ON THE MESSAGE COMPLEXITY AND BIT COMPLEXITY
OF DISTRIBUTED SORTING

by

O. Gerstel and S. Zaks

Technical Report #373
August 1985

1 The work of this author is supported in part by a Technion
grant No. 121-641.
ABSTRACT

We study the sorting problem for asynchronous distributed systems. Upper and lower bounds are derived for families of graphs, where the complexity measures are the bit complexity and message complexity of the algorithm. We show that for every network with a tree topology with diameter $N$, every sorting algorithm must send at least $\Omega(N^2 \log \frac{L}{N})$ bits where $\{0,1,...,L\}$ is the set of possible initial values, and that for a topology of a $k$-connected graph with $N$ vertices every sorting algorithm must send at least $\Omega\left(\frac{1}{k \log k} N^2 \log \frac{L}{N}\right)$ bits in the worst case. These bounds are shown to be optimal in the worst-case graphs. For the message-complexity measure we show a lower bound of $\Omega(\Delta_G)$ messages for a general graph $G$ ($\Delta_G$ is the sum of distances from all vertices to a median of $G$), and show an algorithm that is optimal for any graph.

1. INTRODUCTION

Distributed algorithms play an important role among current researches. In such studies a network of processors is solving a certain problem, when the only means of communication between them is by sending messages.

In this paper we study the sorting problem for asynchronous distributed systems. In this problem each processor initially has an identifier and an initial value, and at the end the initial values have to be rearranged according to the initial identifiers. We study upper and lower bounds for sorting on families of graphs, where the complexity measures are the bit complexity and message complexity of the algorithm.

For the bit complexity measure we discuss several families of graphs (trees with a diameter $N$ and $k$-connected graphs), and show lower bounds for every algorithm on these families; these bounds are shown to be tight for the worst case members of the families, but not for every member of them. More precisely, we show that for every network with a tree topology of diameter $N$, there exists a distribution of initial values and identifiers for which every sorting algorithm must send at least $\Omega(N^2 \log \frac{L}{N})$ bits,
where $\{0, 1, \ldots, L\}$ is the set of possible initial values; we then show an algorithm that sends no more than $O(N^2\log \frac{L}{N})$ bits for such trees. We show that there exists a $k$-connected graph with $N$ vertices and an initial distribution for which every sorting algorithm must send at least $\Omega(\frac{1}{k \log k} N^2 \log \frac{L}{N})$ bits ($L$ is as above).

As for the message-complexity measure, we note that if messages are not of bounded length, a lot of information can be encoded into one message and sorting can be done in $O(N)$ messages for a tree with $N$ vertices. Therefore we restrict the model so that initial values may not be encoded into one message, and every initial value is sent on its own. We show a lower bound of $\Omega(\Delta_c)$ messages for any algorithm that performs sorting on a graph $G$ ($\Delta_c$ is the sum of distances from all vertices to a median of $G$), and present an algorithm to show that this bound is tight for every graph. By this we improve the existing result, where the algorithm is shown to be optimal for a worst case graph only.

In section 2 we present a formal definition of the model and the problem, and discuss related works. The bit complexity of the sorting problem is discussed in section 3, in section 4 we discuss the message complexity of the sorting problem and show an algorithm that is optimal for every graph. In section 5 we summarize the results and present open problems.

2. PRELIMINARIES

2.1. The Computational Model

A distributed system is comprised of identical processors, connected by communication links. Every processor executes the same algorithm; that specifies the actions to be taken upon receipt of a message; these actions are sending messages and doing local computations. It is assumed that a message arrives at its destination after an unknown but finite time; messages on a link arrive at the same order they were sent.

Initially each processor has some information concerning the problem it is about to solve, in our case an identifier and an initial value. We assume that a non-empty sub-
set of the processors start the algorithm. At the end of the algorithm each processor has a final value.

An execution of the algorithm is a possible scenario of sending and receiving messages, according to the algorithm at each processor. An execution has terminated if all the local algorithms have terminated.

The message complexity (bit complexity) of an algorithm is the maximal number of messages (bits) sent during any possible execution.

Formally, a distributed system is $SYS = (PROC, LINKS, MES, ID, IV, FV)$, where

- $PROC$ is a set of processors,
- $LINKS \subseteq \{(p,q) \mid p \neq q, p,q \in PROC \}$ is a set of links,
- $MES$ is a set of messages,
- $ID$ is a set of identifiers,
- $IV$ is a set of initial values and
- $FV$ is a set of final values,

(all sets are finite).

With a distributed system $SYS$ we associate the undirected graph $G=(V,E)$, where $V = \{v_p \mid p \in PROC\}$ and $E = \{(v_p,v_q) \mid (p,q) \in LINKS\}$; this graph is called the underlying graph of the system.

The initial distribution (or simply the distribution) of identifiers and initial values in a distributed system $SYS$ is a function

$\delta : PROC \rightarrow ID \times IV$.

Define the functions $\delta_{ID}, \delta_{IV}$ in the following way: let $p$ be a processor and let $\delta(p) = (ID_p, IV_p)$, where $ID_p \in ID$, $IV_p \in IV$ then

$\delta_{ID}(p) = ID_p$ and $\delta_{IV}(p) = IV_p$.

Two distributions $\delta_1$ and $\delta_2$ are said to agree on a set of processors $P$ if $\delta_1(p) = \delta_2(p)$ for all $p \in P$.

