A Survey of Continuous-Time Computation Theory

Pekka Orponen
Department of Mathematics
University of Jyväskylä*

Abstract

Motivated partly by the resurgence of neural computation research, and partly by advances in device technology, there has been a recent increase of interest in analog, continuous-time computation. However, while special-case algorithms and devices are being developed, relatively little work exists on the general theory of continuous-time models of computation. In this paper, we survey the existing models and results in this area, and point to some of the open research questions.

1 Introduction

After a long period of oblivion, interest in analog computation is again on the rise. The immediate cause for this new wave of activity is surely the success of the neural networks "revolution", which has provided hardware designers with several new numerically based, computationally interesting models that are structurally sufficiently simple to be implemented directly in silicon. (For designs and actual implementations of neural models in VLSI, see e.g. [30, 45]). However, the more fundamental explanation for this development is that as hardware technology has advanced, it has become generally easier to experiment with and manufacture individualized, *special-purpose* computational devices, as opposed to the mass production of general-purpose processors and memory chips. This trend will continue and become even more noticeable in the future, making all manners of special-purpose computational models practically as well as theoretically interesting objects of study.

There will thus be an increasing need of a theoretical understanding of the capabilities, limitations, and effectiveness of various kinds of special-purpose computational models: a challenge directed right at the heart of computational complexity research. While the present paper concentrates on the theoretical issues in analog computing, a similar situation exists, or can be foreseen, in e.g. cellular automata [17] and other "complex systems" models of computation (e.g. [36, 40]), molecular computing [1, 28, 42], optical computing [16, 41], and — somewhat futuristically — quantum computation [6, 49].

While the work on analog computation theory goes back at least to Claude Shannon's papers in the early 1940's [46, 47], the literature is not very extensive, and also does not answer many of the questions that would appear most interesting from a present-day perspective. For instance, while in recent years quite a number of papers have appeared on the computational aspects of discrete-time analog models (e.g. [3, 7, 13, 24, 25, 29, 31, 36, 51]), the number of papers on the implementationally more significant continuous-time models is far fewer. (And

*P. O. Box 35, FIN–40351 Jyväskylä, Finland. E-mail: orponen@math.jyu.fi.

the amount of work on *practically* implementable models is close to nil. In particular, all of these papers mentioned above except [29] ignore the effects on the computing process of *imprecision* and *noise*, two of the most pervasive practical problems in analog computation.) And finally, while some work exists on computatibility in continuous-time systems, there is practically none on the computational *complexity* aspects — even such basic notions as computation time, input size, system size, etc. are still waiting for their proper, reasonably general and implementation-independent definitions.

In this paper we survey the existing research on the general computability and complexity aspects of continuous-time computation models. In Section 2 we outline some mathematically-based, implementationwise unconstrained models [3, 8, 9, 10, 11, 33, 44]. (We are staying here at the level of individual, concretely specified models. In particular, an important collection of results we do *not* cover, although it properly would belong to this context, is the work of Pour-El and Richards [39] on the general computability theory of common analytical and physical operators.) In Section 3 we move to models that have arisen from some real or idealized physical implementations, such as mechanical or electrical differential analyzers [46, 38, 43], or electronically implemented neural networks [20, 34]. In Section 4 we discuss the few existing papers on computational complexity issues. We conclude in Section 5 by listing some of the main open research directions.

2 Unconstrained Models

By a *continuous-time (analog) system* we generally mean an n-dimensional system of autonomous ordinary differential equations (ODE's) of the form

$$\frac{dx}{dt} = f(x), \tag{1}$$

where $f : \mathcal{R}^n \to \mathcal{R}^n$ is the *field* defining the system. If the field f is sufficiently smooth (e.g. continuously differentiable), then the system (1) determines a unique *flow* on $\mathcal{R}^n$, i.e. a function $\phi : \mathcal{R}^{n+1} \to \mathcal{R}^n$ such that for any $x \in \mathcal{R}^n$, $\phi(x, 0) = x$ and for all $\tau \in \mathcal{R}$,

$$\frac{d}{dt}\phi(x,t)|_{t=\tau} = f(\phi(x,\tau)).$$

(To be precise, the existence of a flow may only be guaranteed for τ contained in some interval $I \subseteq \mathcal{R}$ [19].) There are different ways of defining a notion of computation in this context, some of which we shall discuss below. (It is e.g. by no means obvious how one should present the "inputs" to such a system, or how to read the "outputs.") A *discrete-time (analog) system* is defined similarly by an iterated map of the form $x_{n+1} = f(x_n)$.

