \), respectively.

In the same manner, \( \tilde{N}_2 \) is generated from \( N_2 \). Clearly, \( v \) is an input vector of \( \tilde{N}_1 \) and \( \tilde{N}_2 \); furthermore, \( \text{synd}(\tilde{N}_1, v) \) is derived from \( \text{synd}(N_1, v) \) by removing the first element of \( s^{N_1}(v, k') \) and of \( s^{N_1}(v, k'') \). The same holds for \( \text{synd}(\tilde{N}_2, v) \) and so \( \text{synd}(\tilde{N}_1, v) = \text{synd}(\tilde{N}_2, v) \). By the induction hypothesis, there is an isomorphism \( \tilde{g} \) of \( \tilde{N}_1 \) onto \( \tilde{N}_2 \).

Let \( g : N_1 \rightarrow N_2 \) be the extension of \( \tilde{g} \) defined by \( g(c_1) = c_2 \), \( g(d_1') = d_2' \) and \( g(d_1'') = d_2'' \). Clearly, \( g \) preserves the edges/vertices connectivity; moreover, \( g \) preserves the input/output labels and the types of the edges; that is, \( g \) is an isomorphism of \( N_1 \) onto \( N_2 \). \( \square \)
Chapter 3

Keys and Edges

This chapter studies the key values that are transmitted on the edges of a network under certain scenarios.

A network $N$ is usually associated with a certain functionality. This functionality specifies a set of input vectors called valid vectors and specifies a condition that the resulting output vector $T^N(v)$ should satisfy whenever $v$ is a valid vector. Consider, for example, the functionality of merging. Recall that an input vector $v$ of a merging network is a bisequenced vector. With respect to the functionality of merging, a vector $v$ is valid if it is bisequenced and both $\vec{a}$ and $\vec{b}$ are sorted. We refer to such a vector as a bisorted vector. The merging functionality requires that, for any bisorted vector $v$, the outcome $T^N(v)$ is sorted. This functionality, as any other functionality, does not imply anything on $T^N(v)$ when $v$ is a non-valid input vector of $N$.

Let $v$ be an input vector and $e$ an edge of a network $N$. We denote by $\mathcal{V}^N(e,v)$ the key transmitted on $e$ when $v$ is applied to $N$. Let $v^N$ be the extension of $v$ over all the edges of $N$ defined by $v^N(e) = \mathcal{V}^N(e,v)$. A key function is a function $f : \mathcal{K} \rightarrow \mathcal{K}$. (Recall that $\mathcal{K}$ is the set of possible key values.) A monotonic key function is a key function $f$ such that $f(k_1) \leq f(k_2)$, for any $k_1 \leq k_2$. Recall that formally a vector is a mapping from a finite subset of $\mathcal{L}$ (the set of labels) into $\mathcal{K}$. For a key function $f$ and a vector $v$ define $f(v)$ as the vector $u$ that has the same domain as $v$, and satisfies $u(l) = f(v(l))$. In such a case we say that $u$ is a monotonic image of $v$. The following lemma is attributed by Knuth to W.G. Bouricius [10, pp 224].

**Lemma 3.0.2.** For any network $N$ and any monotonic key-function $f$, the functions $f$ and $T^N$ commute; that is, $f(T^N(v)) = T^N(f(v))$ for every input vector $v$.

This lemma implies the well-known “0-1 principle” [10, pp 224]. A generalization of this principle is presented in Section 13.1. We now present a lemma which is a more versatile version of Lemma 3.0.2. Let $g, g' : D \rightarrow \mathcal{K}$ where $D$ is an arbitrary
It suffices to show that for every edge pair \((hypothesis, for the edges such that \(v\)) \(N\) of \(u\) agree on the \(\langle x, y \rangle\) of \(D\) if \(x\) and \(y\) can be named \(z_1\) and \(z_2\) such that \(g(z_1) \leq g(z_2)\) and \(g'(z_1) \leq g'(z_2)\).

If \(g\) and \(g'\) agree on every pair of elements of \(D\), we say that \(g\) and \(g'\) agree. Note that the agree relation is symmetric and reflexive but is not transitive. For example, consider the three sequences \(\langle 0, 1 \rangle, \langle 1, 1 \rangle\) and \(\langle 1, 0 \rangle\). The first agree with the second which agree with the third, but the first does not agree with the third. Note that if \(g'\) is a monotonic image of \(g\) then \(g\) and \(g'\) agree; however, the opposite direction does not hold. For example, the two sequences \(\langle 0, 0, 1 \rangle\) and \(\langle 0, 1, 1 \rangle\) agree but none of them is a monotonic image of the other.

**Lemma 3.0.3.** Let \(v\) and \(u\) agree and be input vectors of a network \(N\). Then \(v\) and \(u\) agree; in particular \(T^N(v)\) and \(T^N(u)\) agree.

**Proof.** It suffices to show that for every edge \(x\) of \(N\) there is an input edge \(y\) of \(N\) such that \(v(x) = v(y)\) and \(u(x) = u(y)\). We prove this claim by induction on the depth of \(x\), i.e. the number of comparators on the longest path to \(x\). The case where \(x\) is an input edge is trivial. Let \(x\) exit a comparator \(c\) and let \(e_1\) and \(e_2\) be the incoming edges of \(c\). Let \(y_1\) and \(y_2\) be the input edges provided by the induction hypothesis, for the edges \(e_1\) and \(e_2\), respectively. The fact that \(v\) and \(u\) agree on the pair \((y_1, y_2)\) implies that \(v\) and \(u\) agree on \(e_1\) and \(e_2\). Hence, there is an \(i \in \{1, 2\}\) such that \(v(x) = v(e_i) = v(y_i)\) and \(u(x) = u(e_i) = u(y_i)\).

For a set \(V\) of input vectors of a network \(N\), define \(\mathcal{V}^N(e, V) \triangleq \{v(e, v) | v \in V\}\). For a comparator \(c\) of \(N\) whose incoming edges are \(e_1\) and \(e_2\), define \(\mathcal{V}^N(c, V) \triangleq \mathcal{V}^N(e_1, V) \cup \mathcal{V}^N(e_2, V)\). A vector \(v\) is a permutation if it is non-repeating and the range of \(v\) is an interval of integers starting at \(0\). For a network \(N\), let \(P^N\) be the set of permutations which are input vectors of \(N\). When the network in question is clear from context we omit it from the above notations and use the shortcuts \(\mathcal{V}(e, v), \mathcal{V}(e, V), \mathcal{V}(c, V)\) and \(P\).

The following lemma was observed by Knuth [10, pp. 239,639].

**Lemma 3.0.4.** For any edge \(e\) of a network \(N\), \(\mathcal{V}(e, P^N)\) is an interval.

We present a variant of this lemma, regarding merging networks. A comparator \(c\) is degenerate with respect to a set of input vectors \(V\) if its incoming edges can be named \(e_1\) and \(e_2\) such that under any input vector \(v \in V\), \(v(e_1) \leq v(e_2)\). A comparator \(c\) of \(N\) is degenerate under a certain functionality if it is degenerate w.r.t. the valid input vectors of this functionality. When the functionality of the network is clear from context we just say that \(c\) is degenerate, without specifying this functionality. Let \(P_{bs}^{2n}\) denote the set of valid input vectors of merging networks of width \(2n\) which are permutations. In other words, \(P_{bs}^{2n}\) is the set of bisorted permutations of width \(2n\). For an edge or a vertex \(x\) of a merging network \(M\) of width \(2n\) define \(\mathcal{V}(x) = \mathcal{V}(x, P_{bs}^{2n})\).
Lemma 3.0.5. Let $M$ be a merging network of width $2n$. Then:
(a) For any edge $e$ of $M$, $V(e)$ is an interval.
(b) Let $e'$ and $e''$ enter a non-degenerate comparator $c$. Then $V(e') \cap V(e'') \neq \emptyset$ (Hence, $V(e)$ is an interval.).

Proof. Consider statement (a). The network $M$ can be extended into a sorting network $S$ by appending to it two sorting networks, which generate the two input sequences of $M$. When the input to $S$ is a permutation then the input to $M$ is a bisorted permutation and any bisorted permutation is generated that way; hence, for any edge $e$ of $M$, $V^M(e, P_{bs}^{2n}) = V^S(e, P^S)$. This and Lemma 3.0.4 imply statement (a). Consider statement (b). Any valid vector is a monotonic image of some valid permutation. This and Lemma 3.0.2 imply that if a comparator is non-degenerate w.r.t. the set of all valid vectors then it is non-degenerate w.r.t. the set of valid permutations. Therefore, statement (b) follows statement (a).

Two edges $e_1$ and $e_2$ of a network $N$ are called disagreeable iff there are two valid input vectors of $N$, $v$ and $u$, such that $v^N(e_1) > v^N(e_2)$ and $u^N(e_1) < u^N(e_2)$; that is, $u$ and $v$ do not agree on the pair $e_1$ and $e_2$. By Lemma 3.0.5(a), if $e_1$ and $e_2$ are disagreeable edges of a merging network $M$ then $V(e_1)$ and $V(e_2)$ are non-disjoint intervals. This implies:

Lemma 3.0.6. Let $e_1$ and $e_2$ be two disagreeable edges of a merging network $M$. Then there is an output edge which is reachable both from $e_1$ and $e_2$.

Two keys, $k'$ and $k''$, are adjacent in a bisorted vector $v = \langle \vec{a}, \vec{b} \rangle$ iff they are distinct, each of them appears exactly once in $v$, one of them appears in $\vec{a}$ and the other appears in $\vec{b}$ and $v$ contain no key which is strictly between $k'$ and $k''$.

Lemma 3.0.7. Let $v$ be an input vector of a merging network $M$ and let $k'$ and $k''$ be two adjacent keys in $v$. Then, under $v$, $k'$ and $k''$ encounter each other.

Proof. Let $u$ be the input vector of $M$ derived from $v$ by swapping the keys $k'$ and $k''$. If $k'$ and $k''$ do not encounter each other then, under $u$, $k''$ traverses the same path $k'$ traverse under $v$. Therefore, $M$ does not sort one of these input vectors. This contradicts the fact that both $v$ and $u$ are valid. \qed
Chapter 4

The bypass transformation

This chapter presents several transformations of a network that produce a smaller network by removing some comparators. This proves helpful for producing networks of a certain functionality out of larger ones, for removing degenerate comparators and for analyzing AMOP merging networks.

The most elementary transformation, called the \textit{bypass transformation}, bypasses a given comparator $c$ as follows. The comparator $c$ is removed and each incoming edge of $c$ is merged with a distinct outgoing edge of $c$; the \texttt{min/max} type of the resulting edge is that of the incoming edge it replaces. A comparator can be bypassed in two different manners as depicted in Figure 4.1. Note that the bypass transformation does not change the width of the network or its input and output labels.

![Figure 4.1: Networks (b) and (c) derive from network (a) by bypassing the comparator $c$.](image)

The concept of a minor network is a generalization of the bypass transformation. Namely, a network $N'$ is a \textit{minor} of a network $N$ if $N'$, or a network isomorphic to $N'$, is derived from $N$ by a sequence of bypass transformations. Sometimes we need to keep track of the association of the members of $N'$ with those of $N$. To this end, for
two networks $N$ and $N'$, we say that the network $N'$ is a minor of the network $N$ via the embedding $\sigma$ if $\sigma$ is an embedding of $N'$ into $N$ having the following properties:

1. The embedding $\sigma$ maps the input/output edges of $N'$ onto the input/output edges of $N$ in a one-to-one manner.

2. The congestion of any edge $e \in N$ is exactly one – there is a unique edge $e' \in N'$ such that the path $\sigma(e')$ passes through $e$.
   
   For any edge $e' \in N'$, let $\sigma(e')_1$ and $\sigma(e')_\infty$ denote the first and last edges of the path $\sigma(e')$. As discussed shortly, $\sigma(e')$ has at least one edge; hence $\sigma(e')_1$ and $\sigma(e')_\infty$ are always defined.

3. For any input (output) edge $e' \in N'$, the two edges $e'$ and $\sigma(e')_1$ ($e'$ and $\sigma(e')_\infty$) have the same label.

4. For any non-input edge $e' \in N'$, $e'$ and $\sigma(e')_1$ have the same min/max type.

We refer to the unique edge provided by requirement (2) as $\sigma^{-1}(e)$. Since $N$ and $N'$ are networks, the incoming degree and the outgoing degree of any internal vertex is exactly two. This fact and requirement (2) imply that the load of any vertex of $N$ is at most one – at most one vertex is mapped to it. By requirement (1), the load of input and output vertices is exactly one. As shown shortly, the comparators of $N$ with zero load correspond to those comparators which were bypassed in the construction of the minor. The above fact that the load is at most one imply that the dilation of any edge $e' \in N'$ is at least one – the path $\sigma(e')$ has at least one edge. The following lemma shows that the above two definitions of a minor, one via a sequence of bypasses and the other via an embedding, are equivalent.

**Lemma 4.0.8.** A network $N'$ is a minor of a network $N$ iff $N'$ is a minor of $N$ via some embedding $\sigma$.

**Proof.** Consider the left to right implication. Clearly, if $N'$ is derived from $N$ by a (single) bypass transformation then it is a minor of $N$ via some embedding $\sigma$. With the obvious definition of composition of embeddings, if $N'$ is a minor of $N$ via the embedding $\sigma'$ and $N''$ is a minor of $N'$ via the embedding $\sigma''$ then $N''$ is a minor of $N$ via the composition of $\sigma'$ and $\sigma''$, denoted $\sigma' \circ \sigma''$. This implies the left to right direction of the lemma.

The right to left implication is proven by induction on the number $k$ of comparators of $N$ whose load is zero. If $k = 0$ then $\sigma$ is an isomorphism. Assume $k > 0$. There exists a network $N^*$ and two embeddings $\sigma^1 : N' \rightarrow N^*$ and $\sigma^2 : N^* \rightarrow N$ such that:

1) $\sigma = \sigma^2 \circ \sigma^1$.
2) $N^*$ is derived from $N$ by a (single) bypass transformation.
3) $N^*$ has $k - 1$ comparators of zero load under $\sigma^1$. 

The induction hypothesis implies that a network isomorphic to $N'$ is derived from $N^*$ by a sequence of bypass transformations; hence a network isomorphic to $N'$ is derived from $N$ by a sequence of bypass transformations.

The following two lemmas are straightforward.

**Lemma 4.0.9.** Let $N'$ be a minor of a network $N$ via an embedding $\sigma$, let $e_1$ and $e_2$ be two edges of $N$ and let $N'$ have a path from $\sigma^{-1}(e_1)$ to $\sigma^{-1}(e_2)$. Then $N$ has a path from $e_1$ to $e_2$.

Recall that a network is AMOP if it has at most one path from every input edge to every output edge.

**Lemma 4.0.10.** Any minor of an AMOP network is AMOP.

### 4.1 Bypass charted by a vector

Recall that a comparator can be bypassed in two different manners. This section shows how to use an input vector $v$ to chart (specify) how to bypass a comparator, or a set of comparators. This bypassing does not disturb the functionality of the network w.r.t. $v$ and to any input vector that agrees with the charting vector $v$. To this end, an input vector $v$ of a network $N$ is called *decisive for* a comparator $c$ if two different keys enter $c$ when $v$ is applied to $N$. Let $v$ be an input vector of a network $N$ which is decisive for a comparator $c$ and let $e_1$ and $e_2$ be the two incoming edges of $c$ such that $\mathcal{V}(e_1, v) < \mathcal{V}(e_2, v)$. The \textit{$v$-charted bypassing of $c$} is the bypassing of $c$ in which $e_1$ is merged with the outgoing min edge of $c$ and $e_2$ is merged with the outgoing max edge of $c$. We refer to the resulting network as $\varphi(N, c, v)$.

Let $C$ be a set of comparators in $N$. If $v$ is decisive for every member of $C$ we say that $v$ is *decisive for* $C$. For such $N, C$ and $v$, let $\Pi(N, C, v)$ denote the set of the paths $p$ of $N$ having the following properties: $p$ has at least two vertices. The first and last vertices of $p$ are not members of $C$, all the other vertices of $p$ are members of $C$ and under $v$ the same key traverses all the edges of $p$. Note that $\Pi(N, C, v)$ covers all the edges of $N$ and each edge is covered exactly once; moreover, if $c$ is an internal vertex of a path $p \in \Pi(N, C, v)$ then there is exactly one more path $p' \in \Pi(N, C, v)$ such that $c$ is an internal vertex of $p'$.

Let $v$ be an input vector of a network $N$ decisive for a set of comparators $C$. The \textit{$N$ minus $C$ $v$-charted minor}, denoted $\varphi(N, C, v)$, is the network $N'$ defined as follows:

- The vertices of $N'$ are the vertices of $N$ except of those which are members of $C$.  

• The edges of $N'$ are the members of $\Pi(N, C, v)$. Let $p \in \Pi(N, C, v)$ lead from $x$ to $y$ in $N$. Then in the context of $N'$, $p$ is an edge from $x$ to $y$. The $\text{min/\max}$ type of $p$ in $N'$ is the $\text{min/\max}$ type of the first edge of $p$ in $N$.

• If $p$ is an input (output) edge of $N'$ then its label is the label of the first (last) edge of the path $p$ in $N$.

It is not hard to establish that $N'$ is a network; to this end, the following items should be checked:

1. $N'$ is acyclic.

2. $N'$ has three types of vertices. The input vertices have in-degree zero and out-degree one; the output vertices have in-degree one and out-degree zero; the internal vertices have in-degree and out-degree two. These internal vertices function as comparators.

3. Of the two edges exiting a comparator of $N'$, one is a $\text{min}$ edge and the other is a $\text{max}$ edge.

It is not hard to check that $N'$ is a minor of $N$ via the embedding $\sigma(N, C, v)$ which is the identity function over the the vertices and the edges of $N'$. (Note that an edge $p$ of $N'$ is a path of $N$.)

We remind the reader that two functions, $g$ and $g'$ agree on a pair of elements $(x, y)$ if both $g$ and $g'$ are defined over $x$ and $y$ and if $x$ and $y$ can be named $z_1$ and $z_2$ such that $g(z_1) \leq g(z_2)$ and $g'(z_1) \leq g'(z_2)$. We henceforth use the term ‘agree’ in an additional manner as follows: Let $v$ and $u$ be two input vectors of a network $N$ and let $e_1$ and $e_2$ be the incoming edges of a comparator $c$ of $N$. We say that $v$ and $u$ agree on $c$ if $v^N$ and $u^N$ agree on the pair $(e_1, e_2)$. Recall that, $v^N$ and $u^N$ denote the natural extension of the input vectors $v$ and $u$ over all the edges of $N$.

Recall that for an input vector $v$ of a network $N$, $T^N(v)$ is the output vector generated by applying $v$ to $N$. The following lemma is straightforward.

**Lemma 4.1.1.** Let $v$ be an input vector of a network $N$, decisive for a set of comparators $C$; let $N' = \varphi(N, C, v)$ and $\sigma = \sigma(N, C, v)$; let $u$ agree with $v$ on each comparator of $C$. Then:

a) If $u$ is decisive for $C$ then $\varphi(N, C, u) = \varphi(N, C, v)$ and $\sigma(N, C, u) = \sigma(N, C, v)$.

b) $u^N(e) = u^{N'}(\sigma^{-1}(e))$ for any edge $e$ of $N$.

c) $T^{N'}(u) = T^N(u)$. 
Bypassing degenerate comparators. The $v$-charted bypassing transformation enables us to remove all the degenerate comparators of a network without disturbing its functionality, as follows. Let $N$ be a network having a certain functionality, let $V$ be the set of its valid vectors and let $C$ be the set of the comparators of $N$ that are degenerate under $V$. Let $v \in V$ be a non-repeating input vector\footnote{We implicitly assume that $V$ has such a vector.}. Since $v$ is non-repeating, it is decisive for all of the comparators of $N$ and so $\varphi(N, C, v)$ is well defined. Any two input vectors of $V$ agree on any comparator of $C$; hence, by Lemma 4.1.1, $\varphi(N, C, v)$ is independent of $v$ (as long as $v$ is a non-repeating member of $V$). Define $\varphi(N, C, V) \triangleq \varphi(N, C, v)$. Lemma 4.1.1 implies the following lemma:

**Lemma 4.1.2.** Let $C$ be the set of comparators of a network $N$ which are degenerate under a set of input vectors $V$. Then:

a) The input/output transformations of $N$ and $\varphi(N, C, V)$ are identical over the members of $V$; that is, $T^N(v) = T^{\varphi(N, C, V)}(v)$ for every $v \in V$.

b) The network $\varphi(N, C, V)$ has no degenerate comparators w.r.t. $V$.

Usually, the functionality of a network, and therefore its valid vectors, are clear from the context. In this case, where $V$ is the set of valid vectors and $C$ is the set of comparators of $N$ degenerate under $V$, we simply denote the network $\varphi(N, C, V)$ by $\text{undeg}(N)$.

### 4.2 Splitting of a network

This section presents a transformation of a given network into a disjoint sum of several smaller networks. To this end, we present a special minor charted by a given input vector $v$. The number of disjoint components of this minor equals the number of different keys in $v$. We usually apply this transformation with input vectors having a small number of different keys, either two or three.

A comparator $c$ of a network $N$ is *mixed under* an input vector $v$ if two different keys enter it under $v$ (i.e. $v$ is decisive for $c$). Let $\text{mix}(N, v)$ denote the set of comparators of $N$ that are mixed under $v$. The *split of $N$ by $v$*, denoted $\text{split}(N, v)$, is the network $\text{split}(N, v) \triangleq \varphi(N, \text{mix}(N, v), v)$. It is easy to see that $\text{split}(N, v)$ is composed of several disjoint components; namely for each different key $k$ that appears in $v$ there is a disjoint component in which all the edges carry the key $k$ under $v$. Next we show two applications of the split transformation that transform any merging network into a smaller network having a meaningful functionality.
4.2.1 Producing half merging networks

Our first application transforms any merging network into a network of a weaker functionality – a network that merges only a subset of the bisorted sequences.

A bisequenced vector \( \langle \vec{a}, \vec{b} \rangle \) of width \( 4n \) is halved if \( a_i, b_i \leq a_j, b_j \) whenever \( i < n \leq j \). A network is a half-merging network if it sorts all vectors which are both bisorted and halved.

For the next lemma we need the following terminology. A network \( N \) is normalized if for every letter \( \alpha \), the two sets \( \{ i \in N | \hat{\alpha}_i \text{ is a label of an input edge of } N \} \) and \( \{ i \in N | \hat{\alpha}_i \text{ is a label of an output edge of } N \} \) are initial intervals of \( N \) (which could be empty). A network \( N' \) is the normalized variant of a network \( N \) if \( N' \) is normalized and is derived from \( N \) by simply replacing the labels of \( N \) via some monotonic function over the labels indices. In other words, let \( e \) be an input (output) edge of \( N \) labelled \( \alpha_i \) and assume there are \( k \) other input (output) edges of \( N \) labelled by \( \alpha_j \) for some \( j < i \). Then the edge \( e \) is labelled by \( \alpha_k \) in \( N' \). The following lemma is straightforward.

**Lemma 4.2.1.** Any non-degenerate half-merging network of width \( 4n \) is composed of two disjoint networks, each of width \( 2n \). One of these networks is a non-degenerate merging network and the normalized variant of the other is a non-degenerate merging network.