An algorithm for a distributed system is a program to be executed by each processor. This program specifies the actions to be taken upon receipt of a message;

(*) In this paper $(x,y)$ always denotes an unordered pair.
these actions are sending messages and doing local computations. An algorithm is a bit algorithm if \( MES = \{0,1\} \).

An execution is a sequence of events, each of which is sending or receiving a message at a processor according to its algorithm; it is assumed that no two events happen simultaneously. The set of all possible executions of an algorithm \( A \) in a distributed system \( SYS \) and with initial distribution \( \delta \) is denoted by \( E(A,SYS,\delta) \). When all other parameters are fixed we use \( E(\delta) \) rather than \( E(A,SYS,\delta) \). The result of an algorithm is specified by a function

\[
\varepsilon : PROC \rightarrow FV
\]

that assigns to each processor a final value.

The message-complexity \( m(SYS,A) \) of an algorithm \( A \) on a system \( SYS \) is the maximum number of messages sent during any possible execution of \( A \). The bit-complexity \( b(SYS,A) \) of the algorithm \( A \) on a system \( SYS \) is the maximum number of bits (among all messages) sent during any possible execution of \( A \).

We study the sorting problem. In this problem we assume that all initial identifiers \( \delta_{ID}(p) \) are distinct, while initial values \( \delta_{IV}(p) \) are not necessarily distinct. An algorithm is solving the sorting problem if it rearranges the initial values according to the initial identifiers; in other words, in every execution in \( E(\delta) \) its result function \( \varepsilon \) satisfies

1. The multiset \( \{\varepsilon(p) \mid p \in PROC\} \) is equal to the multiset \( \{\delta_{IV}(p) \mid p \in PROC\} \), and
2. \( \delta_{ID}(p) < \delta_{ID}(q) \) implies \( \varepsilon(p) \leq \varepsilon(q) \) for all \( p \neq q \).

2.2. Related works

Our definition of the model is based on [GLTWZ]. In [GHS] the problem of constructing a minimum spanning tree is discussed, and the problem of constructing a BFS-tree is studied in [G]. in [KRS1] the problem of finding a median in a tree is solved. Our bit complexity discussion is based on [L]. A closely related problem to the sorting problem is ranking, and it is discussed in [KRS2] and in [ZA]. Our sorting algorithm is a modification of the one in [ZA], where the sorting problem is studied for tree networks.
3. BIT COMPLEXITY

3.1. Preliminaries

In this section we discuss the bit-complexity of the sorting problem. We assume that $MES \subset \{0,1\}^*$, and $IV = \{0,1,...,L\}$ for some given $L$. Note that these restrictions do not limit the generality of the computational model, but only help simplify the definitions.

The discussion in this section follows the one in [L], where a slightly different definition of sorting on networks with a ring topology is studied and lower bounds on the message complexity of that problem are derived; we modify that technique to enable us to deal with the bit complexity.

Let $FSP(n) = (\{0,1\} \times \{1,2,...,n\}^*)$, i.e. all finite sequences of pairs $(i,j)$ such that $i \in \{0,1\}$ and $j \in \{1,2,...,n\}$. Each such pair will be called a component.

Lemma 1: There are fewer than $2(2n)^k$ sequences in $FSP(n)$ with at most $k$ components.

Proof: Since each component in a sequence in $FSP(n)$ is one of $2n$ possibilities, it follows that the number of sequences in $FSP(n)$ with at most $k$ components is smaller than

$$1 + 2n + (2n)^2 + ... + (2n)^k \leq 2(2n)^k$$

Lemma 2 ([L]): Let $S$ be a set of $\sigma$ sequences of $FSP(n)$. The total number $\alpha(\sigma,n)$ of components in all the sequences of $S$ satisfies

$$\alpha(\sigma,n) \geq \frac{4\sigma \log(\sigma/10)}{5 \log(2n)}$$

Let $P$ be a subset of $PROC$. The cut $CUT(P)$ is defined by:

$$CUT(P) = \{ (p,q) \in LINKS \mid \hat{p} \in P, q \notin P \};$$

thus, $CUT(P)$ contains all edges that have one end in $P$ and the other end not in $P$. 

Technion - Computer Science Department - Technical Report CS0373 - 1985
Let be a bit-algorithm, the signature $\text{SIG}(e, C)$ of an execution $e \in E(\delta)$ on a cut $C$ is a sequence in $\text{FPS}(n)$, where $n = 2 |C|$. Each component represents a message arrival event, in the following way: assign the labels $1 \ldots n$ to the links of $C$ (two labels per link), each label representing a direction on that link. During the execution $e$, messages are sent and received on links in $C$. The $i$-th component of $\text{SIG}(e, C)$ is $<m, k>$ if the $i$-th message received is a message with contents $m$, and it was sent on a link and a direction implied by label $k$.

**Lemma 3:** Let $D = \{\delta_1, \ldots, \delta_l\}$ be a set of distributions that agree on a set of processors $P$, and let $A$ be a bit algorithm. Let $E(S(D)) = \{e_1, \ldots, e_l\}$ be a set of executions such that $e_k \in E(\delta_k)$. If the executions in $E(S(D))$ have less than $|D|$ different signatures on $\text{CUT}(P)$, then there exist two distributions $\delta_i, \delta_j \in D$ and two executions $e_i \in E(\delta_i), e_j \in E(\delta_j)$ ($e_i, e_j$ not necessarily in $E(S(D))$), for which the final values for processors in $P$ are the same.