We begin by pointing out that any sufficiently regular discrete-time analog system can theoretically be embedded as a Poincaré section (a "snapshot sequence") of a higher-dimensional continuous-time analog system. (Specifically, a system representable as a diffeomorphic map of the interval $[0,1]^d$ to itself can be embedded as a section of a smooth flow on some $(d+1)$-dimensional manifold [37, p. 111].) Thus, for instance, the Turing machine simulations by iterated piecewise-linear maps on $[0,1]^2$ presented by Moore [31, 32] and Koiran et al. [25] can in principle be extended into smooth, locally three-dimensional systems. (Technically, one needs to require here that the Turing machines to be simulated are *invertible*, but it is well known [4, 5] that any Turing machine can be converted into an invertible one.) Moore

in his paper [32] in fact discusses the issue of continuous-time embedding at some length, and in [31] even presents a quasi-physical "billiard-ball" model for implementing the resulting continuous-time system. In this survey, however, we shall concentrate on explicitly-given continuous-time systems, and not discuss discrete-time systems any further.

We also mention only in passing the simulation of Turing machines by three-dimensional continuous-time systems with piecewise-constant derivatives presented by Asarin and Maler in [3]. A technical point worth noting, though, is the general proof strategy used by both Asarin and Maler, and most of the other recent authors of Turing machine simulations by dynamical systems (e.g. [8, 25, 51], implicitly also [31]). One takes as the starting point the standard correspondence of Turing machines and two-stack pushdown automata. The Turing machine tape is first represented as two opposing stacks, and then the contents of these stacks are encoded in some manner as two real numbers, leading to a representation of the system state as a point in $\mathcal{R}^2$ (usually constrained further to $[0,1]^2$). One then needs an additional dimension to connect the states in a way that corresponds to the Turing machine transition function. (And, in fact, both Asarin and Maler [3] and Koiran et al. [25] prove that, within the class of systems they consider, two dimensions are not sufficient for a continuous-time simulation.)

The same basic idea of two-stack machine simulation is the starting point of the straightforward, but robust and quite elegant simulation of Turing machines by continuous-time systems with continuous vector fields presented by Branicky in [8, 9]. By his technique one can either obtain a simulation by Lipschitz-continuous systems in $\mathcal{R}^5$, or by non-Lipschitz systems in $\mathcal{R}^3$.

Let us briefly outline Branicky's construction in $\mathcal{R}^5$. Given a Turing machine M with at most p tape symbols and states, one represents the instantaneous configuration of M as a pair of integers (x_L, x_R), encoding the tape contents of M to the left and to the right of the tape head, respectively, in a p-adic encoding. (The current state q of M can be encoded as, say, the lowest-order digit of x_R, so that $q = x_R \bmod p$.) The transition function of M then determines a "finite-gain" discrete-time mapping $f(x_L, x_R) = (x'_L, x'_R)$, where by finite gain we mean that there is some constant $M > 0$ such that for any $x \in \mathcal{R}^2$, $\| f(x) \| \leq M \| x \| + M$. Branicky's reason for using integer instead of real-number encodings as in [25, 31, 51] is that this gives the system some degree of robustness against small perturbations.

Now one's first attempt at a continous-time simulation of the discrete-time system given by f might be to define a system with state variables $x_L, x_R \in \mathcal{R}$, and with system equations

$$\begin{aligned} \frac{dx_L}{dt} &= -x_L + f_L(x_L, x_R), \\ \frac{dx_R}{dt} &= -x_R + f_R(x_L, x_R). \end{aligned}$$

This approach to "updating" the state variables does not work, however, because the variables x_L and x_R do not maintain their "old values" while the "new values" are being computed. Branicky solves this dilemma by defining a "two-phase" continuous-time system, introducing an extra pair of state variables $\tilde{x}_L, \tilde{x}_R \in \mathcal{R}$, and an explicit time variable $\tau \in \mathcal{R}$. Using the time variable he then defines two periodic "clock" functions, $S_+(\tau)$ and $S_-(\tau)$, whose values oscillate alternatingly between 0 and 1. Now the state variables can be coupled together so that when the S_+ clock is "high," the new values $(\tilde{x}_L, \tilde{x}_R) = f(x_L, x_R)$ are computed, and when the S_- clock is high, these new values are simply copied from $(\tilde{x}_L, \tilde{x}_R)$ to (x_L, x_R).