Let \( \zeta^{4n} \) be the unique halved bisequenced, \( 0-1 \) vector of width \( 4n \) having exactly \( 2n \) zeroes. (Note that \( \zeta^{4n} \) has exactly \( 2n \) ones and is bisorted.) When the width of \( \zeta^{4n} \) is clear from context, we omit the superscript \( 4n \). We say that a comparator \( c \) of a merging network \( M \) is mixed if it is mixed under \( \zeta \). For a merging network \( M \) of width \( 4n \), we define \( \text{split}(M) \triangleq \text{split}(M, \zeta^{4n}) \). An example of such a splitting is depicted in Figure 4.2. The following lemma is straightforward.

**Lemma 4.2.2.** A bisequenced vector of width \( 4n \) is halved iff it agrees with \( \zeta^{4n} \).

**Lemma 4.2.3.** Let \( c \) be a non-degenerate comparator of a merging network \( M \) of width \( 2n \) which is mixed. Then \( n \in \mathcal{V}^M(c) \).

**Proof.** Let \( v \in \mathbb{P}^{bs}_{2n} \) be halved and let \( h : \mathcal{K} \rightarrow \{0,1\} \) be the monotonic key function defined by:

\[
h(k) = \begin{cases} 
0 & k < n \\
1 & k \geq n . 
\end{cases}
\]

Clearly, \( \zeta = h(v) \). Since \( c \) is mixed, its incoming edges can be named \( e_0 \) and \( e_1 \) so that \( \mathcal{V}^M(e_i, \zeta) = i \). By Lemma 3.0.2, \( h(\mathcal{V}^M(e_i, v)) = \mathcal{V}^M(e_i, \zeta) \) and therefore \( \mathcal{V}^M(e_0, v) < n \) and \( \mathcal{V}^M(e_1, v) \geq n \). By Lemma 3.0.5 (b), \( \mathcal{V}^M(c) \) is an interval; hence \( n \in \mathcal{V}^M(c) \). \( \square \)
Lemmas 4.1.1 and 4.2.2 imply the following lemma.

**Lemma 4.2.4.** Let $M$ be a merging network of width $4n$. Then $\text{split}(M)$ is a half-merging network composed of two disjoint networks, each of width $2n$. One of these networks is a merging network and the normalized variant of the other is a merging network.

When $\zeta$ is applied to $\text{split}(M)$, all the edges of the first network of Lemma 4.2.4 carry the key 0 and all the edges of the second network carry the key 1. We refer to the first network and to the normalized variant of the other as $\text{split}_D(M)$ and $\text{split}_U(M)$, respectively.

**Figure 4.2:** The half-merging network (b) is the split of the merging network (a). This split is generated by bypassing the bold comparators.

### 4.2.2 Pruning a network

Our second application of the split transformation, called *pruning*, changes the width of the network, in contrast to all previous transformations that preserve the width. Pruning is based on the following procedure. All the members of a predefined set of input edges, called $I^{+\infty}$, are wired to receive the fictitious value $+\infty$ (which is greater than all the real keys). Similarly, all the members of a predefined set of input edges, called $I^{-\infty}$, are wired to receive the fictitious value $-\infty$. The other input edges receive real keys. Under this condition, the routing performed by some comparators is predefined and the pruning transformation discards these comparators. A restricted variant of this technique, related to Knuth’s diagram (see Chapter 11) is a common knowledge. See, for example, [16] and [9].
Pruning can be formalized via the split transformation as follows. We use an input vector $v$ of three values ($-\infty, 0, +\infty$). The key $+\infty$ appears on all the members of $I^{+\infty}$, the key $-\infty$ appears on all the members of $I^{-\infty}$ and zero appears on all other input edges. The network $\text{split}(N, v)$ has three disjoint components, one for each of the keys in $v$. All the edges of such a component carry the same value under $v$. We discard the two components associated with the fictitious values $+\infty$ and $-\infty$. The resulting network – the component associated with the value 0 – is the pruning of $N$ by $v$ and is denoted $\text{prun}(N, v)$. If the network $N$ has a known functionality and $v$ is valid for this functionality then we say that the transformation of $N$ into $\text{prun}(N, v)$ is an honest pruning. The following lemma is immediate.

**Lemma 4.2.5.** Let $M$ be a merging network and let $\text{prun}(M, v)$ be derived by an honest pruning. Assume that the key 0 appears the same number of times in the two sequences composing $v$. Then the normalized variant of $\text{prun}(M, v)$ is a merging network.

The above restriction regarding the number of zeroes in $v$ is due to our definition of a merging network. If we allow a ‘merging network’ to merge sequences of different width then this requirement can be lifted.

### 4.3 Minors of merging networks

A merging network is **compact** iff it is both AMOP and non-degenerate. The next lemma shows that the term ‘compact merging network’ is appropriate since comparators of such a network can not be bypassed without disturbing its merging functionality. The proof of this lemma uses the following notation. For an edge or a vertex $x$ of a network $N$, let $IC^N(x)$ denote the subnetwork composed of all edges and vertices of $N$ that have a path to $x$. For such $x$ and $N$, define $O^N(x) \triangleq \{i| \text{there is a path from } x \text{ to the output edge } \hat{o}_i \text{ in } N\}$.

**Lemma 4.3.1.** Let $M$ be a compact merging network and let $M'$ be a minor of $M$ and a merging network. Then $M \cong M'$.

**Proof.** Let the width of $M$ be $2n$. By Lemma 4.0.8, $M'$ is a minor of $M$ via some embedding $\sigma$. Without loss of generality, we may assume that $\sigma$ is the identity function over the vertices of $M'$; hence the vertices of $M'$ are vertices of $M$. Assume, for a contradiction, that $M \not\cong M'$ and let $c$ be a comparator of $M$ of minimal depth which is not a member of $M'$. 

\[\text{Technion - Computer Science Department - Technical Report CS-2007-16 - 2007}\]
Let $e_1$ and $e_2$ be the outgoing edges of $c$, let $d_1$ and $d_2$ be the incoming edges of $c$, let $p_1 = \sigma^{-1}(e_1)$, let $p_2 = \sigma^{-1}(e_2)$ and assume, without loss of generality, that $p_1 = \sigma^{-1}(d_1)$. Since $M$ is AMOP, $O^M(e_1) \cap O^M(e_2) = \phi$. This and Lemma 4.0.9 imply that $O^M(p_1) \cap O^M(p_2) = \phi$. Lemma 3.0.5 and the fact that $c$ is non-degenerate imply that $V^M(d_1) \cap V^M(d_2) \neq \phi$. By the minimality of $c$, $IC^M(d_i) \cong IC^M(p_i)$ for $i \in \{1, 2\}$; therefore $V^M(p_1) \cap V^M(p_2) \neq \phi$. Let $j \in V^M(p_1) \cap V^M(p_2)$. Then for some $p \in \{p_1, p_2\}$: $j \in V^M(p)$ but $j \notin O^M(p)$. This contradicts the fact that $M'$ is a merging network.

A comparator $c$ of a network $N$ having a certain functionality is redundant for this functionality if $c$ (by itself) can be bypassed without disturbing this functionality. Clearly, any degenerate comparator, for a certain functionality, is also redundant under this functionality; however a redundant comparator is not necessarily degenerate. A set of comparators $C$ of a network $N$ having a certain functionality is redundant for this functionality if $C$ can be bypassed without disturbing this functionality. By Lemma 4.3.1, a compact merging network has no single redundant comparator; moreover, it has no redundant (non empty) set of comparators. The second statement does not follow from the first. We now construct a merging network having no single redundant comparator while having a set of redundant comparators.

Following Batcher [1], a sequence is bitonic\(^2\) iff it is a cyclic rotation of a concatenation of an ascending sequence and a descending sequence. A bitonic sorter of width $n$ is a network $B$ which sorts all bitonic sequences of width $n$. That is, its input labels are $\{\hat{x}_i | i \in [0, n]\}$ and its output labels are $\{\hat{y}_i | i \in [0, n]\}$. Whenever its input vector $\vec{x}$ is bitonic, $T^B(\vec{x})$ is sorted. Batcher, in his famous paper [1], also recursively constructed a bitonic sorter of minimal depth.

We first present a bitonic sorter $B$ of width four that has no single redundant comparator but a set of two redundant comparators. The network $B$ works as follows. A comparator $c'$ sorts the pair $x_0$ and $x_1$ into the sequence $\vec{y}$ and a comparator $c''$ sorts the pair $x_2$ and $x_3$ into the sequence $\vec{z}$. The bitonic sequence $\langle y_0, y_1, z_1, z_0 \rangle$ is now sorted by Batcher’s bitonic sorter. Clearly, $B$ is a bitonic sorter (In fact, it is a sorting network.). It is not hard to check that, with respect to the functionality of bitonic sorting, $B'$ has no single redundant comparator while the set $\{c', c''\}$ is redundant.

We next use the network $B$ to generate such an example for the functionality of merging. If a vector $v = \langle \bar{a}, \bar{b} \rangle$ is bisorted then $\bar{a} \cdot \text{rev}(\bar{b})$ is bitonic. This fact implies a natural method to convert via relabelling any bitonic sorter into a merging network. The merging networks generated this way from Batcher bitonic sorter are called Batcher bitonic merging network. Consider the Batcher bitonic merging network of width 16. This network is a layered graph and the first two layers of this

\(^2\)We caution the reader that some authors use the term bitonic with another meaning.
network partition the input into four sequences, each of width 4, s.t. any key in one sequence is less or equal to any key in the following sequence. Furthermore, the two middle sequences are bitonic and any bitonic sequence can be generated this way. Therefore, replacing any of the bitonic sorters used to sort these middle sequences with the above network $B'$ generates a merging network having no single redundant comparator while having a set of two redundant comparators.

We next show that the property of having no redundant set of comparators is too weak to characterize the Batcher merging networks. Fig 4.3 shows how to construct a counter-example of width 8. This network is based on the Tri-section technique (introduced in Chapter 16). In a nutshell, for any $n, j, k$ such that $n = j + k$, the Tri-section technique enables via a depth one network, to transform a bisorted vector of width $2n$ into three sequences $\vec{x}, \vec{y}, \vec{z}$, s.t. all the elements of $\vec{x}$ are greater or equal to all the elements of $\vec{y}$ which are greater or equal to all the elements of $\vec{z}$. Moreover, $\vec{x}$ is ascending-descending and of width $j$, $\vec{y}$ is bitonic and of width $n$ and $\vec{z}$ is descending-ascending and of width $k$. This network for $n = 4, j = 3$ and $k = 1$ is the network $T$ of Fig 4.3. Our counter-example $M$ starts with the network $T$ and then sorts the sequence $\vec{y}$ using Batcher’s bitonic sorter and sorts $\vec{x}$ using the sorting network $X$ of Fig 4.3. The network $X$ is not AMOP and so is $M$. By our main theorem (to be proved in Chapter 10), $M$ is not a Batcher merging network. It is not hard to see that $M$ does not have any non-empty set of redundant comparators.

Figure 4.3: An outline of a merging network having no redundant set of comparators which is not a Batcher merging network.
Chapter 5

The Batcher merging networks

This chapter elaborates on the Batcher merging networks family. As said in the introduction, this family of networks is based on a certain construction to combine two merging networks into a larger merging network via some depth one network. We establish some of their properties and computes the number of such non-isomorphic networks and we show this construction is unique in a strong sense.

Recall that the elements of a sequence \( \vec{a} \) of width \( n \) are denoted \( a_0, a_1, \ldots, a_{n-1} \). Let \( \text{even}(\vec{a}) \) and \( \text{odd}(\vec{a}) \) be the two subsequences of \( \vec{a} \) made of the elements in the even and in the odd positions of \( \vec{a} \), respectively. (Hence, the first member of \( \text{even}(\vec{a}) \) is the first member of \( \vec{a} \).)

A Batcher cleaver is a certain way to partition a bisequenced vector whose width is a multiple of four into two bisequenced vectors of equal width. Namely, there are exactly two Batcher cleavers and each partitions a bisequenced vector \( v = \langle \vec{a}, \vec{b} \rangle \) into two bisequenced vectors \( v^1 = \langle \vec{a}^1, \vec{b}^1 \rangle \) and \( v^2 = \langle \vec{a}^2, \vec{b}^2 \rangle \) in one of the following manners:

- **Parallel manner:** \( \vec{a}^1 = \text{even}(\vec{a}), \vec{a}^2 = \text{odd}(\vec{a}), \vec{b}^1 = \text{even}(\vec{b}), \vec{b}^2 = \text{odd}(\vec{b}) \)

- **Cross manner\(^1\):** \( \vec{a}^1 = \text{even}(\vec{a}), \vec{a}^2 = \text{odd}(\vec{a}), \vec{b}^1 = \text{odd}(\vec{b}), \vec{b}^2 = \text{even}(\vec{b}) \)

For a Batcher cleaver \( r \) and a vector \( v \) as above we denote \( r^1(v) = v^1 \) and \( r^2(v) = v^2 \). The Batcher construction combines two merging network of width \( 2n \), \( M_1 \) and \( M_2 \), into a merging network \( M \) of width \( 4n \). We present this construction in a functional manner; however the network derived from this construction is straightforward.

\(^1\)Beware of the term odd/even. Batcher used this term for the parallel manner while Leighton used the same term for the cross manner. To avoid confusion we decided to use a new terminology altogether.
1. The bisorted input vector $v$ is partitioned into two bisorted input vectors $v^1$ and $v^2$ by a Batcher cleaver $r$.

2. For $i \in \{1, 2\}$, each $M_i$ sorts the input vector $v^i$.

3. It is known [1, 13] that the two output sequences of $M_1$ and $M_2$ can be merged by a network of depth one (which depends on $r$). In $M$, the two output sequences of $M_1$ and $M_2$ are merged via such a network $Z$. (The input labels of $Z$ specifies how these two sequences enter $Z$.) It is also required that none of the comparators of $Z$ is degenerate in $M$.

We refer to the network $M$ constructed as above by $\mathcal{B}(r, M_1, M_2, Z)$. Such a network is defined only when all the four objects in question satisfy the above definition and, in particular, $Z$ does not produce any degenerate comparators.

When $r$ is the parallel cleaver we say that $M$ is constructed in the parallel manner. In this case the even (odd) members of $\vec{a}$ are matched with the even (odd) members of $\vec{b}$. When $r$ is the cross cleaver we say that $M$ is constructed in the cross manner. In this case the even (odd) members of $\vec{a}$ are matched with the odd (even) members of $\vec{b}$. As said, Batcher has presented [1] the parallel construction which uses the parallel cleaver; the cross construction is a straightforward generalization of the parallel one and is mentioned by Leighton [13, pp 623]. We show shortly that each of the two Batcher constructions provide a single way (via a unique network $Z$) to combine the given networks $M'$ and $M''$ into a larger merging network. Furthermore, we show (Lemma 5.0.10) that these constructions are unique in a strong sense.

The family of the Batcher merging networks is defined recursively as follows: A Batcher merging network of width two is a single comparator. A Batcher merging network of width $4n$ is of the form $\mathcal{B}(r, B_1, B_2, Z)$ where $B_1$ and $B_2$ are Batcher merging networks of width $2n$. This implies that the width of a Batcher merging network is a power of two.

We name the Batcher merging networks whose recursive construction uses exclusively the parallel cleaver as the strictly-parallel networks\(^2\). We name the Batcher merging network whose recursive construction uses exclusively the cross cleaver as the strictly-cross network. In [1] Batcher has presented another merging network called the bitonic merging network which is based on his concept of bitonic sequences (see Chapter 14). We show later (Lemma 14.0.8) that the bitonic merging network is isomorphic to the strictly cross network. This isomorphism was also observed by Dalpiaz and Rizzi [17].

A straightforward induction establish the following properties of Batcher merging networks:

\[^2\] These networks are the even-odd networks of Batcher [1]
Lemma 5.0.2. A Batcher merging network is AMOP.

For a bisequenced vector \( v = \langle \tilde{a}, \tilde{b} \rangle \) of width 2\( n \), let 2\( v \) denote the bisequenced vector \( \langle \tilde{a}', \tilde{b}' \rangle \) of width 4\( n \) defined by: \( \tilde{a}'_{2i} = \tilde{a}'_{2i+1} = a_i \) and \( \tilde{b}'_{2i} = \tilde{b}'_{2i+1} = b_i \) for \( i \in [0, n) \). Clearly, for any bisequenced vector \( v \) and for any Batcher cleaver \( r \), \( r^1(2v) = r^2(2v) = v \).

The following discussions concerns all merging networks built by the \( B \) operator rather than only Batcher merging networks.

Lemma 5.0.3. Let \( M = B(r, M_1, M_2, Z) \). A comparator \( c \) is degenerate in \( M \) iff \( c \) belongs to \( M_1 \) or \( M_2 \) and it is degenerate in this network.

Proof. By definition, such a comparator is not a member of \( Z \). Let \( c \) be a comparator of \( M_1 \). When \( M \) receives a valid vector then \( M_1 \) receives a valid vector and any valid vector of \( M_1 \) can be generated this way. The second statement follows from the fact that \( r^1(2v) = v \). This implies that \( c \) is degenerate in \( M_1 \) iff it is degenerate in \( M \). The case where \( c \) belongs to \( M_2 \) is similar. \( \square \)

A straightforward induction using Lemma 5.0.3 imply that:

Lemma 5.0.4. A Batcher merging network has no degenerate comparators.

We use the even-odd vector of width 2\( n \), denoted \( v^{e,o}_{2n} \), which is the bisorted permutation \( \langle \tilde{a}, \tilde{b} \rangle \) of width 2\( n \) in which the sequence \( \tilde{a} \) contains the even numbers and the sequence \( \tilde{b} \) contains the odd ones. When the width of this vector is clear from context, we omit the subscript and use the notation \( v^{e,o} \).

Lemma 5.0.5. Let \( M = B(r, M_1, M_2, Z) \). Then the network \( Z \) is determined by \( r \) and the width of \( M \).

Proof. Note that if \( B(r, M_1, M_2, Z) \) is defined then it is a merging network. We show that, under \( v^{e,o} \), \( r \) determines, for any key \( k \) of \( v^{e,o} \), whether \( k \) enters a comparator in \( Z \) and if so which key \( k \) encounters in \( Z \). This fact completely defines the network \( Z \).

Let the width of \( M_1 \) be 2\( n \). Let \( k \) be a key of \( v^{e,o} \) and \( k \notin \{0, 4n - 1\} \). By Lemma 3.0.7, \( k \) must encounter both \( k - 1 \) and \( k + 1 \). One of these keys, let call it \( k' \), does not enter the same subnetwork (\( M_1 \) or \( M_2 \)) as \( k \); hence it must encounter \( k \) in \( Z \); moreover \( k' \) is determined by \( r \). The same argument holds for \( k \in \{0, 4n - 1\} \) when \( r \) is the cross cleaver. That is, a key \( k' \), determined by \( r \), encounters \( k \) and does not enter the same subnetwork as \( k \).

It remains to consider the case where \( k \in \{0, 4n - 1\} \) and \( r \) is the parallel cleaver. In this case each key of the interval \([1, 4n - 1]\) encounters another key of this interval in \( Z \). The two remaining keys 0 and \( 4n - 1 \) can not meet in a comparator of \( Z \) because, by Lemma 3.0.5, such a comparator is degenerate. \( \square \)
Note that the proof of Lemma 5.0.5 does not use the fact that $M_1$ and $M_2$ are merging networks; it only uses the fact that $M$ is a merging networks.

**Lemma 5.0.6.** Let $N^i = B(r^i, M^i_1, M^i_2, Z^i)$ for $i \in \{1, 2\}$ and let $N^1 \cong N^2$. Then $r^1 = r^2$, $M^1_1 \cong M^2_1$, $M^1_2 \cong M^2_2$ and $Z^1 \cong Z^2$.

*Proof.* It suffices to show that, for any $M = B(r, M_1, M_2, Z)$, synd$(M, v_{e,o})$ determines all the above objects -- $r, M_1, M_2$ and $Z$.

Consider $r$. By the arguments of the proof of Lemma 5.0.5, under $v_{e,o}$, the last key encountered by the key 1 is either 0 or 2. The former case holds when $r$ is a cross cleaver and the latter case holds when $r$ is a parallel cleaver. Hence, $r$ is determined by the syndrome.

Consider $M_1$ and let $v' = r^1(v_{e,o})$. Recall that when $v_{e,o}$ is applied to $M$, $v'$ is the applied to $M_1$. By Lemma 2.0.1, it suffices to show that synd$(M, v_{e,o})$ determines synd$(M_1, v')$. Let $k$ be a key of $v'$. By the proof of 5.0.5, $r$ and $k$ determine whether $k$ enters a comparator of $Z$; hence act$(M_1, v', k)$ is determined by act$(M, v_{e,o}, k)$ and $r$. Since this holds for any $k$, synd$(M_1, v')$ is determined by synd$(M, v_{e,o})$. This determines $M_1$ and the network $M_2$ is determined by the same manner. Lemma 5.0.5 completes the proof.

Induction and Lemma 5.0.6 establish:

**Lemma 5.0.7.** For any $n$, a power of two, there are exactly $2^{n-1}$ non-isomorphic Batcher merging networks of width $2n$.

We say that two input edges $\hat{a}_i$ and $\hat{b}_j$ of a merging network $M$ are matched, if they enter the same comparator. Let the width of $M$ be $2n$. The input matching function (i.m.f.) of $M$, denoted $\text{imf}^M$, is the partial function from $[0, n)$ to itself defined by $\text{imf}^M(i) = j$ iff the input edges $\hat{a}_i$ and $\hat{b}_j$ are matched. Clearly, an i.m.f. is one-to-one. We say that an i.m.f. of a network $M$ is total iff every input edge of $\hat{a}$ is matched with an input edge of $\hat{b}$. The following lemma follows directly from the definition of Batcher merging networks.

**Lemma 5.0.8.** The i.m.f. of any Batcher merging network is total.

**Lemma 5.0.9.** A Batcher merging network $B$ is determined by $\text{imf}^B$. 
Proof. The proof is by induction on $2n$, the width of $B$. The case of $n = 1$ is trivial so let $n > 1$ and $B = B(r, B_1, B_2, Z)$. The cleaver $r$ is determined by the even/odd parity of $\text{imf}^B(0)$. Clearly, $\text{imf}^B$ and $r$ determines $\text{imf}^{B_1}$ and $\text{imf}^{B_2}$. This, by the induction hypothesis, determine $B_1$ and $B_2$. By Lemma 5.0.5, $r$ determines $Z$. \qed

The following lemma states that if a merging network $M$ is constructed from two networks $M'$ and $M''$ of arbitrary functionalities and widths in a manner similar to the Batcher construction than $M'$ and $M''$ must be merging networks of the same width and the construction is actually a Batcher construction. Two edges $e_1$ and $e_2$ of a network $N$ are called disagreeable iff there are two valid input vectors of $N$, $v$ and $u$, such that $v^N(e_1) > v^N(e_2)$ and $u^N(e_1) < u^N(e_2)$. A sandwich is a bisorted vector $\langle \vec{a}, \vec{b} \rangle$ that can be made sorted by sandwiching the entire $\vec{a}$ sequence between an initial part of $\vec{b}$ and the rest of $\vec{b}$; this also includes the case where the initial part of $\vec{b}$ or the rest of $\vec{b}$ is empty.

Lemma 5.0.10. Let $n \in \mathbb{N}$ and let $M$ be a merging network of width $4n$ of the following form:

- The network $M$ is a concatenation of a network $N$ followed by a depth one network $Z$.
- The network $N$ be a disjoint sum of two networks $M'$ and $M''$.
- No comparator of $Z$ is degenerate in $M$.
- The input edge $a_0$ of $M$ enters $M_1$.