**Proof:** By the definition of $E(S(D))$ it is clear that it contains $|D|$ executions, therefore if these executions have less than $|D|$ signatures, there exist two distributions $\delta_i, \delta_j \in D$, with corresponding executions $e_i, e_j \in E(S(D))$ with equal signatures on $\text{CUT}(P)$. We can construct an execution $e'_j \in E(\delta_j)$ for which the events for every processor in $P$ are the same and in the same order as in $e_i$, and hence the final values on these processors are equal to those after $e_i$. The formal details are omitted; intuitively, the processors in $P$ see the same "picture" in the executions $e_i$ and $e'_j$, because they have the same initial configuration, and get the same messages and in the same order from the outside.

3.2. A Lower Bound for Sorting on a Tree

We deal here with systems for which the underlying graph is a tree. Our main result is

**Theorem 1:** Let $A$ be any sorting algorithm on a distributed system for which the
underlying graph is a tree $T$ with diameter $N$ and $IV = \{0, 1, \ldots, L\}$. Then its bit-complexity $b(SYS, A)$ satisfies

$$b(SYS, A) = \Omega(N^2 \log \frac{L}{N})$$

**Proof:** We first show that we do not gain much by allowing messages to be any sequence of bits rather than consisting of just one bit.

**Lemma 4:** For any algorithm $A$, there exists an equivalent bit algorithm $B$ that simulates $A$, and requires exactly twice the number of bits sent by $A$ for every execution.

**Proof:** Given algorithm $A$, we define the bit-algorithm $B$ as follows:

Let $<a_1 a_2 \ldots a_n>$ be a message sent by processor $p$ to processor $q$ in $A$ (where $a_i \in \{0, 1\}$ for $1 \leq i \leq n$). Then in $B$, $p$ will send the following $2n$ bit-messages to $q$: $a_1, 0, a_2, 0, \ldots, a_{n-1}, 0, a_n, 1$, consisting of the $n$ original bits and additional $n$ control bits. Upon receiving the bit $a_1$, $q$ will wait, and receive all bits $a_i$ until the control bit is '1'. After that it will proceed as if it has received the message $<a_1 a_2 \ldots a_n>$. Except for this modification, algorithm $B$ is identical to algorithm $A$. The algorithm $B$ clearly simulates $A$, and sends twice the number of bits sent by $A$ in every execution.

We return now to the proof of the theorem.

We first prove the theorem for the case where the tree $T$ is the path $T_N$ with $N$ processors, namely $T_N = (V, E)$ where

$$V = \{p_1, p_2, \ldots, p_N\}, \text{ and } E = \{ (p_i, p_{i+1}) | 1 \leq i \leq N-1 \}.$$  

For simplicity of the presentation we assume that $N \equiv 0 \mod 3$ and that $L \equiv 0 \mod \frac{N}{3}$; it is easy to extend the discussion for every $N$ and $L$, and we leave out the details.

Define the following sets of processors:

$$P_1 = \{p_i | 1 \leq i \leq \frac{N}{3} \}$$

$$P_2 = \{p_i | \frac{N}{3} + 1 \leq i \leq \frac{2N}{3} \}$$
Let $D$ be the following set of initial distributions: $\delta \in D$ if it satisfies

1. $\delta_{1}(p_i) = i$ for every $i$.
2. $\delta_{IV}(p) = L$ for $p \in P_1 \cup P_2$, and
3. $\delta_{IV}(p_i) \in \{(i - 1 - \frac{2N}{3}) \frac{3L}{N}, \ldots, (i - \frac{2N}{3}) \frac{3L}{N} - 1\}$ for $p_i \in P_3$.

Clearly $|D| = \left(\frac{3L}{N}\right)^\frac{N}{3}$ since for every processor in $P_1$ and $P_2$, the distribution is fixed, and for every processor in $P_3$, every distribution chooses one value out of $\frac{3L}{N}$ possible ones.

From the definition of $D$ it follows that every distribution $\delta \in D$ has different initial values in $P_3$, and by the definition of the sorting problem, will have different final values in $P_1$. Let $S_i = \{p_1, \ldots, p_i\}$. Define the following set of cuts:

$$C = \{\text{CUT}(S_i) \mid \frac{N}{3} + 1 \leq i \leq \frac{2N}{3}\} = \{\langle p_i, p_{i+1} \rangle \mid \frac{N}{3} + 1 \leq i \leq \frac{2N}{3}\},$$

Every cut CUT($S_i$) in $C$ separates the processors in $P_1$ from the processors in $P_3$, furthermore all the distributions in $D$ agree on all processors in $S_i$. Using Lemma 3, if there are less than $|D|$ different signatures on $\text{CUT}(S_i)$ of executions in $\hat{ES}(D)$, then there exist two distributions $\delta_k, \delta_m \in D$ such that there exist executions $e'_k \in E(\delta_k), e'_m \in E(\delta_m)$ for which the final values in $S_i$ are the same, and in particular the values on processors in $P_1$ - a contradiction.