Formally (and ignoring some technical details), the continuous-time system is then constructed as follows. First one defines the clock functions $S_{\pm}(\tau)$ as

$$S_{\pm}(\tau) = h(\sin(\pm\pi\tau)),$$

where

$$h(r) = \begin{cases} 0, & \text{for } r \leq 1/4, \\ 4r - 1, & \text{for } 1/4 < r \leq 1/2, \\ 1, & \text{for } 1/2 \leq r \leq 1, \end{cases}$$

and then connects the state variables (roughly) as follows:

$$\begin{aligned} \frac{d\tilde{x}_L}{dt} &= (-\tilde{x}_L + f_L(x_L, x_R)) \cdot S_+(\tau), \\ \frac{d\tilde{x}_R}{dt} &= (-\tilde{x}_R + f_R(x_L, x_R)) \cdot S_+(\tau), \\ \frac{dx_L}{dt} &= (-x_L + \tilde{x}_L) \cdot S_-(\tau), \\ \frac{dx_R}{dt} &= (-x_R + \tilde{x}_R) \cdot S_-(\tau), \\ \frac{d\tau}{dt} &= c \quad \text{(constant)}. \end{aligned}$$

One aspect of the construction we are ignoring is how to make the time τ run "slowly enough," i.e. how to choose the appropriate constant c so that the other state variables can complete their updates within one clock period. We refer the reader to the original papers [8, 9] for details such as this.

Branicky credits Brockett [10, 11] for first introducing this two-phase trick, in a more specialized context. However, while the construction technically achieves its purpose, it is conceptually somewhat unsatisfactory as it basically digitizes the analog system. (At a fundamental level, our "digital" computers are clocked analog systems too, although with a potentially infinite number of state variables.) It would be of interest to understand the computational capabilities of continuous-time systems without clocks, even if this notion may be difficult to make precise. (One possible, although maybe too restrictive condition would be to require that the system possess a Liapunov function whose value is bounded from below and decreases in time along every system trajectory [19].)

The trick to go from the above presented five-dimensional Turing machine simulation to a three-dimensional simulation [8, 9] is to simply first encode the configuration pair $(x_L, x_R) \in \mathcal{Z}^2$ as a single integer $x = 2^{x_L}3^{x_R}$, and then continue as before. Unfortunately, using this prime-power encoding destroys the finite-gain property of the discrete-time transition mapping, and consequently the resulting continuous-time system will be non-Lipschitzian. It seems to be an open question whether Turing machines can be simulated in three dimensions by robust, Lipschitz-continuous systems. (The continuous embeddings of the piecewise-linear maps of [25, 31, 32] ought to satisfy the Lipschitz condition; however these systems are sensitive to arbitrarily small perturbations.)

Computability by *partial* differential equations has been studied by Omohundro [33] and Rubel [44]. Given an arbitrary two-dimensional cellular automaton M with the Moore (i.e., nine-neighbor) neighborhood, Omohundro constructs a system of ten coupled nonlinear PDE's, with two space variables and one time variable for simulating M. Since two- (and even

one-) dimensional cellular automata can simulate Turing machines [17], Omohundro's construction establishes that also continuous-time PDE's are universal computational systems. However, the simulation is again rather digital, the idea being to evolve localized "bumps" in the XY–space, whose height indicates the state of the simulated automaton at each cell position.

Rubel [44], on the other hand, in his work on the "extended analog computer" (EAC) is interested in the production of real functions as solutions to systems of differential equations set up in a certain systematic "quasi-effective" manner. He starts from the "general-purpose analog computer" (GPAC) systems defined by Shannon [46, 47] and Pour-El [38] (see below), and extends these with the capability to solve boundary-value problems for systems of PDE's defined by lower-order versions of the EAC. Rubel proves that the EAC is a quite powerful generation model, being able to produce many functions which are beyond the GPAC model (for some examples, see below).

3 Constrained Models

Let us then move to analog computation models that correspond to idealized versions of existing devices — which of course does not mean that arbitrary computations in these models could be precisely implemented by any real physical hardware.