Then $M = B(r, M_1, M_2, Z)$ for some $r, M_1, M_2$ and $Z$ where $M_1$ and $M_2$ are derived from $M'$ and $M''$ by a relabelling that normalizes the input edges.

Proof. We first show that the input vector $v = \langle \vec{a}, \vec{b} \rangle$ of $M$ must be partitioned into $M'$ and $M''$ using one of the two Batcher cleavers; that is, even($\vec{a}$) and odd($\vec{a}$) go to different subnetworks and the same holds for $\vec{b}$.

Assume, for a contradiction, that two consecutive keys $a_i$ and $a_{i+1}$ for $i \in [0, n-1)$ enter the same subnetwork, say $M'$ and let $b_j$ enter $M''$. It is not hard to see that such a $b_j$ exists. Clearly, there is a bisorted vector $v$ in which $a_i$ and $b_j$ are adjacent and $b_j$ and $a_{i+1}$ are adjacent. By Lemma 3.0.7, under $v$, $b_j$ encounters both $a_i$ and $a_{i+1}$. Since $M'$ and $M''$ are disjoint these two encounters occur in $Z$. This contradicts the fact that $Z$ is of depth one. By symmetry, the even($\vec{b}$) and odd($\vec{b}$) enter different subnetworks. This implies that the input vector of $M$ is partitioned by some Batcher cleaver.
Next, we show that two merging networks $M_1$ and $M_2$ can be derived from $M'$ and $M''$, respectively, by assigning them output labels and normalizing their input labels. By symmetry, we show it only for $M_1$. We start with a network $\tilde{M}_1$ derived from $M'$ by assigning it temporary arbitrary non-repeating output labels and normalizing its input labels. Next, we show that under the set of bisorted input vectors of $\tilde{M}_1$, no two output edges of $\tilde{M}_1$ are disagreeable. Assume, for a contradiction, that $e'$ and $e''$ are two disagreeable output edges of $\tilde{M}_1$. Hence, there are two valid input vectors $v$ and $u$ of $\tilde{M}_1$ such that $v(e') > v(e'')$ and $u(e') < u(e'')$. Clearly $2v$ and $2u$ are valid input vectors of $M$; furthermore, $e'$ and $e''$ are disagreeable in $M$ under $2v$ and $2u$. Since $M$ is a merging network, by Lemma 3.0.6, $e'$ and $e''$ enter the same comparator of $Z$.

Consider any comparator $c$ of $Z$. Since $c$ is non-degenerate in $M$, $V^M(c) = [k, k+1]$ for some $k \in [0, 4n - 1)$. It is not hard to see that there is a valid permutation input vector of $M$ (in fact, a sandwich vector) in which $k$ and $k+1$ belong to $\vec{a}$ and are consecutive there. Therefore, under any Batcher cleaver used, one key of $\{k, k + 1\}$ goes to $M'$ and the other goes to $M''$. This implies that $c$ receives one edge from $M'$ and the other from $M''$. This contradicts the assumption that $\tilde{M}_1$ has disagreeable edges; hence, a merging network $M_1$ can be derived from $\tilde{M}_1$ by relabelling its output edges.
Chapter 6

Congruent functions

This chapter characterizes, via certain equivalence relations, the input matching functions of the Batcher merging networks. It shows that for any given width, these functions constitute a group.

Let \( N \) denote the set of natural numbers including zero. For \( X, Y \subseteq N \), \( X \) is a congruent class of \( Y \) if \( X \) is an equivalence class of \( Y \) under the equivalence relation “\( p \equiv q \mod 2^j \)” for some \( j \in \mathbb{N} \). A set \( X \) is a congruent class if it is a congruent class of the interval \([0, 2^k)\) for some \( k \in \mathbb{N} \). The following two lemmas are straightforward:

**Lemma 6.0.11.** The relation “\( X \) is a congruent class of \( Y \)” is transitive.

**Lemma 6.0.12.** Let \( Y \subseteq \mathbb{N} \) and \( |Y| > 1 \). Then there is exactly one unordered pair of non-empty sets \((X_1, X_2)\) such that \( X_1 \cap X_2 = \phi, Y = X_1 \cup X_2 \) and \( X_1 \) and \( X_2 \) are congruent classes of \( Y \). Furthermore, if \( Y \) is a congruent class then \( X_1 \) and \( X_2 \) are congruent classes and \( |X_1| = |X_2| \).

We refer to the unordered pair \((X_1, X_2)\), provided by the last lemma, as the congruent partition of \( Y \).

The concept of congruent classes is relevant to the Batcher merging network as follows. Let \( X, Y \subseteq \mathbb{N} \) such that \( |X| = |Y| \). We denote by \( v^{X,Y} \) the bisorted vector \( v^{X,Y} = (\vec{x}, \vec{y}) \) where the image of the sorted sequences \( \vec{x} \) and \( \vec{y} \) are \( X \) and \( Y \) respectively. A vector is congruent if it is of the form \( v^{X,Y} \) where \( X \) and \( Y \) are congruent classes. Batcher merging networks are constructed recursively from smaller Batcher merging networks. By the following lemma, when a Batcher merging network receives a congruent vector, all these smaller Batcher merging networks receive a congruent vector.
Lemma 6.0.13. Let $X, Y \subset \mathbb{N}$ be congruent classes, let $|X| = |Y| > 1$, let $v = v^{X,Y}$, let $r$ be a Batcher cleaver and let $r^1(v) = v^{X_1,Y_1}$ and $r^2(v) = v^{X_2,Y_2}$. Then $(X^1, X^2)$ is a congruent partition of $X$ and $(Y^1, Y^2)$ is a congruent partition of $Y$.

Lemma 6.0.14. Let $(X_1, X_2)$ be the congruent partition of a set $Y$, let $X$ be a congruent class of $Y$ and $X \neq Y$. Then $X$ is a congruent class of either $X_1$ or $X_2$.

Let $X, Y \subset \mathbb{N}$. A congruent function from $X$ onto $Y$ is a bijection from $X$ onto $Y$ under which the image of any congruent class of $X$ is a congruent class of $Y$; we use the notation $f : X \rightleftharpoons Y$ to denote that $f$ is a congruent function from $X$ onto $Y$. A function is a congruent function if it is a congruent function from $[0, 2^j)$ onto $[0, 2^j)$ for some $j \in \mathbb{N}$.

Lemma 6.0.15. For any $j \in \mathbb{N}$ the set congruent functions from $[0, 2^j)$ onto $[0, 2^j)$ is a group under the composition operator.

Proof. It follows from definition that for any two such functions $f$ and $g$, $f \circ g$ is a congruent function and $f^{-1}$ is a congruent function. □

The following lemma presents several congruent functions

Lemma 6.0.16. Let $j, k \in \mathbb{N}$. Then:

a) The permutation $x \mapsto x + k \pmod{2^j}$ of $[0, 2^j)$ is a congruent function.

b) The order reversing\(^1\) permutation of $[0, 2^j)$ is a congruent function.

Proof. Both statements follow from the fact that for every $l$, the relation “$x \equiv y \pmod{2^j}$” is invariant under the permutations in those statements. □

The following lemma provides recursive characterization of congruent functions.

Lemma 6.0.17. Let $X, Y \subset \mathbb{N}$, $|X| = |Y| > 1$. Then $f$ is a congruent function from $X$ onto $Y$ iff there are $(X_1, X_2)$ a congruent partition of $X$, $(Y_1, Y_2)$ a congruent partition of $Y$ and two congruent functions $f_1 : X \rightleftharpoons Y_1$ and $f_2 : X \rightleftharpoons Y_2$ such that $f = f_1 \cup f_2$.

\(^1\)The permutation $x \mapsto 2^j - 1 - x$
Proof. We first prove left to right implication. Let \((X_1, X_2)\) be the congruent partition of \(X\) provided by Lemma 6.0.12. Let \(Y_1\) and \(Y_2\) be the images of \(X_1\) and \(X_2\) under \(f\), respectively. Since \(f\) is a congruent function, \((Y_1, Y_2)\) is a congruent partition of \(Y\).

For \(i \in \{1, 2\}\), let \(f_i\) be the function \(f\) restricted to the set \(X_i\). Clearly, \(f = f_1 \cup f_2\). By Lemma 6.0.14, any congruent class of \(X_i\) is a congruent class of \(X\); hence, the image of any congruent class of \(X_i\) is a congruent class of \(Y_i\), establishing that \(f_1\) and \(f_2\) are congruent functions. The right to left implication follows from Lemmas 6.0.14 and 6.0.11.

Lemma 6.0.17 enables us to compute the number of congruent functions from a congruent class \(X\) onto a congruent class \(Y\). Let \(\Pi(X, Y)\) denote the number of \(X\) to \(Y\) congruent functions. Lemma 6.0.17 imply that \(\Pi(X, Y) = \Pi(X_1, Y_1) \cdot \Pi(X_2, Y_2) + \Pi(X_1, Y_2) \cdot \Pi(X_2, Y_1)\). A simple induction shows that if \(X\) and \(Y\) are congruent classes then \(\Pi(X, Y)\) depends only on \(|X|\) and \(|Y|\). If \(|X| \neq |Y|\) then \(\Pi(X, Y) = 0\). For the other case define \(\Pi(n) = \Pi(X, Y)\) for \(|X| = |Y| = n\). Clearly \(\Pi(1) = 1\) and by Lemma 6.0.17, \(\Pi(2n) = 2 \cdot (\Pi(n))^2\). The solution to this recursive equation is:

Lemma 6.0.18. There are exactly \(2^{n-1}\) congruent functions from \([0, n)\) onto \([0, n)\) when \(n\) is a power of two.

Let \(X, Y \subset \mathbb{N}\) and let \(v^{X,Y} = \langle \vec{a}, \vec{b} \rangle\) be an input vector of a merging network \(M\). Let \(f^{M}(v^{X,Y})\) be the function \(f^{M}(v^{X,Y}) : X \rightarrow X \cup Y\) defined by \(f^{M}(v^{X,Y})(x) = y\) if \(y\) is the first key that the key \(x\) of the \(\vec{a}\) sequence encounters under the input vector \(v^{X,Y}\).

Lemma 6.0.19. A function is a congruent function iff it is the i.m.f. of some Batcher merging network.

Proof. Consider the right to left direction. We first claim that if \(X\) and \(Y\) are congruent classes and \(B\) is a Batcher merging network then \(f^{B}(v^{X,Y})\) is a congruent function from \(X\) onto \(Y\). This claim follows by induction and Lemmas 6.0.17 and 6.0.13. By Lemma 5.0.8, \(\text{imf}^{B}\) is total; and so, for \(X = Y = [0, n)\), \(\text{imf}^{B} = f^{B}(v^{X,Y})\). Therefore, \(\text{imf}^{B}\) is a congruent function. The left to right direction follows from Lemmas 5.0.9, 6.0.18 and 5.0.7. \(\square\)
Chapter 7

The Input Cone

This chapter studies a certain subnetwork of compact merging networks.

The input cone of an edge $e$ of a network $N$, denoted $IC^N(e)$, is the subgraph of $N$ composed of the vertices and edges having a path to $e$. (This subgraph includes $e$.) This section takes special interest of the cone entering the edge $\hat{o}_i$ of a merging network (say $M$) of width $2n$ and denote this cone by $IC^M$. For an edge $e$ of a network $N$, The output cone of $e$, denoted $OC^N(e)$, is the subgraph of $N$ composed of the vertices and edges that can be reached from $e$.

Let $V$ be a set of input vectors of a network $N$ and let $e$ be an edge of $N$. A key $k$ is obliged to $e$ under $V$ if $k$ enters an input edge of $IC^N(e)$. As with other notations, when the network referred to is clear from context we omit it from the above notations and use the shortcuts $IC(e), OC(e)$ and $IC$.

The following lemma follows directly from the definition of the AMOP property.

**Lemma 7.0.20.** Let $e$ be an edge of an AMOP merging network and let $\hat{o}_i \in OC(e)$. Then $i$ is obliged to $e$ under $P^{bs}$.

Let $e_1$ and $e_2$ be the incoming edges of a non-degenerate comparator $c$ of a network $N$ having a certain functionality. Then there are two valid vectors of $N$ that do not agree on the pair $(e_1, e_2)$; that is, these vectors can be named $v'$ and $v''$ such that $v'^N(e_1) > v'^N(e_2)$ and $v''^N(e_1) < v''^N(e_2)$. In this case we say that the unordered pair $(v_1, v_2)$ establishes that $c$ is non-degenerate.

**Lemma 7.0.21.** Let $M$ be an AMOP merging network and let $c$ be a comparator of $IC^M$. Then $c$ is non-degenerate.
Let the width of \( M \) be \( 2n \). Let \( d_1 \) and \( d_2 \) be the two incoming edges of \( c \) and let \( e \) be an outgoing edge of \( c \) where \( e \in IC^M \). By symmetry, it suffices to consider only the case where \( e \) is a \text{min} edge. Since the width of \( M \) is \( 2n \), for each input edge \( d \) there is a valid vector \( v \) s.t. \( v^M(d) = n \). Let \( v_1, v_2 \in P^{bs} \) be two input vectors such that under \( v_i \) the key \( n \) enters an input edge of \( IC^M(v_i) \) for \( i \in \{1, 2\} \). By Lemma 7.0.20, \( n \) is obliged to \( d_1, d_2 \) and \( e \); therefore \( n = \mathcal{V}(e, v_i) = \mathcal{V}(d_i, v_i) \) for \( i \in \{1, 2\} \). Hence, \( \mathcal{V}(d_1, v_1) < \mathcal{V}(d_2, v_1) \) and \( \mathcal{V}(d_1, v_2) > \mathcal{V}(d_2, v_2) \). That is, the unordered pair \( v_1, v_2 \) establishes that \( c \) is non-degenerate.

For \( n \) and \( m \), both powers of two, define an \textit{m-class} of the interval \([0, n)\) to be an equivalence class of \([0, n)\) under the equivalence relation \( "p \equiv q \mod m" \). By definition (Chapter 6), any \textit{m-class} of \([0, n)\) is a congruent class. Recall that a sandwich vector is a bisorted permutation in which the range of the \( \vec{a} \) sequence is an interval. Clearly, there are exactly \( n + 1 \) different sandwich vectors of width \( 2n \); furthermore, for a given width, a sandwich vector is determined by the value of any single element of the \( \vec{a} \) sequence.

Recall that a merging network is regular iff its width is a power of two and that \( O(e) \triangleq \{i|\hat{a}_i \in OC(e)\}, I_A(e) \triangleq \{i|\hat{a}_i \in IC(e)\} \) and \( I_B(e) \triangleq \{i|\hat{b}_i \in IC(e)\} \).

**Lemma 7.0.22.** Let \( M \) be a regular AMOP merging network of width \( 2n \). Then:

\( a) \) \( IC^M \) is a balanced binary tree that contains all the input edges of \( M \).

\( b) \) Let \( e \) be a non-input edge of \( IC^M \). Then \( I_A(e) \) and \( I_B(e) \) are congruent classes of \([0, n)\).

\( c) \) Let \( C \) be a congruent class of \([0, n)\). Then there is a unique non-input edge \( e \in IC^M \) with \( I_A(e) = C \).

\( d) \) The function \( imf^M \) is total and is a congruent function over \([0, n)\).

**Proof.** Consider statement (b). Due to symmetry, it suffices to prove this statement only for \( I_A(e) \). The proof is by induction on the distance from \( e \) to \( \hat{a}_n \). If \( e = \hat{a}_n \), then clearly \( I_A(e) = [0, n) \) which is a 1-class of \([0, n)\). Assume \( e \neq \hat{a}_n \). Let \( e \) and \( e' \) enter the same comparator \( c \) and let \( t' \) and \( t'' \) be the \text{min} and \text{max} outgoing edges of \( c \), respectively. Either \( t' \in IC^M \) or \( t'' \in IC^M \). These two cases are similar and we consider only the former. By the induction hypothesis, \( I_A(t') \) is an \textit{m-class} of \([0, n)\) for some \( m \) a power of two.

Assume, for a contradiction, that \( I_A(e) \) is not a \( 2m \)-class of \([0, n)\). Then there is an edge \( e^* \in \{e, e'\} \) and integer \( i \) such that \( i, i + m \in I_A(e) \) belong to \( IC(e^*) \). Let \( s \) be the sandwich input vector with \( \mathcal{V}((\hat{a}_i, s) = n \). Under this vector, the keys \( n \) and \( n + m \) enter \( IC(e^*) \). By Lemma 7.0.20, \( n \) is obliged to \( e^* \) and so \( \mathcal{V}(e^*, s) = n \). This clearly implies that \( n + m \) is not obliged to \( e^* \) under \( P^{bs} \); therefore, by Lemma 7.0.20, \( \hat{a}_{n+m} \notin OC(e^*) \).
Since $s$ is a sandwich and $I_a(t')$ is an $m$-class of $[0, n)$, under $s$ no key in the interval $(n, n + m)$ enters $IC(t') = IC(t'\prime)$. This and $V(t', s) = n$ imply that $V(t''\prime, s) \geq n + m$. By Lemma 7.0.21, $c$ is non-degenerate and so, by Lemma 3.0.5(b), $V(c) = V(t') \cup V(t''\prime)$ is an interval; therefore $n + m \in V(c)$. This contradicts the above conclusion that $\delta_{n + m} \notin OC(e^*)$ and establishes statement (b). Statements (a) and (c) follow immediately from statement (b) and the properties of congruent classes.

Consider statement (d). Statements (a) and (b) imply that $imf^M$ total. Any input matching function is one-to-one. It remains to show that, for every congruent class $A$, the image of $A$ under $imf^M$ is a congruent class. By statement (c), there exists a non-input edge $e$ such that $I_a(e) = A$. By statement (b), $I_b(e)$ is an congruent class of $[0, n)$. That is, $I_b(e) = imf^M(I_a(e))$ is a congruent class.

The following lemma concerns isomorphism of the structures $IC^{M'}$ and $IC^{M''}$ of two networks $M'$ and $M''$. Note that by our definition (Chapter 2, page 7) these structures are not networks; however the concept of isomorphism is clearly applicable for them. Namely, such an isomorphism has to preserve the edges/vertices connectivity as well as the input labels and the $\text{min}/\text{max}$ types of the edges.

**Lemma 7.0.23.** Let $M'$ and $M''$ be two regular AMOP merging networks with $imf^{M'} = imf^{M''}$. Then $IC^{M'} \cong IC^{M''}$.

**Proof.** We need to show a bijection from $IC^{M'}$ to $IC^{M''}$ which preserves the connectivity, the input labels and the $\text{min}/\text{max}$ types of the edges. Define a mapping $\sigma$ from the edges of $IC^{M'}$ onto the edges of $IC^{M''}$ by:

$$\sigma(e') = \begin{cases} e'' & \text{if } e' \text{ and } e'' \text{ are input edges having the same label.} \\ e'' & \text{if } e' \text{ and } e'' \text{ are not input edges and } I_a(e') = I_a(e''). \end{cases}$$

By Lemma 7.0.22(c), $\sigma$ is well defined and is a bijection from the edges of $IC^{M'}$ onto the edges of $IC^{M''}$. We next show that $\sigma$ preserves the edges connectivity of $IC^{M'}$; i.e. if $\langle e_1, e_2 \rangle$ is a path of length two of $IC^{M'}$ then $\langle \sigma(e_1), \sigma(e_2) \rangle$ is a path of length two of $IC^{M''}$.

Let $y$ be a non-input edge of $IC^{M'}$. By Lemma 7.0.22(b), $I_{a^i}(y)$ is a congruent class of $[0, n)$. First assume that $|I_{a^i}(y)| = 1$; that is $I_{a^i}(y)$ is a singleton, say $\{i\}$. In this case, $\langle \hat{a}_i, y \rangle$ and $\langle \hat{b}_j, y \rangle$ are paths in $IC^{M'}$, for $j = imf^{M'}(i)$. Since $imf^{M'} = imf^{M''}$ the same holds for $\sigma(y), \sigma(\hat{a}_i)$ and $\sigma(\hat{b}_j)$; that is, $\langle \sigma(\hat{a}_i), \sigma(y) \rangle$ and $\langle \sigma(\hat{b}_j), \sigma(y) \rangle$ are paths of $IC^{M''}$.

Assume now that $|I_{a^i}(y)| > 1$, let $\{Y_1, Y_2\}$ be the unique congruent partition of $I_{a^i}(y)$ provided by Lemma 6.0.12. By Lemma 7.0.22(c) there are two unique edges
$y_1$ and $y_2$ s.t. $I_{\text{a}}^{M'}(y_1) = Y_1$ and $I_{\text{a}}^{M'}(y_2) = Y_2$. Clearly, $\langle y_1, y \rangle$ and $\langle y_2, y \rangle$ are pathes in $IC^{M'}$. By definition, $\sigma$ preserves the fact that $(I_{\text{a}}^{M''}(\sigma(y_1)), I_{\text{a}}^{M''}(\sigma(y_2)))$ are the congruent partition of $I_{\text{a}}^{M''}(\sigma(y))$; hence, $\langle \sigma(y_1), \sigma(y) \rangle$ and $\langle \sigma(y_2), \sigma(y) \rangle$ are pathes of $IC^{M''}$. Therefore, $\sigma$ preserves the connectivity of $IC^{M'}$. This implies that $\sigma$ can be extended over the vertices of $IC^{M'}$ while preserving the edges/vertices connectivity.

By definition, $\sigma$ preserves the input labels of $IC^{M'}$. It remains to show that $\sigma$ preserves the min/max type of any edge $e'$ of $IC^{M'}$. This is shown by induction on the depth of $IC(e')$. The case where $e'$ is an input edge is trivial. Let $e'$ emerges from a comparator $c$ and let $e''$ be the edge emerging from $\sigma(c)$ and having the same type as $e'$. We show that $\sigma(e') = e''$.

Let $\bar{\sigma}$ be the mapping from $IC^{M'}(e')$ onto $IC^{M''}(e'')$ such that $\bar{\sigma}$ is identical to $\sigma$ on all the vertices and edges, but $e'$, and $\bar{\sigma}(e') = e''$. By the induction hypothesis, $\bar{\sigma}$ is an isomorphism and so $\bar{\sigma}(e') = e''$. The fact that $n \in V^{M'}(e')$ implies that $n \in V^{M''}(e'')$ and therefore $e'' \in IC^{M''}$. Since $\sigma$ preserves connectivity and since, by Lemma 7.0.22, $IC^{M'}$ and $IC^{M''}$ are trees, it follows that $\sigma(e') = e''$. 

☐
Chapter 8

The i.m.f. of a split

This chapter studies the effect of the split transformation on the i.m.f. of a compact merging network and shows that the resulting i.m.f. is uniquely determined by the original i.m.f.

Lemma 8.0.24. Let $e$ be an edge of a non-degenerate merging network $M$. Then $O(e)$ is an interval.

Proof. Let the width of $M$ be $2n$. The proof is by induction on the depth of $OC(e)$. The case where $e$ is an output edge is trivial. For the general case we use the following fact. For two intervals of integers, $I'$ and $I''$, $I' \cup I''$ is an interval iff there exist $i' \in I'$ and $i'' \in I''$ such that $|i' - i''| \leq 1$.

Let $e$ be an incoming edge of a comparator $c$. Let $e_1$ and $e_2$ be the two outgoing edges of $c$. By the induction hypothesis, $O(e_1)$ and $O(e_2)$ are intervals. By Lemma 3.0.5(a), $V(e_1)$ and $V(e_2)$ are intervals. Clearly, $V(e_i) \subset O(e_i)$ for $i \in \{1, 2\}$. By Lemma 3.0.5(b), $V(e_1) \cup V(e_2)$ is an interval. By the above fact, $O(e_1) \cup O(e_2)$ is an interval.