So far we have shown that there exist at least $|D|$ different signatures of executions in $ES(D)$ on $S_i$ for every $\frac{N}{3} + 1 \leq i \leq \frac{2N}{3}$. Every such signature is in $FSP(2)$, as every cut consists of one link, therefore by Lemma 2 in all these signatures there are at least $\alpha(|D|, 2)$ components. Every component corresponds to sending a bit-message. Let $M(c, \delta)$ be the minimal number of bit-messages sent on cut $c$ among the executions in $E(\delta)$. We have

$$\sum_{\delta \in D} M(c, \delta) \geq \alpha(|D|, 2).$$

The cuts in $C$ are disjoint, and therefore the number $K$ of bit-messages sent on the
cuts in $C$ among the executions in $ES(D)$ satisfies

$$K \geq \sum_{c \in C} \sum_{\delta \in D} M(c, \delta).$$

Hence there exists a distribution $\delta_0 \in D$ for which at least $\frac{K}{|D|}$ messages were sent on the cuts in $C$. Therefore, the bit-complexity $b(SYS,A)$ satisfies

$$b(SYS,A) \geq \sum_{c \in C} M(c, \delta_0) \geq \frac{K}{|D|} \geq \frac{\sum_{c \in C} \sum_{\delta \in D} M(c, \delta)}{|D|} \geq \frac{N}{3} a(|D|, 2) \geq \frac{N}{3} \log\left(\frac{3L}{10N}\right)$$

by Lemma 3

$$\geq \frac{4}{5} \log_{4} \frac{N}{3} \log\left(\frac{|D|}{10}\right) \geq \frac{4}{5} \log_{4} \frac{N}{3} \log_{3} \left(\frac{3L}{10N}\right),$$

so we have

$$b(SYS,A) = \Omega(N^2 \log \frac{L}{N}).$$

This completes the proof for the case when the tree $T$ is a path.

The result can be easily extended to any tree $T$ with diameter $N$ by using labelings as above, and adding the vertices that are not on the diameter to $P_2$. We leave out the details of the formal proof of this case.

\[\square\]

**Example:** Let $L = 15$, $N = 9$, $\frac{3L}{N} = 5$, then the appropriate path and the appropriate distributions in $D$ are depicted in Fig. 1. Here $C = \{ (p_4, p_5), (p_5, p_6), (p_6, p_7) \}$.

$$\delta_{ID} = 1 \quad \delta_{ID} = 2 \quad \delta_{ID} = 3 \quad \delta_{ID} = 4 \quad \delta_{ID} = 5 \quad \delta_{ID} = 6 \quad \delta_{ID} = 7 \quad \delta_{ID} = 8 \quad \delta_{ID} = 9$$

$$\delta_{IV_1} = 15 \quad \delta_{IV_1} = 15 \quad \delta_{IV_1} = 15 \quad \delta_{IV_1} = 8 \quad \delta_{IV_1} = 0.4 \quad \delta_{IV_1} = 5.9 \quad \delta_{IV_1} = 10.14$$

The path $T_N$ and the distributions in $D$

**Figure 1.**
3.3. Extending the Result for Graphs

The bit-complexity of sorting on a general graph strongly depends on the structure of the graph; however, a worst case lower bound can be achieved in a way similar to the one for trees. Recall that a graph $G = (V,E)$ is said to be $k$-connected if the subgraph $G' = (V - W, E \cap (V - W) \times (V - W))$ is connected for every $W \subseteq V$ such that $|W| \leq k$ (see [BM]).

**Theorem 2:** For every $k > 1$ there exists a $k$-connected graph with $N$ vertices and an initial distribution with values in the range $\{0, 1, \ldots, L\}$ for which every sorting algorithm has bit complexity $b(S, T, A) = \Omega(\frac{1}{k \log k} N^2 \log \frac{L}{N})$.

**Proof:** Similar to Theorem 1, for details see appendix A.

3.4. A Tight Upper Bound

We have shown an $\Omega(N^2 \log \frac{L}{N})$ bound on the bit-complexity of sorting on trees with diameter $N$ and initial values in the range $\{0, 1, \ldots, L\}$. We now show that this bound is tight by presenting a modification of the algorithm in [ZA] that requires $O(N^2 \log \frac{L}{N})$ in the worst case. We use an idea from [L] for a simple encoding of a set of numbers $\{n_1, n_2, \ldots, n_k\}$ in the range $\{0, 1, \ldots, L\}$ in $O(k \log \frac{L}{k})$ bits rather than $O(k \log L)$ bits. This encoding is done by sorting the numbers into ascending order $(n_{i_1}, n_{i_2}, \ldots, n_{i_k})$, and sending the sequence $n_{i_1}, n_{i_2} - n_{i_1}, n_{i_3} - n_{i_2}, \ldots, n_{i_k} - n_{i_{k-1}}$ (it is easy to see how the original sequence can be derived from this sequence). In [L] it is shown that this method requires $O(k \log \frac{L}{k})$ bits.

The algorithm starts at the leaf processors, which send their identifiers (IDs) and their initial values (IVs) to their father. Every internal processor waits to receive an encoding of the IDs and IVs accumulated at the sons, decodes them, appends its own ID and IV to the corresponding lists and sends an encoding of these lists to its father.
The root processor receives all the IDs and IVs of the tree, sorts them and sends to the sons the list of the correct IVs (now final values [FVs]) according to the IDs received. Each internal processor can now determine its own FV and routes the remaining FVs to its sons.

During the execution of the algorithm two messages are sent on each edge from son to father (containing IDs and IVs), and one message is sent from father to son (containing FVs). For the worst case tree (a path $P_N$), each message contains at most $O(N \log \frac{L}{N})$ bits, and there are $O(N)$ edges, therefore the bit-complexity of the algorithm is $O(N^2 \log \frac{L}{N})$.

4. MESSAGE COMPLEXITY

In this section we discuss the message-complexity of the sorting problem. We assume that IV is any set of initial values, and IV $\subseteq$ MES, with the assumption that initial values are not encoded but sent each on its own. Without such a restriction, sorting can be performed in $O(N)$ messages on a tree with $N$ vertices, and in $O(N \log N + E)$ messages on a graph with $N$ vertices and $E$ edges, using a spanning tree ([GHS]).

