

The scheduling and organization of periodic associative computation: Essential networks

Timothy Van Zandt*

Department of Economics, Princeton University, Princeton, NJ 08544-1021, USA
(e-mail: tvz@princeton.edu; Web: www.princeton.edu/~tvz)

Received: 15 October 1994 / Accepted: 6 March 1997

Abstract. This paper defines and characterizes essential decentralized networks for calculating the associative aggregate of one or more cohorts of data. A network is essential if it is not possible to eliminate an instruction or manager and still calculate the aggregate of each cohort. We show that for essential networks, the graphs that depict the operations and data dependencies are trees or forests. These results assist in the characterization of efficient networks.

JEL classification: D83, D23

Key words: Organizations, decentralization, parallel processing

1 Introduction

The agents in organizations such as firms jointly make decisions by processing information about their environment. This view of organizations as information-processing networks has a long history, which includes March and Simon (1958), Simon (1976) and Galbraith (1977) and the economics literature surveyed in Van Zandt (1996a). Such joint information processing has been called decentralized information processing in economics, and it is analogous to parallel processing within a multi-processor computer and to distributed processing in a network of computers. The use of formal models of parallel and distributed computing for studying computation in organizations and economic systems was advocated to economists by Mount and Reiter starting in 1982 (see Mount and Reiter

* This is a part 1 of a revision of a paper entitled "Periodic Parallel Addition," which I wrote while I was a Post-Doctoral MTS at AT&T Bell Laboratories (1988–1989). The support of AT&T Bell Laboratories is gratefully acknowledged. This revision has been supported in part by grant SBR-9223971 from the National Science Foundation and a CORE Research Fellowship. I benefited from discussions with Roy Radner and the extensive comments of two anonymous referees.

1990, 1994, 1996). Researchers in computer science (e.g., Huberman and Hogg 1988; Huberman 1990) and management science (e.g., Carley and Prietula 1994) have also seen the natural interpretation of networks of machines as networks of boundedly rational human agents.

A simple but prevalent and important example of the information processing activities in organizations is associative computation, which includes project selection, aggregation of cost functions, and the additive operations in linear decision rules. Associative operations are the leading class of aggregation operations, and are important in many decision problems. For example, in the resource allocation model in Van Zandt (1996a), although there are other elementary operations, it is the associative aggregation of cost functions whose delay increases with the number of operations and leads to decentralized decision making. As a topic of study by economists, the simplicity of associative computation is also a virtue because it provides a vehicle for economists to familiarize themselves with the basic properties and trade-offs of decentralized information processing.

The characterization of decentralized associative computation as a means to understand information processing in organizations was proposed and developed by Radner (1993). Bolton and Dewatripont (1994) contains further research for a class of computation problems that includes, but is more general than, associative computation. It is also possible to interpret Keren and Levhari (1979) as a model of associative computation.

Radner (1993) considered one model in which the organization must aggregate a single cohort of data, and another model in which it must aggregate cohorts of data that arrive periodically. The purpose of the current paper is to define and characterize *essential networks* for these two models, and for more general assumptions about deterministic arrivals of data.

Essential networks are those for which it is not possible to delete a manager or instruction and still obtain a network that calculates the aggregate of each cohort. It is a minimal requirement for efficient networks, if managers and instructions are costly. To characterize the essential networks, we define a type of data-dependency graph, whose nodes are the instructions. We show that for essential networks, this graph is a tree or a forest, with each connected component corresponding to the instructions for aggregating one of the cohorts. This is due to the natural tree structure of associative operations, and is the reason for the prevalence of hierarchies in the literature on associative computation.

This paper also provides a more formal definition of the computation model than that of Radner (1993). This formal model and the characterization of essential networks are useful for stating and proving results about the properties of efficient networks. They are used in a companion paper, Van Zandt (1998), to characterize a class of efficient networks for the case of periodic arrivals. They are also used in Meagher and Van Zandt (1997) to demonstrate some necessary conditions for efficient networks for the case of a single cohort.

2 Computation problem

Time is discrete ($t = 0, 1, \dots$). The unit of time is called a cycle. There is a fixed set $\mathcal{N} = \{1, \dots, N\}$ of data sources of finite size $N \geq 2$. At the beginning of each cycle $\tau \in \mathcal{T}$, for a set \mathcal{T} of arrival times, an organization receives a cohort $\{X_{1\tau}, \dots, X_{N\tau}\}$ of data containing one item from each data source. The cohort that arrives in cycle τ is called cohort τ .

The data take values in a set \mathcal{X} that is endowed with an associative and commutative binary operation, which we denote by \oplus and call aggregation.¹ The operation might be a complex task such as project selection or aggregation of cost functions, or a simple task such as the addition operations in linear decision rules. For each cohort τ , the organization should calculate $X_{1\tau} \oplus \dots \oplus X_{N\tau}$. The calculations for different cohorts are independent, in the sense that each one uses different data, but they overlap temporally, in the sense that the organization may be processing more than one cohort at any given time.

