August 16, 1999

Computational Complexity for Continuous Time Dynamics

Hava Siegelmann, *University of Massachusetts - Amherst*
Asa Ben-Hur
Shmuel Fishman

Computational Complexity for Continuous Time Dynamics

Hava T. Siegelmann and Asa Ben-Hur
Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel

Shmuel Fishman
Department of Physics, Technion, Haifa 32000, Israel
(Received 22 March 1999)

Dissipative flows model a large variety of physical systems. In this Letter the evolution of such systems is interpreted as a process of computation; the attractor of the dynamics represents the output. A framework for an algorithmic analysis of dissipative flows is presented, enabling the comparison of the performance of discrete and continuous time analog computation models. A simple algorithm for finding the maximum of n numbers is analyzed, and shown to be highly efficient. The notion of tractable (polynomial) computation in the Turing model is conjectured to correspond to computation with tractable (analytically solvable) dynamical systems having polynomial complexity.

PACS numbers: 05.45.–a, 89.70.+c, 89.80.+h

The computation of a digital computer, and its mathematical abstraction, the Turing machine is described by a map on a discrete configuration space. In recent years scientists have developed new approaches to computation, some of them based on continuous time analog systems. The most promising are neuromorphic systems [1], models of human memory [2], and experimentally realizable quantum computers [3]. Although continuous time systems are widespread in experimental realizations, no theory exists for their algorithmic analysis. The standard theory of computation and computational complexity [4] deals with computation in discrete time and in a discrete configuration space, and is inadequate for the description of such systems. This Letter describes an attempt to fill this gap. Our model of a computer is based on dissipative dynamical systems (DDS), characterized by flow to attractors, which are a natural choice for the output of a computation. This makes our theory realizable by small-scale classical physical systems (since there dissipation is usually not negligible) [5]. We define a measure of computational complexity which reflects the convergence time of a physical implementation of the continuous flow, enabling a comparison of the efficiency of continuous time algorithms with discrete ones. On the conceptual level, the framework introduced here strengthens the connection between the theory of computational complexity and the field of dynamical systems.

Turing universality of dynamical systems is a fundamental issue; see [6] and a recent book [7]. A system of ordinary differential equations (ODEs) which simulates a Turing machine was constructed in [8]. Such constructions retain the discrete nature of the simulated map, in that they follow its computation step by step by a continuous equation. In the present Letter, on the other hand, we consider continuous systems as is, and interpret their dynamics as a process of computation.

The view of the process of computation as a flow to an attractor has been taken by a number of researchers. The Hopfield neural network is a dynamical system which evolves to attractors which are interpreted as memories; the network is also used to solve optimization problems [2]. Brockett introduced a set of ODEs that perform various tasks such as sorting and solving linear programming problems [9]. Numerous other applications can be found in [10]. An analytically solvable ODE for the linear programming problem was proposed by Faybusovich [11]. Our theory is, to some extent, a continuation of their work, in that it provides a framework for the complexity analysis of continuous time algorithms.

Our model is restricted to exponentially convergent autonomous dissipative ODEs

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}), \tag{1}$$

for $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{F}$, an n-dimensional vector field, where n depends on the input. For a given problem, $\mathbf{F}$ takes the *same* mathematical form, and only the length of the various objects in it (vectors, tensors, etc.) depends on the size of the instance, corresponding to "uniformity" in computer science [12]. We discuss only systems with fixed point attractors, and the term attractor will be used to denote an attracting fixed point. We study only autonomous systems since for these the time parameter is not arbitrary (contrary to nonautonmous ones): under any nonlinear transformation of the time parameter the system is no longer autonomous, as will be explained in what follows. The restricted class of exponentially convergent vector fields describes the "typical" convergence scenario for dynamical systems [13]. Structural stability of exponentially convergent flows is an important property for analog computers. As a further justification we argue that exponential convergence is a prerequisite for efficient computation, provided the computation requires reaching the asymptotic regime, as is usually the case. Asymptotically, $|\mathbf{x}(t) - \mathbf{x}^*| \sim e^{-t/\tau_{\mathrm{ch}}}$ [see Eq. (5)]. When a trajectory is close to its attractor, in a time $\tau_{\mathrm{ch}} \ln 2$ a digit of the attractor is computed. Thus the computation of L

digits requires a time which scales as $\tau_{\rm ch}L$. This is in contrast with polynomially convergent vector fields: Suppose that $|\mathbf{x}(t) - \mathbf{x}^*| \sim t^{-\beta}$ for some $\beta > 0$, then in order to compute $\mathbf{x}^*$ with L significant digits, we need to have $|\mathbf{x}(t) - \mathbf{x}^*| < 2^{-L}$, or $t > 2^{L/\beta}$, for an exponential time complexity.

