

A Modular Network Model of Bounded Rationality

Kenneth R. Mount and Stanley Reiter

Introduction

Many phenomena of economic behavior and economic institutions that are still not well understood might be better analyzed via theories that take into account limitations on the ability of humans to calculate and reason. It has been thought since the 1930s that limitations on information processing capabilities of individuals are fundamental to the existence, structure, and functioning of such economic organizations as firms (see, e.g., Kaldor 1934). If there were no restrictions on the ability of human beings to process information (i.e., to observe, communicate, and compute) then there would be no need for multi-person administrative organizations. Physical and technological limitations require that production involve sharing of tasks among many individuals. The efficiencies resulting from division of labor and specialization create a need for coordination of effort in production. In the absence of information processing limitations, decision making would not in itself require multi-person information processing. The complex internal structures of firms and other administrative organizations cannot be understood without recognizing that information processing tasks therein are distributed among individuals.

The distribution of information which arises in part as the result of limitations on human information processing also contributes to incentive problems, because it can create sources of private information. The analysis of incentives is usually done in game models, typically under the assumption that players are fully rational individuals. This kind of model has often led to new insights into strategic situations that were previously not well understood. If this were always or even generally the case then there would be little pressure to complicate the analysis by introducing considerations of bounded rationality. But often, indeed typically, solutions of repeated games of imperfect information consist of very

large sets of equilibria in which any possible outcome is an equilibrium (the folk theorem). Rationalizing equilibria often involves complex strategic reasoning by the players. The combination of unsatisfactory results and perhaps unreasonable assumptions has led some game theorists to introduce limitations on the ability of a player to figure out his or her strategy (i.e., limitations on the rationality of players) and thereby reduce the size of the set of equilibria.¹

Simon (1972, 1987) has advocated that economic theory take into account the boundedness of human rationality. This has stimulated explorations of organization via behavioral models rather than the rational behavior models based on optimization that are the norm in economic theory. On the other hand, Coase's (1937) hypothesis – that economic institutions are to be understood as optimal adaptations to constraints that limit the possibilities of contracting and exchange – remains the way most economists think about institutions and behavior. The transactions costs approach to institutions, specifically the firm, is in the spirit of this kind of theory, though it is not expressed in a formal model (cf. Williamson 1975).

To carry out the program indicated by Coase's principle, a theory of institutions should be one in which an institution or organization emerges as a solution of a constrained optimization problem. For this it is necessary to have a formal model of information processing, including computing, in which the relevant limitations can be expressed and their consequences analyzed. Information processing necessary to achieve coordinated economic action includes observation, communication, and computing. Communication among economic agents who know or can observe different parameters of the environment has been studied formally in the literature on decentralized mechanisms, starting with Arrow and Hurwicz (1959) and Hurwicz (1960), and in the literature on team theory. The literature on message-space theory in economics is surveyed in Marschak (1986) and in Hurwicz (1986). The same problem has been studied in computer science by Abelson (1980) and in the discrete case by Yao (1979) under the name "communication complexity." In contrast to the literature on communication, computation has received much less attention in economic theory.

The question is: How does one formalize limitations on rationality? A natural first step is to restrict attention to systematic reasoning, such as algorithmic processes as distinct from creative processes or leaps of intu-

¹ It should be noted that nonuniqueness of equilibrium is not always a bad thing. Models with unique solutions are not capable of explaining why different institutions or organizational forms coexist in the same environments. Matsuyama (1993) has emphasized this point.

itive thought. Having done that, it is natural to turn to computer science for guidance. The Turing machine model – and its finite memory version, the finite state sequential machine – or finite automaton are basic representations of algorithmic processes. The finite automaton model is perhaps the one that has received the most attention in economics, and has been particularly well-studied in game theory. Economists have applied the finite automaton model in a number of attempts to analyze the complexity of decision making and the functioning of economic organizations (cf. Futia 1977). The more recent applications of that model in game theory, especially in repeated games, seem to have been motivated by the problem of too many equilibria (Aumann 1981). Because strategies in a (repeated) game are functions from information to actions, restricting the number of states available to a player may restrict the set of strategies that are effectively available to that player, and therefore may restrict the set of equilibria of the game (see Neyman 1985, Abreu and Rubinstein 1988, and Rubinstein 1979, 1986).

Kalai and Stanford (1988) introduced a concept of complexity of a repeated game strategy which is defined independently of the concept of automaton. This concept, which classifies strategies into those whose complexity is n for each natural number n , permits restriction of the rationality of a player to be expressed by the condition that the only strategies available to a player are those whose complexity is at most n .²

The explorations in game theory using finite automata to model players captures some aspects of the complexity or the computational difficulty of a function or problem, but it seems that other aspects of computational difficulty that appear to be particularly important when human beings are doing the thinking or problem solving are not convincingly dealt with. One difficulty seems to be that the models involve a depth of reduction so fine that it seems unconvincing when applied to computations performed by human beings. Some investigators have turned to the “perceptron” (Minsky and Papert 1988), a kind of neural network (Hopfield 1982), as an alternative way of modeling computational limitations of players (see, e.g., Rubinstein 1993, Cho 1993a,b, and Cho and Li 1993).

Furthermore, another difficulty with applying the finite state automaton model more generally in economics is that the functions computed by them are discrete functions; that is, the domain and range of such a function are both discrete sets. This corresponds to the fact that the

² It turns out that the Kalai–Stanford complexity of a strategy is equal to the smallest number of states of a finite automaton that can compute that strategy.

inputs to a sequential machine are strings of symbols from a finite alphabet, such as $\{0, 1\}$, and so are the outputs. This poses no special difficulty in, for example, repeated games in which the stage game (the game being repeated) is a matrix game, but models in other economic settings commonly include functions whose domains and ranges are continua – for example, decision rules that describe behavior of consumers, whether or not strategic. The finite state sequential machine model cannot be applied directly to such functions.

The modular network model of computing is intended to permit analysis of computations performed by human beings (with or without the assistance of computing machines), to be applicable in standard economic models, to allow limitations on computing capabilities to be expressed formally, and to give some degree of control over the level of reduction of analysis required. This model was first proposed in Mount and Reiter (1982). Analysis of the model and its connections with standard models of computing are developed further in Mount and Reiter (1990), and some connections with models that study communication are examined in Mount and Reiter (1993). The modular network model is based on the neural network model of McCulloch and Pitts (1943) and Arbib (1969), and is an extension of that model in several respects. First, it permits computation with real numbers. Second, the set of elementary operations – which in the McCulloch–Pitts model are Boolean functions, including functions with thresholds – are permitted to be vector-valued functions of real variables. The set of elementary operations (functions) is a primitive of the model; that is, it can be specified by the modeler to suit the application in hand. The same is true of the topology of the network. Therefore, the level of reduction of analysis is controlled by the modeler in each application. These features facilitate the application of the model to human agents and to economic models. They also provide formal entities with which to express limitations on computing powers of individuals or other agents. Being a network model, the possibilities of parallel computation are readily expressed, and so is the dispersion of information. In economic terms the latter refers to the dispersion of information among agents (private information) or, in computing terms, to distributed memory.

When the alphabet is finite and the set of elementary operations consists of Boolean functions, the modular network model becomes equivalent to the finite automaton model in the sense that, for every such modular network, there is an equivalent finite automaton and vice versa. Furthermore, the complexity of a (continuous) function (of real variables) as measured in the modular network model (explained in what follows) is the limit of a sequence of complexities of finite approxima-

tions to that function as measured by finite (McCulloch–Pitts) networks, which, as we have said, are equivalent to finite automata.

A modular network is a model of the computation of a function (i.e., of an algorithm for computing a particular function) without regard to how that computation is further organized. In order to apply the model in the context of multi-person organizations, it is necessary to introduce additional structure. This is done by explicitly introducing computational resources to carry out the required computations, each computational agent being characterized by her set of elementary operations (computational capabilities) and by restrictions on the dispersion of initial information (who may observe what input variables or, in economic terms, environmental parameters). In that model, the problem addressed is how best to organize economic activity in a given environment. For example, given the specification of technological possibilities, in what organizational units (“firms”) should production be organized, and how big should they be? This analysis is very briefly summarized in the last section of this paper (see also Reiter 1995).

The aim of this chapter is to provide an informal, intuitive presentation of the modular network model, emphasizing motivation and basic ideas. The exposition is aimed at communicating the model and the ideas embodied in it with a minimum of technical apparatus. We focus here on the use of modular networks to model computations carried out by a human being, illustrating the use of the model by examples. Formal definitions, theorems, and proofs can be found in Mount and Reiter (1982, 1993). Its use in a theory of the structure of firms can be found in Reiter (1995).

The recent work of Radner and of Van Zandt explores information processing within administrative organizations (firms). In Radner (1993), Radner and Van Zandt (1992), and Van Zandt and Radner (1995), the model of computing is one in which the elementary operations consist of binary associative operations on real numbers, such as addition of two numbers. The computational task facing the firm is to evaluate a given function at a given value of its multi-dimensional argument, using the processors available. The inputs to the function to be computed are available to any processor without restriction other than its capacity. In these features the model is a modular network model. However, they consider a problem in which a sequence of function evaluations must be made over time; in each of a succession of periods, a new “cohort” of values of the arguments of the function to be evaluated arrives at the input nodes of the network. The question addressed is how best to organize the sequence of computations when the costs of information processing depend on the time it takes to carry out the computation and the number

of processors used. They also study returns to scale in such computations in relation to the problem of firm size and structure. Van Zandt (1995) has studied models in which the elementary operations are those carried out by simple computers called PRAMS.