4.1. Definitions

The length of the shortest path between the vertices $u$ and $v$ in a graph $G$ is denoted by $d_G(u,v)$. A spanning tree $T$ is called a BFS-tree of vertex $v$ if $d_G(v,w) = d_T(v,w)$ for every vertex $w$.

The weight of a vertex $v \in V$ is

$$\Delta_G(v) = \sum_{u \in V} d_G(u,v).$$

Let

$$\Delta_G = \min_{v \in V} \Delta_G(v).$$

The vertex $m$ for which $\Delta(m) = \Delta_G$ is called a median of the graph.

A pairing $P$ is a partition of the vertices into pairs, leaving at most one vertex free.
(when \(|V|\) is odd), namely,
\[
P = \{ \{a_i, b_i\} \mid i = 1, \ldots, \left\lfloor \frac{|P|}{2} \right\rfloor, a_i, b_i \in V, \ \forall j \neq i; a_i \neq b_i \neq a_j \neq b_j \}.
\]
In the case when \(|V|\) is odd, the vertex unmatched in \(P\) will be denoted by \(\text{free}(P)\).

The \textit{weight} \(T_G(P)\) of a pairing \(P\) in a graph \(G\) is
\[
T_G(P) = \sum_{\{u, v\} \in P} d_G(u, v).
\]

Let
\[
\psi_G = \sum_{\{u, v\} \in V} d_G(u, v).
\]
When \(G\) is understood from the text, we will write \(d(v, w), \Delta(v)\) and \(T(P)\) instead of \(d_G(v, w), \Delta_G(v)\) and \(T_G(P)\), respectively.

4.2. Lower Bound for Message Complexity

4.2.1. Preliminaries

The following relation between the weight of a vertex and the weight of a pairing holds:

\textbf{Lemma 5:} Let \(G = (V, E)\) be a graph, \(v \in V\) and \(P\) a pairing on \(V\). Then
\[
T_G(P) \leq \Delta_G(v)
\]

\textbf{Proof:} Let \(P = \{\{u_1, u_2\}, \{u_3, u_4\}, \ldots, \{u_{2n-1}, u_{2n}\}\}\) (where, \(V = \{u_1, \ldots, u_{2n}\}\) or \(V = \{u_1, \ldots, u_{2n+1}\}\)).

Then
\[
T_G(P) = \sum_{i=1}^{n} d(u_{2i-1}, u_{2i}),
\]
\[
\Delta_G(v) = \sum_{u \in V} d(u, v)
\]

and
\[
T_G(P) = \sum_{i=1}^{n} d(u_{2i-1}, u_{2i}) \leq \sum_{i=1}^{n} [d(u_{2i-1}, v) + d(v, u_{2i})] =
\]
\[
= \sum_{i=1}^{n} d(u_i, v) \leq \sum_{u \in V} d(u, v) = \Delta_G(v)
\]
Corollary 1: $T_G \leq \Delta_G$; Moreover, if $T_G(P) = \Delta_G(v)$ for some vertex $v$ and some pairing $P$, then $T_G(P) = T_G = \Delta_G = \Delta_G(v)$.

The following connection between the graph structure and the worst-case execution of any sorting algorithm holds:

**Lemma 6:** Let $P$ be a pairing on a graph $G$. Then there exists an initial distribution $\delta$ for which every sorting algorithm must use at least $2T_G(P)$ messages during every execution.

**Proof:** Assume that $V = \{v_1, v_2, \ldots, v_n\}$ and consider the distribution $\delta$ defined as follows:

1. For every vertex $v_i$, $\delta_{ID}(v_i) = i$.
2. If $\{v_i, v_j\} \in P$ then $\delta_{IV}(v_i) = j$, $\delta_{IV}(v_j) = i$ and
3. If $|V|$ is odd and $\text{free}(P) = v_k$ then $\delta_{IV}(v_k) = k$.

It is clear from the definition of $\delta$ that every value must travel from a vertex $v$ to a vertex $w$, where $\{v, w\} \in P$, and because every value sent from every vertex is sent on its own, every move of a value from one vertex to an adjacent one requires a message, and therefore every such pair $\{v, w\}$ contributes at least $2d_G(v, w)$ messages to the total number of messages during each execution. This results in a lower bound of $2T_G(P)$, as desired.

From the above lemma it is clear that:

**Corollary 2:** Sorting on a graph $G$ requires at least $\Omega(T_G)$ messages.

In the sequel we shall show that $\Delta_G = 2T_G$ and therefore a lower bound for sorting is $\Omega(\Delta_G)$ for every graph $G$. We also present an algorithm that performs sorting in $O(\Delta_G)$ for every graph $G$ if a median is given, and for trees even without a given median.

### 4.2.2 Sorting on Trees

The problem of sorting on trees was solved by [ZA] and the solution was shown to be optimal in the worst-case tree (for this the property of the center of the tree was used). We show that this algorithm is optimal for every tree, by taking the median of that tree as the root, instead of the center. Finding a median was done by [KRS1] with
\[ \Delta_T \text{ messages. Sorting is achieved by [ZA] with } 4\Delta(v) \text{ messages, if we take } v \text{ as the root of the tree; by this we achieve sorting by } 5\Delta_T \text{ messages for every tree } T. \]

For a tree \( T \) and for a pair of vertices \( v, w \) in \( T \), denote by \( T_{vw} \) the largest subtree including \( w \) but not including \( v \) and denote by \( |T| \) the number of vertices in \( T \).