The earliest theoretical study of the computational capabilities of analog devices seems to have been Shannon's 1941 [46] work on the generative power of Bush's Differential Analyzer [12]. Differential analyzers can be built either mechanically, out of rotating gears and shafts connecting them, or electronically from resistors and capacitors. Bush's original machine was (electro-)mechanical; electronic differential analyzers were developed during World War II initially for fire control purposes, and were then widely used in engineering until the 1960's. We shall discuss here only very briefly the electronic version of the device; for more details we refer the reader to the large literature on the solution of engineering problems on differential analyzers, e.g. any of the textbooks [18, 22, 23, 26].

Electronic differential analyzers are constructed by interconnecting resistors, capacitors, and high-gain operational amplifiers in a systematic manner. Recall that, given a time-varying input voltage $u(t)$, a resistor of resistance R creates (passes) a current $i(t) = u(t)/R$, and a capacitor of capacitance C creates a current $i(t) = C\frac{du(t)}{dt}$. Operational amplifiers act as simple (large) constant voltage multipliers. In particular, since voltage differences across capacitors correspond to integrals of capacitative currents, they can, with the help of resistors and amplifiers, also be used to perform integration over input voltages. Figure 1 illustrates the design of an integrator which, for input voltage $u(t)$, gives the response $v(t) = \frac{-1}{RC}\int u(t)\,dt$, assuming the gain of the amplifier A is very large.

Abstractly, an electronic differential analyzer can be viewed as consisting of the following kinds of computational devices, interconnected into a possibly cyclic network [38]:

1. *Integrator.* A two-input, one-output device producing from input functions $u(t)$, $v(t)$ the output function $\int u(t)\,dv(t) + C$, where C is a constant whose value depends on the initial settings of the device.

2. *Constant multiplier.* A one-input, one-output device producing from input $u(t)$ the output $Cu(t)$, where C is an arbitrarily chosen real constant.

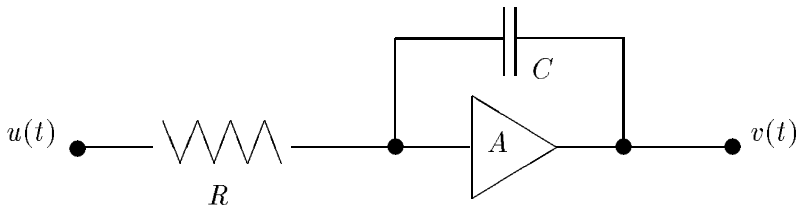

Figure 1: An electronic integrator.

3. *Adder.* A two-input, one-output device producing from inputs $u(t)$, $v(t)$ the output $u(t) + v(t)$.

4. *Variable multiplier.* A two-input, one-output device producing from inputs $u(t)$, $v(t)$ the output $u(t) \cdot v(t)$.

5. *Constant function.* A one-input, one-output device producing from input $u(t)$ the output $C_1(t) \equiv 1$.

Among these devices, the variable multiplier is in fact redundant, because it can be implemented as

$$u(t) \cdot v(t) = \int_0^t u(\tau)\, dv(\tau) + \int_0^t v(\tau)\, du(\tau) + u(0)v(0).$$

Pour-El [38] (and already Shannon [46] in the context of the mechanical differential analyzer) observes that a real function $u(t)$ can be generated from the input t on some interval $[0, T]$ by a network of devices of the above types, if and only if there is a system of ODE's of the form

$$\frac{du_i}{dt} = \sum_{j,k=0}^{n} a_{ijk} u_j \frac{du_k}{dt}, \quad i = 2, \ldots, n, \tag{2}$$

where $u_0(t) = 1$, $u_1(t) = t$, together with initial conditions $u_i(0) = u_i^0$, such that the system has a unique solution $(u_2(t), \ldots, u_n(t))$ for $t \in [0, T]$, and $u(t) = u_n(t)$ on this interval.