Computing over the real numbers makes possible the use of classical mathematics. The modular network model and its analysis presented here (and in the references given) is related to certain classical problems in mathematics. In particular, the modular network model is related to Hilbert's thirteenth problem, and to the analysis of superpositions of functions carried out by Kolmogorov and Arnold. These connections are discussed in Section 5 (see also Blum, Shub, and Smale 1989).

There are nine sections in this chapter. Section 1 consists of a brief sketch of the modular network model of computing. Section 2 discusses the first of two examples of computing in which both humans and machines are elements of a network; Section 3 is the second of the examples in which both machines and humans are involved in a computation. Section 4 introduces some simplifying assumptions that are used in Sections 5 and 6. Section 5 discusses the connection that the definition of complexity arising from the network model has with the mathematical studies of superposition (or function composition) and includes a statement of the Leontief–Abelson theorem. In some special circumstances one can determine uniquely, to within a special equivalence, a network that computes a given function in the least possible time. Section 6 gives an example of the construction of such a network for a special function of eight variables, and Section 7 discusses the selection of the modules of the network of Section 6 as the solutions of a system of equations. In Section 8 we analyze the complexity of computing the equilibrium price for an Edgeworth box economy. Section 9 concludes with a brief account of the application of the model of computing to a theory of the structure of organizations that process information for decision making.

1 The Modular Network Model of Computing

A modular network is specified by a set of *elementary operations*, and a *directed graph* (digraph) that shows constraints that the algorithm imposes on the order (partial) in which elementary operations can be performed. Some operations can be carried out in parallel, whereas others must follow a particular order. Viewed in another way, a *module* (embodying an elementary function) can be visualized as a black box with possibly many input lines and one output line, taking one unit of time to compute its output from its inputs. A *modular network*

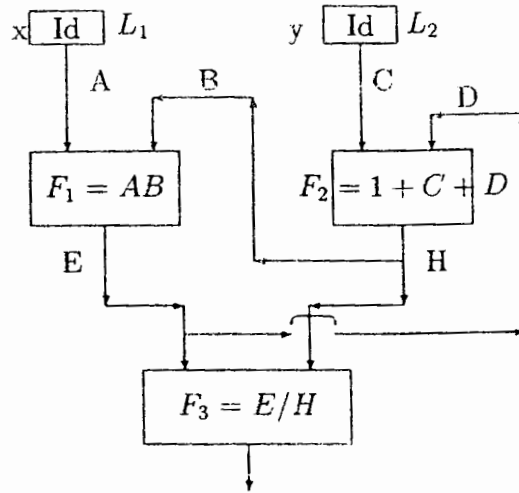


Figure 1

consists of modules wired together subject to the condition that each input wire of a module be connected to at most one output wire of a module.

An example is useful. The diagram in Figure 1 represents a (2,1)-network \mathcal{C} . The class $\mathcal{F}_{\mathcal{C}}$ of functions used in the network consists of four functions of the two real variables A and B :

$$\mathcal{F}_{\mathcal{C}} = \{A + B, AB, A/B, \text{identity function}\}.$$

Each vertex of the digraph that represents \mathcal{C} is denoted by a box with a label that indicates the function assigned to that vertex. The vertices are labeled with upper-case letters (rather than lower-case) identifying the modules; L_1 and L_2 are the input vertices of the network, while the output vertex of the network is labeled F_3 . Each arc of the digraph is labeled by a letter. We use the same labeling for a variable and for the arc that transmits the value from the module that outputs that variable to the one that receives it as an input (this is represented more precisely in Section 6).

We consider next how the $\mathcal{F}_{\mathcal{C}}$ -network \mathcal{C} computes. We have assumed that the output of a module f appears one unit of time after it receives its inputs. The *state of an \mathcal{F} -network* is an array whose entries are the states of the modules of the network in some prescribed order. We assume that the network is initially in some fixed state σ . A network with s input lines that is in state σ' acts on each s -tuple placed on the s input lines. If an s -tuple of values is placed on the network input lines, the

Table 1. Table of States

	L_1	L_2	F_1	F_2	F_3
σ	0	0	0	1	0
t					
0	x	y	0	1	0
1	x	y	x	$(1+y)$	0
2	x	y	$x(1+y)$	$(1+x+y)$	$x/(1+y)$
3	x	y	$x(1+x+y)$	$(1+x)(1+y)$	$\frac{x(1+y)}{(1+x+y)}$
4	x	y	$x(1+x)(1+y)$	$(1+y+x(1+x+y))$	$\frac{x(1+x+y)}{(1+x)(1+y)}$
5	x	y	$x(1+x+y+x^2+xy)$	$(1+y)(1+x+x^2)$	$\frac{x(1+x)(1+y)}{(1+y+x(1+x+y))}$

network will undergo a sequence of changes of state over time. (We assume that if the s -tuple of values on the input lines of the network is changed, the network returns to the fixed initial state σ for the start of the new computation.) As long as the values of the s -tuple on the network input lines remains unchanged, the values produced by the network at the network output vertices at the end of any interval of time are functions of the s -tuple on the network input lines.

Let the initial state in Figure 1 be σ . Assume that, in the initial state σ , the vertices L_1, L_2, F_1, F_2, F_3 have the values 0,0,0,1,0, respectively. We represent the initial state σ by the row matrix (00010). Table 1 shows the sequence of changes of state over time as the network \mathcal{C} computes.

Table 1 can be read as follows. The entry 0 in the column labeled F_1 and the row σ is the state of F_1 in the initial state σ of the network. The second row of Table 1 indicates the new state of the network at time $t = 0$. At that time, the input lines of the network are changed to the state in which input vertex L_1 has state x and input vertex L_2 has state y ; that is, the values x and y are placed on the respective input lines at time 0. During the period from $t = 0$ to $t = 1$, the state of the network changes to a new state in which the state of F_1 is the value of the module $F_1(A, B) = AB$. Because, at $t = 0$, the line A has the value x and B has the value 1 (which is the initial state of the vertex F_2), it follows that F_1 changes to the state x at $t = 1$.

After four units of time (i.e., when $t = 4$, starting from the time that x and y were first placed on the network input lines), the output line F_3 , corresponding to the module F_3 , carries the value

$$h(x, y) = \frac{x(1+x+y)}{(1+x)(1+y)},$$

and $t = 4$ is the earliest time at which this value appears on the output line of the network. The network \mathcal{C} is said to compute the function h in four units of time.

Generally, a modular network \mathcal{C} is said to *compute a function F in time t from initial state σ* if, for each sequence of values (a_1, \dots, a_p) assigned constantly to the network for t units of time, the value on the output lines of the network at time t is the function value $F(a_1, \dots, a_p)$. Usually one is interested in the least time in which a network can compute a function.

It is clear that the least time needed to compute F depends both on the class of elementary operations allowed in the network and the topology of the digraph that represents the network.

As we have indicated, a module represents an elementary computation. The class of elementary functions is a *primitive* of the modular network model. That is, it plays the role of an undefined term in an axiomatic system, such as the term "commodity" in a general equilibrium system, say, as constructed by Debreu (1959). Thus, the set of elementary operations can be different in different applications of the model. For example, what is considered to be elementary may vary with the available means for computing. In some circumstances the basic arithmetic and logical operations may be taken to be the only elementary operations. In other circumstances – say, when the computing is to be done by a person equipped with a personal computer and a program for finding the roots of a polynomial of given degree p in n variables – then a user of the model might want to consider finding roots of such polynomials an elementary operation.

We may want to consider a class of algorithms rather than a particular one, for example, all algorithms corresponding to a neural network with one hidden layer and a given number of input modules. This is expressed by a constraint on the topology of the digraph. We also include the possibility that the functions assigned to the vertices of the digraph may be restricted. For example, in some cases the digraph is restricted to be a tree with the function assigned to the root required to be linear even when the set of elementary operations is not confined to linear modules. Another example is the set of restrictions that define modular:

networks that are simple perceptrons. These restrictions include both the topology of the network and the assignment of modules to vertices of the digraph. Thus, the structure of the allowable class of networks is also a primitive.

That the set of elementary functions is a primitive of the model gives the modeler control over the level of resolution of analyses done with the model. Once the set of elementary operations is specified, analysis of a function requires reduction of the computation to the level of elementary operations. For example, if the elementary operations are the basic logical and arithmetic operations on $\{0, 1\}$, then an algorithm for computation of a function must be expressed as a concatenation, perhaps with loops and branches, of binary operations. When computations are carried out by human beings in the context of an economic model, the level of reduction required by such an analysis can be impractically fine.

A commonly used notion of the complexity of a computation is the time it takes to carry it out. When an algorithm is represented in terms of sequence(s) of operations, the complexity of a function computed by that algorithm is related to the length of the longest such sequence or, alternatively, the average (in some sense) length of all sequences that might arise in the computation of the function. Since the objective is to find a model that allows computational limitations of human beings to be brought into the analysis, and since human beings can easily do some things that are difficult for computers to do (and vice versa), it seems desirable not to require that the elementary operations that are used to model computers also be used to model humans. This supports the idea of making the set of elementary operations a primitive.

The set of elementary functions provides the model with a formal way of expressing limitations on computational powers. The set of functions allowed to be modules might include, for example, Boolean functions, Heavyside or threshold functions, smooth functions, polynomials of no more than a specified degree, or real analytic functions. (For some purposes it is appropriate even to regard continuous functions as elementary.) In particular, the class of elementary operations can also formalize other limitations on computational abilities. For example, psychologists have pointed out that the number of things a person can pay attention to at the same time is limited (Miller 1956, Rosenblatt 1962). This limitation on computational powers can be expressed by the condition that an elementary operation can be a function of no more than a specified number of variables. Let \mathcal{F} be the class of elementary operations. We call a modular network with elementary operations in the class \mathcal{F} an \mathcal{F} -network. If the number of variables that an elementary

function can have as arguments is restricted, say to r variables each of which can be at most d -dimensional (here we have in mind real variables), then we call the network an (r, d) -network with modules in \mathcal{F} . Where there is no ambiguity, we omit explicit reference to \mathcal{F} .