An input edge $\hat{a}_j$ or $\hat{b}_j$ of a merging network of width $4n$ is called small iff $j < n$; otherwise ($n \leq j < 2n$) it is called large. Recall that an input or output edge whose label is $\alpha$ is named $\alpha$. We now name additional edges of a network. Let $N$ be a network with a comparator whose incoming edges are input edges named $\alpha$ and $\beta$ (i.e. $\alpha$ and $\beta$ are matched). In this case, $\min(\alpha, \beta)$ denote the min edge emerging from this comparator and $\max(\alpha, \beta)$ denote the max edge emerging from this comparator. If $\alpha$ and $\beta$ are not matched then $\min(\alpha, \beta)$ and $\max(\alpha, \beta)$ are undefined.
Lemma 8.0.25. Let \( M \) be a merging network of width \( 2n \) and let the input edges \( \hat{a}_j \) and \( \hat{b}_k \) be matched. Then:

a) \( \mathcal{V}(\min(\hat{a}_j, \hat{b}_k)) = [\min(j, k), j + k] \).

b) \( \mathcal{V}(\max(\hat{a}_j, \hat{b}_k)) = [j + k + 1, \max(j, k) + n] \).

c) If \( M \) is AMOP then \( \min(O(\max(\hat{a}_j, \hat{b}_k))) = \max(O(\min(\hat{a}_j, \hat{b}_k))) + 1 = j + k + 1 \).

Proof. Statements (a) and (b) are straightforward so we prove only statement (c). By Lemma 8.0.24, \( O(\min(\hat{a}_j, \hat{b}_k)) \) and \( O(\max(\hat{a}_j, \hat{b}_k)) \) are intervals. Clearly, \( \mathcal{V}(\min(\hat{a}_j, \hat{b}_k)) \subset O(\min(\hat{a}_j, \hat{b}_k)) \) and \( \mathcal{V}(\max(\hat{a}_j, \hat{b}_k)) \subset O(\max(\hat{a}_j, \hat{b}_k)) \). Since \( M \) is AMOP, \( O(\min(\hat{a}_j, \hat{b}_k)) \cap O(\max(\hat{a}_j, \hat{b}_k)) = \emptyset \); therefore statement (c) follows from statements (a) and (b).

Lemma 8.0.26. Let \( M \) be a regular compact merging network of width \( 4n \) having a comparator whose incoming edges are \( \min(\hat{a}_j, \hat{b}_k) \) and \( \min(\hat{a}_{j'}, \hat{b}_{k'}) \). Then:

a) \( j' - j = k - k' \).

b) If one (or more) of the four edges \( \hat{a}_j, \hat{b}_k, \hat{a}_{j'}, \hat{b}_{k'} \) is large then:

1) \( |j - j'| \geq n \).

2) In each one of the four pairs \( (\hat{a}_j, \hat{a}_{j'}) \), \( (\hat{b}_k, \hat{b}_{k'}) \), \( (\hat{a}_j, \hat{b}_k) \) and \( (\hat{a}_{j'}, \hat{b}_{k'}) \), one edge is small and the other is large.

Proof. Consider Statement (a). Let \( e = \min(\hat{a}_j, \hat{b}_k) \) and \( e' = \min(\hat{a}_{j'}, \hat{b}_{k'}) \). Clearly, \( O(e) = O(e') \). By Lemma 8.0.25 (c), \( j + k = \max(O(e)) = \max(O(e')) = j' + k' \) which implies Statement (a).

Consider Statement (b.1). Due to Statement (a), Statement (b.1) is symmetric with respect to \( \hat{a} \) and \( \hat{b} \). Due to this symmetry and the symmetry w.r.t. \( k \) and \( k' \), we can assume that \( \hat{b}_k \) is large and \( k > k' \). Assume, for a contradiction, that \( |j - j'| < n \). In this case the network \( M \) can be pruned into a network \( \bar{M} \) so that the following conditions hold:

1) Half of the edges of each input sequence \( \vec{a} \) and \( \vec{b} \) are pruned.

2) The pruning is honest; that is, the input vector that charts this pruning is valid for merging.

3) Out of the four input edges \( \hat{a}_j, \hat{b}_k, \hat{a}_{j'} \) and \( \hat{b}_{k'} \), only \( \hat{b}_k \) is pruned and is pruned \( +\infty \). This is possible because \( |j - j'| < n \), \( k > k' \) and because \( k \) is large.

By Lemma 4.0.10, \( M \) is AMOP. In \( \bar{M} \), the two edges \( \min(\hat{a}_{j'}, \hat{b}_{k'}) \) and \( \hat{a}_j \) enter the same comparator; therefore \( \text{int} \bar{M} \) is not total. This contradicts Lemma 7.0.22(d) since the normalized variant of \( \bar{M} \) is a regular AMOP merging network. This contradiction establishes Statement (b.1). Statement (b.2) follows from statements (a) and (b.1).
Lemma 8.0.27. Let $M$ be a regular compact merging network whose width is at least four. Let $\min(\hat{a}_j, \hat{b}_k)$ be an edge of $M$ where $\hat{a}_j$ is small and $\hat{b}_k$ is large. Then $M$ has an edge $\min(\hat{a}_{j'}, \hat{b}_{k'})$ where $\hat{a}_{j'}$ is large and $\hat{b}_{k'}$ is small and these two $\min$ edges enter the same comparator.

Proof. Let $\hat{M} = \text{split}(M)$ and let $c$ be the comparator $\hat{a}_j$ enters in $\hat{M}$. By Lemma 4.0.10, $\hat{M}$ is AMOP. By Lemmas 4.2.4 and 7.0.22 (d), $\text{imf}^{\hat{M}}$ is total. Let $\hat{b}_{k'}$ be the other edge entering $c$ in $\hat{M}$. Clearly, $\hat{b}_{k'}$ is a small input edge. Let $e$ and $e'$ be the two edges entering $c$ in $M$, such that $\hat{a}_j \in IC^M(e)$ and $\hat{b}_{k'} \in IC^M(e')$. Since $M$ is AMOP, the edges $e$ and $e'$ are uniquely defined.

Since $\hat{a}_j$ is small, in the bypass process $\hat{a}_j$ follows the $\min$ edges; therefore $e$ is a $\min$ edge. Similarly, $e'$ is a $\min$ edge. It remains to show that each of $IC^M(e)$ and $IC^M(e')$ has exactly two input edges. In $M$ both $\hat{a}_j$ and $\hat{b}_{k'}$ enter a comparator whose second incoming edge is a large input edge; hence the situation is symmetric w.r.t. $\hat{a}_j$ and $\hat{b}_{k'}$. Due to this symmetry it suffices to consider only $IC^M(e)$.

First we show that $\hat{a}_j$ is the only small input edge in $IC^M(e)$. Assume for a contradiction, that $d$ is another small input edge in $IC^M(e)$ and let the width of $M$ be $4n$. Recall that $\text{split}_D(M)$ is a subnetwork of $\hat{M}$ and by Lemma 4.2, $\text{split}_D(M)$ is a width $2n$ merging network. Since $c$ is of depth one in $\text{split}_D(M)$, $c \in IC^{\text{split}_D(M)}$; this implies that $n \in O^M(e)$; therefore $n \in O^M(e)$ and $n \in O^M(d)$. Since $M$ is AMOP, any path $p$ from $d$ to $\hat{o}_n$ passes through $e$. Since $e$ has been reduced in $\hat{M}$ into the input edge $\hat{a}_j$, the path $p$ from $d$ to $\hat{o}_n$ is disconnected in $\hat{M}$. This contradicts the fact that $\hat{M}$ is a half-merging network.

Let $\hat{a}_p$ and $\hat{b}_q$ be two input edges in $IC^M(e)$ which enter the same comparator and let $g = \max(\hat{a}_p, \hat{b}_q)$. Since $\hat{b}_q$ is large, $p + q \geq n$. By Lemma 8.0.25 (c), $\min(O(g)) = p + q + 1 > n$ which implies $g \notin IC^M(e)$ and therefore $\min(\hat{a}_p, \hat{b}_q) \in IC^M(e)$.

Now assume, for a contradiction, that there are more then two input edges in $IC^M(e)$. In this case, $IC^M(e)$ has four input edges which comply with the conditions of Lemma 8.0.26. By the conclusions of this lemma, two of these edges are small. This contradicts the fact that $\hat{a}_j$ is the only small input edge in $IC^M(e)$.

Lemma 8.0.28. Let $M$ be a regular compact merging network whose width is at least four. Then $\text{imf}^M$ uniquely determines $\text{imf}^{\text{split}(M)}$.

Proof. Let $4n$ be the width of $M$. Let $\eta = \text{imf}^M$ and $\eta' = \text{imf}^{\text{split}(M)}$. It suffices to show that:

(a) $\eta'$ is total.
(b) For any $j$: $\eta'(j) < n$ iff $j < n$.
(c) For any $j$: $\eta'(j) \equiv \eta(j)(\text{mod } n)$.
By Lemmas 4.0.10 and 4.2.4, split($M$) is a disjoint sum of two regular AMOP merging networks. By Lemma 7.0.22 (d), $\eta'$ is a total function over $[0, 2n)$. This establishes statement (a). Statement (b) follows from the fact that these two networks are disjoint.

By symmetry, it suffices to prove statement (c) only for $j < n$. If $\eta(j) < n$ then $\hat{a}_j$ enters a comparator which is not mixed; therefore $\eta'(j) = \eta(j)$. Thus, it remains to considers $j$s which are members of $I = \{i | i < n \text{ and } \eta(i) \geq n\}$. Define the function $f : I \rightarrow \mathbb{Z}$ by $f(j) = \eta^{-1}(\eta'(j)) - n$. We first show that $f$ is a permutation of $I$. Clearly, $f$ is one-to-one. Let $j \in I$ and let $k = \eta(j)$. The edges $\hat{a}_j, \hat{b}_k$ satisfy the premise of Lemma 8.0.27 so let $\hat{a}_j'$ and $\hat{b}_k'$ be the two edges provided by this Lemma. By this lemma, $\eta'(j) = k'$ and $\eta^{-1}(k') = j' \geq n$; i.e. $f(j) = j' - n$. It remains to show that $f(j) = j' - n \in I$. By Lemma 7.0.22(d), $\eta$ is a congruent function, and so $j' \geq n$ and $\eta(j') = k' < n$ imply that $\eta(j' - n) = k' + n \geq n$. This and $j' - n < n$ imply that $j' - n \in I$.

If $f$ is not the identity function, then there exists a $j \in I$ such that $f(j) < j$. Let $k = \eta(j)$, $j' = \eta^{-1}(\eta'(j))$ and let $k' = \eta(j')$. Since $j'$ is large, the edges $\hat{a}_j, \hat{b}_k, \hat{a}_{j'}$ and $\hat{b}_{k'}$ satisfy the premise of Lemma 8.0.26 (b). The fact that $f(j) < j$ imply that $\eta^{-1}(\eta'(j)) - n < j$, and so $j' - j < n$; this contradicts Lemma 8.0.26 (b.1). Hence $f$ is the identity function. That is, for every $j \in I$, $j = f(j) = \eta^{-1}(\eta'(j)) - n$. This implies $\eta'(j) = \eta(j + n)$. Since $\eta$ is a congruent function, $\eta(j + n) \equiv \eta(j) \pmod{n}$. □
Chapter 9

Degenerate comparators for half merging.

This chapter investigates which comparators of a merging network are degenerate with respect to the half merging functionality.

Recall that $\zeta^{4n}$ is the unique halved bisorted, 0 − 1 vector of width 4n having exactly 2n zeroes. In cases the width of such a vector is clear from context, we omit the superscript and use the shortcut $\zeta$. Recall that a comparator is mixed if it is mixed under $\zeta$.

**Lemma 9.0.29.** Let $c$ be a comparator of a Batcher merging network $B$ of width 4n. Then $c$ is mixed iff $c$ is degenerate under the functionality of half-merging.

**Proof.** Consider the left to right implication. By Lemmas 4.2.2 and 3.0.3, $\zeta^{4n}$ agree on $c$ with any halved bisorted vector. This implies that $c$ is degenerate under the functionality of half-merging.

To prove the right to left direction we show that if $c$ is not mixed then $c$ is non-degenerate under the functionality of half-merging. This we show by induction on $n$. There are exactly two Batcher merging networks of width 4 and it is easy to verify that this implication holds for both of them. Let $n > 1$ and let $B = B(r, B_1, B_2, Z)$ as defined in Chapter 5 where $r$ is a Batcher cleaver, $B_1$ and $B_2$ are Batcher merging networks and $Z$ is the appropriate depth one network. Let $c$ be a non-mixed comparator of $B$.

First consider the case where $c$ belongs to $B_1$ or $B_2$, say $B_1$. Clearly, $r(\zeta^{4n}) = \langle \zeta^{2n}, \zeta^{2n} \rangle$; namely, when $B$ receives $\zeta^{4n}$, each of the networks $B_1$ and $B_2$ receives $\zeta^{2n}$. Since $c$ is not mixed in $B$ under $\zeta^{4n}$, $c$ is not mixed in $B_1$ under $\zeta^{2n}$. By the induction
hypothesis, $c$ is non-degenerate in $B_1$ for the functionality of half-merging; therefore there are two halved, bisorted input vectors, $v_1$ and $v_2$, of $B_1$ that establish that $c$ is non-degenerate in $B_1$ under the half-merging functionality. Clearly, for any input vector $v$ of $B$, $r(2v) = \langle v, v \rangle$; this implies that when $2v_i$ is applied to $B$ then $v_i$ is applied to $B_1$. Therefore, the pair $(2v_1, 2v_2)$ establishes that $c$ is non-degenerate in $M$ under the half-merging functionality.

Next consider the case where $c \in Z$. The network $Z$ is of depth one and $c$ is non-degenerate under the functionality of merging; hence, by Lemma 8.0.24, its outgoing edges are $\hat{o}_i$ and $\hat{o}_i+1$ for some $i$. Since $c$ is not mixed under $\zeta_{4n}$, we have $i \neq 2n - 1$. Let $v \in \P^\bs_{4n}$ be defined by $b_j = a_j + 1$ for all $j$. Let $v'$ be the permutation derived from $v$ by swapping the keys $i$ and $i+1$. Clearly, $v$ and $v'$ are halved, bisorted permutations. The two incoming cones of the edges entering $c$ are disjoint since one is a subgraph of $M_1$ and the other is a subgraph of $M_2$. Since both $v$ and $v'$ are valid vectors, $c$ receives the keys $i$ and $i+1$ under both input vectors but from different edges; that is, $v$ and $v'$ establish that $c$ is non-degenerate for the half-merging functionality.

**Lemma 9.0.30.** Let $M$ be a regular compact merging network whose width is greater than two. Then $M$ has a minor $M^*$ such that:

a) $\text{split}(M^*)$ is a non-degenerate AMOP half-merging network.

b) $IC^M \cong IC^{M^*}$.

c) $\text{imf}^{\text{split}(M^*)} = \text{imf}^{\text{split}(M)}$.

Note that $M^*$ is not required to be a merging network.

**Proof.** Let the width of $M$ be $4n$ and let $C$ be the set of the comparators of $M$ which are degenerate w.r.t. half-merging and are not mixed under $\zeta_{4n}$. Pick any halved bisorted permutation $v$ and let $M^* = \wp(N, C, v)$; namely, $M^*$ is the product of bypassing all comparators of $C$ in the manner charted by $v$. Since all halved bisorted permutations agree on all comparators of $C$, by Lemma 4.1.1, $M^*$ is independent of which input vector $v$ is chosen. By Lemma 4.1.1, $M^*$ is a half-merging network and all its degenerate comparators (w.r.t. half merging) are mixed under $\zeta_{4n}$, establishing statement (a).

By Lemma 7.0.22(d), $\text{imf}^M$ is a congruent function. By Lemma 6.0.19, there is a Batcher merging network $B$ such that $\text{imf}^B = \text{imf}^M$. By Lemma 7.0.23, $IC^M \cong IC^B$. By Lemma 9.0.29, all the comparators of $B$ which are degenerate w.r.t. half-merging are mixed. This implies that no comparator of $C$ is in $IC^M$, which implies statement (b).

To prove statement (c), let $\mu = \text{imf}^M, \bar{\mu} = \text{imf}^{\text{split}(M)}$ and $\bar{\mu}' = \text{imf}^{\text{split}(M^*)}$. By statement (b), $\text{imf}^{M^*} = \mu$. We have to show that $\bar{\mu}(j) = \bar{\mu}'(j)$ for any $j$. By symmetry, we may assume $j < n$ and let $k = \mu(j)$; that is, the two edges $\hat{a}_j$ and $\hat{b}_k$...
are matched in some comparator $c$ of $M$. Assume first that $k < n$. In this case $c$ is non-degenerate w.r.t. half merging and so $\bar{\mu}(j) = \bar{\mu}'(j) = k$.

Next assume $k \geq n$. In this case $\hat{a}_j$ and $\hat{b}_k$ comply with the premise of Lemma 8.0.27. Let $\hat{a}_{j'}$ and $\hat{b}_{k'}$ be the two edges provided by this lemma, let $\bar{c}$ be the comparator where the edges $\hat{a}_{j'}$ and $\hat{b}_{k'}$ are matched in $M$ and let $\bar{c}$ be the comparator that the edges $\hat{a}_{j'}$ and $\hat{b}_{k'}$ enter. It is not hard to see that $\bar{c}$ is non-degenerate w.r.t. half merging; therefore, $\bar{c} \in M^*$. Since $c$ and $c'$ are not in $\text{split}(M^*)$ and are not in $\text{split}(M)$ and since $\bar{c}$ is in $\text{split}(M^*)$ and in $\text{split}(M)$, the two edges $\hat{a}_{j}$ and $\hat{b}_{k'}$ enter the same comparator both in $\text{split}(M^*)$ and in $\text{split}(M)$. This implies that $\bar{\mu}(j) = \bar{\mu}'(j) = k'$, establishing statement (c). \qed
Chapter 10

Characterization of the Batcher merging networks

This chapter combines the results of previous ones into our main result. To this end, the following theorem is the main result of this work:

**Theorem 10.0.31.** A network is a regular and compact merging network iff it is a Batcher merging network.

**Proof.** The right to left implication is the easy part of this theorem. By their definition, Batcher merging networks are regular. By Lemma 5.0.4 and 5.0.2, all Batcher merging networks are non-degenerate and AMOP.

We prove the left to right implication by induction on the width of the network. Let $M$ be a compact merging network. The case where $M$ is of width 2 is trivial so let the width of $M$ be $4n$. By Lemma 7.0.22(d), $\text{imf}^M$ is a congruent function and so by Lemma 6.0.19, there exists a Batcher merging network $B$ of the same width and i.m.f. as $M$. By Lemma 4.3.1, it suffices to show that some minor of $M$ is isomorphic to $B$.

Let $M^*$ be the minor of $M$ provided by Lemma 9.0.30. Without loss of generality, we may assume that $M^*$ is a minor of $M$ via an embedding $\sigma$ which is the identity function over the vertices of $M^*$; this implies that the vertices of $M^*$ are vertices of $M$. By Lemma 9.0.30, split($M^*$) is a non-degenerate AMOP half-merging network. By Lemmas 4.2.1, split($M^*$) is a disjoint sum of two networks, one is a compact merging network we call $\text{split}_D(M^*)$ and the normalized variant of the other is a compact merging network called $\text{split}_U(M^*)$. By Lemma 9.0.29, all comparators of $B$ which are degenerate w.r.t. half-merging are mixed under $\zeta$, and so split($B$) is a non-degenerate AMOP half-merging network. By Lemmas 4.2.1 and 4.0.10, split($B$) is a
disjoint sum of two networks, one is a compact merging network and the normalized variant of the other is a compact merging network. We name these compact merging networks \( \text{split}_D(B) \) and \( \text{split}_U(B) \), respectively. By the induction hypothesis, each of the four networks, \( \text{split}_D(M^*) \), \( \text{split}_U(M^*) \), \( \text{split}_D(B) \) and \( \text{split}_U(B) \), is isomorphic to some Batcher merging network of width \( 2n \).

By our construction, \( \text{imf}^B = \text{imf}^M \). By Lemma 8.0.28, \( \text{imf}^{\text{split}(B)} = \text{imf}^{\text{split}(M)} \).

By Lemmas 9.0.30 (c), \( \text{imf}^{\text{split}(M)} = \text{imf}^{\text{split}(M^*)} = \text{imf}^{\text{split}(B)} \); therefore, \( \text{imf}^{\text{split}(M^*)} = \text{imf}^{\text{split}_D(B)} \) and \( \text{imf}^{\text{split}_U(M^*)} = \text{imf}^{\text{split}_U(B)} \). By the induction hypothesis and Lemma 5.0.9, \( \text{split}_D(B) \cong \text{split}_D(M^*) \) and \( \text{split}_U(B) \cong \text{split}_U(M^*) \). This implies that \( \text{split}(B) \cong \text{split}(M^*) \).

Recall that it remains to show that \( M^* \cong B \). By Lemma 2.0.1, it suffices to show that \( M^* \) and \( B \) have a common syndrome\(^1\); i.e. \( \text{synd}(M^*, v) = \text{synd}(B, v) \), for some non-repeating input vector \( v \).

Pick a halved input vector \( v \in \mathbb{F}_{4n}^I \). We prove that \( \text{synd}(M^*, v) = \text{synd}(B, v) \), i.e. that for any key \( k \) in \( v \), \( \text{act}(M^*, v, k) = \text{act}(B, v, k) \). Clearly, the first, second and last elements of those acts \( (k, l_1 \) and \( l_2) \) are identical in \( \text{act}(M^*, v, k) \) and \( \text{act}(B, v, k) \). The last identity is due to the fact that \( v \) is valid.

For a network \( N \in \{M^*, B, \text{split}(M^*), \text{split}(B)\} \), let \( \vec{s}^N \) be the sequence of keys that our key \( k \) encounters on its way through \( N \) under the input vector \( v \) and let \( \vec{s}_1^N \) be the initial segment of \( \vec{s}^N \) composed of the keys \( k \) encountered inside \( IC^N \). It remains to show that \( \vec{s}^{M^*} = \vec{s}^B \). By symmetry, we may assume \( k < 2n \). It is not hard to see that \( \vec{s}^{\text{split}(M^*)} \) and \( \vec{s}^{\text{split}(B)} \) are derived from the sequences \( \vec{s}^{M^*} \) and \( \vec{s}^B \), respectively, by purging all keys greater then \( 2n - 1 \); furthermore, all of these purgings concern comparators which are mixed in their networks (which are either \( M^* \) or \( B \)).

Let \( k > 2n - 1 \) appear in \( \vec{s}^{M^*} \) and let this “encounter” occur in a comparator \( c \) of \( M^* \). (Clearly, \( c \) is mixed.) Since \( v \) is valid for half-merging, \( \hat{o}_k \) and \( \hat{\alpha}_k \) are members of \( OC^{M^*}(c) \). Since \( M^* \) is a minor of \( M \), \( \hat{o}_k \) and \( \hat{\alpha}_k \) are members of \( OC^M(c) \). By Lemma 8.0.24, \( c \) is a member of \( IC^M \). Since \( IC^M = IC^{M^*} \), \( c \) is a member of \( IC^{M^*} \). Therefore, all the keys purged from \( \vec{s}^{M^*} \) reside in the sequence \( \vec{s}_1^{M^*} \). Similar and simpler arguments imply that all the keys purged from \( \vec{s}^B \) reside in the sequence \( \vec{s}_1^B \).