Last, we concentrate on ODEs with a formal solution, since for these, complexity is readily analyzed, and it is easy to provide criteria for halting a computation. Dynamical systems with an analytical solution are an exception. But despite their scarcity, we argue later that a subclass of analytically solvable DDS's which converge exponentially to fixed point attractors are a counterpart for the classical complexity class P. This then suggests a correspondence between tractability in the realm of dynamical systems and tractability in the Turing model.

The input of a DDS can be modeled in various ways. One possible choice is the initial condition. This is appropriate when the aim of the computation is to decide to which attractor out of many possible ones the system flows. This approach was pursued in [14]. The main problem within this approach is related to initial conditions in the vicinity of basin boundaries. The flow in the vicinity of the boundary is slow, resulting in very long computation times. In the present Letter, on the other hand, the parameters on which the vector field depends are the input, and the initial condition is a function of the input, chosen in the correct basin, and far from basin boundaries to obtain an efficient computation. For the gradient vector field equation (7), designed to find the maximum of n numbers, the n numbers c_i constitute the input, and the initial condition given by (11) is untypically simple. More generally, when dealing with the problem of optimizing some cost function $E(\mathbf{x})$, e.g., by a gradient flow $\dot{\mathbf{x}} = \mathrm{grad}E(\mathbf{x})$, an instance of the problem is specified by the parameters of $E(\mathbf{x})$, i.e., by the parameters of the vector field.

The vector $\mathbf{x}(t)$ represents the state of the corresponding physical system at time t. The time parameter is thus time as measured in the laboratory, and has a well-defined meaning. Therefore we suggest it as a measure of the time complexity of a computation. However, for nonautonomous ODEs that are not directly associated with physical systems, the time parameter seems to be arbitrary: if the time variable t of a nonautonomous vector field is replaced by $s = g(t)$, where $g(t)$ is strictly monotonic we obtain a nonautonomous system $\frac{d\mathbf{x}}{ds} = \frac{\mathbf{F}(\mathbf{x},t)}{\dot{g}[g^{-1}(s)]} \equiv \mathbf{F}_s(\mathbf{x},s)$. In this way arbitrary speed-up can be achieved in principle. However, the time transformed system $\mathbf{F}_s(\mathbf{x},s)$ is a *new* system. Only once it is constructed does its time parameter take on the role of physical time, and is no longer arbitrary. Therefore speed-up is a relevant concept only within the bounds of physical realizability. We stress the distinction between linear and nonlinear transformations of the time parameter: a linear transformation is merely a change of the time unit; a nonlinear transformation effectively changes the system itself. Therefore we suggest autonomous systems as representing the intrinsic complexity of the class of systems that can be obtained from them by changing the time parameter.

The evolution of a DDS reaches an attractor only in the infinite time limit. Therefore for any finite time we can compute it only to some finite precision. This is sufficient since for combinatorial problems with integer or rational inputs, the set of fixed points (the possible solutions) will be distributed on a grid of some finite precision. A computation will be halted when the attractor is computed with enough precision to infer a solution to the associated problem by rounding to the nearest grid point.

The phase space evolution of a trajectory may be rather complicated, and a major problem is to decide when a point approached by the trajectory is indeed the attractor of the dynamics, and not a saddle point. An attractor is certified by its *attracting region* which is a subset of the trapping set of the attractor in which the distance from the attractor is monotonically decreasing in time. The *convergence time* to an attracting region U, $t_c(U)$ is the time it takes for a trajectory starting from the initial condition $\mathbf{x}_0$ to enter U.