If one in addition requires that the system (2) possess a "domain of generation," meaning that any sequence of initial values sufficiently close to $(u_2^0, \ldots, u_n^0)$ gives rise to a locally unique solution, then one can show that this class of "GPAC-generable" functions $u(t)$ in fact coincides with the class of *differentially algebraic* functions, i.e. those that satisfy some *algebraic differential equation* of the form

$$P(t, u, u', u'', \ldots, u^{(n)}) = 0,$$

where P is a nonzero polynomial in all its variables. (This result was first claimed by Shannon [46], then by Pour-El [38], who argues that Shannon's proof was seriously incomplete, and finally by Lipshitz and Rubel [27], who also claim to have discovered a gap in Pour-El's reasoning.)

One corollary of this characterization of the GPAC-generable functions is that some interesting functions already known to be not differentially algebraic are thus shown to be not generable by differential analyzer -type analog computers. These include, e.g. [46] the Gamma function $\Gamma(s) = \int_0^\infty t^{s-1} e^{-t} dt$, and Riemann's Zeta function $\zeta(s) = \sum_{k=0}^\infty \frac{1}{k^s}$. However, these two functions *can* be generated in Rubel's EAC-model discussed above [44].

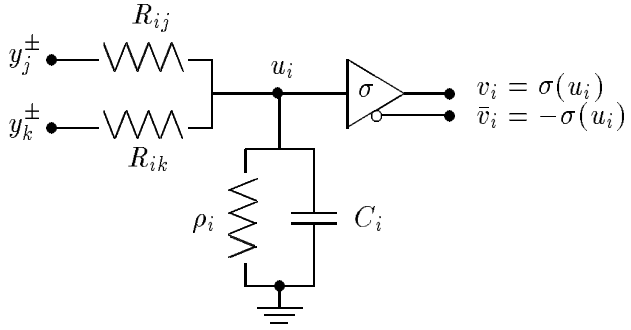

Figure 2: Hopfield's electronic neuron.

A different approach to continuous-time computation is taken in the electronically-based "neural network" model proposed by Hopfield in 1984 [20]. Here the basic computational unit is an electronic "neuron," schematically shown in Figure 2.

As Figure 2 llustrates, Hopfield-type neurons are again constructed from resistors, capacitors, and amplifiers; however this time the amplifiers are assumed to have some saturating nonliner response characteristic σ such as $\sigma(u) = \alpha \tanh(u) + \beta$. A "continuous-time Hopfield network" then consists of some finite number of interconnected units of this type.

Let us consider the behavior of a neuron i in a network of n such neurons. Let ρ_i and C_i be the input resistance and capacitance, respectively, of the amplifier at neuron i. Denote the input voltage of the amplifier by u_i, and the output voltage by v_i. In order to establish also inhibitory interconnections between neurons, also the inverted output voltages $\bar{v}_i = -v_i$ are needed. (For simplicity, we indicate voltages relative to some reference level V_0.)

The neuron i, as shown in Figure 2, draws input from other neurons (indicated as j and k in the Figure) via resistors, whose resistances are denoted by R_{ij} and R_{ik}. The voltages $v_j^{\pm}$ and $v_k^{\pm}$ are obtained from the appropriate output terminals of neurons j and k, depending on whether the interconnections are excitatory or inhibitory.

The circuit equations for a network of n neurons can now be written as

$$C_i \frac{du_i}{dt} + \frac{u_i}{\rho_i} = \sum_{j=1}^{n} \frac{1}{R_{ij}} (v_j^{\pm} - u_i), \qquad \text{for} \quad i = 1, \ldots, n. \tag{3}$$

By choosing the circuit parameters appropriately and normalizing the RC constants to 1, one can use such a network to implement any system of first-order nonlinear differential equations of the form

$$\frac{du_i}{dt} = -u_i + \sum_{j=1}^{n} h_{ij} \sigma(u_j), \qquad i = 1, \ldots, p. \tag{4}$$

(One essentially chooses $R_{ij} = 1/h_{ij}$ and normalizes; for details see [20].)

Hopfield proved in [20], by a Liapunov-function argument, that if the interconnections between the neurons are symmetric, i.e. if $R_{ij} = R_{ji}$ for every pair $i, j = 1, \ldots, n$, then the system (3) is globally asymptotically stable, i.e. from any initial voltage state $(u_1, \ldots, u_n)$ the network relaxes towards some stable equilibrium state. One can thus view such a network as performing an input-output mapping from initial states to their respective equilibrium states. (Note that this point of view completely ignores the time-evolution of the system,

which was the central topic of interest in the study of differential analyzers.) Based on this convergence behavior, and the particular type of Liapunov function used in the proof, Hopfield and others [20, 21, 14] have proposed various special-case uses of such networks for associative memory and combinatorial optimization applications.