By Lemma 7.0.23, \( IC^B \cong IC^M \cong IC^{M^*} \); hence \( \vec{s}_1^{M^*} = \vec{s}_1^B \). This implies that in the above purgings (both in \( \vec{s}^{M^*} \) and in \( \vec{s}^B \)) the same keys have been removed from the same position. Since \( \text{split}(B) \cong \text{split}(M^*) \), the derived sequences, \( \vec{s}^{\text{split}(M^*)} \) and \( \vec{s}^{\text{split}(B)} \), are equal, hence, so must be the original sequences; that is \( \vec{s}^{M^*} = \vec{s}^B \).

\(^1\)Recall that for a network \( N \) and a non-repeating input vector \( v \), \( \text{synd}(N, v) \) is the set \( \{\text{act}(N, v, k)\} \); The key \( k \) appears in \( v \) and that \( \text{act}(k, v, N) \), is the fourtuple \( (k, l_1, \vec{s}, l_2) \), where \( l_1 \) and \( l_2 \) stands for the labels of the input and output edges traversed by \( k \) and \( \vec{s} \) is the sequence of the keys that \( k \) encountered on its way through \( N \).
Chapter 11

oblivious algorithms

In the next chapter we survey a variety of published merging techniques and merging networks. Usually, networks of comparators are presented indirectly via an oblivious algorithm (a.k.a. non-adaptive algorithm) which gives rise to the desired network. This section discusses the concept of oblivious algorithm and how it relates to networks of comparators.

We prefer to start by presenting a new model of oblivious computation, which we believe to be more natural than the accepted model ([16], [10, pp 220] and [13, pp 623]). Our model is more powerful than the accepted one in two aspects; it speeds-up the computation of several functions w.r.t. the accepted model and it allows the computation of new functions which are not computable in the accepted model. Nevertheless, it still obeys the classical 0-1 principle and its known generalizations including several new ones presented in Chapter 13.

Our model of computation, lets call it the min/max oblivious model, has only one data type – a key. These keys are stored in variables and there are only two instructions: “$z \leftarrow \min(x, y)$” and “$z \leftarrow \max(x, y)$”, where $x, y$ and $z$ stand for arbitrary (not-necessarily distinct) variables. Note that this model does not allow for any control operations (such as branching or looping) and especially no conditional instructions of the form “if $(x < y)$ then ...”; therefore, an algorithm in this model, called a min/max oblivious algorithm, is a straight-line code of the above statements.

Clearly, an algorithm is aimed to solve a given problem. To this end, some of the variables are input variables which contain the input; another set of variables, output variables contains the output of the algorithm. All other variables are intermediate variables. The names of the input/output variables reflects the functionality of the algorithm. Consider, for example, an algorithm to merge two sorted sequences, each of length 4, into a single sequence of length 8; in this case, the input variables could be $A_0, A_1, A_2, A_3, B_0, B_1, B_2$ and $B_3$ and the output variables could be $O_0, O_1 \ldots O_7$. 
with the obvious semantics. To avoid the problem of uninitialized variables we add a restriction to this model that a variable must be written before it is read. Clearly, the input variables are considered to be written at the beginning of the algorithm.

Note that the min/max model differs from the accepted model in two significant manners. Firstly, it uses the weak instructions, “min” and “max”, while the accepted model combines these instructions into a single one – the sorting of a pair of keys. Secondly, and more significant, in the min/max model a computed value can be used several times as opposed to once in the accepted model.

To simplify the comparative study of algorithms it is desirable that algorithms will have as little insignificant details as possible. Ideally, two algorithm that essentially “preform the same computation” will be identical. One source of insignificant details is the reuse of variables; namely, using a variable to store several values during the course of the algorithm. To eliminate this source we prefer write-once algorithms; that is, algorithms in which a variable is written only once. Clearly, any min/max oblivious algorithm can be transformed into a write once algorithm.

Any min/max oblivious algorithm can be translated into a min/max network. Such a network is a directed acyclic graph, composed of three types of vertices; input vertices with in-degree 0 and arbitrary out-degree, output vertices with in-degree 1 and out-degree 0 and intermediate vertices with in-degree 2 and arbitrary out-degree. Actually, intermediate vertices are either min elements or max elements. Such an element computes the appropriate value and transmits it on all it’s outgoing edges. The input and output vertices are associated with labels; this association specifies how to apply the input to the network and how to collect the output of the network. There are two differences between min/max networks and networks of comparators. These differences are parallel to the differences between the min/max oblivious model and the accepted one. Namely, the use of min/max elements instead of comparators and the unrestricted fanout.

Any min/max oblivious algorithm can be translated into a min/max network and this translation is immediate when the algorithm is write-once. In this case input vertices correspond to input variables. All other vertices correspond to intermediate vertices that compute the appropriate value. The output variables correspond, not only to intermediate vertices, but also to output vertices. Edges are used to propagate the values within the network.

Translation from a min/max network into a write-once algorithm is also straightforward but unfortunately is not unique; the resulting algorithms may differ in non-significant details of two types; the names of intermediate variables and the relative

---

1A network may have an edge going from an input vertex to an output vertex. In order to translate such an edge we add a special instruction of the form “y \leftarrow x” where x is an input variable and y is an output one.
order of independent\textsuperscript{2} instructions.

So far we described oblivious algorithm as a serial program. However, the big advantage of oblivious algorithms is that they can be computed in a parallel manner. In this parallel model time is divided into steps and a program is divided into a sequence of sections. In each time stamp all instructions of the current section are performed simultaneously; therefore, all instructions of a section are required to be independent. The run time of a parallel algorithm is the number of those steps. Note that the instructions of the same serial oblivious algorithm can be grouped into a sequence of sections in many different ways, all having the same functionality; this holds even if we insist that the resulting parallel oblivious algorithm is of minimal time.

A partition of a serial algorithm into sections induces a partition of the networks vertices into stages s.t. every edge goes from one stage into a later stage. In addition, all input vertices are in the first stage and all output vertices are in the last stage. Each stage corresponds to a different step except of the first stage (containing all input vertices) and the last stage (containing all output vertices). Each partition of a min/max network into stages induces a partition of the serial oblivious algorithm into steps. In both cases, there is no canonical partition of a network into stages or of an oblivious algorithm into steps even if we require minimal number of stages or steps.

The two operations, “min” and “max”, can be combined into a single operation. The resulting model of computation is very similar to our min/max model except that only one operation exists. Namely, “(z, w) ⇐ sort(x, y)” where x, y, z, w stand for arbitrary variables with the only restriction is that z and w are distinct variables. Clearly, this change is superficial and effects neither the computability of the model nor the run time of a serial or parallel algorithm. Networks which represents such algorithms are called unrestricted network of comparators. Clearly, the internal vertices of such a network are comparators; however, there is no restriction on the fanout of these comparators; namely, a comparator has two out-ports corresponding to the minimal and maximal keys. An arbitrary number (including zero) of outgoing edges emerge from any out-port. Such a network was presented by Knuth [10, pp 233]. As said, the difference between unrestricted networks of comparators and min/max networks is superficial and we believe the latter to be a more natural model.

The restricted networks of comparators, which are the main subject of this work and outside this section are simply referred to as networks, are defined in Chapter 2. As said, they are subclass of the unrestricted networks of comparators in which the fanout of inputs and the fanout of out-ports of intermediate vertices equals one. We refer to the model of computation which corresponds to restricted networks of

\textsuperscript{2}Two instructions are dependent if they write to the same variable or if one writes a variable which the other reads.
comparators as the read-once model. Its instructions are of the form \(\langle z, w \rangle \leftarrow \text{sort}(x, y)\) where \(x, y, z, w\) stand for arbitrary variables with the restriction that \(z\) and \(w\) are distinct variables. Furthermore, every computed or input value is used exactly once; therefore, the above \(x\) and \(y\) must be distinct variables.

As said, it is sometimes desirable that the algorithm is a write-once algorithm. However, the common models of oblivious computation took the opposite approach. In these models, let's call them the in-place models, there is only one set of variables. The same variables are used as input, output and as intermediate variables. (This implies that a variable should have two names: one as an input variable and another as an output variable.) The only instruction of the in-place model is the in-place sorting instruction; namely, \(\langle x, y \rangle \leftarrow \text{sort}(x, y)\) where \(x\) and \(y\) are distinct variables. This implies that any computed value is used exactly once and keys are neither duplicated nor lost. This model best corresponds to Knuth’s non-standard diagram [10, pp 237]. This diagram consists of horizontal lines, representing the variables, and vertical arrows representing the sorting operations, connecting the two variables in question with the arrowhead pointing to the variable where the maximal key is assigned. Clearly any Knuth’s diagram can be translated into a unique restricted networks of comparators; however, translation from a restricted network of comparators into Knuth’s diagram is not unique. As mentioned by Bilardi [6], every partition of the network into edge-disjoint paths, combined with an ordering of these paths, produces a distinct Knuth’s diagram.

The accepted model of oblivious computation [16],[13], let's call it the ordered in-place model is a submodel of the previous one in which the set of variables is ordered. An instruction \(\langle x, y \rangle \leftarrow \text{sort}(x, y)\) is allowed only if \(x < y\). This model corresponds to the well-known variant of Knuth’s diagram [10, pp 222]. In this diagram, let's call it Knuth’s standard diagram, the layout of the horizontal lines follows the order of variables; the arrowheads are redundant in this diagram; therefore, sort operations are represented as vertical lines.

As before, any standard diagram can be translated into a unique restricted networks of comparators. Translation in the other direction (from a restricted network of comparators into Knuth’s standard diagram) is meaningful only when there is a natural order on the output vertices of the restricted network; such an order exists when the restricted network produces a single sequence rather than, for example, two sequences. Clearly, we would like to translate the network into a Knuth’s standard diagram in which the layout of the horizontal lines follows the order of output vertices of the network. Following Knuth [10, pp 239] and Bilardi [6], such a translation can be performed as follows. Let \(N\) be such a restricted network and let \(v\) be a permutation input vector which \(N\) sorts (according to the order of the output vertices mentioned above). Then the path traversed by key \(i\) is associated with the \((i + 1)\)’th horizontal line. This induces the vertical lines representing the comparators. If no such vector
exists, the network cannot be translated into a Knuth’s standard diagram. A network usually has many such input permutations and therefore many translations into a Knuth’s standard diagram.

This variety of representations have lead to confusion. Bilardi [6], for example, shown that two distinct Knuth’s standard diagrams, one of the Balanced network [8] and the other of Batcher’s bitonic sorter [1] actually represent networks which are isomorphic up to a relabelling of their inputs.

This variety of representations is especially noticeable in the context of merging networks. Let $N$ be a merging network of width $2n$. Clearly, this network sorts any bisorted permutation and each of them leads to a different Knuth’s standard diagram. Namely, for any partition of the horizontal lines into two sets of size $n$ there is a Knuth standard diagram of $N$ in which the sequence $\vec{a}$ enters the horizontal lines via the first set, the sequence $\vec{b}$ enters via the second set and within each set the corresponding sequence enters the diagram in the natural order.

As said, the min/max model is more powerful than the accepted model in two aspects. Firstly, as observed by Knuth [10, pp 241], there are some functions which are computable in the former and are not computable in the latter. This is due to the following property of such a function $f : \mathcal{K}^n \rightarrow \mathcal{K}$, computable in the accepted model. The function $f$ is either a projection or it has two distinct arguments $x_i$ and $x_j$ s.t. $f(x_0, \ldots, x_{n-1})$ is invariant under a transposition of the values of $x_i$ and $x_j$, and this for any $x_0, \ldots, x_{n-1} \in \mathcal{K}$. The min/max model allow functions which violate this property, for example

$$f(x_0, x_1, x_2, x_3) = \max(\min(x_0, x_1), \min(x_1, x_2), \min(x_2, x_3)).$$

The second aspect in which the min/max model is more powerful than the accepted one is the computation time of certain functions. Assume we have several functions, all from $\mathcal{K}^n$ to $\mathcal{K}$ and each of them can be computed in time $t$. In the min/max model all of them can be computed simultaneously in the same time $t$. In the accepted model it is not assured that all these functions can be computed together (ignoring the computation time) or if all of them can be computed in time $t$.

A concrete example, already observed by Batcher and presented in [10, pp 233], has to do with insertion. An inserter of width $n$ merges a sorted sequence of length $n-1$ and a single key into a sorted sequence of length $n$. By a straightforward reachability argument, the depth of a network of comparators having this functionality is at least $\lceil \log(n) \rceil$. Now consider the min/max model. Any output key of an inserter depends on either two or three input keys. Not surprisingly, any output key can be computed from the relevant input keys in the min/max model; this in a constant run time. Hence, all output keys can be computed simultaneously in a constant time.
Chapter 12

Variety of Merging Techniques

Many ingenious algorithms for oblivious merging have been invented. However, it is a common phenomena that radically different algorithms produce identical networks. For many applications the networks themselves are of importance and their properties are investigated. In order to prevent a duplication of effort it is desired to tell when different algorithms produce identical networks. This chapter shows that all published merging networks are Batcher merging networks. As said, this diagnostics does not require a complete understanding of the technique in question; in fact, a very superficial understanding of the algorithm suffices to establish that a merging technique produces generalized Batcher merging network.

In this chapter we use Theorem 10.0.31 to prove such claims. We survey several published techniques and demonstrate that this tool does not require a full understanding of the oblivious algorithm in question but only a few details that the authors explicitly state. In order to use Theorem 10.0.31 we need to establish that a network is both AMOP and non-degenerate. It is usually easy to show that a network is AMOP but it is tedious to show it has no degenerate comparators; this is not critical since if a network would have a degenerate comparators they can be bypassed, by Lemma 4.1.2, without disturbing the AMOP property or the merging functionality of the network; therefore, we do not concern ourselves with showing them to be non-degenerate.

A few words about non-Batcher merging networks are in order. Prior to this work there was no technique to produce a non-Batcher merging network which is non-degenerate and of minimal depth; moreover, it was not known if such a network exists. Such a technique is presented in Chapter 16.

1Most published oblivious algorithms are only partly-specified; for example, they say "sort this set of keys" without specifying how to do so; when such missing details are filled in (in a natural manner) the generated networks are Batcher merging network.
12.1 pre-recursive and post-recursive

As explained above, our main goal in the following discussion is to show that certain oblivious algorithms produce AMOP networks; therefore, during the following discussion, we ignore the functionality of the networks, their input and output labels and the \textbf{min/max} type of edges.

Many oblivious recursive algorithms (not necessarily for merging) fall into one of the following two classes. In the first class, the given input is partitioned, in a data independent manner, into several smaller problems; each of these problems is solved recursively and, finally, some post-processing combines the solutions of the smaller problems into a solution of the original one. We call such algorithms \textit{pre-recursive}. In the second class, some pre-processing partitions the given problem into several smaller problems which are then solved recursively and those solutions combined into a solution of the original problem without any post-processing. We call algorithms of the second class \textit{post-recursive}. An oblivious algorithm of the above types can also be a \textit{multi-choice} algorithm. Such an algorithm can choose, in a data independent manner, one of several modes of operations. For example, the generalized Batcher merging technique is a multi-choice pre-recursive algorithm.

A restricted form of a multi-choice pre-recursive oblivious algorithms produces networks of the following type. A family (i.e. a set) of networks $\mathcal{R}$ is \textit{pre-recursive} if every member of $\mathcal{R}$ is composed of a \textit{pre-processing network $A$} followed by a \textit{post-processing network $B$} such that:

1. The network $A$ is a disjoint sum of two (non empty) networks $N_1, N_2$ s.t. each $N_i$ is either a single edge or a member of $\mathcal{R}$.

2. The depth of $B$ is at most one.

3. Out of any two edges entering the same comparator of $B$ one arrives from $N_1$ and the other from $N_2$.

Note that in this definition, the functionality of the networks is completely irrelevant and therefore, so are the input/output labels and the \textbf{min/max} type of the edges. Three famous pre-recursive families are the set of the strictly-parallel merging networks (Batcher’s odd/even merging networks [1]), the set of the strictly-cross merging networks [13] and, as said, the set of all the Batcher merging networks.

A \textit{post-recursive} family of networks is defined analogously. In this case the pre-processing network is called $B$ and the post-processing network is called $A$. The rest of the definition stays as it is except that condition (3) is modified to:

$3'$. Out of any two edges exiting the same comparator of $B$ one enters $N_1$ and the other enters $N_2$. 
The following lemma is straightforward:

**Lemma 12.1.1.** All members of a pre-recursive family or a post-recursive family are AMOP.

Lemma 12.1.1 implies, for example, that all Batcher merging networks are AMOP. Knuth [10, pp 232] presented the following oblivious algorithm for sorting bitonic sequences whose length is a power of two. Let \( \vec{z} \) be such a bitonic sequence:

1. Sort each of the sequences even(\( \vec{z} \)) and odd(\( \vec{z} \)) recursively, producing the sorted sequences \( \vec{x}_1 \) and \( \vec{x}_2 \).

2. Merge \( \vec{x}_1 \) and \( \vec{x}_2 \) into a sorted sequence. As Knuth has shown, this can be done by a depth one network.

Naturally, the resulting family of networks is pre-recursive. Clearly, any bitonic sorter can be transformed into a merging network via relabelling; therefore, assuming the resulting networks are non-degenerate, these merging networks are Batcher merging networks.

There are \( 4n \) ways\(^2 \) to rearrange a bisorted sequence of width \( 2n \) into a bitonic one. Each of them provides a relabelling of a bitonic sorter into a merging network. We show that, under all of these natural relabellings, the resulting merging networks are all isomorphic to one another and are all isomorphic to the strictly-cross network. This is done without assuming that the resulting merging networks are non-degenerate, as follows. A simple induction shows that in Knuth’s bitonic sorter of width \( 2n = 2^k \) the two input edges \( \hat{x}_i \) and \( \hat{x}_{i+n} \) are matched, for every \( i < n \). Let \( M \) be a merging network derived from the Knuth bitonic sorter by one of the above relabellings. It is easy to verify that \( \text{imf}^M \) is the order reversing permutation. By Theorem 10.0.31, some minor of \( M \) (which bypassed all degenerate comparators w.r.t. merging) is a Batcher merging network and by Lemma 5.0.9 this minor is the strictly-cross network. Since the strictly-cross network and Knuth’s bitonic sorter have the same number of comparators, the above minor is \( M \) itself.

All techniques considered so far were pre-recursive. We now consider a post-recursive merging technique. Batcher [1] introduced the concept of bitonic sequences and the idea of recursively sorting bitonic sequences, as follows. Let \( \vec{z} \) be a a bitonic sequence of width \( 2n = 2^k \).

1. Compute the two sequences \( \vec{x} = \min(a[0, n], a[n, 2n]) \) and \( \vec{y} = \max(a[0, n], a[n, 2n]) \).

\((\text{It holds that } \vec{x} \ll \vec{y} \text{ and that both } \vec{x} \text{ and } \vec{y} \text{ are bitonic sequences.})\)

2. Sort \( \vec{x} \) and \( \vec{y} \) recursively.

\(^2\text{There are two ways to transform a bisorted vector into an ascending-descending one and all the } 2n \text{ rotations of the latter are bitonic.}\)
Clearly, the resulting family of networks is post-recursive and by Lemma 12.1.1, all those networks are AMOP. Our discussion about Knuth’s bitonic sorter implies that all the merging networks derived from Batcher’s bitonic sorter are strictly-cross networks. A stronger result, concerning bitonic sorters, is presented in Chapter 14; namely, there is exactly one minimal depth bitonic sorter of width $2^k$.

As said, an advantage of our characterization tool is that it requires only a superficial understanding of a given merging technique in order to determine that it produces Batcher merging networks. Consider the Balanced networks introduced by Rudolph, Dowd, Perl and Sacks [8]; the authors state the following properties of their merging algorithm.

1. By a pre-processing network of depth one, the keys are divided into two sets $X$ and $Y$. Henceforth, keys from one set are not compared with keys from the other set.

2. The algorithm is preformed recursively for $X$ and for $Y$.

Furthermore, each of the comparators of the pre-processing network sends one key to $X$ and the other key to $Y$.

These properties alone are sufficient\(^3\) to determine that this algorithm is post-recursive, and that the family of the networks produced by it is post-recursive. The authors state that the Balanced networks are merging networks\(^4\) and by Lemma 12.1.1 and Theorem 10.0.31 those networks are Batcher merging networks. A stronger result, regarding the identity of the Balanced network is due to Bilardi [6]. Using the full description of the algorithm, he shows that the Balanced network, under a certain relabelling, is Batcher’s bitonic sorter.

Becker, Nassimi and Perl\([4]\) introduced a more general family of merging networks in \([7]\). Their multi-choice oblivious merging algorithm is based on the concept of $g$-chain (defined there). By definition, it is pre-recursive and produce a pre-recursive family of networks; hence, their networks are AMOP and therefore Batcher merging networks. A more careful examination reveals that their algorithm generates the entire family of Batcher merging networks.

Bender and Williamson \([5]\) defined a certain family of recursive networks; let us call it the BW-family. These networks are merging networks in a more general sense; that is, the two input sequences are not necessarily of the same length. This family is almost pre-recursive; the main violation is that the post-processing network is not

\(^{3}\)There are many other details which are not necessary for this classification such as: the two sets $X$ and $Y$ are of the same width, every key in one set is greater or equal to every key in the other set, etc.

\(^{4}\)In most of their work they view the Balanced network’s input as a single sequence; therefore, it is a merging network only under a relabelling.
restricted to be of depth one; however, they also considered a subset of the BW-family, let us call it the BW'-family, which is constructed of members in which

- The length of the input sequences is equal and is a power of two.
- The depth of the network is minimal.\(^5\)

They establish [5, Corollary 3.1.1], that the BW'-family is pre-recursive. Therefore the resulting networks are AMOP and therefore Batcher merging networks\(^6\). A careful examination reveals that the BW'-family contains all the Batcher merging networks.

As said, the same family of networks can be constructed in many ways. Knowing that such families are identical is important. it enables us to investigate a single family rather than an assortment of families. Here is a case in point; Bender and Williamson [5] left the following open question: Does a network of the BW'-family have redundant comparators? This question has been answered negatively in a strong way by Lemma 4.3.1. Namely, a Batcher merging network does not have a comparator, or a non-empty set of comparators, that can be bypassed without disturbing the merging functionality of the network.

Most of Bender and Williamson’s work concerns merging networks having the following three properties: a recursive structure of a specific form, a width which is a power of two and a minimal depth. A natural question is whether the first property, of recursive structure, is redundant. That is, does any non-degenerate merging network of minimal depth and whose width is a power of two must have a recursive structure, either of their form or of some other form. This question is answered negatively in Chapter 16. Namely, there are non-degenerate merging networks of minimal depth and whose width is a power of two which are not Batcher merging networks and do not possess any recursive structure.

### 12.2 Powerful building blocks

Many descriptions of oblivious algorithms do not go into details as to the level of a single comparator and instead use powerful building blocks (such as “sorting a sequence” or “merging two sequences”), without specifying how to perform these tasks. Such a description naturally lead to a network, not of comparators, but of elements having more powerful functionality. We call such a network a skeleton. Formally, a skeleton is a directed acyclic graph (DAG) having 3 types of vertices:

---

\(^5\)This restriction apply only to the entire network and not to its internal merging subnetworks.