When the computation has reached the attracting region of a fixed point, and is also within the precision required for solving the problem, ϵ_p, the computation can be halted. We thus define the *halting region* of a DDS with attracting region U and required precision ϵ_p as $H = U \cap B(\mathbf{x}^*, \epsilon_p)$, where $B(\mathbf{x}^*, \epsilon_p)$ is a ball of radius ϵ_p around the attractor $\mathbf{x}^*$. The *computation time* is the convergence time to the halting region, $t_c(H)$, given by

$$t_c(H) = \max[t_c(\epsilon_p), t_c(U)], \tag{2}$$

where $t_c(\epsilon_p)$ is the convergence time to $B(\mathbf{x}^*, \epsilon_p)$.

In general, we cannot calculate $t_c(H)$ for a DDS algorithm. Thus we resort to halting by a bound on the computation time of all instances of size L:

$$T(L) = \max_{|\Pi|=L} t_c(H(\Pi)), \tag{3}$$

where Π denotes the input, and $L = |\Pi|$ is its size in bits. The definition of the input size depends on the input space considered. The bit size (used here) is the suitable measure for unbounded integers. See [15] for more details.

Time complexity is a dimensionless number, whereas $T(L)$ depends on the time units of the system at hand. To make it dimensionless we express it in terms of the time scale for convergence to the fixed point. Let $\mathbf{x}^*(\Pi)$ be the attracting fixed point of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ on input Π. In the vicinity of $\mathbf{x}^*$ the linear approximation $\dot{\delta\mathbf{x}} = D\mathbf{F}|_{\mathbf{x}^*}\delta\mathbf{x}$ holds, where $\delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$. Let λ_i be the real part of the ith eigenvalue of $D\mathbf{F}|_{\mathbf{x}^*}$. We define:

$$\lambda = \min_i |\lambda_i|. \tag{4}$$

λ determines the rate of convergence to an attractor, since in its vicinity $|\mathbf{x}(t) - \mathbf{x}^*| \sim e^{-\lambda t}$, leading to the definition of the characteristic time

$$\tau_{\rm ch} = \frac{1}{\lambda}. \tag{5}$$

We finally define the *time complexity* of a DDS algorithm:

$$T'(L) = \frac{T(L)}{\tau_{\rm ch}(\Pi_0)}, \tag{6}$$

where Π_0 is a fixed instance of the problem. This is a valid definition of complexity since it is invariant under linear transformations of the time parameter.

We demonstrate our approach with a simple DDS algorithm for the "MAX" problem, which is the problem of finding the maximum of n numbers. Let the numbers be $c_1, \ldots, c_n$, and define the linear cost function $E(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$. The MAX problem can be formulated as a constrained optimization problem: find the maximum of $E(\mathbf{x})$ subject to the constraints $\sum_{i=1}^n x_i = 1$, $x_i \geq 0$. This is recognized as the linear programming problem on the $n-1$ dimensional simplex $\Delta_{n-1} = \{\mathbf{x} \in \mathbb{R}^n\colon\ x_i \geq 0, \sum_{i=1}^n x_i = 1\}$. We use the vector field

$$F_i = \left(c_i - \sum_{j=1}^n x_j c_j\right) x_i, \tag{7}$$

which is the gradient of the function E on Δ_{n-1} relative to a Riemannian metric which enforces the positivity constraints [10]. This flow is a special case of the Faybusovich vector field [11], which we analyze elsewhere [16].

We denote by $\mathbf{e}_1, \ldots, \mathbf{e}_n$ the standard basis of $\mathbb{R}^n$. The fixed points of $\mathbf{F}$ are the vertices of the simplex $\mathbf{e}_1, \ldots, \mathbf{e}_n$. We first assume a unique maximum. Also suppose that $c_1 > c_2$ and $c_2 \geq c_j, j = 3, \ldots, n$. Under this assumption the flow converges exponentially to $\mathbf{e_1}$ as witnessed by the analytical solution

$$x_i(t) = \frac{e^{c_i t} x_i(0)}{\sum_{j=1}^n e^{c_j t} x_j(0)}, \tag{8}$$

where $x_i(0)$ are the components of the initial condition. We note that the analytical solution does not help in determining which of the fixed points is the attractor of the system: the solution to the specific instance of the problem is required for that. Thus the analytical solution is only formal, and one has to follow the dynamics of the vector field (7) to find the maximum. However, the formal solution is useful in obtaining tight bounds on $T(L)$.