3 Computation model

The computation model is that of Radner (1993). (See Van Zandt (1996b) for a discussion of alternate models.) The computation is performed by identical managers who synchronously execute instructions, along with an input and an output device. Each manager has (i) an infinite buffer (an in-box) to which messages arrive from the input device and from other managers and (ii) a register that can hold a single item and that is used to aggregate the data. We assume that messages are sent instantly and costlessly and that managers have random access to the messages in their buffers. Therefore, we can imagine that messages are broadcast so that we do not have to specify their recipients. Yet we can determine who reads a message in a network, and hence we can track communication. To model the random access that managers have to their buffers, messages are given unique labels or ID's by which managers identify them.

The alphabet for the instructions consists of (i) the set $\{\text{INPUT, SEND, OUTPUT, LOAD, ADD}\}$ of instruction types, (ii) the set \mathbb{N} of cycles, (iii) the set \mathcal{N} of data sources, (iv) an infinite set \mathbb{M} of manager names (disjoint from \mathbb{N}), (v) an infinite set \mathbb{A} of message ID's, and (vi) the set $\{(\cdot, \cdot)\}$ of connectives. Instructions are strings from this alphabet of the following forms:

$$\begin{array}{lll} \text{INPUT}(t, \tau, n, a) & \text{SEND}(t, m, a) & \text{OUTPUT}(t, m, \tau) \\ \text{LOAD}(t, m, a) & \text{ADD}(t, m, a) & \end{array}$$

where $t \in \mathbb{N}$, $m \in \mathbb{M}$, $a \in \mathbb{A}$, $\tau \in \mathcal{T}$ and $n \in \mathcal{N}$. The argument t is the execution time, m is the manager who executes the instruction, and a is the message ID. Instructions of types INPUT, SEND and OUTPUT are called *messages* and instructions of types LOAD and ADD are called *operations*. $\text{INPUT}(t, \tau, n, a)$ is called an INPUT instruction for item n of cohort τ , and $\text{OUTPUT}(t, m, \tau)$ is called an OUTPUT instruction for cohort τ .

Networks, delay and managerial costs are defined axiomatically, but we pause to give an informal description of the computation in a network. This description presumes that no two INPUT and SEND instructions have the same message ID. As long as no confusion should arise, a message ID a may be treated as if it were the unique INPUT or SEND instruction with that ID, or as if it were the location in a manager's buffer where the value of the message with ID a is stored (e.g., "manager m reads message a ").

The execution of the instructions in each cycle is divided into two phases.

1. In the first instant, messages are executed (sent) with no delay and no cost.
 - INPUT(t, τ, n, a) means that the input device sends $X_{n\tau}$, with message ID a , to the managers.
 - SEND(t, m, a) means that manager m sends the value of her register, with message ID a , to the other managers.
 - OUTPUT(t, m, τ) means that manager m sends the value of her register, identified as the aggregate of cohort τ , to the output device.
2. In the remainder of the cycle, operations are executed (performed). Each manager m can retrieve from her buffer the value of a single message a and, like an adding machine, either
 - store it directly in her register (LOAD(t, m, a)), or
 - aggregate it to the current value in her register (ADD(t, m, a)).

Manager m is said to process message a . If a is an INPUT instruction, the operation is called a *preprocessing* operation, and if a is a SEND, the operation is called a *postprocessing* operation.

Instructions are partially ordered by their execution times and by the fact that a message is executed before an operation that has the same execution time. This is the presumed ordering when we refer, for example, to the next, previous, first or last instruction.

Definition 3.1 A (programmed) network is a pair $\langle \mathcal{M}, \mathcal{I} \rangle$, where \mathcal{M} is a set of managers and \mathcal{I} is a set of instructions, with the following properties:

1. Each manager who executes an instruction in \mathcal{I} is in \mathcal{M} .
2. Each manager performs at most one operation in \mathcal{I} during each cycle.
3. No two INPUT or SEND instructions in \mathcal{I} have the same message ID.
4. If there is an operation in \mathcal{I} at time t with message ID a , then there is an INPUT or SEND in \mathcal{I} at time $t' \leq t$ with message ID a .
5. If a manager executes instructions in \mathcal{I} , then her first instruction is a LOAD.

A *one-shot* (respectively, *periodic*) network means a network in the one-shot (respectively, *periodic*) model.

In a network $\langle \mathcal{M}, \mathcal{I} \rangle$, each instruction $i \in \mathcal{I}$ has a *value* $V(i)$, which formally is a function from the set \mathcal{X}^{NT} of possible realizations of the raw data into \mathcal{X} , and hence is analogous to a random variable. If i is a message with ID a , then we also denote the value of i by $V(a)$. Each instruction i that is not an INPUT instruction has one or two *operands*, which are prior instructions whose values

are the inputs of i . The operands of each instruction and the rules that recursively define the values of each instruction as a function of the values of its operands are the obvious ones and they are listed in Table 1.

Table 1. Operands and values of instructions for a network $\langle \mathcal{M}, \mathcal{I} \rangle$. For a SEND, OUTPUT or ADD instruction i executed by a manager m in cycle t , $L(i)$ denotes the last operation performed by manager m before cycle t .