The *complexity* of a function relative to an \mathcal{F} -network is the minimum of the computing time required over all \mathcal{F} -networks that compute the given function. A function is *not computable* relative to \mathcal{F} if there is no \mathcal{F} -network that computes it in finite time.

The extension of computation from integers and discrete functions to continuous variables and functions is a natural one. Limit theorems (Mount and Reiter 1990) confirm the intuition that computing with real numbers is an acceptable idealization of computing with integers, much as measurement with real numbers is an accepted idealization of measurement with rational numbers. These theorems tell us that the measure of complexity of a real-valued function f of real variables relative to (r, d) -networks with elementary operations \mathcal{F} is a limit of measures of complexity, relative to finite (r, d) -networks, of discrete functions that approximate f . These results relate the extended notion of computing by (r, d) -networks to the finite state sequential machine model.

However, the model we have described so far is not yet adequate for representing directly some computations performed by humans. For that purpose we extend the model to include more abstract computations, provided that they can in a sense be reduced to computations with real variables. This is the purpose of introducing the idea of computing an *encoded version of a function*. The formal definition of this concept is presented in Section 3. For now we make do with an informal description that is sufficient for understanding the examples to follow. The idea is that a function to be computed may map a domain that is not a Euclidean space into another space, perhaps also not Euclidean. Human beings recognize or construct patterns, or other relationships, which generally are not presented as points of Euclidean spaces. For example, a set of points in the plane may be perceived as a two-dimensional representation of a box. Recognition of this visual pattern, a subset of Euclidean 2-space, can be thought of as computing a function that expresses the relation among the points in the subset that constitutes what is meant by saying it is a pattern – that is, a function whose value for that subset is the word “BOX.” In such a case it may be possible to encode the more abstract domain and range of the function being computed in such a way as to transform the problem into one of computing a function between Euclidean spaces, and decoding the result. Figure 2 shows the scheme.

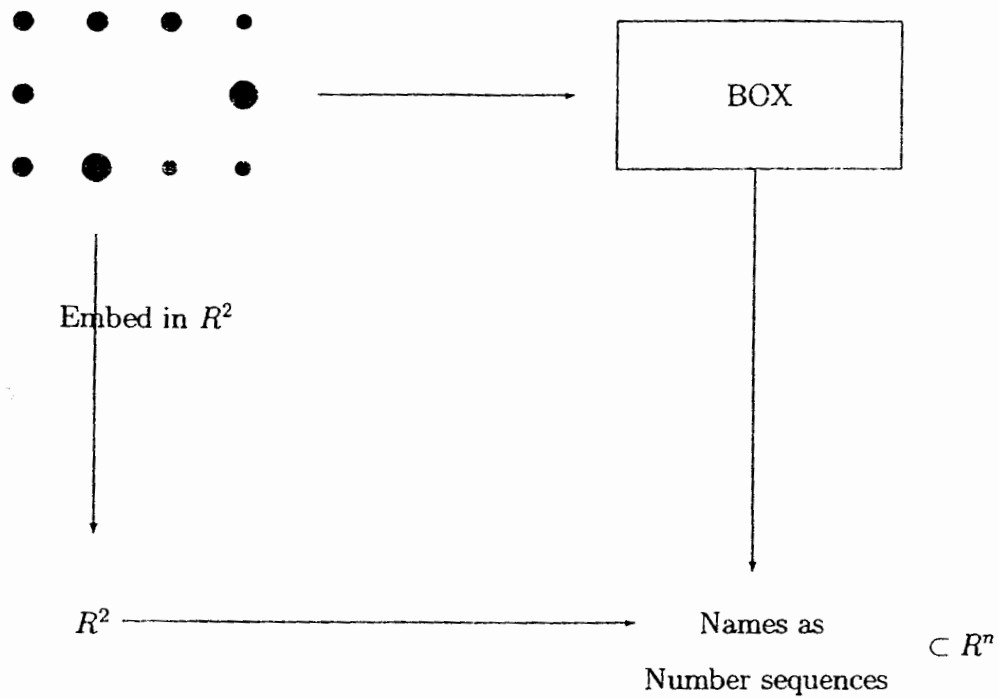


Figure 2

Before going further with the model, we take up two examples. These examples illustrate the application of the modular network model to computations performed by human beings and machines, and by human beings alone, and help clarify the ideas we have already discussed and that are subsequently presented in a more formal way.

2 Example I: Chernoff Faces

As a first example of the way in which humans and computers can interact to analyze complex information, we give an example in which the class of functions \mathcal{F} used in the construction of an \mathcal{F} -network contains functions that are more easily evaluated by humans than by computers.

Human beings are good at seeing patterns. The ability to see patterns seems to depend on the structure and functioning of the human visual apparatus. Consequently, this ability applies to patterns in the space, or space-time, in which the visual system evolved, that is, in at most three or four dimensions (ignoring color).

On the other hand, situations arise in which we would like to detect patterns in high-dimensional data, say observations represented by points in R^k for k a positive integer much larger than 4. Computers are good at handling data of high dimensionality. Although there exist algorithmic processes, such as discriminant analysis or cluster analysis, for detecting patterns or regularities in some cases, we do not have pattern-recognizing algorithms that do as well as humans when restricted to low dimensions. Therefore, the idea of combining the power of computers to manipulate data with the ability of humans to see patterns is appealing. Indeed, the practice of making graphical representations of data as a way of bringing to bear the human visual system predates the electronic computer, and is widely used in physical, biological and social science, and mathematics, as well as in business and everyday affairs.

Chernoff (1973) introduced the idea of combining human beings and computers to detect patterns in a sample of observations of a relatively large number of variables. Specifically, he introduced the graphical representation of multi-dimensional data as *cartoon faces* drawn in two dimensions, and illustrated its use by two examples. These are:

- (i) a set of 8 measurements made on each of 87 fossils; and
- (ii) a set of 53 observations of 12 variables taken from mineral analysis of a 4,500-foot core drilled from a Colorado mountain side.

The data are encoded as faces by a program that provides for up to 18 parameters that govern 18 features of a face. For example, one variable determines the horizontal distance between the eyes; another determines the height of the eyes; another determines the curvature of the arc that forms the mouth; and so forth. If the number of variables observed is $k \leq 18$, then $18 - k$ variables are fixed at some value and the remaining k variables determine the variable features of a face for each point observed. The computer prints out the set of faces, and a human being looks for a pattern in them. In the example with measurements made on fossils, the pattern sought was a classification of the fossils into groups of similar ones. In the second example, the observations were assumed to be generated by a multivariate stochastic process, and the problem was to detect a point in the time series of observations at which the process changed character.

Let

$$S \subset R^k, S = \{x^1, \dots, x^n\},$$

be the sample of n observations, each a k -dimensional point. Let $\eta: R^k \rightarrow R^2$ be a correspondence that assigns to k variables the subset of

R^2 that consists of the visual image encoding the variables. Set $\eta(x^1, \dots, x^n) = (y^1, \dots, y^n)$, where $y^i = \eta x^i$. (It is implicit in this notation that distinct points of S are assumed to be mapped to distinct subsets of R^2 .) Thus, in Chernoff's first example ($k = 8$), x^i is the vector of eight measurements made on the i th fossil and y^i is the cartoon face that encodes those measurements.

The problem is to classify the fossils, so we seek a partition of the set S or (correspondingly) a partition of the set $\{y^1, \dots, y^n\}$. Because S has n elements, the number of nonempty subsets in a partition of S , and a fortiori in a partition of $\{y^1, \dots, y^n\}$, is at most n . Therefore, a partition of S (or of $\{y^1, \dots, y^n\}$) can be represented by characteristic functions as follows. Let $\xi: S \rightarrow \{0, 1\}^n$ where $\xi = (\xi_1, \dots, \xi_n)$, let $\xi_i: S \rightarrow \{0, 1\}$, and define $Q_i = \{x \in S \mid \xi_i(x) = 1\} = \xi_i^{-1}(1) \subset S$. Then $Q = \{Q_1, \dots, Q_n\}$ is a partition of S , where (possibly after a renumbering of the characteristic functions) Q_i is the i th nonempty subset defined by ξ . If we suppose that Y is the collection of all possible n -tuples of faces, then a partition

$$P = \{P_1, \dots, P_r\}$$

of Y is defined by characteristic functions

$$\chi: Y \rightarrow \{0, 1\}^n,$$

where

$$\chi = (\chi_1, \dots, \chi_n), \quad \chi_i: Y \rightarrow \{0, 1\},$$

and

$$P_i = \{y \in Y \mid \chi_i(y) = 1\} = \chi_i^{-1}(1) \subset Y.$$

Consider the entire computation as carried out by a machine. The machine would execute a program that embodied an algorithm for, say, a cluster analysis of S . The machine would compute a (vectorial) characteristic function ξ from inputs that consisted of the coordinates of the 8-dimensional points that characterize a fossil. An \mathcal{F} -network model would represent this computation in terms of the class of elementary operations \mathcal{F} .