**Lemma 7 [ZE]:** A vertex \( m \) in \( T \) is a median iff \( |T_{mw}| \leq \frac{1}{2}|T| \) for every neighbor \( w \) of \( m \).

We can now prove:

**Theorem 3:** \( \Delta_T = T_T \) for every tree \( T \).

**Proof:** Let \( m \) be a median of the tree \( T \), we look for a pairing with \( T(P) = \Delta(m) \) (and then \( \Delta_T = T_T \) by corollary 1). We first show:

**Lemma 8:** There exists a pairing \( P \) such that the path between every \( u, w : \{u, w\} \in P \) goes through \( m \), and such that \( \text{free}(P) = m \) in the case when \( |V| \) is odd.

**Proof:** Assume, by contradiction, that for every pairing \( P \) there exist a pair of vertices \( u, w : \{u, w\} \in P \), the path between which does not go through \( m \). Let \( P' \) be such that \( T(P') = T_T \), \( P' \) has a pair \{\( u, w \)\} as above; namely, there exists a neighbor \( k \) of \( m \) such that \( u, w \) are both in \( T_{mk} \). There exists another pair \( \{z, y\} \in P \) such that \( z, y \) are both not in \( T_{mk} \), since otherwise every vertex not in \( T_{mk} \) would have a match in \( T_{mk} \), and so \( |T_{mk}| > |T - T_{mk}| \), which contradicts Lemma 7.

Now construct \( P'' \) from \( P' \) by exchanging \( x \) and \( w \), that is:

\[
P'' = P' \cup \{\{x, u\}, \{y, w\}\} - \{\{x, y\}, \{u, w\}\}
\]

Clearly

\[
T(P'') = T(P') - d(x, y) - d(u, w) + d(x, u) + d(y, w)
\]

but

\[
d(x, u) + d(y, w) = d(x, m) + d(m, u) + d(y, m) + d(m, w) > d(x, y) + d(u, w),
\]

so we have \( T(P'') > T(P') \) which contradicts the maximality of \( P' \).

It remains to show that \( \text{free}(P') = m \) in the case when \( |V| \) is odd, otherwise there exist two vertices \( u, w \neq m \) so that \( \{m, u\} \in P' \) and \( \text{free}(P') = w \). Let

\[
P'' = P' \cup \{\{v, w\}\} - \{\{m, v\}\}
\]

Clearly \( \text{free}(P'') = m \).

**Case 1:** \( v, w \) are not in the same subtree \( T_{mk} \) for every neighbor \( x \) of \( m \), then

---

Technion - Computer Science Department - Technical Report CS0373 - 1985
\[ T(P'') = T(P') - d(m, v) + d(v, w) \text{ and } d(v, w) = d(v, m) + d(m, w) > d(v, m). \]

Therefore \( T(P'') > T(P') \) - a contradiction.

**Case 2:** \( v, w \) are in the same subtree \( T_m \) for some \( z \), choose some pair \( \{a, b\} \in P' \) both not in the same subtree and both not in \( T_m \), and exchange \( w \) and \( a \) and then exchange \( a \) and \( m \) in the above way, that is
\[ P'' = P' \cup \{ \{a, v\}, \{b, w\} \} - \{ \{m, v\}, \{a, b\} \}. \]

In this case
\[ T(P'') = T(P') - d(a, b) - d(m, v) + d(w, b) + d(a, v) \]
and as before we conclude that \( T(P'') > T(P') \) - a contradiction.

It remains to show that \( \Delta(m) = T(P) \):

If the path between a pair of vertices \( u, w \) goes through a vertex \( m \), then
\[ d(u, w) = d(u, m) + d(m, w). \]

The pairing \( P' \) has this property, and therefore,
\[ T(P) = \sum_{u, w : E_P} d(u, w) = \sum_{u, w : E_P} (d(u, m) + d(m, w)) = \sum_{v \in V} d(v, m) = \Delta(m) \]

This completes the proof of the theorem.

From theorem 3 and corollary 2 we conclude:

**Theorem 4:** For every tree \( T \) there exists a distribution \( \delta \) for which every sorting algorithm requires at least \( \Omega(\Delta_T) \) messages.

4.2.3. **Sorting on General Graphs**

In order to perform sorting on a general graph it might be possible to construct an algorithm that uses the topology of the graph in order to minimize the number of messages. We show that it is sufficient to build a special spanning tree, and then use the algorithm in [ZA] on that tree. This algorithm will not only be efficient on worst-case graphs but we show that for every graph there exists a distribution for which the algorithm is optimal up to a constant. We first show:

**Theorem 5:** For every graph \( G \) the following holds:
\[ T_G \leq \Delta_G \leq 2T_G. \]

**Proof:** The fact that \( T_G \leq \Delta_G \) is proved in Lemma 5; in order to prove the other inequality we need some definitions:

A set of pairings \( S \) over some set \( V \) is said to be **full** if

\[ \forall u, w \in V \exists P \in S : \{u, w\} \in P. \]

Two pairings \( P_1, P_2 \) are said to be **distinct** if \( P_1 \cap P_2 = \emptyset \).

**Lemma 9:** For every set \( V \) there exists a full set of distinct pairings \( S \).

**Proof:** This problem is equivalent to the 1-factorization of the complete graph \( K_{2n} \) (for the odd case \( K_{2n+1} \), it is equivalent to the 1-factorization of \( K_{2n+2} \)). There are few solutions to it known in the literature (see [BM]).