Instruction i	Operands		Value $V(i)$
	Operation	Message	
INPUT(t, τ, n, a)			$X_{n\tau}$
SEND(t, m, a)	$L(i)$		$V(L(i))$
OUTPUT(t, m, τ)	$L(i)$		$V(L(i))$
LOAD(t, m, a)		a	$V(a)$
ADD(t, m, a)	$L(i)$	a	$V(L(i)) \oplus V(a)$

Definition 3.2 A network $\langle \mathcal{M}, \mathcal{I} \rangle$ is **functional** if and only if, for each $\tau \in \mathcal{T}$, there is a unique OUTPUT instruction $i(\tau)$ for cohort τ , whose value is $\bigoplus_{n \in \mathcal{N}} X_{n\tau}$. The **delay** of cohort $\tau \in \mathcal{T}$ is then $D(\tau) \equiv t(\tau) - \tau$, where $t(\tau)$ is the execution time $i(\tau)$. The **delay** of $\langle \mathcal{M}, \mathcal{I} \rangle$ is $D : \mathcal{T} \rightarrow \mathbb{N}$.

4 Execution graph

The operand relations for a network $\langle \mathcal{M}, \mathcal{I} \rangle$ can be represented by a directed acyclic graph, called the *execution graph* and denoted $\mathcal{E}(\mathcal{M}, \mathcal{I})$, in which the set of nodes is \mathcal{I} and there is an edge from $i_1 \in \mathcal{I}$ to $i_2 \in \mathcal{I}$ if and only if i_1 is an operand of i_2 .² Figure 1 shows three different representations of the execution graph of a one-shot network that processes 4 items with 2 managers.

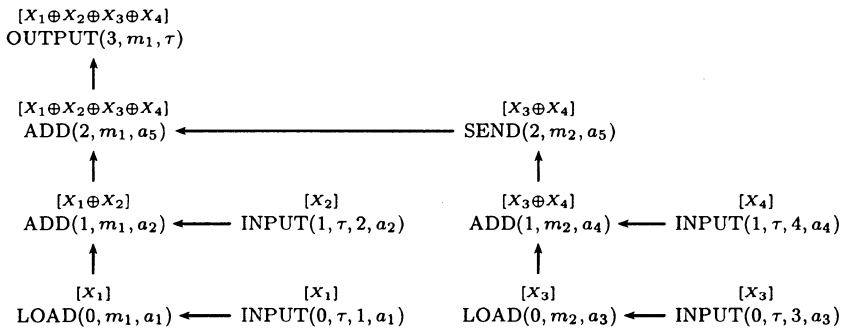
Remark 4.1 The execution graph is similar to what are called data dependence graphs, data flow graphs, precedence graphs or simply DAG's in the scheduling and compiler literatures, except that these would not include the messages nor the specification of execution times and managers for each operation; instead, the edges (which still represent data dependence between operations) would be assigned a communication cost that is incurred when the adjoining operations are not performed by the same processor element. Such graphs are used to represent parallel algorithms in Mount and Reiter (1990) and Reiter (1996).

From Table 1, we can see that a node of $\mathcal{E}(\mathcal{M}, \mathcal{I})$ is a source if and only if it is an INPUT instruction, and we can derive the following:

Lemma 4.1 Let $\langle \mathcal{M}, \mathcal{I} \rangle$ be a network. Let $i \in \mathcal{I}$ be an instruction whose type is not INPUT. Let $\mathcal{P}(i)$ be the set of paths in $\mathcal{E}(\mathcal{M}, \mathcal{I})$ from INPUT instructions that are predecessors of i to i , and, for $p \in \mathcal{P}(i)$, let $n(p)$ and $\tau(p)$ be the data source and cohort, respectively, of the INPUT instruction that is the initial vertex of p . Then $V(i) = \bigoplus_{p \in \mathcal{P}(i)} X_{n(p)\tau(p)}$.

Proof. See Appendix. □

The execution graph, with the value of each instruction in brackets:



Two compact representations of the execution graph:

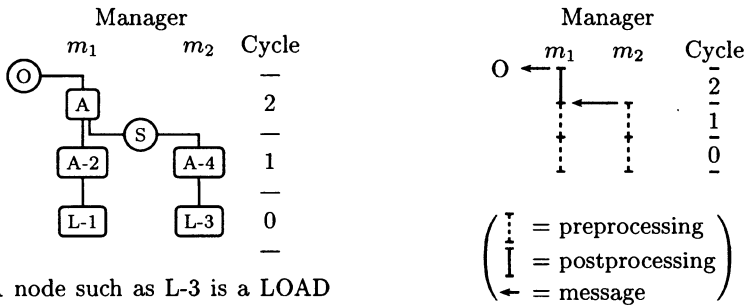


Fig. 1. Three representations of the execution graph of a one-shot network with two managers, m_1 and m_2 , for $N = 4$. Each manager loads one raw item in her register during cycle 0 and aggregates another raw item to her register during cycle 1. At the beginning of cycle 2, manager m_2 sends the value of her register, which is equal to $X_3 \oplus X_4$, to manager m_1 . Then, during cycle 2, manager m_1 aggregates this to the value $X_1 \oplus X_2$ in her register. The value of this postprocessing operation is the aggregate of the data, which manager m_1 sends to the output device at the beginning of cycle 3.

We define another graph that is a static representation of the communication channels implicit in a network.