The general computational power of Hopfield's network model was studied in [34], where it was shown that arbitrary polynomially space-bounded Turing machines can be simulated by polynomial-size networks with the piecewise-linear amplifier response function

$$\sigma(u) = \begin{cases} -1, & \text{for } u < -1, \\ u, & \text{for } -1 \leq u \leq 1, \\ 1, & \text{for } u > 1. \end{cases}$$

However, the networks constructed in [34] are asymmetric, and the computational power of polynomial-size symmetric networks remains an open question. (On the other hand, in [35] it was shown that in the corresponding discrete-time model, asymmetric and symmetric networks are computationally equivalent.) Also, the simulation in [34] uses a similar two-phasing trick as Branicky's construction in [8, 9], and is thus in the same way somewhat unsatisfactory. And finally, the result is only a *lower bound* on the computational power: the possibility still remains that polynomial-size Hopfield networks might be even more powerful than polynomial-space Turing machines: conceivably even finite networks might have universal power, as was the case with the models discussed in Section 2.

4 Computational Complexity

Very little work has been done on the potentially most fruitful field of computational complexity analysis of continuous-time systems. Even the basic definitions have not yet been fixed in a universally acepted manner.

Apparently, the only published paper in this area is that of Vergis et al. [52], where the authors study the possibility of using GPAC-type systems (cf. equation (2)), or more generally Lipschitz-continuous systems of ODE's, for solving combinatorial problems faster than is possible by digital means. By a standard numerical-integration argument, they come to the conclusion that any analog computation can be simulated (integrated on a given interval $[0, t]$), to an arbitrary precision ε by a digital computer in a number of steps that is polynomial in $1/\varepsilon$ and R, the maximum magnitude of the second derivative of the simulated system. The intended implication is then that Lipschitzian analog systems cannot be superpolynomially more efficient than digital computers for solving limited-precision problems.

However, looking more carefully at the argument in [52], one notices that the number of steps in the digital simulation is in fact *exponential* in the length of the analog time interval $[0, t]$, which is assumed predetermined in the proof. Of course, an analog computation can be artificially sped up to occur within any given time interval, but then the maximum second derivative of the system during this interval increases. Thus, it seems that the length of the interval $[0, t]$ should also appear as a parameter in the result, and the argument in [52] is inconclusive.

A promising approach to defining a general notion of analog computation time is suggested (based on discussions with the present author) in [50]. Let us assume that a system given by a field $\frac{dx}{dt} = f(x)$ relaxes from an initial state $x(0) = x$ towards a stable equilibrium state $x(\infty) = x^*$. Define the computation time for input x and precision $\varepsilon > 0$ to be the

smallest $t_\epsilon \geq 0$ such that for all $t > t_\epsilon$, $\| x(t) - x^* \| \leq \varepsilon$. To obtain a natural *time scale*, one linearizes the field around the stable state x^* to obtain its stability matrix M (so that close to x^*, $\frac{dy}{dt} \approx My$, where $y = x - x^*$). If all of the eigenvalues of M have negative real parts, and $-\lambda$ is the largest of them, then the state of the system approaches x^* locally as $|x(t) - x^*| \sim e^{-\lambda t}$ [19]. (All the eigenvalues must have nonpositive real parts, since the system is stable at x^*.) Thus, for every increase of $1/\lambda$ in t, the state gets closer to x^* by a factor of e, and it is natural to choose $1/\lambda$ as the (local) unit of time.

Obviously, this approach is still preliminary: the time scales obtained are valid only locally, in the vicinity of each individual stable state (although one can argue [50] that at least simple systems "usually" converge "quickly" to one of these neighborhoods); and also it is not clear how to define time scales for systems some of whose stability matrix eigenvalues vanish. There is also the question how (and whether) to define a notion of *input size* in this model. The authors of [50] propose considering the computation relative to a *grid*, and defining the input size as $\log_2 1/\varepsilon$, where $\varepsilon \leq 1$ is the largest gridsize for which the computation is performed correctly, when both input and output are observed with precision ε.