\(^6\)In their work, they implicitly allow degenerate comparators. Assuming those degenerate comparators are bypassed and eliminated, then by Lemma 4.1.2, the resulting merging networks are Batcher merging networks.
• Input vertices with in-degree 0 and out-degree 1.
• Output vertices with in-degree 1 and out-degree 0.
• Internal vertices which represent building blocks. The in-degree and out-degree of each internal vertex are equal.

Note that although a vertex of a skeleton may process many keys, each of its edges carry a single key. A skeleton can be expanded to a network of comparators by replacing each internal vertex with a network of comparators of the appropriate functionality and width. We say that $N$ is an AMOP expansion of a skeleton $S$ if $N$ is derived from $S$ by replacing each of the internal vertices with AMOP network. The following Lemma is straightforward

**Lemma 12.2.1.** An AMOP expansion of an AMOP skeleton is AMOP.

### 12.2.1 Matrix technique

A very elegant oblivious algorithm for bitonic sorting was invented by Nakatani, Huang, Arden, and Tripathi [12] and works as follows. Let $\vec{z}$ be a bitonic sequence of width $j \cdot k$.

**Stage (1):** Arrange $\vec{z}$ in a $j \times k$ matrix in a row major fashion. (Each column of this matrix is clearly bitonic.)

**Stage (2):** Sort each column by a bitonic sorter. (Now, each row is a bitonic sequence and for any two consecutive rows $X_i$ and $X_{i+1}$, $X_i \ll X_{i+1}$).

**Stage (3):** Sort each row by a bitonic sorter. (Now the matrix is sorted in a row major fashion.)

Naturally, this algorithm leads to a layered skeleton $S$ having four layers in which:

• The first and last layers contains the $j \cdot k$ input and $j \cdot k$ output vertices, respectively.
• The second layer has $k$ bitonic sorters, each of width $j$.
• The third layer has $j$ bitonic sorters, each of width $k$.
• The second and third layers are connected as a complete bipartite graph.

This skeleton is AMOP. Assume that each internal vertex is replaced by an AMOP network of the appropriate functionality. By Lemma 12.2.1, any AMOP expansion of this skeleton is AMOP and therefore, this network is a Batcher merging network. This technique is discussed more thoroughly in Chapter 16.
12.2.2 Modulo Merge

Batcher and Liszka [3] presented an oblivious merging algorithm based on different ideas but actually similar to the Matrix technique of Nakatni et al. Their algorithm first rearranges the bisorted input $⟨\vec{a}, \vec{b}⟩$ into the bitonic sequence $\vec{a}$ followed by rev$(\vec{b})$. Then their algorithm applies the three stages of Nakatani’s matrix technique with the following change. Under such input and prior to stage (2), each column is not only bitonic but also essentially bisorted; therefore, in Stage (2) their algorithm uses merging networks instead of bitonic sorters; that is, they use the same skeleton except that vertices of the second layer are mergers rather than bitonic sorters. Useful and interesting properties of some networks produced by this method are described in Chapter 16.
Chapter 13

Conclusive sets

This chapter studies generalizations of the famous 0-1 principle. For a given functionality it seeks a small set \( V \) of valid (input) vectors s.t. any network, has this functionality w.r.t. \( V \) iff it has this functionality w.r.t. all its valid input vectors. Formally, let \( F \) be a functionality of networks. As defined in Chapter 2, \( F \) specifies a set \( V \) of valid vectors and a desired condition that the resulting output vector should satisfy whenever the input vector is a member of \( V \). A set \( V' \) is conclusive for \( F \) if the following conditions hold:

1) \( V' \subset V \).

2) For every network \( N \), if \( T^N(v') \) satisfies the desired condition for every \( v' \in V' \) then \( N \) has the \( F \) functionality.

As demonstrated by the 0 – 1 principle, small and simple conclusive sets simplify the design and study of useful networks. Moreover, such sets can be used to shorten an exhaustive procedure to verify that a given network has a certain functionality. Although all the lemmas in this chapter refer to and proven for networks of comparators, they hold also for min/max networks. Among other results, this chapter presents a conclusive set of \( n+1 \) vectors for the functionality of merging two sequences of width \( n \) each. It also shows that the cardinality of a conclusive set for this functionality, restricted only to 0 – 1 vectors, is at least \((N + 1)^2 - 2\).

All previous works in this field consider only conclusive sets restricted to 0 – 1 vectors in contrast to our work. Previous generalizations of the 0 – 1 principle considered only the functionality of merging (for example, see [11]). In contrast, our generalizations concern a large variety of functionalities. Rajasekaran and Sen [14] extended the 0 – 1 principle for “almost sorting networks” – networks which sort a large fraction of their 0 – 1 input vectors. Rice [15] have proved a weaker version of Lemma 13.1.10 and his contribution is discussed there.
13.1 conclusiveness by monotonic functions

The following lemma, attributed by Knuth [10, page 224] to W.G. Bouricius, is stated in Chapter 3.

Lemma 3.0.2 For any network $N$ and any (weakly) monotonic key-function $f$, the functions $f$ and $T^N$ commute; that is, $f(T^N(v)) = T^N(f(v))$ for every input vector $v$.

Recall that a network $N$ of width $n$ sorts a vector $v$ if $v$ is an input vector of $N$, the output labels of $N$ are $\{\hat{o}_i|i \in [0,n]\}$ and $T^N(v)$ is a sorted sequence. A network $N$ sorts a vector set $V$ if it sorts all vectors in $V$. A network which sorts all its input vectors is a sorting network. Lemma 3.0.2 implies the following well-known theorem by Knuth [10, page 224].

Theorem 13.1.1 (The 0-1 principle). A network is a sorting network iff it sorts all its 0-1 input vectors.

Recall that a monotonic image of a vector $v$ is a vector of the form $f(v)$ for some monotonic key function $f$. Let $V$ be a set of vectors. The monotonic-closure of $V$, denoted $\overline{V}$, is the set of all monotonic images of all members of $V$. A set of vectors $V$ is monotonically-closed iff it is the monotonic-closure of itself. For a function $g$ and a set $X$ define:

$$g(X) \triangleq \{g(x)|x \in X \text{ and } g(x) \text{ is defined} \}$$

The following lemma follows immediately from Lemma 3.0.2.

Lemma 13.1.2. Let $V$ be a set of input vectors of a network $N$ and let $S$ be a monotonically-closed set of vectors. Then $T^N(V) \subset S$ iff $T^N(\overline{V}) \subset S$.

For $V$ and $S$ of Lemma 13.1.2, let $F$ be the functionality whose set of valid vectors is $\overline{V}$ and the desired condition is to be a member of $S$. Lemma 13.1.2 states that $V$ is a conclusive set for $F$. Lemma 13.1.2 and the fact that any vector is the monotonic image of some permutation establishes the following well-known lemma.

Lemma 13.1.3. A network sorts all its permutation input vectors iff it is a sorting network.

For a set of vectors $V$, let $V^{0-1}$ denote the subset of the $0-1$ vectors of $V$. The following lemma is an extension of Lemma 13.1.1 and has a similar proof; furthermore, it follows from Lemma 13.1.8 which we prove ahead and whose proof is not based on Lemma 13.1.4; hence, the proof of 13.1.4 is omitted.
Lemma 13.1.4. Let \( V \) be a monotonically-closed set of input vectors of a network \( N \). Then \( N \) sorts \( V \) iff \( N \) sorts \( V^{0−1} \).

We now present several natural monotonically-closed sets of vectors, most of them were presented earlier in this work. A sandwich is a bisorted vector \( ⟨\vec{a}, \vec{b}⟩ \) that can be made sorted by sandwiching the entire \( \vec{a} \) sequence between an initial part of \( \vec{b} \) and the rest of \( \vec{b} \); this also includes the case where the initial part of \( \vec{b} \) or the rest of \( \vec{b} \) is empty. A sequence is ascending-descending iff it is a concatenation of 2 sequences, the first ascending, the second descending. A sequence is descending-ascending iff it is a concatenation of 2 sequences, the first descending, the second ascending. Following Batcher [1], a sequence is bitonic iff it is a rotation of an ascending-descending sequence. This includes the case of a null rotation, that is, any ascending-descending sequence is bitonic. A sequence is unitonic iff it is a rotation of an ascending sequence. A sequence is halved if its width is even, say \( 2n \), and \( a_i \leq a_j \) for every \( i < n \leq j \). The following lemma is straightforward

Lemma 13.1.5. For every \( n \in \mathbb{N} \), the following sets are monotonically-closed:
(a) The set of sorted sequences of width \( n \).
(b) The set of unitonic sequences of width \( n \).
(c) The set of bitonic sequences of width \( n \).
(d) The set of ascending-descending sequences of width \( n \).
(e) The set of descending-ascending sequences of width \( n \).
(f) The set of halved sequences of width \( 2n \).
(g) The set of bisorted vectors of width \( 2n \).
(h) The set of sandwiches of width \( 2n \).

A halver is network which halves any input vector. A network that sorts all bitonic (or unitonic or ascending-descending, etc.) sequences of width \( n \) is called a bitonic (or unitonic or ascending-descending, etc.) sorter of width \( n \), respectively. Note the difference between a bitonic sorter which is a network that sorts all bitonic sequences of a given width and the bitonic sorting network which is a sorting network invented by Batcher [1] and is built from bitonic sorters. By Lemma 13.1.4 and Lemma 13.1.5 we get that:

Lemma 13.1.6. For any \( X \), a set of vectors mentioned in Lemma 13.1.5, a network \( N \) sorts \( X \) iff it sorts \( X^{0−1} \).

A vector set \( X \) supprots a vector \( v \) if \( (\{v\})^{0−1} \subseteq X \). A vector set \( X \) supprots a vector set \( V \) if it supprots every vector of \( V \). A vector set \( X \) is complete if for every vector \( v \), \( X \) supports \( v \) iff \( v \in X \). Clearly, any complete set of vectors is also
monotonically-closed. Many natural sets of vectors are complete as stated by the following lemma

**Lemma 13.1.7.** All sets mentioned in Lemma 13.1.5, except sets (b) and (h) (the set of unitonic sequences and the set of sandwiches), are complete.

**Proof.** We consider only the hardest case of bitonic sequences. It suffices to show that any non-bitonic sequence $\vec{v}$ has a non-bitonic $0-1$ monotonic image. Without loss of generality, we assume that $v_0$ is a minimal key in $\vec{v}$; let $j$ be such that $v_j$ is a maximal key in $\vec{v}$. Let $\vec{x}$ and $\vec{y}$ be the subsequences of $\vec{v}$, the first from $v_0$ to $v_j$ and the second from $v_{j+1}$ to the end of $\vec{v}$. Since $\vec{v}$ is non-bitonic, either $x$ is not ascending or $y$ is not descending. Both cases are similar so we consider only the former. If $x$ is not ascending, there is $i \in [1, j-1)$ such that $v_i > v_{i+1}$. This implies that $\vec{v}$ has a $0-1$ monotonic image $\vec{u}$ in which $u_0, u_{i+1} = 0$ and $u_i, u_j = 1$; hence, $\vec{u}$ is not bitonic.

It is not hard to see that not all natural sets of vectors are complete; for example:

a) For $n > 2$, the set of unitonic sequences of width $n$ is not complete.

b) For $n > 1$, the set of sandwich vectors of width $2n$ is not complete.

The following lemma generalizes Lemma 13.1.4 to cases where the output of the network is not required to be sorted, but to be a member of a complete set of vectors.

**Lemma 13.1.8.** Let $X$ be a set of input vectors of a network $N$, let $X$ support $Y$, let $S$ be complete and let $T^N(X) \subset S$. Then $T^N(Y) \subset S$.

**Proof.** Assume for a contradiction, that $T^N(Y) \not\subseteq S$. Hence, there is a vector $v \in Y$ s.t. $T^N(v) \not\in S$. Since $S$ is complete, $S$ does not support $T^N(v)$; that is, there is a monotonic key function $f$ s.t. $f(T^N(v))$ is a 0-1 vector and $f(T^N(v)) \not\in S = S$. By Lemma 3.0.2, $T^N(f(v)) = f(T^N(v))$ and so $T^N(f(v)) \not\in S$. This implies that $f(v) \not\in X$. Moreover, by Lemma 13.1.2, $f(v) \not\in \overline{X}$, contradicting the fact that $X$ supports $v$.

Lemma 13.1.4 follows from Lemma 13.1.8 when $Y = V, X = V^{0-1}$ and $S$ is the set of the sorted sequences and from the fact that if a set of vectors $V$ is monotonically-closed then $V^{0-1}$ supports $V$.

We present two applications of Lemma 13.1.8 which enable us to construct conclusive sets much smaller then those restricted to $0-1$ vectors. Consider the two straightforward facts:

1. The set of unitonic permutations supports the set of bitonic sequences.
2. The set of sandwich permutations supports the set of bisorted sequences.

Lemma 13.1.8 and the facts above imply that:

**Lemma 13.1.9.**

a) A network is a merging network iff it is a sandwich-permutation sorter.

b) A network is a bitonic sorter iff it is a unitonic-permutation sorter.

In contrast to unitonic sorters, not all ascending-descending sorters are bitonic sorters; that is, for every \( n \geq 3 \), it is easy to construct an ascending-descending sorter of width \( n \), which is not a bitonic sorter.

As said, the above techniques are powerful tools for constructing small conclusive sets – much smaller than those restricted to \( 0-1 \) vectors. A natural question in this context is what is the smallest set of \( 0-1 \) vectors which is conclusive for sorting, merging, etc. By Lemma 13.1.9, there is a conclusive set for merging two sequences of width \( n \) each into a sorted sequence which has \( n+1 \) members; however, by the following lemma, 13.1.10, the smallest conclusive set of \( 0-1 \) vectors for the same functionality has \((n+1)^2 - 2\) vectors. For the functionality of sorting bitonic sequences of width \( n \), Lemma 13.1.9 implies there is a conclusive set of size \( n \), while, again by Lemma 13.1.10, the smallest conclusive set of \( 0-1 \) vectors is of size \( n \cdot (n - 1) \).

**Lemma 13.1.10.** For any non-constant \( v \in \{0,1\}^n \) there is a function \( f : \{0,1\}^n \rightarrow \{0,1\}^n \), computable by a network of comparators, s.t. \( f(v) \) is not sorted while \( f(v') \) is sorted for any other \( v' \in \{0,1\}^n \).

**Proof.** Given such a vector \( v \) let \( f = T^N \) where \( N \) is constructed as follows. First of all \( N \) statically (without any comparisons) partitions its input vector \( u \in \{0,1\}^n \) into two sequences \( \bar{u}^0 \) and \( \bar{u}^1 \) s.t. \( \bar{u}^0 \) contains all the elements \( u_j \) where \( v_j = 0 \) and \( \bar{u}^1 \) contains all the elements \( u_j \) where \( v_j = 1 \). Since \( v \) is non-constant none of the sequences \( \bar{u}^0 \) and \( \bar{u}^1 \) is empty. Let \( n^0 \) and \( n^1 \) be the width of \( \bar{u}^0 \) and \( \bar{u}^1 \), respectively.

The network \( N \) computes \( x^0 \) and \( x^1 \) which are the maximal element of \( \bar{u}^0 \) and the minimal element of \( \bar{u}^1 \), respectively. Let \( \bar{w} \) be the sequence composed of all elements of \( u \) except \( x^0 \) and \( x^1 \). The network \( N \) partitions \( \bar{w} \) into \( \bar{w}^0 \) and \( \bar{w}^1 \) s.t. \( |\bar{w}^0| = n^0 - 1 \), \( |\bar{w}^1| = n^1 - 1 \) and \( \bar{w}^0 << \bar{w}^1 \). The network \( N \) sorts the sequence \( x^1 \cdot \bar{w}^0 \) and these elements are the lowest \( n^0 \) keys of the output and \( N \) sorts the sequence \( x^0 \cdot \bar{w}^1 \) and these elements are the highest \( n^1 \) keys of the output. It is not hard to see that \( x^1 \cdot \bar{w}^0 << x^0 \cdot \bar{w}^1 \) iff \( u \neq v \).

A weaker version of Lemma 13.1.10 was proved by Rice [15]. In his version the term “is computable by a network of comparators” is replaced by the weaker condition
“is a member of the IP class” where the IP class is defined as follows. A mapping \( f : \mathcal{K}^n \to \mathcal{K}^n \) is isomeric if for every \( v \in \mathcal{K}^n \) any key \( k \in \mathcal{K} \) appears the same number of times in \( v \) and in \( f(v) \). The IP\(^{[15]} \) class is the set of real functions from \( \mathbb{R}^n \) to \( \mathbb{R}^n \) which are continuous and isomeric. Clearly, any mapping which is computable by network of comparators is a member of the IP class; on the other hand, as we now show, for any \( n > 3 \), there are members of the IP class which are not computable by a network of comparators. We present only the case of \( n = 4 \) and for the time being consider only boolean functions (where the set of possible keys is \( \{0, 1\} \)).

Let \( g : \{0, 1\}^4 \to \{0, 1\} \) be a mapping computable by a min/max network but is not computable by a network of comparators. Such a function was shown by Knuth and an example of such a function was presented in Chapter 11. Let \( h : \{0, 1\}^4 \to \{0, 1\}^4 \) be the unique function satisfying the following requirements for every argument \( v \):

- \( h(v)_0 = g(v) \).
- \( h(v)_1 \leq h(v)_2 \leq h(v)_3 \)
- \( h \) is isomeric.

It is not hard to see that all the projections of \( h \) are monotonic and non-constant. It is well known that any such boolean function is computable by a digital network of \textbf{and} and \textbf{or} gates; that is, it is computable by a min/max network. Hence, \( h \) is computable by a min/max network and not by a network of comparators.

The function \( h \) (as well as any other mapping which is computable by a min/max network) has a unique extension to a function \( \bar{h} : \mathbb{R}^4 \to \mathbb{R}^4 \). By a straightforward application of Lemma 3.0.2, the function \( \bar{h} \) is isomeric. Hence, \( \bar{h} \) is a member of the IP class and is not computable by a network of comparators.

### 13.2 conclusiveness by agreement

Recall that two functions \( g \) and \( g' \), defined over a domain \( D \), agree on a pair of elements \( (x, y) \) of \( D \) if \( x \) and \( y \) can be named \( z_1 \) and \( z_2 \) such that \( g(z_1) \leq g(z_2) \) and \( g'(z_1) \leq g'(z_2) \). If \( g \) and \( g' \) agree on every pair of elements of \( D \) we say that \( g \) and \( g' \) agree. For a vector set \( X \), let \( \tilde{X} \) denote the set of vectors which agree with some vector of \( X \). Note that the ‘\( \sim \)’ operator is not idempotent; namely, for most sets \( \tilde{X} \neq \tilde{X} \). The format of the following lemma is different from the format of the previous lemmas concerning conclusive sets. This lemma follows immediately from Lemma 3.0.3.
Lemma 13.2.1. Let $V$ be a set of input vectors of a network $N$, let $S$ be a set of vectors and let $T^N(V) \subseteq S$. Then $T^N(\tilde{V}) \subseteq \tilde{S}$.

We use Lemma 13.2.1 to present a conclusive set for the functionality of halving. A network $N$ halves a vector $v$ if $v$ is an input vector of $N$ and $T^N(v)$ is halved. A network $N$ halves a vector set $V$ if it halves every member of $V$. A vector is balanced if it is a $0-1$ vector having the same number of zeroes and ones.

Lemma 13.2.2. A network $N$ halves all its balanced 0–1 input vectors iff it is a halver.

Proof. The right to left implication is trivial. For the other direction, let $2n$ be the width of $N$, let $V$ be the set of all balanced input vectors of $N$ and let $h$ be the only balanced sorted sequence of width $2n$. The required implication follows by applying Lemma 13.2.1 to $V$ and $S = \{h\}$ and using the following two facts. A sequence of width $2n$ is halved iff it agrees with $h$ and any input vector of $N$ agrees with some member of $V$. 

\qed
Chapter 14

The unique bitonic sorter

This chapter studies bitonic sorters of minimal depth and shows that when the width is a power of two, such a network is unique. This result is based on Theorem 10.0.31 and demonstrates the power of this theorem for analyzing comparator networks.

For a sequence \( \mathbf{a} \) let \( \text{rev}(\mathbf{a}) \) denote the reverse of \( \mathbf{a} \); that is, \( \text{rev}(\mathbf{a})_i = a_{n-1-i} \) for \( i \in [0,n) \) where \( n \) is the length of \( \mathbf{a} \). Recall that for any two sequences \( \mathbf{a} \) and \( \mathbf{b} \), \( \mathbf{a} \cdot \text{rev}(\mathbf{b}) \) denotes the concatenation of the sequences \( \mathbf{a} \) and \( \mathbf{b} \); namely, the sequence \( \mathbf{a} \) followed by the sequence \( \mathbf{b} \).

Clearly, if \( \mathbf{a} \) and \( \mathbf{b} \) are ascending sequences then the sequence \( \mathbf{a} \cdot \text{rev}(\mathbf{b}) \) is a bitonic sequence. For a bitonic sorter \( N \) of even width, let \( \hat{N} \) denote the merging network derived from \( N \) by the last fact. That is, assume that the input labels of \( N \) are \( \langle \hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{2n-1} \rangle \) \( (N \) receives a sequence \( \mathbf{x} \) of width \( 2n \)). The network \( \hat{N} \) is derived from \( N \) by a relabelling so that whenever \( \hat{N} \) receives the bisequenced vector \( \langle \mathbf{a}, \mathbf{b} \rangle \), the network \( N \) receives the sequence \( \mathbf{x} = \mathbf{a} \cdot \text{rev}(\mathbf{b}) \).

For two sequences \( \mathbf{a} \) and \( \mathbf{b} \) of the same length \( n \) define \( \min(\mathbf{a}, \mathbf{b}) \) and \( \max(\mathbf{a}, \mathbf{b}) \) to be the two sequences of length \( n \) defined by \( \min(\mathbf{a}, \mathbf{b})_i = \min(a_i, b_i) \) and \( \max(\mathbf{a}, \mathbf{b})_i = \max(a_i, b_i) \). For a vector \( v \) and \( u \), let \( v \ll u \) denote the fact that any member of \( u \) is not smaller than any member of \( v \). Let \( \mathbf{a} \) be a sequence of length \( n \) and let \( i < j < n \). Let \( \mathbf{a}(i, j) \) denote the subsequence of \( \mathbf{a} \) containing all elements of \( \mathbf{a} \) from position \( i \) until \( j \) (not including) position \( j \). The cornerstone of Batcher’s technique to construct a bitonic sorter [1] is the following lemma.

**Lemma 14.0.3.** Let \( \mathbf{a} \) be a bitonic sequence of width \( 2n \), and let \( \bar{x} = \min(\mathbf{a}[0,n), \mathbf{a}[n, 2n)) \) and let \( \bar{y} = \max(\mathbf{a}[0,n), \mathbf{a}[n, 2n)) \). Then:

a) \( \bar{x} \ll \bar{y} \).

b) Both sequences \( \bar{x} \) and \( \bar{y} \) are bitonic.

63
Batcher’s construction is based on a recursive applications of Lemma 14.0.3; for every \( n \) a power of two, it produces a bitonic sorter of width \( n \) and of depth \( \log(n) \) called the \textit{Batcher bitonic sorter} and denoted \( B^n \). By the following lemma, \( B^n \) is of a minimal depth.

**Lemma 14.0.4.** Let \( n \) be a power of two and let \( N \) be a minimal depth bitonic sorter of width \( 2n \). Then:

1. The depth of \( N \) is \( \log(2n) \).
2. The network \( N \) is AMOP.
3. Any input edge is matched with another input edge.
4. For all \( i \in [0, n) \) the two input edges \( \hat{x}_i \) and \( \hat{x}_{i+n} \) are matched.