Definition 4.1 Let $\langle \mathcal{M}, \mathcal{I} \rangle$ be a network. The **communication graph** for $\langle \mathcal{M}, \mathcal{I} \rangle$ is the directed graph in which the set of nodes is $\mathcal{N} \cup \mathcal{M}$ and in which (i) an edge connects a data source $n \in \mathcal{N}$ to a manager $m \in \mathcal{M}$ if and only if \mathcal{M} processes an INPUT message for source n , and (ii) an edge connects manager $m_1 \in \mathcal{M}$ to $m_2 \in \mathcal{M}$ if and only if m_1 sends a message that is processed by m_2 . $\langle \mathcal{M}, \mathcal{I} \rangle$ is **hierarchical** if and only if its communication graph is a tree.

Figure 2 shows the communication graph of the one-shot network in Fig. 1.

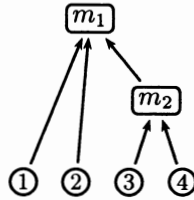


Fig. 2. The communication graph of the one-shot network in Fig. 1.

5 Essential networks

Definition 5.1 defines an *essential* network to be one without superfluous instructions or managers, meaning that it is not possible to delete a manager or an instruction and still obtain a functional network. Figure 3 shows a one-shot network that is not essential.

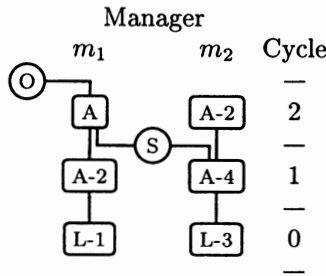


Fig. 3. The execution graph of a non-essential one-shot network with two managers, for $N = 4$. Deleting manager m_2 's last ADD operation yields the network shown in Figure 1, which has the same delay

Definition 5.1 A network $\langle \mathcal{M}, \mathcal{I} \rangle$ is **essential** if it is functional and if, for each $m \in \mathcal{M}$ and $i \in \mathcal{I}$, $\langle \mathcal{M} \setminus \{m\}, \mathcal{I} \rangle$ and $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ are not functional networks.

Our interest in essential networks comes from the characterizations given in Theorem 5.1 and Corollary 5.1 below. These results are interesting because they show how much structure is imposed by the rather weak condition of essentiality. These results are also used in Van Zandt (1998) and Meagher and Van Zandt (1998) to characterize efficient networks.

Whether essentiality is a *necessary* condition for efficiency with respect to delay and computational costs depends on how computational costs are defined. For example, if there is no accounting for the number of instructions in a network or for communication transmission costs, then efficient networks may have irrelevant messages that appear to involve communication between managers but actually do not because the messages are never processed. If managerial costs are measured by the number of managers, then even in efficient networks managers may fill up potential idle time with operations that do not affect the output.³

However, for any functional network $\langle \mathcal{M}, \mathcal{I} \rangle$, we can obtain an essential network with the same delay by eliminating instructions that do not affect the

value of any OUTPUT instruction, and eliminating managers who perform no instructions. (This can be shown using the characterization of functional networks in the next paragraph.) This reduction may reduce, but not increase, managerial costs when these are defined appropriately. Hence, to search for efficient networks one may restrict attention to essential networks. To show that every network is weakly dominated by a network in some class, it is easier to only have to show this for every network that has the properties derived below for essential networks. When constructing a network that dominates another network, one can verify that the constructed network is functional by checking that it satisfies the properties of essential networks. These facts are applied extensively in Van Zandt (1998) and Meagher and Van Zandt (1997).

Let $\langle \mathcal{M}, \mathcal{I} \rangle$ be a network in which there is a unique OUTPUT instruction $i(\tau)$ for each cohort $\tau \in \mathcal{T}$. For $\tau \in \mathcal{T}$, let $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ be the induced subgraph of $\mathcal{E}(\mathcal{M}, \mathcal{I})$ containing $i(\tau)$ and the predecessors of $i(\tau)$. If $\langle \mathcal{M}, \mathcal{I} \rangle$ is functional, then the value of $i(\tau)$ is $\bigoplus_{n \in \mathcal{N}} X_{\tau n}$. Therefore, according to Lemma 4.1, the sources of $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ must be N INPUT instructions, one for each item n in cohort τ , and there must be a unique path from each of these sources to $i(\tau)$. Since also $i(\tau)$ is the unique sink in $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$, $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ is a tree⁴ whose root is $i(\tau)$ and whose leaves are the N INPUT instructions. The following theorem characterizes the essential networks in terms of these trees.

Theorem 5.1 *A network $\langle \mathcal{M}, \mathcal{I} \rangle$ is essential and has delay D if and only if the following hold:*

1. *Each manager $m \in \mathcal{M}$ executes an instruction in \mathcal{I} .*
2. *For each $\tau \in \mathcal{T}$, there is a unique OUTPUT instruction for cohort τ , which is executed in cycle $\tau + D(\tau)$.*
3. *For each $\tau \in \mathcal{T}$, $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ is a tree whose leaves consist of one INPUT instruction for each item of cohort τ .*
4. *$\mathcal{E}(\mathcal{M}, \mathcal{I})$ is a forest whose connected components are $\{\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})\}_{\tau \in \mathcal{T}}$.*

Proof. See Appendix. □

When there is a single cohort, the execution graph is a tree, as is seen, for example, in Fig. 1. Although it is also true that the communication graph in Fig. 2 is a tree, this need not be the case, as is discussed and illustrated in Van Zandt (1998) and Meagher and Van Zandt (1997).

