1 Introduction

1.1. THE MODELING OF COMPUTING AND ECONOMIC AGENTS

This book presents a model of computing, called the modular network model, and a measure of complexity, intended to apply to computations performed by a mixed system of people and machines. Applications of the model to problems in game theory and economics are also presented.

The model has two primitives: a set, usually denoted $\mathcal{F}$ of elementary functions or elementary operations, and a set of directed graphs. The modeler can choose the set of elementary operations to fit the problem at hand. It is assumed that an elementary operation is carried out in one unit of time. Every computation is described as a superposition of elementary operations. Choice of the set of elementary operations permits limitations on computing capabilities to be expressed formally. It also gives the modeler control over the level of reduction of analysis. The topology of the directed graph can also be restricted by the modeler, though in this book we do not do so; instead we assume that any directed graph is available.

These features facilitate the application of the model to human agents and to economic models. Parallel and distributed computing and the dispersion of information among agents are naturally expressed in this model. In each application the modeler can choose the set of elementary functions to suit the problem. The class of elementary operations may include functions of real variables, as well as functions whose domains are discrete sets, as, for instance, functions defined on strings from a finite alphabet. When the alphabet is finite and the elementary functions are Boolean, the model is equivalent to the finite-state automaton model.

Computing with real numbers offers some important advantages in the context of scientific computing (see Blum et al., 1998). It is also relevant to applications in economic theory. Economic models typically use real variables and functions of them. A model of computing in which the elementary operations are functions of real variables allows that model to be directly applied to

standard economic models, without requiring an analysis of approximations in each application. In cases in which the analysis in the economic model is itself numerical, then, as is the case with numerical analysis generally, computation is necessarily finite and typically is carried out by persons who use a finite-state machine. This raises a question about the relationship between the analysis of complexity in our model when real variables are involved and the complexity of the corresponding computation performed by a finite-state automaton that uses a finite alphabet. Instead of analyzing this question case by case, we show (in Chapter 6, Theorem 6.2.1) that the measure of complexity of a function obtained from the continuum model (real variables and smooth functions) is a limit of a sequence of measures of complexity obtained from finite networks, equivalent to sequential machines, computing approximations to the smooth function in question. The limit theorem presented in Chapter 6 shows that when regularity assumptions are satisfied, our model of computation, and the measure of complexity obtained in it, is an idealization of finite computing in the same sense in which measurement with real numbers is an idealization of measurement with rational numbers.

Real computing opens connections to classical mathematics. When computing is done over a finite alphabet, the technical machinery of analysis available is combinatorial mathematics, which is difficult and, in the setting of standard economic models, awkward. In contrast, when the alphabet is the real numbers, or real vectors, the apparatus of classical analysis is available. In this book the class of elementary operations is often taken to be the class of functions of at most r variables, each a d-dimensional real vector, whose value is also a vector in a d-dimensional Euclidean space. In terms of machines, the parameter d can be thought of as the *size* of a register, and the parameter r as the number of registers. When a human being is doing the computing, the parameter d may refer to the number of modalities of perception via the senses. Smoothness conditions are sometimes imposed. A person typically receives visual, auditory, and other sensory inputs simultaneously. In our model, the number of these is d. Further, a person can perceive more than one input at a time, but not very many. In our model the number of these is r. According to a classic paper (Miller, 1956) for a human being the number r is approximately seven. A modular network whose elementary functions satisfy the restrictions imposed by r and d is called an (r, d) network. Much of the analysis in this book concentrates on analysis of (r, d) networks.

How can an (r, d) network represent computations performed by a system consisting of human beings and machines? When the class of elementary functions consists of functions between Euclidean spaces (or between smooth manifolds), it is not obvious that the (r, d)–network model can represent computations performed by human beings, or by a combination of people and machines. However, we can extend the model so that more abstract computations can be reduced to computations with real quantities. For this purpose

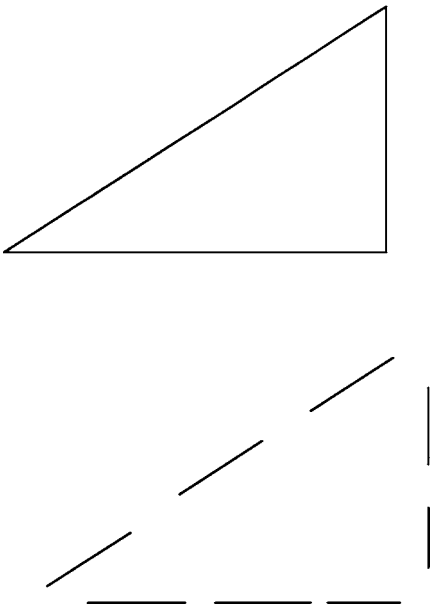

Figure 1.1.1.

we introduce the idea of an *encoded version of a function*, and its computation. The formal definition appears in the first section of Chapter 4. Here we make do with an informal description sufficient to understand the example that follows.