Suppose on the other hand that, instead of doing a cluster analysis, a human being produced a partition of the set Y using as inputs the set Y of cartoon faces encoding S produced by the computer. We could view the human being as an agent capable of "computing χ directly." We would like our model to represent the entire computation – namely, the part that produces the cartoon faces (which is carried out by the machine) and the part that produces the classification of faces (carried out by the human being) – in a seamless way. In that case, even if there

$$\begin{array}{ccc}
 S & \xrightarrow{\xi = \text{id}\chi\eta} & \{0,1\}^n \\
 \eta \downarrow & & \downarrow \text{id} \\
 Y & \xrightarrow{\chi} & \{0,1\}^n
 \end{array}$$

Figure 3

were no algorithm for performing the required analysis on S , the function ξ could be defined as in Figure 3 and incorporated into the set \mathcal{F} . However, the time scale for elementary operations carried out by a person might be different from those carried out by a machine. We shall continue to assume that an elementary operation takes one unit of time.

For our purposes, Chernoff's second example differs from the first only in that the sets S and Y are ordered according to the time sequence of the observations and the functions ξ and χ are step functions, with the step at the point at which the stochastic process is deemed to change character.

3 Example II: Reading Handwriting

Another example of a computation that humans perform routinely and in many cases easily and yet is very difficult for computers is reading handwriting. The phrase "reading handwriting" can mean several different things. Here we take it to mean writing in noncursive form (i.e., printing) the (English) expressions indicated by a given sample of cursive script.

The translation of cursive script into printed form is still extremely difficult, complex, and problematic for machines, although it is routinely performed by literate persons (though not without error). An impressive example is the reading of physicians' prescriptions by pharmacists. Imagine a typesetter, a person or machine, who has before him (it) a manuscript (cursive statement) and a font of type, and whose task is to produce a sequence of type elements (upper- and lower-case letters, punctuation marks, and spaces) that correctly translate the cursive manuscript into printed form.

A cursive writing sample is a plane "curve," one that may have discontinuities (e.g., gaps between adjacent letters) due to the idiosyncrasies of handwriting of a particular person, or the normal spaces between words; it may have isolated points (e.g., the dot over the letter "i" or the full stop that marks the end of a sentence); it may also have crossing strokes, such as that for the letter "t."

We may consider these curves to be concatenations of elements of a finite-dimensional space consisting of conceivable finite samples of cursive script of no more than a given (unit) length. Thus we assume the writing to be constructed of some collection of curves capable of being represented by a subset of a finite-dimensional Euclidean space. Denote this space by C . (Other properties of curves that can be cursive writing samples, such as being of bounded variation or having uniform upper and lower bounds on the height of letters, could also be considered.) The space T of printed text consists of a null element as well as all finite strings over the alphabet made up of the font elements.

The act of reading a cursive statement may be represented by a function $\varrho: C \rightarrow T$. Typically, this function will be many-to-one. Unreadable cursive samples are mapped to the null element of T . (If it is useful to do so, the function ϱ may be assumed to depend on the person who writes, as well as on the person who reads, or both.)

First, however, we consider how a machine might perform this act of reading a given cursive writing sample. The curve that constitutes a cursive writing sample must be presented to the computer in a form the computer can accept. This may be done by a device like a scanner that converts the curve into a string of symbols from the alphabet recognized by the computer, or perhaps by a sensitive tablet on which the sample is written using some sort of stylus or light pen. The result of either of these input devices is an *encoded representation of the curve*. This may be as a graphic image or as an object specified by the equations that define the curve as a locus in 2-space, perhaps relative to some given coordinate system. Another possibility (which involves more information, however) is to describe the curve parametrically, by equations

$$x(t) = \phi_1(t), \quad y(t) = \phi_2(t), \quad t_1 \leq t \leq t_2,$$

where (x, y) denotes a point in the plane and t is in the interval between t_1 and t_2 in the real line. This representation, or a discrete approximation to it, might describe someone writing cursively on a sensitive tablet.

Given this input, the computer would need a program to process the input into the ASCII code for the string of font symbols that constitute the output desired. Because the task is a complex one for a computer, the program is likely to be long and perhaps likely to produce incorrect results on many writing samples. (In the present discussion we may regard an incorrect result as equivalent to infinite computing time, i.e., we would have to wait forever for the correct result. A more satisfactory approach would be to measure the degree to which the output approximates the correct result, but this seems too complicated for the present purpose.) The diagram in Figure 4 represents the situation.

$$\begin{array}{ccc}
 C & \xrightarrow{p} & T \\
 e \downarrow & & \downarrow a \\
 R^{l_1} \times \dots \times R^{l_n} & \xrightarrow{f_p} & R^{l_1} \times \dots \times R^{l_m}
 \end{array}$$

Figure 4

In Figure 4, the function $e: C \rightarrow R^{l_1} \times \dots \times R^{l_n}$ is an encoding of the elements of C (i.e., cursive statements) into elements of $R^{l_1} \times \dots \times R^{l_n}$. (Note that if the encoding is done by a device such as a scanner, the encoding may depend on the position of the cursive statement on the screen of the device. In that case the coding would not be unique unless the positioning of the cursive sample is standardized. We may either assume that this is the case, or define a set of transformations in the plane that leave the cursive sample invariant except for position and define the encoding to be the same for any element of the equivalence class so generated. Evaluating this equivalence relation describes something humans do regularly – e.g., in stabilizing the visual field – but it is a complex task for a machine.)

Furthermore, the function a in the diagram is an encoding of T in $R^{l_1} \times \dots \times R^{l_m}$. The ASCII code for alphanumeric characters is an example. (Here the null element of T is mapped to any element of $R^{l_1} \times \dots \times R^{l_m}$ that is not the image of any character.) The inverse of the encoding a , performed by a device such as a printer, would produce the final result, the translation of the cursive writing sample into a printed writing sample.

If the computer cannot read a cursive writing sample in one step then the function f_p would not be elementary for that computer – that is, not be in the set \mathcal{F} consisting of the operations that are elementary for that computer. There would have to be a program written that computes f_p from the inputs using the operations that are elementary for that computer system.

The computation may be represented by an (r, d) -network, with modules from the class \mathcal{F} , that computes f_p . The complexity of f_p is likely to be very high, if indeed that function is at all computable relative to the modules in \mathcal{F} . If, for instance, the elementary operations consist of the arithmetic and logical operations, then a program that can read handwriting is likely to be long and involved, and the time required to compute f_p is likely to be long.

Consider next a person reading the cursive writing sample. That a person can read cursive script may be expressed by saying that the evaluation of the function ρ is an elementary operation for that person. That is, the person does it immediately or directly without any apparent inter-

mediate steps, taking only a small (unit) interval of time per unit length of curve. Another way to describe this is that we do not analyze the process into steps that are internal to the reader.

Although the function ϱ is clearly a representation of the act of reading cursive writing, it is not in itself a useful model; it does not yet connect with any other model of computation, nor does there appear to be a way to use it in the analysis of economic models.

On the other hand, in modeling a person reading handwriting we may consider the function ϱ to be equivalent to the composition

$$\alpha^{-1} \circ f_{\varrho} \circ e \equiv \varrho \quad (\text{E1})$$

in Figure 4. To say that a person can evaluate ϱ in one unit of time can be interpreted as saying that the composition (E1) can be evaluated in one unit of time. This amounts to saying that the function f_{ϱ} cannot take more than one unit of time. Hence it may be included as an elementary operation (i.e., a member of \mathcal{F}) in any application of the model in which a human being capable of reading cursive writing is among the computational resources.

For example, at the local drug store there is a pharmacist who reads the prescription form given to her by the customer, a form written by a physician in longhand. The pharmacist enters the prescription and other relevant information into a desktop computer by typing it on the keyboard. The computer processes this input according to its internal program, a computation representable by an (r, d) -network. The act of translating a unit length of the handwritten prescription into type (key-strokes) is elementary, and in the (r, d) -network model is formally seamless with the rest of the computation.

4 Analyzing the Complexity of a Function

With the preceding examples in mind, we now return to the modular network model. Specifically, we take up the question of how to use the modular network model to analyze the complexity of a function. In this, while we go into more detail, we make some simplifying assumptions in the interest of clarity. In particular, we take the concept of a directed graph as an intuitive one and do not give a mathematical definition; see Mount and Reiter (1990) for formal definitions of the concepts employed. Furthermore, we restrict the modules (elementary functions) to functions of at most r inputs, each of which is a d -dimensional vector; in fact, we take $r = 2$ and $d = 1$, restricting attention to elementary functions that are real-valued functions of at most two real variables. We also restrict attention to the computation of real-valued functions. Figure 1 showed a network that satisfies all these assumptions. Therefore,

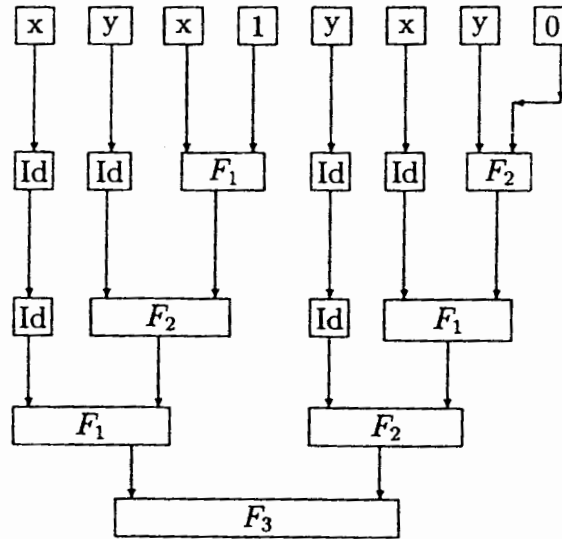


Figure 5

we may take the network shown in Figure 1 as a prototypical example. As seen in Table 1, that network computes many different functions from the same inputs as time increases. This property, which is inconvenient for analysis of the complexity of a function, is a result of the fact that the network in Figure 1 contains loops (i.e., its digraph contains cycles). However, if the function F to be analyzed is given then *a network C with loops that computes F in time t can be replaced by an equivalent network that computes F in time t , has the same modules as C (perhaps with the addition of projections and constants), and is free of loops.*

5 Modular Networks and Superpositions

The network in Figure 1 computes h (from the initial state shown in Table 1), so it is immediate from the graph that $h(x, y)$ can be written as

$$h(x, y) = F_3 \left[F_1 \left(x, F_2 \left(y, F_1 \left(x, 1 \right) \right) \right), F_2 \left(y, F_1 \left(x, F_2 \left(y, 0 \right) \right) \right) \right]$$

for any x and y .