The main new idea in Corollary 5.1 is that in an essential network, each manager executes one or more sequences of instructions, each of which begins with a LOAD, followed by zero or more ADD's, and concludes with a message. Such a sequence is called a task. For example, each manager's instructions in Fig. 1 constitute a task.

Definition 5.2 *Let $\langle \mathcal{M}, \mathcal{I} \rangle$ be an essential network. A **task** is a set $\mathcal{H} \subset \mathcal{I}$ of instructions performed by the same manager $m \in \mathcal{M}$ in different cycles, such that (i) the first instruction in \mathcal{H} is a LOAD, the last is a message and all others are ADD's, and (ii) there is no instruction in $\mathcal{I} \setminus \mathcal{H}$ executed by m after the LOAD and before the message in \mathcal{H} .*

Corollary 5.1 *A network $\langle \mathcal{M}, \mathcal{I} \rangle$ is essential and has delay D if and only if each manager executes an instruction in \mathcal{I} , and there is a partition $\{\mathcal{I}(\tau)\}_{\tau \in \mathcal{T}}$ of \mathcal{I} such that, for each $\tau \in \mathcal{T}$, $\mathcal{I}(\tau)$ has the following properties:*

1. *Each manager's instructions in $\mathcal{I}(\tau)$ are partitioned into tasks.*
2. *$\mathcal{I}(\tau)$ has a single OUTPUT instruction, which is executed in cycle $\tau + D(\tau)$.*
3. *$\mathcal{I}(\tau)$ has a single INPUT instruction for each item in cohort τ and no other INPUT instructions.*
4. *Each INPUT and SEND in $\mathcal{I}(\tau)$ is processed by a single operation in $\mathcal{I}(\tau)$ and by no other operation in \mathcal{I} .*

Proof. See Appendix. □

Appendix: Proofs

Proof of Lemma 4.1. Let $\mathcal{I}^* \subset \mathcal{I}$ be the set of instructions whose types are not INPUT. For $i_1, i_2 \in \mathcal{I}^*$ such that i_2 is an immediate successor of i_1 and for a path p from a source to i_1 , let $\langle p, i_2 \rangle$ be the extension of p to i_2 . For $i \in \mathcal{I}^*$, let $\text{PREC}(i)$ be the set of immediate predecessors of i , and let $\delta(i)$ be the length of the longest path in $\mathcal{P}(i)$.⁵ The proof is by induction on $\delta(i)$.

Suppose $\delta(i) = 1$. Then i 's immediate predecessors are INPUT instructions, which, by Table 1, implies that i is a LOAD with a single immediate predecessor $\text{INPUT}(t, n, \tau, a)$. Then $V(i) = X_{n\tau}$, which is trivially equal to $\bigoplus_{p \in \mathcal{P}(i)} X_{n(p)\tau(p)}$.

Let $\delta^* \geq 1$. Suppose $V(i) = \bigoplus_{p \in \mathcal{P}(i)} X_{n(p)\tau(p)}$ for all $i \in \mathcal{I}^*$ such that $\delta(i) \leq \delta^*$. Let $i \in \mathcal{I}^*$ be such that $\delta(i) = \delta^* + 1$. Then we have

$$V(i) \stackrel{(a)}{=} \bigoplus_{i' \in \text{PREC}(i)} V(i') \stackrel{(b)}{=} \bigoplus_{i' \in \text{PREC}(i)} \left(\bigoplus_{p' \in \mathcal{P}(i')} X_{n(p')\tau(p')} \right) \stackrel{(c)}{=} \bigoplus_{p \in \mathcal{P}(i)} X_{n(p)\tau(p)},$$

as follows. (a) holds according to Table 1. (b) holds because for each $i' \in \text{PREC}(i)$, $\delta(i') \leq \delta^*$. (c) holds because $\{\{\langle p', i \rangle \mid p' \in \mathcal{P}(i')\}\}_{i' \in \text{PREC}(i)}$ is a partition of $\mathcal{P}(i)$ and because $n(\langle p', i \rangle) = n(p')$ and $\tau(\langle p', i \rangle) = \tau(p')$. □

Proof of Theorem 5.1. Suppose $\langle \mathcal{M}, \mathcal{I} \rangle$ is essential and has delay D . We show that Properties 1–4 hold.

Property 1 holds because for any $m \in \mathcal{M}$, $\langle \mathcal{M} \setminus \{m\}, \mathcal{I} \rangle$ is not a functional network. Properties 2 and 3 hold because $\langle \mathcal{M}, \mathcal{I} \rangle$ is functional, as discussed in the paragraph preceding the theorem.