5 Conclusion and Open Problems

We have surveyed the so far rather sparsely and unsystematically researched field of continuous-time computation theory. As has become apparent, most of the interesting research problems are still open, and in some cases even the proper definitions have not yet been established. The most significant unexplored area is surely the computational complexity theory of continuous-time systems: here one should first find the correct definitions for the basic notions of computation time, input size, etc., and then develop techniques for global analysis of interesting concrete systems, in the spirit of traditional discrete algorithm analysis. (For an initial step, one might look into the *local* analysis of the Hopfield associative memory presented in [50].) In parallel, one should develop the appropriate notions of complexity classes, reductions, and hard problems for continuous-time computation.

Also many interesting, more specialized problems remain open. In Section 2 we surveyed some finite-dimensional systems capable of simulating Turing machines. However, in each of these simulations there was little concern about implementability. Are any of the more implementation-based finite-dimensional systems, e.g. finite Hopfield networks, computationally universal? Also, many of the Turing machine simulations presented were unsatisfactory in being based on an explicitly constructed "system clock." What is the computational power of continuous-time systems without such a clock, e.g. of systems that possess Liapunov functions? Again, the most interesting concrete example is the class of Hopfield networks with symmetric neuron interconnections.

References

[1] L. M. Adleman, Molecular computation of solutions to combinatorial problems. *Science 266* (11 Nov. 1994), 1021–1024.

[2] J. A. Anderson and E. Rosenfeld (eds.), *Neurocomputing: Foundations of Research.* The MIT Press, Cambridge, MA, 1988.

[3] E. Asarin, O. Maler, On some relations between dynamical systems and transition systems. *Proc. 21st Internat. Colloq. on Automata, Languages, and Programming*, 59–72. Lecture Notes in Computer Science 820, Springer-Verlag, Berlin, 1994.

[4] C. H. Bennett, Logical reversibility of computation. *IBM J. Res. Develop. 17* (1973), 525–532.

[5] C. H. Bennett, Time/space trade-offs for reversible computation. *SIAM J. Comput. 18* (1989), 766–776.

[6] E. Bernstein, U. Vazirani, Quantum complexity theory. *Proc. 25th ACM Symp. on Theory of Computation*, 11–20. ACM Press, New York, NY, 1993.

[7] L. Blum, M. Shub, S. Smale, On a theory of computation over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the Amer. Math. Soc. 21* (1989), 1–46.

[8] M. Branicky, Analog computation with continuous ODEs. *Proc. Workshop on Physics and Computation 1994,* 265–274. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[9] M. Branicky, Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoret. Comput. Sci. 138* (1995), 67–100.

[10] R. W. Brockett, Smooth dynamical systems which realize arithmetical and logical operations. *Three Decades of Mathematical System Theory* (H. Nijmeijer, J. M. Schumacher, eds.) Lecture Notes in Control and Information Sciences 135, Springer-Verlag, Berlin, 1989.

[11] R. W. Brockett, Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems. *Linear Algebra and Its Applications 146* (1991), 79–91.

[12] V. Bush, The differential analyzer, a new machine for solving differential equations. *J. Franklin Inst. 212* (1931), 447–488.

[13] M. Casey, The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation 8* (1996), 1135–1178.

[14] A. Cichocki, R. Unbehauen, *Neural Networks for Optimization and Signal Processing.* Wiley/Teubner, Stuttgart, 1993.

[15] Hopfield, J. J. and Tank, D. W. Neural computation of decisions in optimization problems. *Biological Cybernetics 52* (1985), 141–152.

[16] D. G. Feitelson, *Optical Computing: A Survey for Computer Scientists.* The MIT Press, Cambridge, MA, 1988.

[17] M. Garzon, *Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks.* Springer-Verlag, Berlin, 1995.

[18] A. Hausner, *Analog and Analog/Hybrid Computer Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1971.

[19] M. W. Hirsch, S. Smale, *Differential Equations, Dynamical Systems, and Linear Algebra.* Academic Press, San Diego, CA, 1974.

[20] J. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Nat. Acad. Sci. USA 81* (1984), 3088–3092. Reprinted in [2], pp. 579–583.

[21] J. J. Hopfield, D. W. Tank, Neural computation of decisions in optimization problems. *Biological Cybernetics 52* (1985), 141–152.

[22] A. S. Jackson, *Analog Computation.* McGraw-Hill, New York, NY, 1960.