Human beings are good at detecting patterns in low dimensions. For instance, people have little trouble recognizing the pattern shown in Figure 1.1.1 as a triangle.

We continue to recognize a triangle even if the corners, and perhaps other pieces of the perimeter, are missing, or if the sides are slightly curved. Recognizing a pattern can be thought of as computing a function that expresses the relation between a subset of the plane, and the act of saying that it is a particular pattern, in our example a triangle. Thus, recognizing a pattern is represented as evaluating a function, ρ, whose value at the subset shown in Figure 1.1.1 is the word *triangle*. The domain of ρ is the set of subsets of the plane, not a Euclidean space, and the range of ρ is the set of English words, or some suitable subset of it, also not a Euclidean space. In the case of pattern recognition by a human being, it is natural to consider ρ to be an elementary function. When it is possible to encode the more abstract domain and range of the function in terms of elements of Euclidean spaces, then, as Figure 1.1.2 shows, evaluating ρ becomes equivalent to evaluating a function, h, whose domain and range are Euclidean spaces. Although h may be complex if evaluated by a machine,

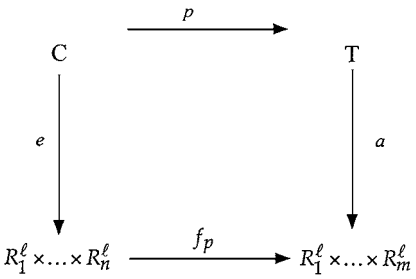

Figure 1.1.2.

when ρ is elementary for a human agent, and the (fixed) encoding and decoding functions are elementary, the function h becomes an elementary function in a modular network model that applies to the system consisting of human agent and machine. Figure 1.1.2 shows the scheme.

More broadly, the standard models of computer science are not convenient for expressing the characteristics of human beings, in particular, economic agents, as information processors. A machine can easily do some things that people find difficult. Other things are easy and natural for people and difficult for machines. Machines have no special difficulty in handling high-dimensional data, whereas human beings do. Human beings are good at recognizing patterns in low-dimensional data, whereas that task can be quite difficult for a machine. Reading handwriting is relatively easy for literate human beings, and quite complex for machines. The widespread and heavy use of computers to generate two-dimensional visual representations of high-dimensional or complex data so that a human being can study them graphically, rather than look at a printout of the underlying function in the form of numerical data, testifies to the special ability of humans to see patterns in low dimensions; it also testifies to the limitations of humans to comprehend the same information presented in another form. The standard models of computing represent these tasks independently of who is to perform them, and the complexity of a task is the same whether the computation is done by a human or a machine. A model in which reading handwriting can be a simple task if done by a human being and a complex one if done by a digital computer would come closer to capturing computation done by persons equipped with machines.

Computers are sometimes used directly by human beings, who determine and control their inputs, and receive and use their outputs. (Some computational devices are internal to other machinery, such as the computers used to control automobile engines and the like. In these cases the interaction with humans is more remote.) In some cases when a human being is an active participant in the course of a computation, with the result that the whole computation depends on both the actions of the machine and of the person, it is nevertheless possible to focus entirely on the analysis of the computation performed by the machine.