For $\tau \in \mathcal{T}$, let $\mathcal{I}(\tau)$ be the nodes of $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$. Then Property 4 holds if (i) $\{\mathcal{I}(\tau)\}_{\tau \in \mathcal{T}}$ is a partition of \mathcal{I} and (ii) there is no edge in $\mathcal{E}(\mathcal{M}, \mathcal{I})$ connecting a node in $\mathcal{I}(\tau_1)$ to a node in $\mathcal{I}(\tau_2)$ for some $\tau_1 \neq \tau_2$. Observe that (i-a) $\{\mathcal{I}(\tau)\}_{\tau \in \mathcal{T}}$ are disjoint, because the sources that are predecessors in $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ to $i \in \mathcal{I}(\tau_1) \cap \mathcal{I}(\tau_2)$ must be in both $\mathcal{E}_{\tau_1}(\mathcal{M}, \mathcal{I})$ and $\mathcal{E}_{\tau_2}(\mathcal{M}, \mathcal{I})$, whereas Property 3 implies that the sets of leaves of $\mathcal{E}_{\tau_1}(\mathcal{M}, \mathcal{I})$ and $\mathcal{E}_{\tau_2}(\mathcal{M}, \mathcal{I})$ are disjoint for $\tau_1 \neq \tau_2$. Then (ii) must also hold, because otherwise an initial vertex of the edge would belong

to $\mathcal{I}(\tau_1) \cap \mathcal{I}(\tau_2)$. It remains to be shown that (i-b) $\mathcal{I} = \bigcup_{\tau \in \mathcal{T}} \mathcal{I}(\tau)$, i.e., that $\mathcal{I}^* \equiv \mathcal{I} \setminus \bigcup_{\tau \in \mathcal{T}} \mathcal{I}(\tau) = \emptyset$.

For this, we invoke essentiality, showing that if $\mathcal{I}^* \neq \emptyset$, then \mathcal{I}^* contains a superfluous instruction. First we show (Step 1) that if $i \in \mathcal{I}^*$ is such that $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is a network, then $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is functional. Then we show (Step 2) a minor but more tedious detail: That $\mathcal{I}^* \neq \emptyset$ implies that there is $i \in \mathcal{I}^*$ such that $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is a network.

Step 1. An equivalent statement is that if $i \in \mathcal{I}$ is such that $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is a network that is not functional, then $i \notin \mathcal{I}^*$. Let $i \in \mathcal{I}$ be as indicated. Since $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is not functional, there is $\tau \in \mathcal{T}$ such that either $i = i(\tau)$, in which case $i \notin \mathcal{I}^*$, or the value of $i(\tau)$ is not the same in $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ as in $\langle \mathcal{M}, \mathcal{I} \rangle$. In the latter case, there must be a predecessor of $i(\tau)$ in $\mathcal{E}(\mathcal{M}, \mathcal{I})$, i.e., an instruction $i' \in \mathcal{I}(\tau)$, whose immediate predecessors are not the same in $\mathcal{E}(\mathcal{M}, \mathcal{I})$ as in $\mathcal{E}(\mathcal{M}, \mathcal{I} \setminus \{i\})$. According to Table 1, this implies that $i = L(i')$, in which case i is the immediate predecessor of i' in $\mathcal{E}(\mathcal{M}, \mathcal{I} \setminus \{i\})$ and hence $i \in \mathcal{I}(\tau)$ and $i \notin \mathcal{I}^*$.

Step 2. Suppose $\mathcal{I}^* \neq \emptyset$. We consider three exhaustive cases: (A) \mathcal{I}^* contains no operations, (B) \mathcal{I}^* contains an operation that is not a manager's first instruction in \mathcal{I} , and (C) \mathcal{I}^* contains operations, each of which is a manager's first instruction in \mathcal{I} . *Case A:* Let $i \in \mathcal{I}$. Then i is a message. There are no operations in \mathcal{I}^* , and for $\tau \in \mathcal{T}$, there can be no operation in $\mathcal{I}(\tau)$ that processes i because then i would belong to $\mathcal{I}(\tau)$. Therefore, $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ satisfies Property 4 of the definition of a network (Definition 3.1). The remaining properties of a network obviously hold as well. *Case B:* Let $i \in \mathcal{I}^*$ be an operation that is not a manager's first instruction in \mathcal{I} . Then $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ clearly satisfies Properties 1–4 that define a network, and each manager's first operation in $\mathcal{I} \setminus \{i\}$ is her first operation in \mathcal{I} , and hence is a LOAD. *Case C:* Then no manager can have more than one operation in \mathcal{I}^* , and hence there is an operation $i \in \mathcal{I}^*$ that is performed at that same time as or after all other operations in \mathcal{I}^* . Let m be the manager who executes i . Consider two exhaustive subcases: (i) If m has a subsequent instruction i' in \mathcal{I}^* , then i' is a message that cannot be processed by any of the operations in \mathcal{I}^* . As in Case A, $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is a network. (ii) Otherwise, i is m 's only instruction in \mathcal{I}^* . As in Case B, $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ satisfies Properties 1–4 of a network and is a network if m 's first instruction in $\mathcal{I} \setminus \{i\}$, if it exists, is a LOAD. m 's first instruction in $\mathcal{I} \setminus \{i\}$ is her first instruction in $\mathcal{I}(\tau)$ for some $\tau \in \mathcal{T}$. The latter must be a LOAD, because any other instruction would have an immediate predecessor that is an operation performed by the same manager and is also in $\mathcal{I}(\tau)$.

This completes the proof that an essential network with delay D has Properties 1–4 listed in the theorem. Now we prove the converse.