That is, h can be written as a *superposition* (composition) of the functions that are modules of the network that computes h . A network that is represented by a connected tree with a single root computes functions that are *superpositions* of the functions in \mathcal{F} (i.e., superpositions of functions that are the modules of the network). Figure 5 shows such a

network that computes h . The *depth of the superposition* – that is, the number of levels of functions used – is equal to the length of the tree. Furthermore, if the network inputs are constantly on the input lines for a time that exceeds the length of the tree, then the network output is constant for all times thereafter and the depth of superposition is the length of the tree.

Thus, we see that the complexity of a function F relative to (r, d) -networks with modules in the class \mathcal{F} is equivalent to the minimum depth t such that F can be written as a superposition of functions from the class \mathcal{F} with depth t . This formulation is useful for obtaining analytical conditions for a lower bound on the complexity of F when F is sufficiently smooth.

The complexity of F relative to (r, d) -networks with modules in \mathcal{F} is related to a classical problem in mathematics: Hilbert's thirteenth problem (see Lorentz 1966). The essential substance of the problem is to decide whether a given function F of n variables can be written as a superposition of functions of fewer than n variables from a given class. It is implicit in the literature that the depth of the superposition expresses an intuitive notion of computational complexity.

When the functions in the superposition are restricted to be continuous and have fewer variables than F , Arnold and Kolmogorov have shown (cf. Lorentz 1966, p. 168; Vituskin 1961, Introduction) that F can always be written as a superposition of continuous functions of fewer variables. Indeed they have shown that each function of n variables can be written as a superposition of continuous functions of *two* variables. Furthermore, the depth of the superposition is bounded above by a constant that depends only on the number of variables of F . If, on the other hand, F is required to be smooth (i.e., continuously differentiable to some specified order), and if the functions used in the superposition are required to have the same degree of smoothness as F , then it is known that in general such a superposition representation cannot be guaranteed (cf. Lorentz 1966, Vituskin 1961).

Even if we reduce the class of functions and ask, as Hilbert did, whether an arbitrary analytic function of n variables can be written as a superposition of analytic functions of at most two variables, then an argument of Hilbert shows that the answer is No (see Lorentz 1966).

We may interpret these results as follows. The Arnold and Kolmogorov results suggest that there are *too many* continuous functions of two variables, and Hilbert's argument suggests that there are *too few* power series in two variables. Furthermore, in being too large, the class of continuous functions of two variables includes many functions that it would strain

credulity to regard as elementary. On the other hand, real analytic functions of two variables can be considered as extensions or idealizations of arithmetic operations, especially if we restrict them to be truncated power series.

In any case, for the purpose of illustrating the analysis of complexity of functions, including the construction of minimal (2,1)-networks, we shall take the class \mathcal{F} of elementary functions to consist of power series in two variables up to degree M . Specifically, we assume as follows.

- A1 \mathcal{F} consists of real analytic functions (power series) in two variables, truncated at degree M , with constant term identically zero and linear term not zero.
- A2 The functions to be computed (i.e., whose complexity is to be analyzed) consist of real analytic functions of n variables that vanish at the origin. They are to be computed only up to degree M .

We then say that a function F can be computed in time T to degree M if it can be written, to degree M , as a superposition of length T of functions from \mathcal{F} but not as a superposition of length $T - 1$.

Are there conditions on the function F that inform us whether it can be written as a superposition of length T of functions from \mathcal{F} but not as a superposition of length $T - 1$? And further, how can we construct a superposition of length T for F or (equivalently) an (r, d) -network with modules in F that computes F in time T ?

For this informal presentation we restrict attention to representing F as a superposition of analytic functions of two variables (see Mount and Reiter 1990 for a more general treatment). Accordingly, throughout this discussion the function F and the class of elementary functions \mathcal{F} satisfy assumptions A1 and A2. We use the following theorem (Leontief 1947 and Abelson 1980). Although we apply the theorem here only to the special case defined by A1 and A2, we state the theorem in more general terms. We first state the theorem for a function $F: R^m \times R^n \rightarrow R$, and then illustrate in the special case how it can be used. We suppose that R^m has coordinates $\underline{x} = (x_1, \dots, x_m)$ and R^n has coordinates $\underline{y} = (y_1, \dots, y_n)$. In order to state the theorem, we define two matrices associated with the function F . These are:

$$BHF(\underline{x}, \underline{y}) = \begin{pmatrix} \partial F / \partial x_1 & \partial^2 F / \partial x_1 \partial y_1 & \dots & \partial^2 F / \partial x_1 \partial y_n \\ \vdots & \vdots & \dots & \vdots \\ \partial F / \partial x_m & \partial^2 F / \partial x_m \partial y_1 & \dots & \partial^2 F / \partial x_m \partial y_n \end{pmatrix}.$$

where $\underline{x} = (x_1, \dots, x_m)$ and $\underline{y} = (y_1, \dots, y_n)$; and $BHF(\underline{y}, \underline{x})$, constructed by interchanging \underline{x} and \underline{y} in the construction of the matrix $BHF(\underline{x}, \underline{y})$.

The Leontief result (as used by Abelson 1980) implies: A necessary condition that a continuously differentiable function F can be written in the form $G(A(\underline{x}), \underline{y})$, where A is a function with continuous first derivatives in a neighborhood of a point $(\underline{a}, \underline{b})$ with $\underline{a} = (a_1, \dots, a_m)$ and $\underline{b} = (b_1, \dots, b_n)$, is that the matrix $BHF(\underline{x}, \underline{y})$ have rank at most 1. This rank condition is also a sufficient condition that F can be written in the form $G(A(\underline{x}), \underline{y})$ if F has nonvanishing first partials in the x_i at $(\underline{a}, \underline{b})$.

Abelson used the necessary condition to analyze the information transfer of a multi-stage distributed computation of the function F . He analyzed the informational exchange requirements when two processors PX and PY compute the function $F(\underline{x}, \underline{y})$ when only PX has the values x_i and only PY has the values y_j . He treated informational exchange as the communication of real number values. Abelson showed that if the matrix derived from $BHF(\underline{x}, \underline{y})$ by deleting the first column has rank R , then each multi-stage distributed computation that computes F in a neighborhood of $(\underline{a}, \underline{b})$ must have a total information transfer of at least R between the two processors.

Abelson's analysis of the communication required to compute a function given distributed information uses a submatrix of the bordered Hessian $BHF(\underline{x}, \underline{y})$ and bounds the communication by a rank condition on a submatrix of the Hessian. We also use the matrix $BHF(\underline{x}, \underline{y})$ and its twin $BHF(\underline{y}, \underline{x})$ to analyze computational complexity. However, the rank conditions we require are not the same as Abelson's, and further we must analyze a *sequence* of such matrices derived from F .

Theorem (Leontief–Abelson). *The function F equals $H(A(\underline{x}), B(\underline{y}))$ only if both matrices $BHF(\underline{x}, \underline{y})$ and $BHF(\underline{y}, \underline{x})$ have rank at most 1, and these rank conditions are also sufficient when first partials in x_i and in y_i are nonvanishing. (A complete proof can be found in Mount and Reiter 1990.)*

Suppose further that, in addition to satisfying the hypotheses of the theorem, F is a function of 2^N variables, that the linear part of F depends on all 2^N variables, and that (as assumed previously) $F(0, \dots, 0) = 0$. It is easy to see that the shortest time in which F can possibly be computed is N . To see this, note that to compute the linear combination of the 2^N variables x_i ($j = 0, \dots, 2^N - 1$) that is the linear part of F in minimal time, using $(2, 1)$ -modules in \mathcal{F} , one should compute as many of these as

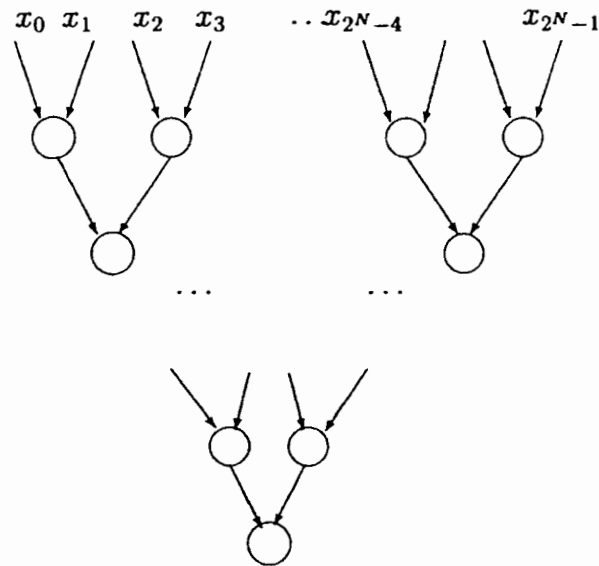


Figure 6

possible at the same time. This is done by the $(2, 1)$ -network shown in Figure 6, a $(2, 1)$ -fan-in of length N .