Consider a full set \( S \) of distinct pairings. It follows from the definition that

\[ \chi_G = 2 \sum_{P \in S} T(P). \]

since \( T(P) \leq T_G \) for every \( P \in S \), and since \( |S| \leq |V| \), we get:

\[ \chi_G \leq 2|V|T_G. \]

On the other hand,

\[ \chi_G = \sum_{v \in V} \sum_{u \in V \setminus \{v\}} d(v, u) = \sum_{v \in V} \Delta(v), \]

but

\[ \Delta_G \leq \Delta(v) \text{ for all } v \in V. \]

so

\[ \chi_G \geq |V| \Delta_G \]

therefore

\[ |V| \Delta_G \leq \chi_G \leq 2|V|T_G. \]

\[ \Delta_G \leq 2T_G. \]

It is easy to verify that in any tree or a ring with an even number of vertices, \( T_G = \Delta_G \), and that in a complete graph with an odd number of vertices \( K_n \), \( \Delta_G = 2T_G \).

Hence the above two inequalities are the best possible.
From Theorem 5 and corollary 2 we conclude:

**Theorem 4:** For every graph $G$ there exists a distribution $\delta$ for which every sorting algorithm requires at least $\Omega(\Delta_G)$ messages.

4.3. A General Graph Sorting Algorithm:

So far we have shown that for every graph $G$, sorting requires $\Omega(\Delta_G)$ messages; in this section we show that sorting can be done in $O(\Delta_G)$ messages if the median is given.

The algorithm basically consists of two stages. In the first stage a spanning BFS-tree is constructed from a given median $m$, and in the second stage sorting is performed on this tree. We give here an informal description of the algorithm, from which an exact code can be derived.

The BFS algorithm builds a directed tree layer by layer; it consists of expand phases, where messages go down the tree, and synchronization phases, where messages travel up the tree. Initially the tree consists of the single vertex $m$.

An expand phase starts at the root. It sends expand messages to its sons in the tree, that in turn propagate the messages to their sons in the tree, until the messages reach "leaf" processors in the current tree, which in turn send the expand messages to their neighbors in the graph. These processors are now added as new "leaves" to the tree (except for the ones who were already in the tree), as sons of the processors that first sent them these expand messages. This terminates the expand phase.

A synchronization phase starts at the new "leaf" processors towards the root, by sending back messages to their fathers in the tree. At this phase every processor waits until it gets back messages from all its sons and then sends a back message to its father. The phase terminates when the root receives back messages from all its sons, and starts a new expand phase.

The synchronization phase is used to synchronize the next expand phase, and also to inform processors of the growth of the tree in the last expand phase (in particular, to inform the former "leaf" processors which of their neighbors were really added to the tree as their sons, and which were already in the tree before the last expand phase).
its direction during the last expand phase, its father will not send it *expand* messages in the next phases. The BFS algorithm terminates when the root has received information from all its sons, that there has been no growth in their directions.

In the second stage, sorting is performed on the tree constructed in the BFS stage, using the algorithm in [ZA].

The message complexity of the first stage is bounded by \(2\Delta C\) because *expand* (back) message are sent no more than \(d(m,v)\) of each type for every vertex \(v\) (as a matter of fact, for all the vertices with the same father in the tree, only one *expand* (back) message is sent, from (to) the root to (from) the father of these vertices). The message complexity of the second stage is bounded by \(4\Delta C\).

It is easy to improve the algorithm to perform both stages in one stage, that is during the construction of the tree, the initial values are accumulated in the root (these values can reach the root by the back messages). At the end of the construction of the tree, the root will already have all initial values and all that remains is to distribute the values to their destination. This modification reduces the message complexity of the algorithm from \(6\Delta C\) to \(3\Delta C\).

5. SUMMARY AND OPEN PROBLEMS

We have shown lower bounds on the sorting problem, when the complexity measure is either the bit complexity, or the message complexity. For the bit complexity measure we have shown that the lower bound is tight for several graphs, and it remains to show a tight lower bound for every graph. For the message complexity measure we have shown that the lower bound is tight for every graph, given a median of the graph. If we are not given the median, the task of finding it is an expensive one: the direct method of constructing a BFS tree from every vertex and choosing the vertex with a minimum sum of distances costs more than \(|V|\Delta C\) messages. No better algorithm is known to us. One interesting problem is to design a better median finding algorithm.

As pointed out in [KRS1], the simple heuristic method of finding a BFS tree of an arbitrary vertex, and then finding the median of that tree and repeating this process until the BFS tree from the suspected median has the same suspected median as its
median, does not necessarily give the median of the graph. It is however interesting to characterize the graphs for which this method gives the median.

Another interesting question is to characterize families of graphs in which the task of finding a median is an easy task, like those where every vertex is a median (it is clear that these graphs include all edge-transitive graphs, and there are also non edge-transitive graphs that have this property).
REFERENCES


[ZE] B. Zelinka, Medians and peripherals on trees, Arch. Math. (Brno); 1968, pp. 87-95.
Appendix A: Bit Complexity for $k$-connected Graphs

We present the proof of Theorem 2:

**Theorem 2:** There exists a $k$-connected graph with $N$ vertices and an initial distribution with values in the range \(0, 1, \ldots, L\), for which every sorting algorithm has bit complexity $b(SYS, A) = \Omega(\frac{1}{k \log k} N^2 \log \frac{L}{N})$.

**Proof:** For simplicity we assume that $N = 0 \mod k$, (otherwise the last clique $G_{kN}$ [see fig. 2] will include all $N - k \cdot \frac{N}{k}$ remaining processors; the rest of the details are omitted).