A linearization of $\mathbf{F}$ shows that the time scale for convergence to the attractor $\mathbf{e}_1$ is

$$\tau_{\rm ch} = \frac{1}{c_1 - c_2}. \tag{9}$$

By solving for t in the equation $\|\mathbf{x}(t) - \mathbf{e}_1\| < \epsilon$, an upper bound on the time to reach an ϵ vicinity of the vertex $\mathbf{e}_1$ is found [10]:

$$t_c(\epsilon) \leq \tau_{\rm ch} |\ln(x_1(0)\epsilon^2)|. \tag{10}$$

The divergence as $x_1(0)$ tends to zero is due to initial conditions close to the basin boundary. To minimize the contribution of flow near basin boundaries we choose as an initial condition the symmetric vector

$$\mathbf{e} = \frac{1}{n}(1, \ldots, 1)^T. \tag{11}$$

The coordinates of the possible solutions (vertices of the simplex) are integer, and therefore $\epsilon_p = 1/2$. Using Eq. (10) we obtain that the convergence time to the ϵ_p-vicinity of $\mathbf{e}_1$ is bounded by

$$t_c(\epsilon_p) < \tau_{\rm ch} \ln 4n. \tag{12}$$

Next we show a bound on the convergence time to the attracting region. The attracting region of the problem is the region in which $\dot{x_i} < 0$ for $i > 1$. By the positivity of the x_i's, this is satisfied if $\sum_{j=1}^n c_j x_j > c_i$, $i = 2, \ldots, n$. Inserting the analytical solution yields a bound on $t_c(U)$:

$$t_c(U) \leq \tau_{\rm ch}(\ln \tau_{\rm ch} c_2 + \ln n), \tag{13}$$

The maximum of (12) and (13) gives the bound

$$t_c(H(\mathbf{c})) \leq \tau_{\rm ch}(\ln \tau_{\rm ch} c_2 + \ln 4n). \tag{14}$$

If integer inputs are considered then $\tau_{\rm ch} \leq 1$. The size of the input in bits is $L = \sum_{i=1}^n [1 + \log_2(c_i + 1)]$. Expressing the bound on $t_c(H(\mathbf{c}))$ in terms of L:

$$t_c(H(\mathbf{c})) = O(\ln L). \tag{15}$$

A sublinear (logarithmic) complexity arises because the model is inherently parallel: the variables of a DDS algorithm can be considered as processing units, and their number in this algorithm increases with the size of the input. When the inputs are bounded integers the complexity becomes $O(\log n)$, similar to the complexity obtained in models of parallel computation in the Turing framework. The analysis of the MAX problem was presented as an illustration. In [16] we analyze the maximum network flow problem and the general linear programming problem.

In the following, we no longer assume integer inputs (see [15]). Suppose that $c_1 = c_2$, then the maximum is not unique, and arbitrarily small perturbations of such an instance have different attractors as the solution. The bound on $t_c(U)$ also shows that in such a case it takes a long time for the system to distinguish between attractors with close values of the cost function, since when c_2 is close to c_1 we have a large time scale, $\tau_{\rm ch}$. Thus problems which are near to problems with a nonunique maximizer are "hard." Instances of a problem which have a nonunique solution are termed "ill-posed" in the numerical analysis literature [17] and the difficulty of problems which are close to ill-posed ("ill-conditioned") is expressed in "condition number theorems" which state that the inverse of the distance of an instance from the set of ill-posed problems is equal to its condition number. In our context we define $\tau_{\rm ch}$ as the condition number and indeed find that it is the inverse of the distance of an instance from the set of problems in which $c_1 = c_2$ [15]. A more general condition number theorem is also argued

[15]. Ill-conditioned problems for models where the input is the parameters, are reminiscent of problems with the initial condition near the basin boundary in models where the initial condition is the input.

In the following, we compare complexity in our model with the classical theory. The complexity class P in classical computational complexity is the class of problems for which there exists an algorithm which runs in polynomial time on a Turing machine. Its counterpart in our framework is called CP (continuous P), and contains the set of problems which have a DDS algorithm with a polynomial number of variables and polynomial time complexity. Note that since the variables play the role of processing units, their number needs to be limited as well. In [16] we present a DDS algorithm for the maximum network flow problem which has polynomial time complexity. In addition we define the class CLOG (continuous log) of problems that have a DDS algorithm with a polynomial number of variables and logarithmic time complexity. We have shown that MAX is in CLOG. We note that for a comparison with the classical theory to be meaningful it is necessary to impose constraints on the complexity of the vector field. Otherwise, the computational power of the model can be attributed to the complexity of the vector field. We assume in the following that the vector field is in the parallel complexity class *NC* [12], of computations in polylogarithmic (polynomial of log) time with a polynomial number of processors; *NC* is the complexity class just below P.