Now suppose that F (i.e., not just the linear part of F) can be computed up to degree M in the minimal time N . Then the graph of the $(2, 1)$ -network that computes it must also be a fan-in of depth N . We need only specify the module assigned to each vertex of the fan-in in order to specify the network completely, that is, to write F as a superposition of functions in \mathcal{F} . We next show how to construct a proposed assignment. The Leontief-Abelson theorem can then be applied to verify that the assignment proposed can in fact be made.

6 Assigning Modules to the Vertices of the Fan-In

We begin by labeling the vertices of the fan-in. This is done as follows. If the vertex is a leaf (an input vertex) then it is labeled with the index of the variable that is input there. If the vertex is not an input vertex then it is labeled with an ordered pair. There are two cases.

- (1) The two inputs to the vertex v come from vertices labeled (i, j) and (k, l) , respectively, where $i < k$. Then the label attached to v is (i, l) .
- (2) The input lines of v are network input lines. Network input lines are labeled by the variables they carry. Thus, the inputs to v are

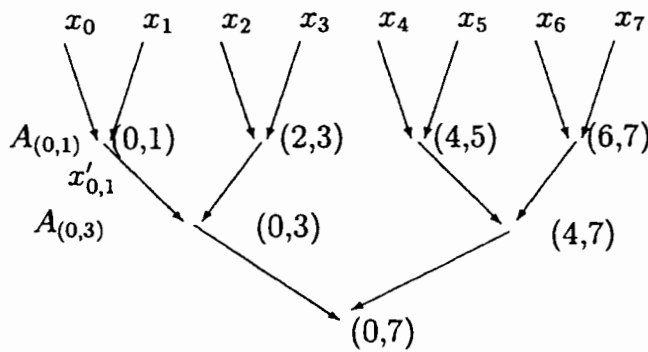


Figure 7

the variables in an adjacent pair (x_{2j}, x_{2j+1}) for some j in $0, \dots, 2^{N-1} - 1$. In this case the vertex v has the label $(2j, 2j + 1)$.

This labeling is illustrated in Figure 7 for $N = 3$. The leaves are labeled in order from left to right, $0, 1, \dots, 2^N - 1 = 0, 1, \dots, 7$. The vertex whose inputs are x_0 and x_1 is labeled $(0, 1)$; the vertex whose inputs are the outputs of $(0, 1)$ and $(2, 3)$ is labeled $(0, 3)$; and so on. (There is no ambiguity in this labeling scheme, because if a vertex has inputs from vertices (i, j) and (k, l) then $i < k$ implies that $(i, l) \neq (i, j')$ for any vertex labeled (i, j') .)

Under the assumptions we have made, a function F computed by a $(2, 1)$ -network in minimal time determines the modules of the network essentially uniquely. The qualification “essentially” refers to the fact that the modules are determined up to an equivalence relation, according to which two functions are equivalent if they are the same except for changes of variables in the domain and range. This is defined more explicitly in what follows. In the interest of clarity and to minimize notational complexity, we illustrate the process of assigning modules for $N = 3$ using the tree in Figure 7. We suppose that $F: R^8 \rightarrow R$, where $F(x_0, \dots, x_7)$ is computed by a network whose tree is shown in Figure 7 with modules in \mathcal{F} .

The module assigned to vertex (i, j) is denoted $A_{(i,j)}$. Consider first the vertex $(0, 1)$, which has as inputs the variables x_0 and x_1 and is therefore assigned the function $A_{(0,1)}$. Consider how the network evaluates F at the point $(x_0, x_1, 0, \dots, 0)$. Because the functions $A_{(i,j)}$ in F have the property that $A_{(i,j)}[0, 0] = 0$, it follows that, at each vertex (i, j) such that $i \neq 0$, the output of the module assigned to that vertex must be 0. Furthermore, at each vertex $(0, j)$ where $j > 1$, only the left input line carries a value different from 0. Thus, each module $A_{(0,j)}$ where $j > 1$ acts like a (truncated)

power series in one variable. Therefore, the composition of the modules from $A_{(0,3)}$ to the module assigned to the root acts like a power series in one variable. Denote this composition $h(z)$ (not to be confused with the function h of Table 1). Thus

$$h(z) = A_{(0,7)}(A_{(0,3)}(z, 0), 0). \quad (\text{E2})$$

Write $F(x_0, x_1, 0, \dots, 0) = F_{(0,1)}(x_0, x_1)$. Then

$$h(A_{(0,1)}(x_0, x_1)) = F(x_0, x_1, 0, \dots, 0) = F_{(0,1)}(x_0, x_1).$$

Similarly, for $F(0, 0, x_2, x_3, 0, \dots, 0) = F_{(2,3)}(x_2, x_3)$,

$$g(A_{(2,3)}(x_2, x_3)) = F_{(2,3)}(x_2, x_3),$$

where

$$g(z) = A_{(0,7)}[A_{(0,3)}(z), 0] = h(z).$$

The last equality is by equation (E2). Therefore,

$$h(A_{(2,3)}(x_2, x_3)) = F_{(2,3)}(x_2, x_3).$$

That is, the composition of the modules $A_{(0,j)}$, which is the function h , when composed with $A_{(0,1)}$ yields $F_{(0,1)}$ and yields $F_{(2,3)}$ when composed with $A_{(2,3)}$. Thus, if $h(z)$ is known and has an inverse, then we have determined both $A_{(0,1)}$ and $A_{(2,3)}$ from F once we know h . In the same way, the modules $A_{(4,5)}$ and $A_{(4,7)}$ can be assigned by using $F_{(4,5)}$ and $F_{(6,7)}$ defined analogously.

Next, define new variables $y_{i,j}$ by

$$y_{i,j} = A_{(i,j)}(x_i, x_j), \quad \text{where } (i, j) = (0, 1), (2, 3), (4, 5), \text{ or } (6, 7),$$

and consider the network shown in Figure 8.

The same process that was used to determine $A_{(0,1)}$ can be applied to the determination of $A_{(0,3)}$, using the function

$$G(y_{0,1}, y_{2,3}) = F(x_0, x_1, x_2, x_3, 0, \dots, 0),$$

where $y_{0,1} = A_{(0,1)}(x_0, x_1)$ and $y_{2,3} = A_{(2,3)}(x_2, x_3)$. The other modules in Figure 8 can be assigned in the same way.

We consider next the extent to which the functions we have constructed are unique. Let us define an equivalence relation on real analytic functions of two real variables as follows.

Two (real analytic) functions $B(x, y)$ and $C(x, y)$ are *equivalent* if there are nonsingular (i.e., having nonzero linear terms) analytic functions u, k, l of one variable such that

$$B(x, y) = u(C(k(x), l(y))).$$

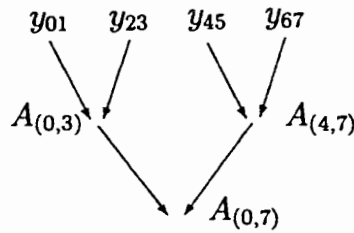


Figure 8

The constructions illustrated in Figures 7 and 8 indicate that if the function F_i can be computed by a superposition as in Figure 7, then the modules $A_{(i,j)}$ are unique to within equivalence; that is, they are equivalent to the functions $F_{(i,j)}$.

7 Determining the Modules by Solving Equations

We digress here to present a determination of the modules $A_{(0,j)}$ in Figure 7 that makes explicit use of the power series expressions for all the functions involved. This material can be skipped by readers who find the foregoing sufficiently explicit.

We assume that $M = 2$ – in other words, that F is to be computed up to degree 2 and the modules are power series truncated at degree 2. Then

$$F(x_0, \dots, x_7) = \sum_{i=0}^7 a_i x_i + \sum_{i=0}^7 b_i x_i^2 + 2 \sum_{i=0 \dots 7, j=0 \dots 7, i < j} c_{i,j} x_i x_j.$$

Using the power series for F and the modules $A_{(0,j)}$, we have

$$F_{(0,1)}(x_0, x_1) = a_0 x_0 + a_1 x_1 + b_0 x_0^2 + b_1 x_1^2 + c_{0,1} x_0 x_1,$$

$$A_{(0,1)}(x_0, x_1) = \alpha_0 x_0 + \alpha_1 x_1 + \beta_0 x_0^2 + \beta_1 x_1^2 + \chi_{0,1} x_0 x_1,$$

$$A_{(0,3)}(y) = \alpha'_0 y + \beta'_0 y^2,$$

$$A_{(0,7)}(y', 0) = \alpha''_0 y' + \beta''_0 y'^2,$$

where

$$y = \alpha_0 x_0 + \alpha_1 x_1 + \beta_0 x_0^2 + \beta_1 x_1^2 + \chi_{0,1} x_0 x_1,$$

$$y^2 = \alpha_0^2 x_0^2 + 2\alpha_0 \alpha_1 x_0 x_1 + \alpha_1^2 x_1^2,$$

and hence

$$y' = A_{(0,3)}(y, 0) = \alpha'_0(\alpha_0 x_0 + \alpha_1 x_1 + \beta_0 x_0^2 + \beta_1 x_1^2 + \chi_{0,1} x_0 x_1) \\ + \beta'_0(\alpha_0^2 x_0^2 + 2\alpha_0 \alpha_1 x_0 x_1 + \alpha_1^2 x_1^2).$$

Using the expression for y' and calculating y'^2 , we evaluate $A_{(0,7)}(y', 0)$ to obtain

$$z = A_{(0,7)}(y', 0) \\ = \alpha''_0(\alpha'_0 \alpha_0 x_0 + \alpha'_0 \alpha_1 x_1 + (\alpha'_0 \beta_0 + \beta'_0 \alpha_0^2) x_0^2 \\ + (\alpha'_0 \beta_1 + \beta'_0 \alpha_1^2) x_1^2 + (\alpha'_0 \chi_{0,1} + 2\beta'_0 \alpha_0 \alpha_1) x_0 x_1) \\ + \beta''_0((\alpha'_0 \alpha_0)^2 x_0^2 + 2\alpha_0 \alpha_1 (\alpha'_0)^2 x_0 x_1 + (\alpha'_0 \alpha_1)^2 x_1^2).$$

