Comments on Attractor Computation

Hava T. Siegelmann and Asa Ben-Hur
iehava@iehava.technion.ac.il
Faculty of Industrial Engineering and Management,
Technion, Haifa 32000, Israel.

Shmuel Fishman
fishman@physics.technion.ac.il
Department of Physics, Technion, Haifa 32000, Israel.

Abstract

Dissipative flows model a large variety of physical systems. In this paper 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.

Keywords: Analog Computation, Dynamical Systems, Complexity Theory.

1 Introduction

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 (Mead, 1989), models of human memory (Hopfield and Tank, 1985), and experimentally realizable quantum computers (Williams, 1998). Although continuous time systems are widespread in experimental realizations, no theory exists for their algorithmic analysis. The standard theory of computation and

International Journal of Computing Anticipatory Systems, Volume 6, 2000
Edited by D. M. Dubois, CHAOS, Liège, Belgium, ISSN 1373-5411 ISBN 2-9600179-8-6

computational complexity [1] deals with computation in discrete time and in a discrete configuration space, and is inadequate for the description of such systems. This paper 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 (Ott, 1993). 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 is a fundamental issue, see (Moore, 1990) and a recent book (Siegelmann, 1999). A system of ODEs which simulates a Turing machine was constructed in (Branicky, 1994). 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 paper 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 (Hopfield and Tank, 1985). Brockett introduced a set of ODEs that perform various tasks such as sorting and solving linear programming problems (Brockett, 1991). Numerous other applications can be found in (Helmke and Moore, 1994). An analytically solvable ODE for the linear programming problem was proposed by Faybusovich (Faybusovich, 1991). 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.

2 The Model

We base our model on exponentially convergent autonomous dissipative ODEs

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

where $\mathbf{x} \in \mathbb{R}^n$,$\mathbf{F}$ is an n-dimensional vector field. For a given problem, $\mathbf{F}$ takes the *same* mathematical form, and only the length of the various objects in it (vectors, matrices etc.) depends on the size of the instance, corresponding to "uniformity" in computer science (Papadimitriou, 1995). In this paper we discuss only dissipative systems with point attractors (equilibrium points). In fact,

[1] The term *computational complexity* in computer science denotes the scaling of the resources needed to solve a problem with its size (Papadimitriou, 1995).

our considerations will be valid for systems with exponentially stable equilibrium which are not necessarily dissipative. We study only autonomous systems since for these the time parameter is not arbitrary (contrary to non-autonmous ones): under any nonlinear transformation of the time parameter the system is no longer autonomous, as will be explained in the next section. The restricted class of exponentially convergent vector fields describes the "typical" convergence scenario for dynamical systems (Hunt et al., 1992). 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, $|\mathbf{x}(t) - \mathbf{x}^*| \sim e^{-t/\tau_{ch}}$ (see eqn. (6)). When a trajectory is close to its attractor, in a time $\tau_{ch} \ln 2$ a digit of the attractor is computed. Thus the computation of L digits requires a time which is proportional to $\tau_{ch} L$. 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 stable equilibrium points 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 (Siegelmann and Fishman, 1998). 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 paper, 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 (10), designed to find the maximum of n numbers, the n numbers c_i constitute the input, and the initial condition is a constant vector. 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.

3 Computational Complexity for Continuous Time Systems?

We are interested in ODEs as models of physical systems. The vector $\mathbf{x}(t)$ then 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 non-autonomous ODEs that are not directly associated with physical systems, the time parameter seems to be arbitrary: if the time variable t of a non-autonomous vector field is replaced by another variable s, where $t = g(s)$, and $g(s)$ is strictly monotonic, we obtain another non-autonomous system

$$\frac{d\mathbf{x}}{ds} = \mathbf{F}(\mathbf{x}, g(s))g'(s) \; . \tag{2}$$

The above system will be called the time transformed version of $\mathbf{F}$. If we take for example, $t = e^s$, then the transformed system computes exponentially faster. This way arbitrary speed-up can be achieved in principle. However, the time transformed system 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 non-linear 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.

4 Halting a Computation

The evolution of a DDS reaches an attractor only in the infinite time limit. Therefore for any finite time it can be computed to some finite precision. This is sufficient since for combinatorial problems with integer or rational inputs, the set of equilibrium 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 an equilibrium 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:

$$t_c(H) = \max(t_c(\epsilon_p), t_c(U)) \,, \tag{3}$$

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

The attracting region of a DDS algorithm is some vicinity of the attractor, which is associated with the solution to the problem and is not easier to compute than solving the problem. Thus it is not feasible to specify halting by convergence to the halting region of a specific instance, but rather to specify halting by a bound on the computation time of all instances of size L:

$$T(L) = \max_{|\Pi|=L} t_c(H(\Pi)) \tag{4}$$

where Π denotes the input, and $|\Pi|$ is its size in bits. On input Π of size L, the computation will be halted after a time $T(L)$.

5 Time Complexity

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 equilibrium point. Let $\mathbf{x}^*(\Pi)$ be the attracting equilibrium point of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ on input Π. In the vicinity of $\mathbf{x}^*$ the linear approximation $\delta\dot{\mathbf{x}} = D\mathbf{F}|_{\mathbf{x}^*}\, \delta\mathbf{x}$ 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:

$$\lambda = \min_i |\lambda_i| \tag{5}$$

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

$$\tau_{ch} = \frac{1}{\lambda} \,. \tag{6}$$

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

$$T'(L) = \frac{T(L)}{\tau_{ch}(\Pi_0)} \,, \tag{7}$$

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 or equivalently multiplying the vector field by a constant:

Claim 5.1 Let $\mathbf{F}, \mathbf{F}'$ be two vector fields related by $\mathbf{F}' = \frac{1}{a}\mathbf{F}$ for some constant $a > 0$, then they have the same time complexity.

Proof. We denote by primes properties of the the vector field $\mathbf{F}'$. Multiplying the vector field by $\frac{1}{a}$ is equivalent to multiplying the time parameter by a. Therefore the computation times in the two systems are related by: $t'_c(\Pi) = at_c(\Pi)$, for every input Π. Let M, M' be the stability operators of $\mathbf{F}, \mathbf{F}'$ on input Π, respectively. Clearly $M' = \frac{1}{a}M$ so that $\tau'_{ch}(\Pi) = a\tau_{ch}(\Pi)$, and in particular for Π_0. We conclude:

$$\frac{t_c(H(\Pi))}{\tau_{ch}(\Pi_0)} = \frac{t'_c(H(\Pi))}{\tau'_{ch}(\Pi_0)} .$$

This holds when taking the maximum as well. ∎

6 Solving the MAX Problem

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

$$f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} . \tag{8}$$

The MAX problem can be formulated as a constrained optimization problem: find the maximum of f subject to the constraints

$$\sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0, \; i = 1, \ldots, n. \tag{9}$$

This is recognized as the linear programming problem on the $n-1$ dimensional simplex $\Delta_{n-1} = \{\mathbf{x} \in \mathbb{R}^n : \; x_i \geq 0, \; \sum_{i=1}^{n} x_i = 1\}$. We use the vector field

$$F_i = (c_i - \sum_{j=1}^{n} x_j c_j)\, x_i , \tag{10}$$

which is the gradient of the function E on Δ_{n-1} relative to a Riemannian metric which enforces the positivity constraints (Helmke and Moore, 1994).

We denote by $\mathbf{e}_1, \ldots, \mathbf{e}_n$ the standard basis of $\mathbb{R}^n$. The equilibrium points of $\mathbf{F}$ in Δ_{n-1} are the vertices of the simplex $\mathbf{e}_1, \ldots, \mathbf{e}_n$. We assume a unique maximum. See (Ben-Hur et al., 1999) for the general case. 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

$$x_i(t) = \frac{e^{c_i t} x_i(0)}{\sum_{j=1}^{n} e^{c_j t} x_j(0)} , \tag{11}$$

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 equilibrium points is

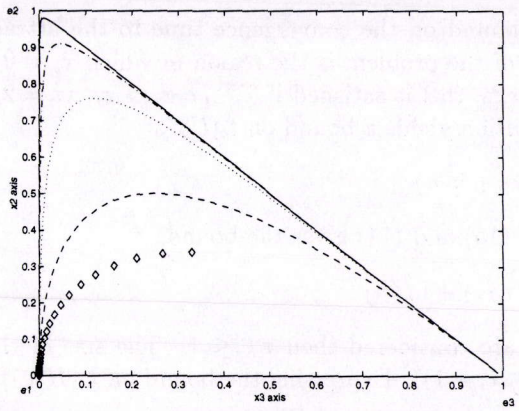

Figure 1: Phase space behavior of the flow generated by (10) on the two dimensional simplex with $c_1 > c_2 > c_3$. A projection onto the plane of the x_2 and x_3 coordinates is shown. A number of trajectories which start near the minimum vertex $\mathbf{e}_3$ are plotted. Such trajectories reach a vicinity of the saddle point $\mathbf{e}_2$ before flowing in the direction of the maximum which is projected onto the origin. The trajectory of the interior initial condition $\mathbf{e} = \frac{1}{3}(1,1,1)$ is denoted by diamonds.

the attractor of the system: one needs the solution to the specific instance of the problem for that. Thus the analytical solution is only formal, and one has to follow the dynamics of the vector field (10) to find the maximum. However, the formal solution is useful in obtaining tight bounds on $T(L)$.

From the asymptotic behavior of the analytic solution (11) it is seen that the time scale for convergence to the attractor $\mathbf{e}_1$ is

$$\tau_{ch} = \frac{1}{c_1 - c_2} \,. \tag{12}$$

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 (Helmke and Moore, 1994):

$$t_c(\epsilon) \leq \tau_{ch} |\ln(x_1(0)\epsilon^2)| \,. \tag{13}$$

The divergence as $x_1(0)$ tends to zero is due to initial conditions close to the basin boundary (see Figure 1). To minimize the contribution of flow near basin boundaries we choose as initial condition the symmetric vector $\mathbf{e} = \frac{1}{n}(1,\ldots,1)^T$.

The coordinates of the possible solutions (vertices of the simplex) are integer, and therefore $\epsilon_p = 1/2$. Using equation (13) we obtain that the convergence time to the ϵ_p-vicinity of $\mathbf{e}_1$ is bounded by

$$t_c(\epsilon_p) < \tau_{ch} \ln 4n \tag{14}$$

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 $t_c(U)$:

$$t_c(U) \leq \tau_{ch}(\ln \tau_{ch} c_2 + \ln n) \,, \tag{15}$$

The maximum of (15) and (14) gives the bound

$$t_c(H(\mathbf{c})) \leq \tau_{ch}(\ln \tau_{ch} c_2 + \ln 4n) \,. \tag{16}$$

If integer inputs are considered then $\tau_{ch} \leq 1$. 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:

$$t_c(H(\mathbf{c})) = O(\ln L) \,. \tag{17}$$

A sub-linear (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.

7 Complexity Classes

In this section 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 (Ben-Hur et al., 1999) we show 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 time required to compute the vector field, even though it is unclear what complexity should be ascribed to the equations of motion of physical systems. 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 (Papadimitriou, 1995), of computations in poly-logarithmic (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 [2] problem of maximum network flow is in CP (Ben-Hur et al., 1999). 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. But we believe that a polynomial time simulation of the ODE with some numerical integration scheme should be possible because of the convergence to equilibrium points.

8 Discussion

In this paper 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 an attracting equilibrium 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 equilibrium point is attracting. If the ball has expanded in some direction, then the equilibrium point is a saddle. This yields a Co-RP type of complexity class (Papadimitriou, 1995) which is applicable to gradient flows for example (Siegelmann and Fishman, 1998).

In this paper only point attractors were considered. In another paper (Siegelmann and Fishman, 1998) computation of chaotic attractors is discussed. Such attractors are found to be computable efficiently by means of nondeterminism (in a different sense then the one mentioned above). The inherent difference between point attractors and chaotic attractors may shed light on the P vs. NP question which is a main open problem in computer science.

Acknowledgments

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.

[2] A problem in P is called P-complete if any problem in P can be reduced to it (Papadimitriou, 1995).

References

Ben-Hur, A., Siegelmann, H., and Fishman, S. (1999). A theory of complexity for continuous time dynamics.

Branicky, M. (1994). Analog computation with continuous ODEs. In *Proc. IEEE Workshop Physics and Computation*, pages 265–274, Dallas, TX.

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

Faybusovich, L. (1991). Dynamical systems which solve optimization problems with linear constraints. *IMA Journal of Mathematical Control and Information*, 8:135–149.

Helmke, U. and Moore, J. (1994). *Optimization and Dynamical Systems.* Springer Verlag, London.

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

Hunt, B., Sauer, T., and Yorke, J. (1992). Prevalence: A translational-invariant "almost every" on infinite dimensional spaces. *Bulletin Of the American Mathematical Society*, 27(2):217–238.

Mead, C. (1989). *Analog VLSI and Neural Systems.* Addison-Wesley.

Moore, C. (1990). Unpredictability and undecidability in dynamical systems. *Phys. Rev. Lett.*, 64:2354–2357.

Ott, E. (1993). *Chaos in Dynamical Systems.* Cambridge University Press, Cambridge.

Papadimitriou, C. (1995). *Computational Complexity.* Addison-Wesley, Reading, Mass.

Siegelmann, H. (1999). *Neural Networks and Analog Computation: Beyond the Turing Limit.* Birkhauser, Boston.

Siegelmann, H. T. and Fishman, S. (1998). Computation by dynamical systems. *Physica D*, 120:214–235.

Williams, C. P., editor (1998). volume 1509 of *Lecture Notes in Computer Science*, Palm Springs. Springer.