Suppose first that $\langle \mathcal{M}, \mathcal{I} \rangle$ has Property 1, i.e., each manager executes an instruction in \mathcal{I} . Then for each $m \in \mathcal{M}$, $\langle \mathcal{M} \setminus \{m\}, \mathcal{I} \rangle$ is not a network.

Suppose now that $\langle \mathcal{M}, \mathcal{I} \rangle$ has Properties 2–4. From Lemma 4.1, $\langle \mathcal{M}, \mathcal{I} \rangle$ is functional and has delay D . Let $i \in \mathcal{I}$. There is $\tau \in \mathcal{T}$ such that i is a node in

$\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$. We show that $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is not a functional network. (i) If i 's type is INPUT or SEND, then $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ is not a network, because each such message in $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ has a parent in $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ and hence is processed by an operation in \mathcal{I} . (ii) If i is the OUTPUT instruction for cohort τ , then $\langle \mathcal{M}, \mathcal{I} \setminus \{i\} \rangle$ never sends the aggregate of cohort τ to the output device. (iii) If i is an operation, then the message processed by i has no other immediate successor and hence the INPUT instructions that are (weakly) predecessors of the message in $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ are not predecessors of $i(\tau)$ in $\mathcal{E}(\mathcal{M}, \mathcal{I} \setminus \{i\})$. Hence, the value of $i(\tau)$ is not the aggregate of cohort τ . \square

Proof of Corollary 5.1. Suppose $\langle \mathcal{M}, \mathcal{I} \rangle$ is essential. Then each manager executes an instruction in \mathcal{I} . For $\tau \in \mathcal{T}$, let $\mathcal{I}(\tau)$ be the nodes of $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$. It follows directly from Theorem 5.1 that $\{\mathcal{I}(\tau)\}_{\tau \in \mathcal{T}}$ is a partition of \mathcal{I} and that, for $\tau \in \mathcal{T}$, $\mathcal{I}(\tau)$ has Properties 2–4 listed in the corollary. That $\mathcal{I}(\tau)$ has Property 1, i.e., that each manager's instructions in $\mathcal{I}(\tau)$ are partitioned into tasks, is shown below.

Define a *semi-task* for manager m in $\langle \mathcal{M}, \mathcal{I} \rangle$ to be a set of instructions consisting of a LOAD in \mathcal{I} performed by m and all instructions in \mathcal{I} executed by m before m 's next LOAD instruction (if m has one in \mathcal{I}). Since m 's first instruction in \mathcal{I} is a LOAD, the set of m 's semi-tasks is a partition of m 's instructions in \mathcal{I} . If \mathcal{H} is one of m 's semi-tasks, then each instruction in \mathcal{H} after the LOAD is an immediate successor of the previous operation (see Table 1) and, by induction, is a successor of the LOAD. Hence, \mathcal{H} belongs to the same connected component of $\mathcal{E}(\mathcal{M}, \mathcal{I})$, i.e., $\mathcal{H} \subset \mathcal{I}(\tau)$ for some $\tau \in \mathcal{T}$. It follows that for $\tau \in \mathcal{T}$, each manager's instructions in $\mathcal{I}(\tau)$ are partitioned into semi-tasks. We conclude by showing that each semi-task is a task. That is, each semi-task \mathcal{H} contains a single-message, which is the last instruction in \mathcal{H} . Let $i \in \mathcal{H}$ be a message. If \mathcal{H} contains other instructions executed at the same time as or after i , then either there is $i' \in \mathcal{H}$ that is the first operation in \mathcal{H} executed after i or there is a message in $i' \in \mathcal{H}$, with $i' \neq i$, executed at the same time as or after i and there are no operations in \mathcal{H} executed after i . In both cases, $L(i) = L(i')$ and hence i and i' are immediate successors of the same operation. This violates the assumption that each $\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$ is a tree. Hence \mathcal{H} can have at most one message which is the last instruction in \mathcal{H} . Because only OUTPUT instructions are sinks in $\mathcal{E}(\mathcal{M}, \mathcal{I})$, each operation must have an immediate successor. Therefore, the last operation i in \mathcal{H} must be followed by a message that is executed before the manager's subsequent LOAD (if she has one), and this message is in \mathcal{H} .

For the converse, we need to show (by Theorem 5.1) that if the partition $\{\mathcal{I}(\tau)\}_{\tau \in \mathcal{T}}$ has Properties 1–4, then for each $\tau \in \mathcal{T}$, the induced subgraph of $\mathcal{E}(\mathcal{M}, \mathcal{I})$ whose nodes are $\mathcal{I}(\tau)$ is a tree, and for $\tau_1, \tau_2 \in \mathcal{T}$ such that $\tau_1 \neq \tau_2$ and for $i_1 \in \mathcal{I}(\tau_1)$ and $i_2 \in \mathcal{I}(\tau_2)$, there is no edge from i_1 to i_2 . Since $\mathcal{E}(\mathcal{M}, \mathcal{I})$ is acyclic and each OUTPUT instruction is a sink, it suffices to show that every other instruction $i \in \mathcal{I}(\tau)$ has a unique immediate successor in $\mathcal{E}(\mathcal{M}, \mathcal{I})$, which is also in $\mathcal{I}(\tau)$. (i) If i 's type is INPUT or SEND, then Property 4 implies that i 's unique immediate successor is the single operation in $\mathcal{I}(\tau)$ that processes i . (ii) If i 's type is LOAD or ADD, then let m be the manager who executes i and

let \mathcal{H} be the task containing i . Property 1 implies that m 's next instruction is an ADD or message $i' \in \mathcal{H} \subset \mathcal{I}(\tau)$, which is thus an immediate successor of i , and i does not have another immediate successor because if i' is a message, then either i' is m 's last instruction or m 's next instruction after i' is a LOAD. \square