Because the superposition we have obtained is required to compute $F_{(0,1)}$, that is, because

$$F_{(0,1)}(x_0, x_1) = z \quad \text{for all } (x_0, x_1),$$

we may equate coefficients of like terms; this yields the following set of equations:

$$\begin{aligned} a_0 &= \alpha''_0 \alpha'_0 \alpha_0, \\ a_1 &= \alpha''_0 \alpha'_0 \alpha_1, \\ b_0 &= \alpha''_0(\alpha'_0 \beta_0 + \beta'_0 \alpha_0^2) + \beta''_0(\alpha'_0 \alpha_0)^2, \\ b_1 &= \alpha''_0(\alpha'_0 \beta_1 + \beta'_0 \alpha_1^2) + \beta''_0(\alpha'_0 \alpha_1)^2, \\ c_{0,1} &= \alpha''_0(\alpha'_0 \chi_{0,1} + 2\beta'_0 \alpha_0 \alpha_1) + \beta''_0(2\alpha_0 \alpha_1 (\alpha'_0)^2). \end{aligned} \quad (\text{E3})$$

These are five equations in nine unknowns. We specify the values of four of the variables arbitrarily and solve for the remaining five. Thus, let

$$\alpha'_0 = \alpha''_0 = \alpha'_1 = \beta''_0 = 1. \quad (\text{E4})$$

Then the equations (E3) reduce to

$$\begin{aligned} a_0 &= \alpha_0, & a_1 &= \alpha_1, \\ b_0 &= \beta_0 + 2\alpha_0^2, & b_1 &= \beta_1 + 2\alpha_1^2, \\ c_{0,1} &= \chi_{0,1} + 4\alpha_0 \alpha_1, \end{aligned}$$

or, equivalently, to

$$\begin{aligned} \alpha_0 &= a_0, & \alpha_1 &= a_1, \\ \beta_0 &= b_0 - 2a_0^2, & \beta_1 &= b_1 - 2a_1^2, \\ \chi_{0,1} &= c_{0,1} - 4a_0 a_1. \end{aligned} \quad (\text{E5})$$

Equations (E4) and (E5) determine the modules $A_{(0,j)}$ for $j = 1, 3, 7$. To verify that these modules do in fact compute $F(x_0, x_1, 0, \dots, 0)$, we substitute from (E4) and (E5) into (E2). Then the expression for z becomes

$$\begin{aligned} z &= x_0 a_0 + x_1 a_1 + x_0^2 (b_0 - 2a_0^2 + 2a_0^2) \\ &\quad + x_1^2 (b_1 - 2a_1^2 + 2a_1^2) + x_0 x_1 (c_{0,1} - 4a_0 a_1 + 4a_0 a_1) \\ &= a_0 x_0 + a_0 x_0 + b_0 x_0^2 + b_1 x_1^2 + c_{0,1} x_0 x_1 \\ &= F(x_0, x_1, 0, \dots, 0). \end{aligned}$$

It should be noted that this algebraic method does not work for larger problems. We now return to the main line of exposition.

The process for determining the modules of the fan-in that computes F in the general case yields the following information about the expression of F as a superposition of functions in \mathcal{F} . Looking first at the root of the tree for F , we see that

$$\begin{aligned} &F(x_0, \dots, x_{2^{N-1}}) \\ &= H\left(F(x_0, \dots, x_{2^{N-1}-1}, 0, \dots, 0), F(0, \dots, 0, x_{2^{N-1}}, \dots, x_{2^{N-1}})\right), \end{aligned}$$

where H is shorthand for $A_{(0,2^{N-1})}$.

The function $F(x_0, \dots, x_{2^{N-1}-1}, 0, \dots, 0)$, or a function equivalent to it, is computed in minimal time by the subtree whose root is the vertex $(0, 2^{N-1} - 1)$. Similarly, the function $F(0, \dots, 0, x_{2^{N-1}}, \dots, x_{2^{N-1}})$, or a function equivalent to it, is computed in minimal time by the subtree whose root is the vertex $(2^{N-1}, 2^{N-1})$. In Figure 7 these are the subtrees with root $(0, 3)$ and $(4, 7)$, respectively.

Recall that the Leontief–Abelson theorem gives necessary and sufficient conditions for these computations. Therefore that theorem gives us two ways to verify whether the computation indicated in Figure 7 can actually be carried out, that is, whether the functions involved exist.

We illustrate the application of the theorem in a simple example.

8 Computing Equilibrium Price in an Edgeworth Box Economy

The function P to be computed gives the price as a function of the parameters characterizing the agents in a two-person, two-good exchange economy in which the utility functions of the agents are quadratic quasilinear. Denoting these parameters by (x, z) for agent 1 and by (x', z') for agent 2, the function P is given by

$$P(x, z, x', z') = \frac{xz' - x'z}{x - x'}$$

(see Mount and Reiter 1982, p. 124, for the derivation of P). The function P is to be computed by a $(2, 1)$ -network whose inputs are the variables x, z, x', z' in some order. To avoid singularity, we make a coordinate translation to coordinates R, S, T, U , where

$$R = x - 1, \quad S = z, \quad T = x' + 1, \quad U = z'.$$

In the new coordinates,

$$P(R, S, T, U) = \frac{S + U + RU - ST}{2 + R - T}.$$

Here the number of variables is $2^N = 4$, so that $N = 2$. Hence a lower bound for the depth of $(2, 1)$ -networks that compute P is 2. If P is computable in time 2, then in terms of superpositions there must exist real analytic functions A', B', C' , or A'', B'', C'' , defined on a neighborhood of the origin in R^2 , such that P can be written as

$$P(R, S, T, U) = C[A'(S, T), B'(R, U)] \tag{E6}$$

or

$$P(R, S, T, U) = C''[A''(R, T), B''(S, U)] \tag{E7}$$

Consider the case in which P is given by equation (E6). The Leontief-Abelson theorem states that a necessary condition for the existence of A', B', C' satisfying (E6) is that the matrix

$$BHP(S, T; R, U)_{(0,0,0,0)} = \begin{pmatrix} \left(\frac{\partial P}{\partial S}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial R \partial S}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial S \partial U}\right)_{(0,0)} \\ \left(\frac{\partial P}{\partial T}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial R \partial T}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial U \partial T}\right)_{(0,0)} \end{pmatrix} \tag{E8}$$

have rank at most 1. But P , being real analytic, can be written in power series in the form

$$P = (S + U + RU - TS) \left(\sum_{j=0}^{\infty} (-1)^j \left(\frac{1}{2}\right)^{j+1} (T - R)^j \right) \\ = \left(\frac{1}{2}\right) \left[(S + U + RU - TS) + \frac{(S + U)(T - R)}{2} \right] + \theta,$$

where θ is a sum of monomials in R, S, T, U of degree at least 3. Evaluating the matrix in (E8), we see that

$$BHP(S, T; R, U)_{(0,0,0,0)} = \begin{pmatrix} \frac{1}{2} & -\frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix},$$

which has rank 2.

We try next the possibility that

$$P(R, S, T, U) = C''[A''(R, T), B''(S, U)]$$

In this case the matrix BHP has the form

$$BHP(S, T; R, U)_{(0,0,0,0)} = \begin{pmatrix} \left(\frac{\partial P}{\partial S}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial R \partial S}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial S \partial T}\right)_{(0,0)} \\ \left(\frac{\partial P}{\partial U}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial R \partial U}\right)_{(0,0)} & \left(\frac{\partial^2 P}{\partial T \partial U}\right)_{(0,0)} \end{pmatrix}.$$

When evaluated as before, this matrix is

$$\begin{pmatrix} \frac{1}{2} & -\frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \end{pmatrix},$$

which has rank 2. Thus, the necessary condition of the Leontief-Abelson condition is not satisfied in either case. Therefore P cannot be computed in a neighborhood of the origin by a (2, 1)-network with real analytic modules in less than three units of time from the inputs R, S, T, U . However, P clearly can be computed from R, S, T, U in three units of time.³

9 Concluding Remarks

The value of the modular network approach to modeling information processing will, as with other approaches, depend on results achieved with it. In this paper we have presented the basic ideas of the model and of the analysis of computational complexity in that model. We have also given two examples, Chernoff faces and reading handwriting, to illustrate its applicability to computing performed by human beings. Interesting as these applications may be, they are not applications to economics.

³ A more complete analysis of this example is given in Mount and Reiter (1990). There the possible trade-off between communication in the form of message-space size and computational complexity is analyzed and the efficient frontier is derived.

Therefore, we conclude this paper with a very brief account of application of the model to a problem of economic theory, namely, a theory of the structure and size of organizational units, such as firms or divisions of firms. This discussion is necessarily brief; a full discussion would exceed the scope of this paper.

The problem addressed in Reiter (1995) is to construct a theory that carries out Coase's program, or principle, which is that economic institutions (such as firms) are optimal adaptations to certain constraints. As pointed out in the introduction of this chapter, firms typically have administrative or managerial structures in which information processing for decision making is shared among many people, the necessity for sharing arising from limitations on the information processing capacities of individuals. Therefore the theory is one that obtains the set of firms (or informationally independent organizational units) as solutions, not necessarily unique, of a constrained optimization problem. In that problem there should be:

- (1) a "variable" whose values correspond to different organizational units (e.g., different firms);
- (2) a set of constraints on this variable expressing technological possibilities and resource availability;
- (3) constraints on information processing; and
- (4) a criterion of performance that expresses the goals of action.