**Proof.** Any bitonic sorter has a path from any input edge to every output edge. Hence the depth of any bitonic sorter of width \( 2n \) is at least \( \log(2n) \). Since \( B^{2n} \) is of depth \( \log(2n) \), this is the minimal depth of such a bitonic sorter. Statement (2) and (3) follow from the above facts.

Assume, for a contradiction, that statement (4) does not hold. Then, there are \( j, k < n \) such that \( \hat{x}_j \) and \( \hat{x}_{j+k} \) enter the same comparator. Since a cyclic rotation of a bitonic sequence is a bitonic sequence, a cyclic rotation of the indices of the input edges of \( N \) produces a minimal depth bitonic sorter. Therefore, we may assume, without loss of generality, that \( j = 0 \). In the merging network \( \hat{N} \) the two input edges \( \hat{a}_0 \) and \( \hat{a}_k \) are matched. The networks \( \hat{N} \) and \( N \) are isomorphic, up to relabelling; hence, \( N \) is AMOP and satisfies the premise of Lemma 7.0.21. By this Lemma, \( \hat{a}_0 \) is matched with some \( \hat{b}_q \). \( \square \)

Recall the two Batcher cleavers (the parallel cleaver and the cross cleaver, defined in Chapter 5) used in the construction of Batcher merging network. Recall that the cross network is the Batcher merging network whose recursive construction uses exclusively the cross cleaver. Let \( X^{2n} \) denote the cross network of width \( 2n \). The following lemma is proved by a straightforward induction.

**Lemma 14.0.5.** For any \( n \) a power of two:

1. The i.m.f. of \( X^{2n} \) is the order inverting mapping of \([0, n)\). That is \( \text{imf}^{X^{2n}} \) is defined by \( i \mapsto n - 1 - i \).
2. Every path in \( X^{2n} \) from an input edge to an output edge has exactly \( \log(2n) \) comparators.
The following theorem is the main result of this chapter.

**Theorem 14.0.6.** For any \( n \), a power of two, there is a unique bitonic sorter of width 2 and of minimal depth.

**Proof.** The case where \( n = 1 \) is trivial so assume \( n > 1 \). Let \( N \) be a minimal depth bitonic sorter of width \( n \). By Lemma 14.0.4 (4), \( \text{imf}^N \) is the order inverting mapping of \([0, \frac{n}{2})\); that is, \( \text{imf}^N = \text{imf}^X \). Let \( N' \) be derived from \( \hat{N} \) by bypassing all degenerate comparators of \( \hat{N} \) w.r.t. merging. Clearly, \( \text{imf}^{N'} = \text{imf}^N \). By Lemma 14.0.4 (2), \( N \) is AMOP. Therefore, \( N' \) is a regular compact merging network. By Theorem 10.0.31, \( N' \) is a Batcher merging network.

As said, \( \text{imf}^{N'} = \text{imf}^X \) and both networks are Batcher merging networks; by Lemma 5.0.9, \( N' \cong X^n \). By Lemma 14.0.5 (2), each path in \( N' \) from an input edge to an output edge has exactly \( \log(n) \) comparators. Since the depth of \( \hat{N} \) is \( \log(n) \), no comparator has been bypassed while deriving \( N' \) from \( \hat{N} \). That is, \( \hat{N} \cong N' \cong X^n \).

Since \( N \) is derived from \( \hat{N} \cong X^n \) by a predefined relabelling, \( N \) is unique. \( \square \)

We now show that the requirement “\( n \) is a power of two” in Lemma 14.0.6 is mandatory. Consider the case of \( n = 3 \). Clearly, any sequence of length 3 is bitonic and so any bitonic sorter of width 3 is a sorting network. It is easy to verify that the depth of a sorting network of width 3 is at least 3. Two non-isomorphic bitonic sorters of width 3 are depicted in figure 14.1. These networks are not isomorphic, even after any relabeling of their input and output edges, since the left one has a path of edges of type \((\text{max}, \text{min}, \text{max})\) and the network on the right does not have such a path.

![Figure 14.1: Two non-isomorphic bitonic sorters of width 3](image)

By Lemma 14.0.4 (1), the minimal depth of a bitonic sorter of width 4 is 2 while by the above argument the minimal depth of a bitonic sorter of width 3 is 3. This yields the following surprising result:
Lemma 14.0.7. The minimal depth of a bitonic sorter of width $n$ is not a monotonic function of $n$.

Clearly, the functionality of bitonic sorting is stronger than merging. As discussed in Chapter 14, there are $4n$ natural ways to relabel a bitonic sorter of width $2n$ into a merging network. However, if the bitonic sorter is of minimal depth, then by Lemma 14.0.6 it is unique. Moreover, this unique network halves the inputs after a depth one network. The only minimal depth merging network with this property is the strictly-cross network. Therefore

Lemma 14.0.8. Let $B$ be a bitonic sorter of minimal depth whose width is a power of two and let $M$ be a merging network derived from $B$ only by relabelling of its input edges. Then $M$ is the strictly-cross network.
Chapter 15

Zipper sorters

This chapter introduces a new merging technique which is based on a certain way to quantify the amount of “unsortedness” of a bysorted vector and a network of depth one which halves this quantity. This technique produces minimal depth merging networks which, not surprisingly, are Batcher merging networks.

A bisequenced vector \( \langle \vec{x}, \vec{y} \rangle \) of width 2\( n \) is a \((k\pm \frac{t}{2})\)-zipper iff all following conditions hold

a ) \( t \in \mathbb{N} + 1, k + \frac{t}{2} \in \mathbb{Z}, |k| \leq \frac{t}{2} \). The parameters \( t, k \) are called the tolerance and the offset of the zipper.

b ) \( \vec{x} \) and \( \vec{y} \) are sorted; hence, a zipper is bisorted.

c ) \( \forall i, j \in [0, n) : j = i + k - \frac{t}{2} \Rightarrow y_j \leq x_i \)

d ) \( \forall i, j \in [0, n) : j = i + k + \frac{t}{2} \Rightarrow x_i \leq y_j \)

A pair \( \langle k, t \rangle \) that satisfies condition (a) is called (zipper) admissible. In this discussion, the only interesting case where the tolerance \( t \) is odd is for \( t = 1 \). Such a zipper is essentially sorted; namely, it can be transformed into a sorted sequence by rearrangement of its elements (without any comparisons). By our definition, any \( \langle k, t, 2n \rangle \) as above are associated with a set \( \Gamma^{k,t,2n} \) of weak inequalities of the form \( \alpha_i \leq \beta_j \) where \( \alpha, \beta \in \{x, y\} \). A vector \( \langle \vec{x}, \vec{y} \rangle \) is a \((k \pm \frac{t}{2})\)-zipper of width 2\( n \) if it satisfies all the inequalities of \( \Gamma^{k,t,2n} \). In our drawings of a zipper, we use directed edges to denote the inequalities of condition (c) and (d) as above. An edge from \( \alpha_i \) to \( \beta_j \) denotes that \( \alpha_i \geq \beta_j \).

All of the inequalities of \( \Gamma^{k,t,2n} \) are clearly preserved under any monotonic key functions; moreover, if a bisequenced vector violates an inequality of \( \Gamma^{k,t,2n} \), it has a 0 – 1 monotonic image which violates this inequality. Therefore:
Lemma 15.0.9. For any $n \in \mathbb{N} + 1$ and an admissible $(k, t)$, the set of $(k \pm \frac{t}{2})$-zippers of width $2n$ is monotonically-closed and complete.

We present a number of straightforward properties of zippers. To this end, we remind the reader of the following notations. For two sequences $\vec{a}$ and $\vec{b}$, $\vec{a} \cdot \vec{b}$ denotes the concatenation of the sequence $\vec{a}$ followed by the sequence $\vec{b}$. Let $\vec{a} \ll \vec{b}$ denote that any element of $\vec{a}$ is smaller or equal to any element of $\vec{b}$. For a sequence $\vec{a}$ of width $n$ and for $i < j < n$, let $\vec{a}[i, j)$ denote the subsequence of $\vec{a}$ containing all elements of $\vec{a}$ from position $i$ until (not including) position $j$.

Lemma 15.0.10. $\langle \vec{x}, \vec{y} \rangle$ is a $(k \pm \frac{t}{2})$-zipper iff $\langle \vec{y}, \vec{x} \rangle$ is a $(-k \pm \frac{t}{2})$-zipper.

Lemma 15.0.11. Let $\langle \vec{x}, \vec{y} \rangle$ be a $(k \pm \frac{t}{2})$-zipper, let $\vec{a}, \vec{b}$ be two sorted sequences of width $j$ such that $\vec{a} \ll \vec{x}, \vec{y}$ and $\vec{b} \gg \vec{x}, \vec{y}$ and let $(k - j, t)$ be admissible. Then $\langle (\vec{a} \cdot \vec{x}), (\vec{y} \cdot \vec{b}) \rangle$ is a $((k - j) \pm \frac{t}{2})$-zipper.

Lemma 15.0.12. Let $\vec{a}, \vec{b}$ be two sequences of width $j$, let $\vec{x}$ and $\vec{y}$ be two sequences such that $\langle (\vec{a} \cdot \vec{x}), (\vec{y} \cdot \vec{b}) \rangle$ is a $(k \pm \frac{t}{2})$-zipper and let $(k + j, t)$ be admissible. Then $\langle \vec{x}, \vec{y} \rangle$ is a $((k + j) \pm \frac{t}{2})$-zipper.

15.1 Tolerance-halvers

Let $(t, k)$ be admissible, let $t$ be even and let $n \in \mathbb{N}$. A tolerance-halfer with parameters $(t, k, 2n)$ is a mapping that transforms any $(k \pm \frac{t}{2})$-zipper of width $2n$ to a $(\tilde{k} \pm \frac{t}{4})$-zipper, where $\tilde{k} = |k| - \frac{t}{4}$. Note that the pair $(\tilde{k}, \frac{t}{4})$ is admissible. A network $N$
is a tolerance-halfer for the parameters \((t, k, 2n)\) iff its input/output transformation \(T^N\) is a tolerance-halfer with those parameters.

We now present a conclusive set for the functionality of tolerance halving. A bisequenced vector \(v = (\vec{v}, \vec{y})\) is called arithmetic iff the sequences \(\vec{v}\) and \(\vec{y}\) contain only integer numbers\(^1\), both \(\vec{v}\) and \(\vec{y}\) are arithmetic progressions with difference one and \(x_0 = 0\). Arithmetic zippers are easy to study due to the following trivial fact:

**Lemma 15.1.1.** An arithmetic bisequenced vector \((\vec{v}, \vec{y})\) is a \((k \pm \frac{t}{2})\)-zipper iff \(y_0 \in [-\frac{t}{2} - k, \frac{t}{2} - k]\).

Note that the set of arithmetic \((k \pm \frac{t}{2})\)-zippers supports the set of all \((k \pm \frac{t}{2})\)-zippers. This fact and Lemmas 15.0.9 and 13.1.8 imply the following lemma.

**Lemma 15.1.2.** For any even \(t\) and \(k, n \in \mathbb{N}\), a network \(N\) is a tolerance-halfer with parameters \((k, t, 2n)\) iff it has this functionality when the input vectors are further restricted to be arithmetic.

For any \(t, k, 2n\) as above, we construct a tolerance-halfer denoted \(h^{k,n}\). Note that the tolerance \(t\) is missing in \(h^{k,n}\), since our mapping is independent of \(t\). As said, the argument of \(h^{k,n}\) is a bisequenced vector, say \((\vec{x}, \vec{y})\). We divide the definition of \(h^{k,n}\) to three cases according to the offset \(k\).

- **Case 1**, \(0 \leq k \leq n\): let \(h^{k,n}(⟨\vec{x}, \vec{y}⟩) ≜ ⟨\vec{u}, \vec{v}⟩\) where
  \[
  \begin{align*}
  \vec{u} &= \min(\vec{x}[0, n - k], \vec{y}[k, n]) \cdot \vec{x}[n - k, n] \\
  \vec{v} &= \vec{y}[0, k] \cdot \max(\vec{x}[0, n - k], \vec{y}[k, n]).
  \end{align*}
  \]

- **Case 2**, \(n < k\): Let \(h^{k,n}(⟨\vec{x}, \vec{y}⟩) ≜ h^{n,n}(⟨\vec{x}, \vec{y}⟩)\).
  (Note that \(h^{n,n}(⟨\vec{x}, \vec{y}⟩) = ⟨\vec{x}, \vec{y}⟩\).

- **Case 3**, \(k < 0\): Let \(h^{k,n}(⟨\vec{x}, \vec{y}⟩) = h^{-k,n}(⟨\vec{y}, \vec{x}⟩)\).

Clearly, there is a unique non-degenerate depth one network which implements \(h^{k,n}\). We denote this network by \(H^{k,n}\). The network \(H^{2,5}\) is depicted in Figure 15.2. As in most of our figures, lines represent comparators, a hollow arrowhead represent the outgoing min edge of a comparator and a solid arrowhead represent its outgoing max edge. Arrows with plain arrowheads represent edges which do not exit a comparator; i.e. edges which are both input edges and output edges.

\(^1\)So far in this work, the keys were natural numbers.
Lemma 15.1.3. Let \( t \) be even and let \( (t, k) \) be admissible. Then \( H^{k,n} \) is a tolerance-halfer with parameters \( (t, k, 2n) \).

As said, the same network \( H^{k,n} \) is a tolerance-halfer for any even \( t \).

Proof. Let \( \langle \vec{x}, \vec{y} \rangle \) be a \((k \pm \frac{t}{2})\)-zipper. By Lemma 15.0.10 we may assume that \( k \geq 0 \). By Lemma 15.1.2, we may assume without loss of generality that \( \langle \vec{x}, \vec{y} \rangle \) is arithmetic.

By Lemma 15.1.1, \( y_0 \in \left[ -\frac{t}{2} - k, \frac{t}{2} - k \right] \). Let \( \tilde{k} = k - \frac{t}{2} \) and let \( \langle \tilde{u}, \tilde{v} \rangle = h^{k,n}(\langle \vec{x}, \vec{y} \rangle) \).

First, consider the case where \( k = 0 \). Clearly, \( \langle \tilde{u}, \tilde{v} \rangle = \langle \vec{x}, \vec{y} \rangle \) or \( \langle \tilde{u}, \tilde{v} \rangle = \langle \vec{y}, \vec{x} \rangle \). In the case of \( \langle \tilde{u}, \tilde{v} \rangle = \langle \vec{x}, \vec{y} \rangle \), \( \langle \tilde{u}, \tilde{v} \rangle \) is arithmetic and \( 0 = x_0 = u_0 \leq v_0 = y_0 \); hence, \( v_0 \in [0, \frac{t}{4}] \). By Lemma 15.1.1, \( \langle \tilde{u}, \tilde{v} \rangle \) is \(-\frac{t}{4} \leq \tilde{k} \leq \frac{t}{4}\)-zipper. Next, consider the case of \( \langle \tilde{u}, \tilde{v} \rangle = \langle \vec{y}, \vec{x} \rangle \). By similar arguments, \( y_0 \in [-\frac{t}{2}, 0] \). By Lemmas 15.0.10 and 15.1.1, \( \langle \tilde{u}, \tilde{v} \rangle \) is a \((-\frac{t}{4} \pm \frac{t}{4})\)-zipper. This concludes the case of \( k = 0 \).

Next consider the case of \( k > 0 \). Let \( \tilde{a} \) be the sequence of width \( k \) composed only of the key \(-\infty\) and let \( \tilde{b} \) the sequence of width \( k \) composed only of the key \(+\infty\). By Lemma 15.0.11, \( \langle \tilde{a} \cdot \vec{x}, \vec{y} \cdot \tilde{b} \rangle \) is a \((0 \pm \frac{t}{2})\)-zipper. Let \( \langle \tilde{p}, \tilde{q} \rangle = H^{0,k,n}(\langle \tilde{a} \cdot \vec{x}, \vec{y} \cdot \tilde{b} \rangle) \). By the case of \( k = 0 \), \( \langle \tilde{p}, \tilde{q} \rangle \) is a \((-\frac{t}{4} \pm \frac{t}{4}\))-zipper of width \( 2n + 2k \). It is not hard to see that \( \tilde{p} = \tilde{a} \cdot \tilde{q} \) and \( \tilde{q} = \tilde{v} \cdot \tilde{b} \). By Lemma 15.0.12, \( \langle \tilde{u}, \tilde{v} \rangle \) is a \((\tilde{k} \pm \frac{t}{4})\)-zipper.

Let \( (t, k) \) be admissible. A network \( Z \) is a \((k \pm \frac{t}{2})\)-zipper sorter of width \( 2n \) iff it sorts any \((k \pm \frac{t}{2})\)-zipper of width \( 2n \). Recursive applications of Lemma 15.1.3 produce a \((k \pm \frac{t}{2})\)-zipper sorter as stated in the following lemma:

Lemma 15.1.4. Let \( (t, k) \) be admissible and let \( n \in \mathbb{N} + 1 \). Then there exists a \((k \pm \frac{t}{2})\)-zipper sorter of width \( 2n \) and of depth \( \lceil \log t \rceil \).

Lemma 15.1.4 for the special case of \( k = \frac{t}{2} \) has been observed by Batcher and Lee [2]. It is not hard to see that, when \( t \leq n \), the \((k \pm \frac{t}{2})\)-zipper sorter provided by Lemma 15.1.4 is of minimal depth.
15.2 The zipper merging technique

The concept of a zipper offers the following “zipper merging technique” to construct minimal depth merging networks. The simple variant of this technique is based on the fact that any bisorted vector is a $(0 \pm \frac{n}{2})$-zipper; therefore it can be sorted by a $(0 \pm \frac{n}{2})$-zipper sorter. As Batcher has observed [2], the resulting network when $n$ is a power of 2 is exactly his odd/even network. The general and more interesting variant is based on the fact, shown shortly, that any bisorted vector of width $2n$ can be transformed, by a depth one network, into a $(j \pm \frac{n}{2})$-zipper for any given $j \in [-\frac{n}{2}, \frac{n}{2})$.

By Lemma 15.1.4, the resulting vector can be sorted by a network of depth $\log n$. To define the above depth one network, we proceed as follows. Let $l \in [0, n)$ and let $(\vec{x}, \vec{y})$ be a bisorted vector of width $2n$. Define

$$
\vec{x}' \triangleq x[0, l], \quad \vec{x}'' \triangleq x[l, n), \quad \vec{y}' \triangleq y[0, n-l), \quad \vec{y}'' \triangleq y[n-l, n)
$$

$$
\vec{u}' \triangleq \min(\vec{x}', \vec{y}''), \quad \vec{u}'' \triangleq \max(\vec{x}'', \vec{y}''), \quad \vec{v}' \triangleq \min(\vec{x}', \vec{y}'''), \quad \vec{v}'' \triangleq \max(\vec{x}'', \vec{y}'')
$$

$$
\text{zip}^{l,n}(⟨\vec{x}, \vec{y}⟩) \triangleq ⟨\vec{u}', \vec{u}'', \vec{v}', \vec{v}''⟩
$$

Clearly there is a unique network of depth one that implements the $\text{zip}^{l,n}$ operator (See Figure 16.1). Let $G^{l,n}$ be that network.

![Figure 15.3: The $G^{3,5}$ network](image)

**Lemma 15.2.1.** Let $0 \leq l \leq n$. Then $⟨\vec{u}, \vec{v}⟩$ is a $((\frac{n}{2} - l) \pm \frac{n}{2})$-zipper of width $2n$ iff $⟨\vec{u}, \vec{v}⟩ = \text{zip}^{l,n}(⟨\vec{x}, \vec{y}⟩)$ for some bisorted vector $(\vec{x}, \vec{y})$.

**Proof.** Consider the right to left implication. Let $⟨\vec{u}, \vec{v}⟩ = \text{zip}^{l,n}(⟨\vec{x}, \vec{y}⟩)$ and let $\vec{x}', \vec{x}'', \vec{y}', \vec{y}'', \vec{u}', \vec{u}'', \vec{v}', \vec{v}''$ be as above. We first show that $⟨\vec{u}, \vec{v}⟩$ is bisorted. Clearly,
the four sequences $\bar{u}', \bar{u}'', \bar{v}', \bar{v}''$ are ascending. By definition $\bar{x}' \ll \bar{x}''$; this implies that $\bar{u}' = \min(\bar{x}', \bar{y}'') \ll \max(\bar{x}'', \bar{y}') = \bar{u}''$. By symmetry, $\bar{v}' \ll \bar{v}''$. Hence, $\langle \bar{u}, \bar{v} \rangle$ is bisorted.

Consider condition (c) in the zipper definition for $t = n$ and $k = \frac{n}{2} - l$ and let $i, j \in [0, n)$ where $j = i + k - \frac{t}{2} = i - l$. This implies that $u_i$ is a member of $\bar{u}''$ and that $u_i$ and $v_j$ are the max and min of the same pair of keys; therefore, $u_i \geq v_j$. That is, condition (c) holds. By symmetry, condition (d) holds.

To establish the left to right implication we show that any $((\frac{n}{2} - l) \pm \frac{n}{2})$-zipper, $\langle \bar{u}, \bar{v} \rangle$, is a fixed point of $\text{zip}^{l,n}$; that is, $\langle \bar{u}, \bar{v} \rangle = \text{zip}^{l,n}(\langle \bar{u}, \bar{v} \rangle)$. The (c) condition of a zipper implies that $\bar{u}'' = \max(\bar{u}'', \bar{v}')$ and that $\bar{v}' = \min(\bar{u}', \bar{v}'')$. The (d) condition of a zipper implies that $\bar{u}' = \min(\bar{u}', \bar{v}'')$ and that $\bar{v}'' = \max(\bar{u}', \bar{v}'')$. Altogether, $\langle \bar{u}, \bar{v} \rangle$ is a fixed point of $\text{zip}^{l,n}$.

Our zipper merging technique is based on Lemmas 15.2.1 and 15.1.4. The network $G^{l,n}$ transforms any width $2n$ bisorted vector into a $((\frac{n}{2} - l) \pm \frac{n}{2})$-zipper which can be sorted by a depth $\log n$ network provided by Lemma 15.1.4.

### 15.3 Unique zipper sorter

We now show that minimal depth zipper sorters with no degenerate comparators are unique. To this end, we consider minors of zippers sorters having the following functionality. An inserter of width $n$ is a network $I$ that receives a single key $x$ and an ascending sequence $\hat{y}$ of width $n - 1$ and sorts any such vector. Its input edges are labeled $\bar{x}$ and $\hat{y}_i$ for $0 \leq i < n - 1$.

As we show, any zipper sorter contains several inserters which can be generated by the pruning operation introduced in Chapter 4. Let $\langle k, t \rangle$ be admissible and let $Z$ be a $(k \pm \frac{t}{2})$-zipper sorter of width $2n$. For any edge $\hat{x}_i$, $Z$ can be pruned into an inserter whose edge $\hat{x}$ is the edge $\hat{x}_i$ of $Z$ and all the other $\hat{x}_j$ edges of $Z$ are pruned. This pruning is charted by the input vector $v^i$ of $Z$ defined as follows: $v^i = (\bar{x}, \hat{y})$ is the $(k \pm \frac{t}{2})$-zipper of width $2n$ composed only of the keys $\{-\infty, 0, +\infty\}$; $x_i = 0$; any other element $\alpha_j$ of $v^i$ is either $-\infty$ or $+\infty$; it is $-\infty$ when $\alpha_j \leq x_i$ and it is $+\infty$ when $\alpha_j \geq x_i$. If the zipper conditions do not settle the relative order of $x_i$ and $\alpha_j$, then $\alpha_j = 0$. Note that the key 0 appears at least once and at most $t$ times in $v^i$. Let $I^i$ be the normalized variant of $\text{prun}(Z, v^i)$. Since $v^i$ is a $(k \pm \frac{t}{2})$-zipper, this pruning is honest and $I^i$ is an inserter of width $w$ where $w$ is the number of zeroes in $v^i$. (Note that $w$ may equal one.)