Endnotes

1. It will be apparent that commutativity is not important to the results in this paper, but gives us one less thing to keep track of.
2. This digraph is acyclic because each edge in $\mathcal{E}(\mathcal{M}, \mathcal{I})$ connects a message sent in cycle t_1 to an operation performed in cycle $t_2 \geq t_1$, or an operation performed in cycle t_1 to a message or an ADD instruction in cycle $t_2 > t_1$.
3. For example, suppose that $N = 4$ and a cohort arrives every 3 cycles. Two managers m_1 and m_2 can process each cohort in the way that is shown in Fig. 1, with a delay of 3. Manager m_1 is then always busy, but manager m_2 is idle every third cycle. The managers can instead process each cohort in the way that is shown in Fig. 3. Manager m_2 is now always busy, but the extra operation does not increase the number of managers in the network.
4. Any finite directed graph with a unique sink and a unique path from each source to the sink is a tree.
5. Which is finite for the reason given in Endnote 2.

List of symbols

Cohorts

t	Index for cycles ($t = 0, 1, \dots$).
\mathcal{T}	Set of cycles in which cohorts arrive.
τ	Index for cohorts ($\tau \in \mathcal{T}$).
T	Number of cycles between cohorts in periodic mode.
N	Number of items per cohort.
\mathcal{N}	Set of data sources ($\mathcal{N} = \{1, \dots, N\}$).
n	Index for items in a cohort ($n \in \mathcal{N}$).

Networks

\mathcal{M}	Set of managers in a network.
m	A manager in a network.
\mathcal{I}	Set of instructions in a network.
i	An instruction in a network.

Values

χ	Set from which data are drawn.
X_m	Value of item n in cohort τ .
$V(i)$	Value of instruction i .
$L(i)$	Previous operation performed by the manager who executes i .

Execution graphs

$\mathcal{E}(\mathcal{M}, \mathcal{I})$	Execution graph of network $\langle \mathcal{M}, \mathcal{I} \rangle$.
$\mathcal{E}_\tau(\mathcal{M}, \mathcal{I})$	Execution subgraph for cohort τ .
$\mathcal{I}(\tau)$	Instructions in $\mathcal{E}(\tau)$.
\mathcal{H}	Task.

Performance

D	Delay of a network or ($D : \mathcal{T} \rightarrow \mathbb{N}$).
M	Number of managers in a network.

References

- Bolton, P., Dewatripont, M. (1994) The firm as a communication network. *Qu. J. Econ.* 109: 809–839
- Carley, K. M., Prietula, M. J. (eds) (1994) *Computational Organization Theory*. Lawrence Erlbaum, Hillsdale, NJ
- Galbraith, J. (1977) *Organization Design*. Addison-Wesley, Reading, MA
- Huberman, B. A. (1990) The performance of cooperative processes. *Physica* 42: 38–47
- Huberman, B. A., Hogg, T. (1988) The behavior of computational ecologies. In: B. A. Huberman (ed.) *The Ecology of Computation*. North Holland, Amsterdam
- Keren, M., Levhari, D. (1979) The optimum span of control in a pure hierarchy. *Manag. Sci.* 11: 1162–1172
- March, J. G., Simon, H. A. (1958) *Organizations*. Wiley, New York
- Meagher, K., Van Zandt, T. (1997) Managerial costs for one-shot decentralized information processing. Australian National University and Princeton University
- Mount, K., Reiter, S. (1990) A model of computing with human agents. The Center for Mathematical Studies in Economics and Management Science, Discussion Paper No. 890, Northwestern University, Evanston, Illinois
- Mount, K. R., Reiter, S. (1994) On modeling computing with human agents. In: M. Majumdar (ed.) *Organizations with Incomplete Information*. Cambridge University Press, Cambridge
- Mount, K. R., Reiter, S. (1996) A lower bound on computational complexity given by revelation mechanisms. *Econ. Theory* 7: 237–266
- Radner, R. (1993) The organization of decentralized information processing. *Econometrica* 62: 1109–1146
- Reiter, S. (1996) *Coordination and the structure of firms*. Northwestern University
- Simon, H. A. (1976) *Administrative Behavior: A Study of Decision-Making Processes in Administrative Organizations*. Free Press, New York
- Van Zandt, T. (1996a) Decentralized information processing in the theory of organizations. In: M. Sertel (ed.) *Contemporary Economic Development Reviewed, Volume 4: The Enterprise and its Environment*. MacMillan, London
- Van Zandt, T. (1996b) Organizations with an endogenous number of information processing agents. In: M. Majumdar (ed.) *Organizations with Incomplete Information*. Cambridge University Press, Cambridge
- Van Zandt, T. (1998) The scheduling and organization of periodic associative computation: Efficient networks. *Rev. Econ. Design* 3 (2)