Incentive constraints on individuals and groups, which properly belong in the problem, are not addressed.

In order to solve the model we must find a subset of the set of possible firms that maximizes the criterion over the set of environments, subject to all the constraints.

Desired economic action is represented by a decision function, for example, a function that associates to each environment actions resulting in a Pareto optimal (or, alternatively, a profit maximizing) outcome. Information about the environment is distributed among agents. Coordination is defined in terms of the decision function, and the decision-making task is to compute the value of the decision function in each environment. The problem addressed is how best to organize decision making in these circumstances, taking account of the limitations on the information processing capabilities of individuals.

The information processing capabilities of an individual include:

- (a) conditions on the capability of the individual to observe environmental variables;

- (b) the set of computational operations (modules) that are elementary for that individual (e.g., individual i may be capable of observing the value of a parameter x or of y but not both); and
- (c) the ability to carry out at most one elementary operation in a unit of time.

A mode of organization in its informational aspect consists of:

- (i) an algorithm for computing its decision function; and
- (ii) an assignment to individual agents of the steps required to execute the algorithm.

An algorithm is modeled as a modular network. An assignment of the steps of the computation to individuals means that the module at each node of the digraph that represents the algorithm is assigned to an individual to carry out. The assignment of modules of the network to individuals must satisfy three conditions as follows:

- (i) the assignment of input nodes to individuals must satisfy the constraints on what individuals may observe;
- (ii) a module assigned to an individual must be one of her elementary operations;
- (iii) the scheduling of execution of elementary operations in time must satisfy the condition that two or more operations assigned to the same individual cannot be carried out at the same time.

An assignment of a modular network to individual agents determines four characteristics on which information processing costs depend, each measured by a variable. These are: delay; communication; access to memory; and number of agents (processors) used.

The cost function is assumed to be linear in these variables. It is shown that cost minimization is achieved by efficient assignments, that is, assignments resulting in vectorially minimal (four-dimensional) vectors whose components are the arguments of the cost function.

Necessary and sufficient conditions for efficient assignments are derived, and algorithms are given for computing the set of efficient assignments. An efficient assignment of a modular network is represented by an efficient assigned directed acyclic graph – an EADAG. Thus, each EADAG represents an informationally efficient organization of the economic activity possible in the given class of environments. An EADAG may represent one or more informationally separate organizational units, depending on its structure. The idea behind the formal definition given in Reiter (1995) is that difficult coordination problems, described by the necessity to exchange a great deal of information among

the individuals involved, require a highly integrated organization (one organizational unit), whereas other coordination problems can be handled with limited exchange of information. For example, in a two-person two-good exchange economy, individual preference relations may be very complicated, requiring an arbitrarily large number of parameters to specify, but the information that must be communicated between the two agents in order to achieve a Pareto outcome in each such environment can consist of just two numbers. If we were to represent this situation by a modular network describing the internal calculations needed to derive excess demands, as well as the calculations needed to find a market equilibrium, the solution EADAG would consist of two components: one corresponding to the internal calculations of each of the two economic agents. The internal communication in each component would increase with the number of parameters needed to specify the preference relations, while the number of variables that need to be transmitted between the components would be constant, independent of the number of parameters.

The model is applied to examples, showing circumstances in which the solution is one firm, with a bound on its size, and others in which it is two firms. Finally, some general results are discussed in which the prevalence of centralized or administrative rather than decentralized organization is related to the nature of the coordination problem presented by the class of environments to be organized.

References

- H. Abelson (1980), *Lower Bounds on Information Transfer in Distributed Computations*, *Journal of the Association for Computing Machinery* 27: 384-92.
- D. Abreu and A. Rubinstein (1988), *The Structure of Nash Equilibrium in Repeated Games with Finite Automata*, *Econometrica* 56: 1259-88.
- M. A. Arbib (1969), *Theories of Abstract Automata*, Prentice-Hall, Englewood Cliffs, NJ.
- K. J. Arrow and L. Hurwicz (1959), *On the Stability of the Competitive Equilibrium II*, *Econometrica* 27: 82-109.
- R. J. Aumann (1981), *Survey of Repeated Games*, in *Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern*, Bibliographisches Institut, Mannheim, pp. 11-42.
- L. Blum, M. Shub, and S. Smale (1989), *On a Theory of Computation and Complexity Over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines*, *Bulletin of the American Mathematical Society* 21: 1-46.
- R. Coase (1937), *The Nature of the Firm*, *Economica* 4: 386-405.

- H. Chernoff (1973), *The Use of Faces to Represent Points in 6-Dimensional Space Graphically*, *Journal of the American Statistical Association* 68: 361–8.
- I. K. Cho (1993a), *Perceptrons Play Repeated Games with Imperfect Monitoring*, Mimeo, Department of Economics, University of Chicago.
- I. K. Cho (1993b), *Perceptrons Play the Repeated Prisoner's Dilemma*, Mimeo, Department of Economics, University of Chicago.
- I. K. Cho and H. Li (1993), *Complexity and Neural Network in Repeated Games*, Preprint, Department of Economics, University of Chicago.
- G. Debreu (1959), *Theory of Value*, Wiley, New York.
- C. Futia (1977), *The Complexity of Economic Decision Rules*, *Journal of Mathematical Economics* 4: 289–99.
- J. J. Hopfield (1982), *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, *Proceedings of the National Academy of Science* 79: 2554–8.
- L. Hurwicz (1960), *Optimality and Informational Efficiency in Resource Allocation Processes*, in *Mathematical Methods in the Social Sciences* (K. J. Arrow, S. Karlin, and P. Suppes, eds.), Stanford University Press.
- L. Hurwicz (1986), *On Informational Decentralization and Efficiency*, in *Studies in Mathematical Economics* (S. Reiter, ed.), Mathematical Association of America, Washington, DC.
- E. Kalai and W. Stanford (1988), *Finite Rationality and Interpersonal Complexity in Repeated Games*, *Econometrica* 56: 397–410.
- N. Kaldor (1934), *The Equilibrium of the Firm*, *Economic Journal* 44: 70–80.
- W. Leontief (1947), *A Note on the Interrelation of Subsets of Independent Variables of a Continuous Function with Continuous First Derivatives*, *Bulletin of the American Mathematical Society* 53: 343–50.
- G. G. Lorentz (1966), *Approximation of Functions*, Holt, Rinehart and Winston, New York.
- T. Marschak (1986), *Organizational Design*, in *Handbook of Mathematical Economics*, vol. II (K. J. Arrow and M. D. Intriligator, eds.), Elsevier, Amsterdam, pp. 1358–440.
- K. Matsuyama (1993), *Toward an Economic Theory of Pattern Formation*, Discussion Paper no. 1079, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL.
- W. McCulloch and W. Pitts (1943), *A Logical Calculus of the Ideas Imminent in Nervous Activity*, *Bulletin of Mathematical Biophysics* 5: 115–33.
- G. A. Miller (1956), *The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*, *Psychological Review* 63: 108.
- M. L. Minsky and S. A. Papert (1988), *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA.
- K. R. Mount and S. Reiter (1982), *Computation, Communication, and Performance in Resource Allocation*, Paper presented at the CEME–NBER Decentralization Seminar, University of Minnesota, Minneapolis.
- K. R. Mount and S. Reiter (1990), *A Model of Computing with Human Agents*, Discussion Paper no. 890, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL.
-

- K. R. Mount and S. Reiter (1993), *Essential Revelation Mechanisms and Computational Complexity*, Discussion Paper no. 1047, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL.
- A. Neyman (1985), *Bounded Complexity Justifies Cooperation in the Finitely Repeated Prisoner's Dilemma*, *Economic Letters* 19: 227-9.
- R. Radner (1993), *The Organization of Decentralized Information Processing*, *Econometrica* 62: 1109-46.
- R. Radner and T. Van Zandt (1992), *Information Processing in Firms and Returns to Scale*, *Annales d'Economie et la Statistique* 25/26: 265-98.
- S. Reiter (1995), *Coordination and the Structure of Firms*, Discussion Paper no. 1121, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL.
- F. Rosenblatt (1962), *Principles of Neurodynamics*, Spartan, Washington, DC.
- A. Rubinstein (1979), *Equilibrium in Supergames with the Overtaking Criterion*, *Journal of Economic Theory* 21: 1-9.
- A. Rubinstein (1986), *Finite Automata Play the Repeated Prisoner's Dilemma*, *Journal of Economic Theory* 39: 83-96.
- A. Rubinstein (1993), *On Price Recognition and Computational Complexity in Monopolistic Model*, *Journal of Political Economy* 101: 473-84.
- H. A. Simon (1972), *Theories of Bounded Rationality*, in *Decision and Organization* (C. B. McGuire and R. Radner, eds.) North-Holland, Amsterdam, pp. 161-76.
- H. A. Simon (1987), *The Sciences of the Artificial*, 2nd ed., MIT Press, Cambridge, MA.
- T. Van Zandt (1995), *Heirarchical Computation of the Resource Allocation Problem*, *European Economic Review* 39: 700-8.
- T. Van Zandt and R. Radner (1995), *Information processing and returns to scale in a statistical decision problem*, Princeton University and AT&T Bell Laboratories.
- A. G. Vituskin (1961), *Complexity of Tabulation Problems*, Pergamon, New York.
- O. E. Williamson (1975), *Markets and Hierarchies, Analysis and Antitrust Implications*, New York, Free Press.
- A. C. C. Yao (1979), *Some Complexity Questions Related to Distributed Computing*, *Journal of the Association for Computing Machinery* 22: 209-13.