We argue that CP = P. For the inclusion P $\subseteq$ CP we rely on the claim that the P-complete [18] problem of maximum network flow is in CP [16]. If we use the Turing reductions from a P problem to maximum network flow we have in fact shown that all efficient Turing computations can be performed polynomially in our framework. However, relying on Turing reductions which are external to our model might be considered unsatisfactory. As of yet we have no argument that CP $\subseteq$ P. However, we believe that a polynomial time simulation of the ODE with some numerical integration scheme should be possible because of the convergence to fixed points.

In this Letter we concentrated on analytically tractable dynamical systems, a property which helped us in computing bounds on the convergence time to the attracting region, and find initial conditions far from basin boundaries. For a large class of systems without an analytical solution one can resort to probabilistic verification of an attractor: when it is suspected that a fixed point is approached, a number of trajectories are initiated in an ϵ ball around the trajectory. If this ball shrinks, then with high probability the fixed point is an attractor. If the ball has expanded in some direction, then the fixed point is a saddle. This yields a Co-RP type of complexity class [12] which is applicable to gradient flows, for example [14].

Only fixed point attractors were considered here. In [14] computation of chaotic attractors is discussed. Such attractors are found to be computable efficiently by means of nondeterminism. The inherent difference between fixed points and chaotic attractors may shed light on the P vs NP question which is a main open problem in computer science.

The authors would like to thank E. Sontag, K. Ko, C. Moore, and J. Crutchfield for helpful discussions. This work was supported in part by the U.S.-Israel Binational Science Foundation (BSF), by the Israeli Ministry of Arts and Sciences, by the Fund for Promotion of Research at the Technion and by the Minerva Center for Nonlinear Physics of Complex Systems.

[1] C. Mead, *Analog VLSI and Neural Systems* (Addison-Wesley, Reading, Massachusetts, 1989).
[2] J. J. Hopfield and D. W. Tank, Biol. Cybernet. **52**, 141 (1985).
[3] C. P. Williams, in *Proceedings of Quantum Computing and Quantum Communications: First NASA International Conference, QCQC'98*, Springer Lecture Notes in Computer Science Vol. 1509 (Springer, Berlin, 1998).
[4] The term computational complexity in computer science denotes the scaling of the resources needed to solve a problem with its size [12].
[5] E. Ott, *Chaos in Dynamical Systems* (Cambridge University Press, Cambridge, 1993).
[6] C. Moore, Phys. Rev. Lett. **64**, 2354 (1990).
[7] H. T. Siegelmann, *Neural Networks and Analog Computation: Beyond the Turing Limit* (Birkhauser, Boston, Massachusetts, 1999), and references therein.
[8] M. S. Branicky, in *Proceedings of the IEEE Workshop on Physics and Computation* (Dallas, Texas, 1994), pp. 265–274.
[9] R. W. Brockett, Linear Algebr. Appl. **146**, 79 (1991).
[10] U. Helmke and J. B. Moore, *Optimization and Dynamical Systems* (Springer-Verlag, London, 1994).
[11] L. Faybusovich, IMA J. Math. Control Inf. **8**, 135 (1991).
[12] C. Papadimitriou, *Computational Complexity* (Addison-Wesley, Reading, Massachusetts, 1995).
[13] B. R. Hunt, T. Sauer, and J. A. Yorke, Bull. Am. Math. Soc. **27**, 217 (1992).
[14] H. T. Siegelmann and S. Fishman, Physica (Amsterdam) **120D**, 214 (1998).
[15] H. T. Siegelmann, A. Ben-Hur, and S. Fishman (to be published).
[16] A. Ben-Hur, H. T. Siegelmann, and S. Fishman (to be published).
[17] S. Smale, SIAM Rev. **32**, 211 (1990).
[18] A problem in P is called P-complete if any problem in P can be reduced to it [12].