The following lemma follows from a straightforward reachability argument.

**Lemma 15.3.1.** Let $I$ be an inserter of width $n$ and let $\tilde{n} = 2^{\lceil \log(n) \rceil}$. Then:
a) The depth of \( I \) is at least \( \log(n) \).

b) If the depth of \( I \) is \( \log(n) \) then there is a \( j \) such that \( \hat{x} \) and \( \hat{y}_j \) are matched and
\[ j \in [n - \frac{n}{2} - 1, \frac{n}{2} - 1]. \]
In particular, if \( n \) is a power of two, then \( \hat{y}_j \) is exactly the center member of the \( \hat{y} \) sequence.

We use Lemma 15.0.10 to study the structure of minimal depth zipper sorters. To this end, we generalize the definition of an input matching function to zipper sorters and tolerance-halvers; in this case the sequence \( \bar{x} \) replaces the sequence \( \bar{a} \) and \( \hat{y} \) replaces \( \hat{b} \). The following lemma states that the i.m.f. of a minimal depth \((k \pm \frac{1}{2})\)-zipper sorter is determined by \( k \) and its width \( n \).

**Lemma 15.3.2.** Let \((k, t)\) be admissible, let \( t \) be a power of 2 and \( n \geq t > 1 \). Let \( Z \) be a non-degenerate \((k \pm \frac{1}{2})\)-zipper sorter of depth \( \log(t) \) and of width \( 2n \). Then \[ \text{imf}^Z = \text{imf}^{H_{k,n}}. \]

**Proof.** By Lemma 15.0.10, we may assume \( k \geq 0 \). We show that two edges, \( \hat{x}_i \) and \( \hat{y}_j \), are matched iff \( j = i + k \). Let \( i \in [0, n) \), let \( v^i \) be defined as above and let \( P^i \triangleq \text{prun}(Z, v^i) \); let \( w^i \) denote the width of \( P^i \). Recall that \( 1 \leq w^i \leq t \). Let the input edges of \( P^i \) be \( \hat{x}_i \) and \( \hat{y}_{i'}, \hat{y}_{i'+1}, \ldots \hat{y}_{im'} \); then, \( l^i = \max(0, i + k - \frac{f}{2} + 1) \) and \( m^i = \min(n - 1, i + k + \frac{f}{2} - 1) \). We consider two cases according to \( w^i \).

**Case 1 , \( w^i > \frac{f}{2} \):** Lemma 15.3.1, applied on the normalized version of \( P^i \) and for \( n = t \), implies that \( P^i \) and \( Z \) are of the same depth, \( \log(t) \); therefore, the edge \( \hat{x}_i \) is matched, both in \( P^i \) and in \( Z \), with the same \( \hat{y} \) edge, say \( \hat{y}_j \). First assume that \( l^i \neq 0 \) and that \( m^i \neq n - 1 \). In this case \( w^i = t \). By Lemma 15.3.1, \( j \) is the center member of the interval \([l^i, m^i]\); i.e., \( j = i + k \).

Assume now that either \( l^i = 0 \) or \( m^i = n - 1 \) (or both). The two cases are similar and we consider only the first. Since \( n \geq t \), the condition of \( l^i \geq 0 \) and \( w^i > \frac{f}{2} \) holds exactly when \( i \in [0, \frac{f}{2} - k) \triangleq Q \). Let \( f: Q \to Z \) be defined by \( f(q) = \text{imf}^Z(q) - k \). As said, \( f \) is total and clearly, \( f \) is one-to-one. Let \( q \in Q \). Lemma 15.3.1 and the fact that \( w_i - 2 = m^i = i + k + \frac{f}{2} - 1 \) imply that \( \hat{x}_i \) is matched with an edge \( \hat{y}_j \) where \( i + k = m^i + 2 - \frac{f}{2} - 1 = w_i - \frac{f}{2} - 1 \leq j < \frac{f}{2} \); hence, \( f \) is a permutation of \( Q \) and \( f(q) \geq q \). This implies that \( f \) is the identity function; i.e., \( \text{imf}^Z(i) = i + k \).

**Case 2 , \( w^i \leq \frac{f}{2} \):** In this case \( i \in [n - k, n) \). By case (1), all input edges \( \hat{y}_j \) for \( j \in [k, n) \) are matched to previous input edges. Let \( \langle \bar{x}, \bar{y} \rangle \) be a \((k \pm \frac{1}{2})\)-zipper. Condition (c) of a zipper and the facts that \( (k, t) \) is admissible and that \( n \geq t \) imply that for \( j \in [0, k) \), \( x_i \geq y_{i+k} \) and \( y_{i+k} \geq y_{(n-k)+k} = y_{n-k} \geq y_{k} \geq y_j \). Since \( Z \) has no degenerate comparators, the input edge \( \hat{x}_i \) is not matched. \( \square \)
Let \( \langle k, t \rangle \) be admissible and let \( \tilde{k} = |k| - \frac{t}{4} \). Recall that \( h^{k,n} \) transforms a \( (k \pm \frac{t}{2}) \)-zipper into a \( (\tilde{k} \pm \frac{t}{4}) \)-zipper. The next lemma provides some fixed points of \( h^{k,n} \) and is useful to establish the uniqueness of certain zipper sorters.

**Lemma 15.3.3.** Let \( t \) be even, let \( \langle k, t \rangle \) be admissible, let \( \tilde{k} = |k| - \frac{t}{4} \) and let \( \langle \tilde{x}, \tilde{y} \rangle \) be a \( (\tilde{k} \pm \frac{t}{4}) \)-zipper of width \( 2n \). Then \( h^{k,n}(\langle \tilde{x}, \tilde{y} \rangle) = \langle \tilde{x}, \tilde{y} \rangle \).

**Proof.** By Lemma 15.0.10, we may assume \( k \geq 0 \). Let \( \langle \tilde{u}, \tilde{v} \rangle = h^{k,n}(\langle \tilde{x}, \tilde{y} \rangle) \). First consider all \( i \in [n-k, n) \). By definition of \( h^{k,n} \), \( u_i = x_i \) and \( v_{n-i} = y_{n-i} \). Next consider all \( i \in [0, n-k) \). Since \( \langle \tilde{x}, \tilde{y} \rangle \) is a \( (\tilde{k} \pm \frac{t}{4}) \)-zipper, \( x_i \leq y_{i+k} + \tilde{k} \). By Lemma 15.3.2, \( Z \) is concatenation of \( H^{k,n} \) and a network \( Z' \). By Lemma 15.3.3, when \( Z \) receives a valid input then its subnetwork \( Z' \) receives a \( (\tilde{k} \pm \frac{t}{4}) \)-zipper; furthermore, any \( (\tilde{k} \pm \frac{t}{4}) \)-zipper can enter \( Z' \) this way. Therefore, \( Z' \) is a non-degenerate \( (\tilde{k} \pm \frac{t}{4}) \)-zipper sorter of depth \( \log(t) - 1 \). By the induction hypothesis, \( Z' \) is unique and therefore \( Z \) is unique.

**Lemma 15.3.4 (zipper sorters are unique).** Let \( t \) be a power of 2 and \( n \geq t \). Then there is a unique non-degenerate \( (k \pm \frac{t}{2}) \)-zipper sorter of depth \( \log(n) \).

**Proof.** The existence of such a sorter is provided by Lemma 15.1.4. We prove the uniqueness by induction on \( t \). Let \( Z \) be a non-degenerate \( (k \pm \frac{t}{2}) \)-zipper-sorter. For \( t = 1 \), any valid input of \( Z \) is essentially sorted; therefore, \( Z \) has no comparators and its input/output labels are predefined. Assume \( t > 1 \). By Lemma 15.3.2, \( Z \) is concatenation of \( H^{k,n} \) and a network \( Z' \). By Lemma 15.3.3, when \( Z \) receives a valid input then its subnetwork \( Z' \) receives a \( (\tilde{k} \pm \frac{t}{4}) \)-zipper; furthermore, any \( (\tilde{k} \pm \frac{t}{4}) \)-zipper can enter \( Z' \) this way. Therefore, \( Z' \) is a non-degenerate \( (\tilde{k} \pm \frac{t}{4}) \)-zipper sorter of depth \( \log(t) - 1 \). By the induction hypothesis, \( Z' \) is unique and therefore \( Z \) is unique.

We now show that all merging networks produced by the “zipper merging technique” are Batcher merging networks. By Lemma 15.2.1 and arguments which appeared in the last proof, for every \( k \in [0, n) \) there is exactly one merging network \( M \) such that the following holds

1. \( \text{imf}^M = \text{imf}^{G^{k,n}} \).
2. \( M \) is non-degenerate.
3. Its depth is \( \log(n) \).

It is not hard to see that \( \text{imf}^{G^{k,n}} \) is of the form \( x \mapsto x + k(\mod n) \). By Lemma 6.0.16, it is a congruent function and so if \( \text{imf}^M \); hence, there is a Batcher merging network which satisfies the above conditions.
Chapter 16

Merging by tri-section

This chapter presents a method for constructing merging networks which is based on partitioning, by a depth one network, the bisorted input vector into three sequences, $\vec{x}, \vec{y}, \vec{z}$, such that $\vec{x} \ll \vec{y} \ll \vec{z}$. This allows us to sort each of these sequences separately and in our technique the resulting network is of minimal depth $\log(2n)$; furthermore the sequences $\vec{x}$ and $\vec{z}$ are sorted by networks of depth $\lceil \log(|\vec{x}|) \rceil$ and $\lceil \log(|\vec{z}|) \rceil$, respectively. There is also a certain freedom concerning the length of these sequences; in particular, the length of $\vec{x}$ or $\vec{z}$ can be any number of $[0, n)$ where $2n$ is the width of the network.

This technique is of interest by itself and has the following practical advantage. Let the length of $\vec{x}$ or $\vec{z}$ be $k$ which is much smaller then $n$; the $k$ smallest or largest keys exit the merging network after a delay of at most $1 + \lceil \log(k) \rceil$ comparators; i.e. the resulting merging network produces the $k$ smallest keys or the $k$ largest keys faster then the well-known bound of $\log(2n)$.

This technique also answers the following question. Most of oblivious merging algorithms published are recursive (or can be presented as recursive) and produce minimal depth merging networks. A question arises whether these two properties are dependent on each other or whether there exists a minimal depth merging network (without degenerate comparators) which has no recursive structure. We show that the latter is correct.

We actually present two tri-section methods. In the symmetric method, $|\vec{x}| = |\vec{z}|$ and this number is a power of two. This is the only restriction regarding on the length of these sequences; moreover, $\vec{x}$ and $\vec{z}$ are sorted by isomorphic networks. In the asymmetric method, usually $|\vec{x}| \neq |\vec{z}|$ and the only restriction regarding the length of these above three sequences is $|\vec{y}| = n$.

1Producing such a network with degenerate comparators is trivial.
In both tri-section methods, the lowest $|\vec{x}|$ keys and highest $|\vec{z}|$ keys exit the merging network after a delay of at most $\lceil \log(|\vec{x}|) \rceil + 1$ and $\lceil \log(|\vec{z}|) \rceil + 1$ comparators, respectively; i.e. the longest path ending at an output edge $\hat{o}_i$, corresponding to one of the $|\vec{x}|$ smallest keys or to the $|\vec{z}|$ largest keys has at most the above number of comparators. The two best known Batcher merging network (the strictly-cross and strictly-parallel networks) are rather weak in this regard. In the former, every path from input to output goes through exactly $\log(2^n)$ comparators. In the latter the lowest and highest keys come out after only one comparator. For any other output edge, there is a path of length $\log(2^n)$ comparators which ends at this edge.

In our tri-section techniques the $\vec{x}$, $\vec{y}$ and $\vec{z}$ sequences can be sorted s.t. the resulting merging network is a Batcher merging network. It is not clear whether these sequences can be sorted, s.t. the resulting merging network is not a Batcher merging network, while maintaining the above depth restrictions. However, if we drop the restriction of sorting $\vec{x}$ by a network of depth $\lceil \log(|\vec{x}|) \rceil$, we can produce minimal depth merging networks which are not Batcher merging networks, as shown ahead. That is, for any $n$ a power of two greater then 8, there is a minimal depth merging network of width $2^n$ which is not a Batcher merging network; however, we do not know of any such network having useful or interesting properties.

16.1 The asymmetric tri-section method

We now define the $tsec$ operator which partitions a bisorted vector into three sequences $\vec{x}$, $\vec{y}$ and $\vec{z}$ as above. Let $k \in [0, n)$ and let $\langle \vec{a}, \vec{b} \rangle$ be a bisorted vector of width $2n$. Define

$$\vec{a}' \triangleq a[0,k), \quad \vec{a}'' \triangleq a[k,n), \quad \vec{b}' \triangleq \text{rev}(b[0,k)), \quad \vec{b}'' \triangleq \text{rev}(y[k,n])$$

$$\vec{x} \triangleq \text{rev}(\min(\vec{a}', \vec{b}')), \quad \vec{y} \triangleq \max(\vec{a}', \vec{b}') \cdot \min(\vec{a}'', \vec{b}'') \quad \vec{z} \triangleq \text{rev}((\vec{a}'', \vec{b}''))$$

and finally $tsec^{k,n}(\langle \vec{a}, \vec{b} \rangle) \triangleq \langle \vec{x}, \vec{y}, \vec{z} \rangle$

Clearly there is a unique network of depth one that implements the $tsec^{k,n}$ operator. Let $T^{k,n}$ be that network. See Figure 16.1.

Lemma 16.1.1. Let $\langle \vec{a}, \vec{b} \rangle$ be a biosorted vector of width $2n$, let $k \in [0, n)$ and let $\langle \vec{x}, \vec{y}, \vec{z} \rangle = tsec^{k,n}(\langle \vec{a}, \vec{b} \rangle)$. Then

1. $\vec{x} \ll \vec{y} \ll \vec{z}$.
2. $|\vec{x}| = k$, $|\vec{y}| = n$, $|\vec{z}| = n - k$.
3. $\vec{x}$ is ascending-descending, $\vec{z}$ is descending-ascending.
4. $\vec{y}$ is bitonic.
5. If \( n \) is a power of 2, then \( \text{imf}_{T_k,n} \) is a congruent function.

Proof. Items (1) and (3) follow from the generalized 0-1 principle (Lemmas 13.1.8 and 13.1.7). Item (2) follows immediately from the definition above. To show item (5), it is not hard to see that \( \text{imf}_{T_k,n}(i) = n - 1 - i + k \mod n \); hence, \( \text{imf}_{T_k,n} \) is a composition of the order reversing permutation and the cyclic rotation by \( k \) permutation. By Lemma 6.0.16, these two permutations are congruent functions and by Lemma 6.0.15, \( \text{imf}_{T_k,n} \) is a congruent function.

It remains to show item (4). The conventional manner to prove such a claim is via the 0-1 principle; however, this leads to many special cases which need to be verified. In contrast, we use the set of permutation sandwich vectors which supports the set of bisorted vectors. We use Lemma 13.1.8 and the fact that the set of bitonic sequences is complete. Note that a permutation sandwich vector \( \langle \vec{a}, \vec{b} \rangle \) is determined by the key \( a_0 \). There are two (overlapping) cases; either \( a_0 \leq k \) or \( a_0 \geq k \). The two cases are similar and we consider only the first. Let \( j = a_0 \leq k \). Since \( \langle \vec{a}, \vec{b} \rangle \) is a sandwich, \( \vec{b}[0, j] \ll \vec{a} \ll \vec{b}[j, n] \); therefore, the following equations hold:

\[
\begin{align*}
y[0, k - j] &= \text{rev}(b[j, k]), \\
y[k - j, k] &= \max(a[k - j, k), \text{rev}(b[0, j])) = a[k - j, k] \\
y[k, n] &= \min(a[k, n], \text{rev}(b[k, n])) = a[k, n].
\end{align*}
\]

Altogether, \( \vec{y} = \text{rev}(\vec{b}[j, k]) \cdot \vec{a}[k - j, n] \) which is a concatenation of a descending and ascending sequences; i.e. \( \vec{y} \) is bitonic. (The sequence \( \vec{y} \) is not always descending-ascending; in the other case, where \( a_0 \geq k \), the sequence \( \vec{y} \) comes out ascending-descending.)
We henceforth assume that \( n \) is a power of two and show how to sort the sequences provided by the previous lemma. The sequence \( \vec{y} \) is bitonic and of length \( n \); therefore, by Lemma 14.0.6, it can be sorted by (the unique) bitonic sorter of depth \( \log(n) \).

For the sequence \( \vec{x} \) consider the following fact. Any ascending-descending sequence can be expanded into a bitonic sequence of a desired width by adding keys of value \(-\infty\) at the end of the sequence. This implies any wide enough bitonic sorter can be pruned into an ascending-descending sorter of a smaller given width; therefore, \( \vec{x} \) can be sorted by depth \( \lceil \log(\|\vec{x}\|) \rceil \). Note that the fact that \( \vec{x} \) is ascending-descending, rather than bitonic, is critical; there is no way to expand a bitonic sequence into a wider bitonic sequence by adding the keys \(-\infty\) and \(+\infty\) at predefined positions; moreover, when \( n \) is not a power of two, the minimal depth of a width \( n \) bitonic sorter might exceed \( \lceil \log(n) \rceil \) (see Lemma 14.0.7). By symmetry, the sequence \( \vec{z} \) can be sorted in a similar manner. In this merging network the \( \|\vec{x}\| \) lowest keys and the \( \|\vec{z}\| \) highest keys exit the network after a delay of \( \lceil \log(\|\vec{x}\|) \rceil + 1 \) and \( \lceil \log(\|\vec{z}\|) \rceil + 1 \) comparators, respectively. This is useful when \( \|\vec{x}\| \) or \( \|\vec{z}\| \) are small relative to \( n \).

It is not hard to see that the resulting merging network is AMOP and has no degenerate comparators w.r.t. merging (and if it has such degenerate comparators then they can be bypassed, as per Lemma 4.1.2); therefore, by Theorem 10.0.31, our asymmetric tri-section merging network are Batcher merging network.

If we drop the request that the sequence \( \vec{x} \) is sorted by a network of depth \( \lceil \log(\|\vec{x}\|) \rceil \), we can construct a non-degenerate, minimal depth merging network \( M \) which is not a Batcher merging network. This construction is based on the fact that when \( \|\vec{x}\| \) is small w.r.t. \( n \), the sequence \( \vec{x} \) can be sorted in an inefficient manner (by a network of excessive depth) while maintaining the minimal depth of the entire merging network. Since we are free to choose how to sort \( \vec{x} \) with very little restrictions, we may sort \( \vec{x} \) using a non-AMOP network. For \( \|\vec{x}\| = 3 \) such a network \( X \) is depicted in Figure 16.2. Let \( M \) be the merging network constructed by the asymmetric tri-section method as described above except that \( M \) sorts the sequence \( \vec{x} \) using the network \( X \). When \( n > 8 \), \( M \) is a minimal depth merging network. Note that none of the comparators of \( X \) is degenerate in \( M \) (w.r.t. merging); moreover, the network \( X \) is not AMOP; therefore, by Theorem 10.0.31, \( M \) is not a Batcher merging network.

Assume that \( \|\vec{x}\| \) is large and still much smaller than \( n \); than it is possible to sort \( \vec{x} \) by a network which is clearly of no recursive structure of the “divide and conquer” form; two good examples are Knuth’s bubble-sort network and Knuth’s odd-even transposition sort [10, pp 223,241]. This provides a non-degenerate merging network of minimal depth having an arbitrary large subnetwork which posses no recursive structure.
16.2 The symmetric tri-section method

Nakatni et al. [12] introduced an elegant manner to construct a bitonic sorter which is based on the following lemma.

**Lemma 16.2.1 ([12]).** Let \( \vec{x} \) be a bitonic sequence of width \( j \cdot k \) and \( \vec{M} \) be the \( j \times k \) matrix having the \( \vec{x} \) sequence in a row major fashion. Then:

1) Every row in \( \vec{M} \) is a bitonic sequence.
2) Every column in \( \vec{M} \) is a bitonic sequence.

Let \( \vec{M}' \) be derived from \( \vec{M} \) by sorting each column separately. Then:

3) Every row in \( \vec{M}' \) is a bitonic sequence.
4) Every element of \( \vec{M}' \) is smaller or equal to any element of the next row.

Nakatni’s matrix technique to sort bitonic sequences of length \( j \times k \) is based on the above lemma and is composed of three stages:

**Stage 1:** Arrange the bitonic sequence in a \( j \times k \) matrix in a row major fashion.

**Stage 2:** Independently, sort every column by a bitonic sorter.

**Stage 3:** Independently, sort every row by a bitonic sorter.

Following those stages the resulting matrix is sorted in a row major fashion.

Assume henceforth, that \( j \) and \( k \) are a power of two and that the bitonic sorters used in stage (2) and stage (3) are of minimal depth. The depth of the entire network is \( \log(j) + \log(k) \) which is minimal. The technique of Nakatani et al. is new and elegant; however, by Lemma 14.0.6 and the fact that their bitonic sorter is of minimal depth their sorter is the Batcher bitonic sorter; that is, their technique does not produce new bitonic sorter but sheds new light on the Batcher bitonic sorter.
Our\(^2\) symmetric tri-section technique is a specialization of Nakatani’s method to the special case where the input is a bitonic sequence of special form; namely, of the form \(\vec{a} \cdot \text{rev}(\vec{b})\) where \(\langle \vec{a}, \vec{b} \rangle\) is bisorted. Our technique differs from Nakatani’s only in stage (2). We use the fact that a column, after stage (1), is not only bitonic but also bisorted; hence it can be sorted by any merging network. We choose a Batcher merging network in which the lowest and highest keys exit after only one comparator (for example, the strictly-parallel network).

Let \(\vec{x}\) and \(\vec{z}\) be the keys of the first and last row, respectively, after the above Stage 2; let \(N'\) be the depth one network which produces \(\vec{x}\) and \(\vec{z}\). (each comparator of \(N'\) produce a member of these sequences.) Clearly, \(N'\) partitions the keys into the sequences \(\vec{x}, \vec{z}\) and \(\vec{y}\) which is composed of all other keys. Since keys of \(\vec{x}\) are compared from this stage forward only among themselves and the same holds for \(\vec{z}\); this implies that the keys are tri-sected. As promised, the smallest \(k\) and the largest \(k\) keys exit the entire merging networks after a delay of exactly \(1 + \log(k)\) comparators. It is not hard to see that the resulting merging network is both AMOP and non-degenerate; therefore, by Theorem 10.0.31, it is a Batcher merging network.

\(^2\)Batcher and Lizka [3] presented an oblivious merging algorithm based on different ideas but actually similar to the technique we present now. They did not observe that this is a tri-section technique and that it has the practical advantage of producing the \(k\) smallest keys and \(k\) largest keys faster than \(\log(2n)\).
Bibliography


[17] Dalpiaz F. & Rizzi R. Two algorithms, One sorting network. (Personal communication. Available from“Romeo.Rizzi@unitn.it”)