But when, as in some economic models, the object of the analysis includes the person, it is not appropriate to separate the person from the machine. When these situations are important, there is need for a model of computation that facilitates analysis of computational problems to be solved by a combination of human and machine resources. The modular network model allows operations that can be carried out by a person to be included in the set of elementary functions. When an encoded version of such an elementary function exists, it can be included in the set of elementary functions of an (r, d) network. Then the analysis of the entire computational task of human and machine can be modeled and analyzed seamlessly. (Analysis of some examples of computations performed by a combination of humans and machines are presented in the first section of Chapter 4. Also see Mount and Reiter, 1998.) Of course, the entire computation might also be represented in one of the classical models of computing, a Turing machine, or a finite-state automaton. Some work along this line is in the literature. The finite-state automaton model and the Turing machine model have been used in game theory (Neyman, 1985), where computational issues also arise. In game theory, as in economic theory generally, it is assumed that players are fully rational and have unlimited computational abilities and resources. These assumptions provide a basis for deep analysis of situations in which the interests of agents may run together, and also conflict to some extent. As in economic theory, there are attempts to weaken the assumptions of full rationality and unlimited computational capabilities. The finite-state automaton model has been used to analyze the complexity of strategies, and the Turing machine model has been used to study the complexity of games. The questions addressed are, for a given game: "How difficult is it for a player to carry out a given strategy in that game?" and: "How difficult is it to solve the game?"[1] Regarding the first question, Aumann (1981) suggested using finite-state automata to measure the complexity of strategies. The measure is the number of states of the smallest automaton that can execute the strategy. (see Gilboa and Zemel, 1989). This is called *strategic complexity*. Infinitely repeated games have been studied in this way (Rubenstein, 1979, 1986; Abreu and Rubenstein, 1988; and Cho, 1995). There is substantial literature in which finite-state automata or Turing machines are used to restrict strategic complexity (Kalai and Stanford, 1988; Kalai, 1996). With respect to the second question, the standard model of computational complexity in computer science, namely the classification of computations based on the asymptotic theory of complexity classes, has been used to analyze the complexity of solving games. (Also see Ben-Porath, 1989; Kalai et al., 1993; Papadimitriou, 1992.) Ben-Porath (1986) showed that there is an advantage to a player in a repeated two-person zero-sum game with patient players from having a larger automaton. The

[1] Kalai (1996) surveys the literature relevant to these questions and provides a bibliography listing the basic papers.

advantage can be considerable, but the automaton must have an exponentially larger number of states. We address these questions in Chapter 5 in the context of the modular network model of computing.

The idea that human cognition is modeled well by Turing machines, or finite-state versions of them, is widely asserted, but it remains controversial.[2]

There is another issue that deserves a brief comment here. An economic agent experiences his environment directly. How that agent represents his environment in his own mind is usually not observable by others. The economic analyst considering the agent's behavior in his environment "constructs" her own model of the situation in which she can deduce the optimal action of the agent. We apply the (r, d)-network model to the analyst's model in order to analyze the computational complexity of the decisions of the agent. Because the analyst's choice of how to model the agent's situation may have arbitrary elements in it, the measure of complexity might reflect those arbitrary elements, and consequently might be misleading. It is therefore important that the model of computation and the complexity measure it defines do not depend on a particular parameterization of the problem. We want the measure of complexity to be invariant with respect to transformations of the problem that preserve its essential elements. Specifically, we want the measure of complexity to be the same under coordinate changes in the spaces of the variables that define the computation. The methods of analysis in Chapter 3 are coordinate free; in Chapter 4, where the model is applied to analyzing the trade-off between communication and computation in finding the equilibrium of a decentralized message process, we show explicitly that the result is invariant under appropriate coordinate transformations of the underlying spaces.[3] There is a second reason why invariance of the measure of complexity under coordinate transformations is important. Changing coordinate systems can implicitly perform computations without our taking explicit notice of them. For instance, to solve a system of linear equations without doing any work, just assume that the coordinate system is one that makes the matrix of the system diagonal. There are also other ways of "smuggling" computation, but these are ruled out in our analyses by regularity conditions.

[2] Put briefly, the assertion is that the human brain (mind) is a Turing machine. Among others, Roger Penrose does not subscribe to this assertion. His book (Penrose, 1994) is an excellent guide to the issues and to the literature. It is not necessary for our purpose here to take a position on this question. Even if it were the case that the brain is a Turing machine, it would not necessarily be useful in applications to economic agents or economic organizations to model them as executing algorithms in which the elementary steps are evaluations of Boolean functions over a finite alphabet.

[3] This is done in steps, beginning with linear coordinate transformations, and ending with general nonlinear coordinate transformations. We show the invariance under linear coordinate transformations explicitly. The proof for nonlinear transformations is tedious and does not lead to any new insight. We therefore refer the reader to Mount and Reiter (1990), where it is presented in full.

1.2. COMPLEXITY, MATHEMATICS, AND HUMAN CAPACITIES

There is a direct connection between the (r, d)–network model and certain classical problems in mathematics. For instance, when we restrict attention to analytic functions, computation of a function F by an $\mathcal{F}$ network, where $\mathcal{F}$ is taken to be the class of analytic functions of two variables, is related to Hilbert's 13th problem. That problem asks whether an analytic function of several variables can be expressed as a superposition of analytic functions of two variables. This is the same as asking whether an analytic function of several variables can be computed by a (2, 1) network whose modules are analytic functions. There is literature stemming from Hilbert's 13th problem that includes contributions by Kolmogorov and others, such as Kolmogorov (1961a, 1961b) and Arnol'd (1963); also see Lorentz (1966). Kolmogorov first showed that each continuous function of n real variables could be written as a superposition of functions of three variables. Arnol'd showed that only functions of two variables are required. Kolmogorov refined this result and showed that each continuous function of n variables could be written as a superposition of continuous functions of one variable and the binary function of addition. In general, the functions required for superposition, besides addition, are not differentiable. The situation is more complicated when the functions in the superposition are required to be smooth. It is known that there are s times differentiable functions of n variables that cannot be written as a finite superposition of s times differentiable functions of fewer variables (see Lorentz, 1966 or Vitushkin, 1961). In this book we work mostly with elementary functions that are twice continuously differentiable, d-vector–valued functions of r variables, each a d-dimensional real vector. Sometimes real analytic functions are used as elementary functions, as in the paper by Mount and Reiter (1998). That paper presents some of the ideas of our model in a less technical setting, and it also presents some applications of the model to human computing, specifically to Chernoff faces in pattern-recognition problems (Chernoff, 1973).

In our model, *computability* and *complexity* are *relative* concepts. The complexity of a given computation can be different depending on the class $\mathcal{F}$ of elementary functions (and, if relevant, the class of graphs permitted). Consider a polynomial of degree ten in one variable, and consider the function that associates the array of its roots with the vector of its coefficients. A person who knows the coefficients and must compute the roots can be in one of at least three situations. She may not have a computer available, or she may have a computer but not have a program for the task, or she may have a computer equipped with a program for computing the roots of polynomials from the coefficients, for example, the program Gauss, or Mathematica. A person without access to a computer, or one using a personal computer that lacks a program for that purpose, would find this a time-consuming task – a complex task. However, that same person using a computer equipped with Gauss or Mathematica could accomplish the task with a few keystrokes. In that case it would be sensible in

many situations to regard the function that associates the roots to the coefficients as an elementary operation, and not pursue the analysis to more detailed levels of the program doing the work. To require that every computation be reduced to a fixed given level of elementary operations, such as Boolean functions over a finite alphabet, results in a model that is awkward to apply to computations done in the context of economic problems by economic agents or by economic theorists, whether they do or do not have access to computing machines.

The idea that complexity is a relative concept is in keeping with practice in mathematics. There the notion of solution to a problem is also relative. For example, what does it mean to solve a differential equation? Among other possibilities, it can mean to find an integral, perhaps expressed in some abstract form, or it can mean to find the solution trajectories numerically. The complexities of these two tasks can be quite different. In mathematics a problem can be considered as solved in cases that are quite different. For instance, a problem might be considered solved if it is shown to be equivalent to another problem for which a solution is known, or to one that is known to have a solution, or to one for which there is an algorithm.

1.2.1. Complexity and Computability

In the $\mathcal{F}$-network model, the complexity of a function F relative to the class $\mathcal{F}$ ($\mathcal{F}$ complexity) is the minimum over all $\mathcal{F}$ networks $\mathcal{N}$ of the number of sequential steps in the longest path in the network $\mathcal{N}$. We also refer to this as the *time* it takes $\mathcal{N}$ to compute F. When we take account of the resources used to evaluate elementary functions, the time can vary depending on the assignment of elementary operations to agents (see Chapter 7). If the $\mathcal{F}$ complexity of F is infinite, then F is not $\mathcal{F}$ computable; that is, it is not computable by networks with modules in the class $\mathcal{F}$. We will sometimes refer to the complexity of F, omitting reference to the class $\mathcal{F}$ when it is clear which class is being used.

Note that the complexity of a function depends only on the class of elementary functions, and not on a particular algorithm that might be used to compute it. In some cases, in which the functions being computed are smooth and the set of elementary functions is appropriately specified, we are able to give lower bounds, sometimes exact, on the complexity of a function F in terms of properties of F alone, that is, independently of the algorithms used. This means that the complexities of, say, different polynomial or rational functions (functions in the standard complexity class P) can be compared without having to count the steps of the algorithms being used to compute them. In the case of smooth functions between Euclidean spaces (or, more generally, smooth manifolds) the lower bound on computational complexity is determined by the number of variables that the function *actually* depends on (as distinct from the number of its nominal arguments). The calculation of the lower bound does not require

specification in detail of the computations performed by any algorithm that computes the function. This analysis is presented in Chapter 6.

One of our aims in proposing the modular network model is to provide a formal structure in which to study the process of figuring out what actions are best, or at least desired, for agents in given situations, when that process is subject to constraints that in one way or another impose costs. Thus, our approach can be located within the rational-choice paradigm. It is a model of rational choice when there are costs of figuring out what to choose. Radner has called this approach one of costly rational choice (Radner, 1997, 2000). This is rational choice in a model that includes constraints on information processing (see Rubinstein, 1998, p. 22; Gilboa and Samet, 1989).

The idea that elementary operations can be chosen at the discretion of the analyst may appear objectionable. Wouldn't it be sounder to identify the fundamental capabilities of human beings and to base the set of elementary operations on them? If the level of resolution they determine is too fine, then higher-level compositions of them could be formed in a way analogous to the way computer instructions in a high-level language are ultimately related to machine language instructions. A difficulty with this idea is that if we mean by the inherent capabilities of human beings those that are genetically determined, then those capabilities are likely the same today as they were fifty thousand years ago. However, we know that there are information-processing tasks that are elementary for people in contemporary societies but were impossible for people who lived only a few hundred years ago. It seems appropriate that the difficulty of an information-processing task should depend on the currently operative capabilities, recognizing that those capabilities change from time to time.

1.3. COMPUTING AND ECONOMIC ORGANIZATION

Historically, the main analysis of economic organization and of the coordination of economic activity has been General Equilibrium Theory. Adam Smith's insight into the workings of a market economy was that persons – economic agents – motivated by self-interest and guided by market prices are led to take actions that produce socially efficient outcomes. *Socially efficient* has come to mean *Pareto optimal*. In the 1950s this insight was given a rigorous but somewhat incomplete formal treatment, known as the classical theorems of welfare economics. The First Welfare Theorem asserts that in a certain class of economic environments, a competitive equilibrium corresponds to or determines an efficient social state. This result makes Smith's insight rigorous, but it is incomplete in several ways. First, *market economy* means perfectly competitive markets, in which all economic actors are price-takers who do not consider the effects of their actions on the actions of others. In economies with finitely many agents, price-taking behavior is not in general consistent with fully rational self-interest. Second, the proposition that prices established in competitive markets

suffice to guide economic action to socially efficient outcomes is not true in all economic environments, partly because equilibrium competitive prices do not exist in some environments, and partly because in some environments where they do exist they do not necessarily correspond to socially efficient outcomes. For instance, the existence of significant indivisibilities in commodities can allow nonefficient competitive equilibria. Furthermore, in environments with externalities, competitive market prices are generally not adequate to guide economic agents to socially efficient actions.

Even in environments in which the First Welfare Theorem is valid, it does not adequately support the claim that competitive prices guide economic agents to take socially efficient actions. The theorem tells us that if the economic agents are taking competitive equilibrium actions, then the outcome will be efficient. It says nothing about how the market process will find a competitive equilibrium even when it is unique. (Competitive equilibria are not in general unique, even in the most classical environments.) Thus, there must be some process that arrives at and selects a particular equilibrium, even when the equilibrium prices are different, and even more so when different equilibria are associated with the same prices.

One of the important properties claimed for the market organization is that it economizes on the information that agents need to know, and on the need to transfer information among agents. It is therefore essential that the process of finding an equilibrium from a possible multiplicity of them does not compromise this claim. In the case of a single market, the idea of adjusting the price to reduce the difference between supply and demand was known to the English classical economists, and in the case of multiple markets, Walras' *tatonnements* extended the idea to simultaneous adjustments of all prices. Pareto (1927, pp. 233–34) compared the market mechanism to a machine for computing equilibria. The economic environment was viewed in a way that modern computing theory would call *distributed*. Memory or observation of the environment is local; each individual economic agent observes only that part of the environment that pertains to him, such as his own preferences, or production possibilities. All other information must be communicated among the agents. The computation was conceived as an iterative process of price adjustment beginning in some state of the economy and eventually converging to a competitive equilibrium price. In environments in which the equilibrium corresponding to a given price is unique, and in which the First Welfare Theorem is valid, we are assured that finding a competitive equilibrium is equivalent to finding an efficient profile of actions and hence an efficient economic allocation.

This approach was formalized by Samuelson (1947, pp. 269–75) in terms of systems of differential equations. It was subsequently shown that the class of environments for which systems of differential equations such as the ones proposed by Samuelson converge even locally is quite restricted (see Scarf, 1960; Arrow and Hurwicz, 1958; and Arrow et al., 1959). More elaborate