[23] C. L. Johnson, *Analog Computer Techniques, 2nd Ed.* McGraw-Hill, New York, NY, 1963.

[24] P. Koiran, Dynamics of discrete time, continuous state Hopfield networks. *Neural Computation 6* (1994), 459–468.

[25] P. Koiran, M. Cosnard, M. Garzon, Computability with low-dimensional dynamical systems. *Theoret. Comput. Sci. 132* (1994), 113–128.

[26] G. A. Korn, T. M. Korn, *Electronic Analog and Hybrid Computers, 3rd Ed.* McGraw-Hill, New York, NY, 1964.

[27] L. Lipshitz, L. Rubel, A differentially algebraic replacement theorem, and analog computability. *Proc. Amer. Math. Soc. 99* (1987), 367–372.

[28] R. J. Lipton, DNA solution of hard computational problems. *Science 268* (28 Apr. 1995), 542–545.

[29] W. Maass, P. Orponen, On the effect of analog noise in discrete-time analog computation. *Proc. Neural Information Processing Systems 1996,* to appear.

[30] C. Mead, *Analog VLSI and Neural Systems.* Addison-Wesley, Reading, MA, 1989.

[31] C. Moore, Unpredictability and undecidability in physical systems. *Phys. Review Letters 64* (1990), 2354–2357.

[32] C. Moore, Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity 4* (1991), 199–230.

[33] S. Omohundro, Modelling cellular automata with partial differential equations. *Physica 10D* (1984), 128–134.

[34] P. Orponen, On the Computational Power of Continuous Time Neural Networks. Project NeuroCOLT Report NC-TR-95-051, Royal Holloway College, Univ. of London, Dept. of Computer Science, 1995. 18 pp.

[35] P. Orponen, The computational power of discrete Hopfield nets with hidden units. *Neural Computation 8* (1996), 403–415.

[36] P. Orponen, M. Matamala, Universal computation by finite two-dimensional coupled map lattices. *Proc. Workshop on Physics and Computation 1996,* to appear.

[37] J. Palis, Jr., W. de Melo, *Geometric Theory of Dynamical Systems: An Introduction.* Springer-Verlag, New York, NY, 1982.

[38] M. B. Pour-El, Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Trans. Amer. Math. Soc. 199* (1974), 1–28.

[39] M. B. Pour-El, J. I. Richards, *Computability in Analysis and Physics.* Springer-Verlag, Berlin, 1989.

[40] P. Pudlák, Complexity theory and genetics. *Proc. 9th Ann. IEEE Conf. on Structure in Complexity Theory*, 383–395. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[41] J. H. Reif, J. D. Tygar, A. Yoshida, The computability and complexity of optical beam tracing. *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, 106–114. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[42] D. Rooß, K. Wagner, On the Power of Bio-Computers. Technical Report, Universität Würzburg, Inst. für Informatik, Feb. 1995.

[43] L. A. Rubel, Some mathematical limitations of the general-purpose analog computer. *Adv. in Appl. Math. 9* (1988), 22–34.

[44] L. A. Rubel, The extended analog computer. *Adv. in Appl. Math. 14* (1993), 39–50.

[45] E. Sánchez-Sinencio, C. Lau, *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations.* IEEE Press, New York, 1992.

[46] C. E. Shannon, Mathematical theory of the differential analyzer. *J. Math. Phys. MIT 20* (1941), 337–354. Reprinted in [48], 496–513.

[47] C. E. Shannon, The theory and design of linear differential equation machines. Report to the National Defense Research Council, January 1942. Reprinted in [48], pp. 514–559.

[48] C. E. Shannon, *Collected Papers* (N. J. A. Sloane, A. Wyner, eds.). IEEE Press, Piscataway, NJ, 1993.

[49] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring. *Proc. 35th Ann. IEEE Symp. on Foundations of Computer Science*, 124–134. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[50] H. T. Siegelmann, S. Fishman, Analog computing and dynamical systems. Manuscript, April 1996. 34 pp.

[51] H. T. Siegelmann, E. D. Sontag, On the computational power of neural nets. *J. Comput. System Sciences 50* (1995), 132–150.

[52] A. Vergis, K. Steiglitz, B. Dickinson, The complexity of analog computation. *Math. and Computers in Simulation 28* (1986), 91–113.