Define the graph $G = (V, E)$ as follows:

1. $V = \{p_{ij} \mid 1 \leq i \leq k, 1 \leq j \leq \frac{N}{k}\}$.

2. $(p_{ij}, p_{ik}) \in E$ for all $i, j, k$ such that $1 \leq j \leq \frac{N}{k}$, $1 \leq j \neq k \leq k$ and $1 \leq i \leq k$.

3. $(p_{ij}, p_{j(i+1)}) \in E$ for all $i, j, k$ such that $1 \leq i \leq \frac{N}{k} - 1$, $1 \leq j \leq k$.

The above graph $G$ is depicted in Fig. 2.

![Diagram of the $k$-connected graph $G$](image)

The $k$-connected graph $G$

Figure 2.

We assume that $\frac{N}{k} = 0 \mod 3$, define the following sets of processors of the graph:

- $P_1 = \{p_{ij} \mid 1 \leq i \leq k, 0 \leq j \leq \frac{N}{3k} - 1\}$
- $P_2 = \{p_{ij} \mid 1 \leq i \leq k, \frac{N}{3k} \leq j \leq \frac{2N}{3k} - 1\}$
Let $D$ be the following set of initial distributions: $\delta \in D$ iff:

1. $\delta_{ID}(p_y) = i + (k+1)(j-1)$ for $1 \leq i \leq k$, $1 \leq j \leq \frac{N}{k}$,

2. $\delta_{IV}(p) = L$ for $p \in P_1 \cup P_2$ and

3. $\delta_{IV}(p_y) \in \{ (\delta_{ID}(p_y) - \frac{2N}{3}) \cdot \frac{3L}{N}, \ldots, (\delta_{ID}(p_y) + \frac{2N}{3}) \cdot \frac{3L}{N} \}$ for $p_y \in P_3$.

From the definition of $D$ every $\delta \in D$ has different initial values in $P_3$, and by the definition of the sorting problem, will have different final values in $P_1$. Define the following set of cuts:

$$C_i = \{ (p_{yi}, p_{y(j+1)}) \mid 1 \leq j \leq k \} \text{ for every } i,$$

$$C = \{ C_i \mid \frac{N}{3k} < i \leq \frac{2N}{3k} \}.$$

By similar considerations to those in Theorem 1, it can be shown that a signature on a cut $C_i$ is in $FSP(2k)$, and that $|C_i| = \frac{N}{3k}$, so that at the worst case distribution $\delta_0$, for every bit-algorithm there exists an execution in which at least $b(SYS,A)$ messages are sent, and $b(SYS,A)$ satisfies

$$b(SYS,A) \geq \sum_{C_i \in C} M(C_i, \delta_0) \geq \frac{N}{3k} \cdot \frac{\alpha(\lceil \frac{N}{2k} \rceil, 2k)}{|D|} \geq \frac{4}{5 \log 4k} \cdot \frac{N}{3k} \cdot \frac{N}{3} \cdot \log \left( \frac{3L}{10N} \right) = O\left( \frac{1}{k \log k} N^2 \log \frac{L}{N} \right).$$
Appendix B: A Formal Presentation of the Sorting Algorithm

The following algorithm performs (in phase 2) sorting on a BFS-tree constructed in phase 1. The variables and message used in the algorithm of the first phase are:

**VARIABLES:**

- **NS**: the set of all neighbors in which direction the next BFS phase may expand the tree. Initially NS contains all neighbors of the processor.
- **father**: the neighbor which is the father of the processor in the tree. This is the first neighbor that tried to expand the tree to include the processor.
- **someone**: any processor other than father which attempts to expand the tree to include the processor (which is already in the tree).
- **stop**: the flag returned with a back message from neighbor n.

**MESSAGES:**

- **expand**: the message which is sent in the direction of a neighbor when attempting to expand the tree in its direction.
- **back**: the response to an expand message. If sent with a false flag then the last expand did increase the BFS-tree, if sent with a true then there is no way to expand the tree in the direction of this neighbor, in particular - if the neighbor to which the corresponding expand was sent already belongs to the tree (from another direction).

**A CODE FOR THE ALGORITHM:**

First Phase:

```plaintext
First phase:
for median:
    NS := set of all neighbors;
    while NS ≠ Ø do
        for n in NS do
            SEND("expand") to n;
            for n in NS do begin
                RECEIVE("back", stop);
                if stop = true then NS := NS - {n};
            end;
        end;
end;
```
for other vertices:

\[ NS := \text{set of all neighbors}; \]

RECEIVE("expand") from \textit{father};

\[ NS := NS - \{ \textit{father} \}; \]

SEND("back",false) to \textit{father};

while \( NS \neq \emptyset \) do begin

. . .

RECEIVE("expand") from \textit{someone};

if \textit{someone} \neq \textit{father} then SEND("back",true) to \textit{someone};

else begin

for \( n \) in \( NS \) do

SEND("expand") to \( n \);

for \( n \) in \( NS \) do begin

. . .

RECEIVE("back",\textit{stop}) from \( n \);

if \textit{stop} = \textit{true} then \( NS := NS - \{ n \}; \)

. . .

SEND("back",false) to \textit{father};

end;

. . .

end;

while \textit{true} do begin

RECEIVE("expand") from \textit{someone};

SEND("back",true) to \textit{someone};

end;

\[ \text{Second Phase:} \]

Preform sorting as in [ZA] on the tree constructed by phase 1,

with the median as the root: