THE COMPLEXITY OF ANALOG COMPUTATION †

Anastasios VERGIS
Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.

Kenneth STEIGLITZ
Department of Computer Science, Princeton University, Princeton, NJ 08544, U.S.A.

Bradley DICKINSON
Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, U.S.A.

We ask if analog computers can solve NP-complete problems efficiently. Regarding this as unlikely, we formulate a strong version of Church's Thesis: that any analog computer can be simulated *efficiently* (in polynomial time) by a digital computer. From this assumption and the assumption that P $\neq$ NP we can draw conclusions about the operation of physical devices used for computation.

An NP-complete problem, 3-SAT, is reduced to the problem of checking whether a feasible point is a local optimum of an optimization problem. A mechanical device is proposed for the solution of this problem. It encodes variables as shaft angles and uses gears and smooth cams. If we grant Strong Church's Thesis, that P $\neq$ NP, and a certain ''Downhill Principle'' governing the physical behavior of the machine, we conclude that it cannot operate successfully while using only polynomial resources.

We next prove Strong Church's Thesis for a class of analog computers described by well-behaved ordinary differential equations, which we can take as representing part of classical mechanics.

We conclude with a comment on the recently discovered connection between spin glasses and combinatorial optimization.

1. Introduction

Analog devices have been used, over the years, to solve a variety of problems. Perhaps most widely known is the Differential Analyzer [4,26], which has been used to solve differential equations. To mention some other examples, in [25] an electronic analog computer is proposed to implement the gradient projection method for linear programming. In [18] the problem of finding a minimum-length interconnection network between given points in the plane is solved with movable and fixed pegs interconnected by strings; a locally optimal solution is obtained by pulling the strings. Another method is proposed there for this problem, based on the fact that soap films form minimal-tension surfaces. Many other examples can be found in books such as [14] and [16], including electrical and mechanical machines for solving simultaneous linear equations and differential equations.

Given the large body of work on the complexity of Turing-machine computation, and the recent interest in the physical foundations of computation, it seems natural to study the complexity of analog computation. This paper pursues the following line of reasoning: it is generally regarded as likely that

† This work was supported in part by ONR Grants N00014-83-K-0275 and N00014-83-K-0577, NSF Grant ECS-8120037, U. S. Army Research-Durham Grant DAAG29-82-K-0095, and DARPA Contract N00014-82-K-0549. It appeared in *Mathematics & Computers in Simulation* 28 (1986) 91-113.

$P \neq NP$ — that certain combinatorial problems cannot be solved efficiently by digital computers. (Here we use the term *efficient* to mean that the time used by an ‘‘ideal’’ digital computer is bounded by a polynomial function of the size of the task description. See [9] for discussion of this criterion.) We may ask if such problems can be solved efficiently by other means, in particular, by machines of a nature different from digital computers. We thus come to ask if NP-complete problems can be solved efficiently by physical devices that do not use binary encoding (or, more generally, encoding with any fixed radix). We lump such devices together under the term *analog computer*; in what follows we will use the term *analog computer* to mean any deterministic physical device that uses a fixed number of physical variables to represent each problem variable. This description is admittedly vague and certainly non-mathematical — we mean it to capture the intuitive notion of a ‘‘non-digital’’ computer. (More about this in the next section.)

We want to emphasize that the question of whether an analog computer can solve an NP-complete problem ‘‘efficiently’’ is a question about the physical world, while the $P = NP$ question is a mathematical one. However, mathematical models of various kinds provide a formalism that is apparently indispensable for the understanding of physical phenomena. An important connection between the mathematical world of computation and the physical world of computing hardware was discussed by Church. In his 1936 paper [6] he equated the intuitive notion of effective calculability with the two equivalent mathematical characterizations of λ-definability and recursivity. Turing [28] then showed that this notion is equivalent to computability by what we have come to call a Turing machine, so that the intuitive notion of effective calculability is now characterized mathematically by ‘‘Turing-computability.’’ This is generally referred to as ‘‘Church’s Thesis,’’ or the ‘‘Church-Turing Thesis.’’ In our context we express this as follows:

Church’s Thesis (CT): Any analog computer with finite resources can be simulated by a digital computer.

What we will come to demand is more than that: we are interested in efficient computation, computation that does not use up resources that grow exponentially with the size of the problem. This requirement leads us to formulate what we call

Strong Church’s Thesis (SCT): Any finite analog computer can be simulated *efficiently* by a digital computer, in the sense that the time required by the digital computer to simulate the analog computer is bounded by a polynomial function of the resources used by the analog computer.

Evidently we will need to give a characterization of analog computers and the resources that they use. This is discussed in the next section. Following that, we argue that certain numerical problems are inherently difficult (i.e. not polynomial) for analog computers, even though they are easy for digital computers.

Something like our Strong Church’s Thesis was discussed recently by Feynman [8] in connection with the problem of building a (digital) computer that simulates physics. He says:

‘‘The rule of simulation that I would like to have is that the number of computer elements required to simulate a large physical system is only to be proportional to the space-time volume of the physical system. I don’t want to have an explosion.’’

We would argue that ‘‘proportional to’’ be replaced by ‘‘bounded by a polynomial function of,’’ in the spirit of modern computational complexity theory.

A class of mechanical devices is proposed in Section 5. Machines in this class can be used to find local optima for mathematical programming problems. We formalize the physical operation of these machines by a certain ‘‘Downhill Principle.’’ Basically, it states that if, in our class of mechanical devices, there are feasible ‘‘downhill’’ directions, the state vector describing the physical system moves in such a direction. We also discuss measuring the resources required by these machines.

In Section 6 we reduce 3-SAT (the problem of whether a Boolean expression in 3-conjunctive normal form has a satisfying truth assignment), to the problem of checking whether a given feasible point is a local optimum of a certain mathematical programming problem. This shows that merely checking for local optimality is NP-hard.

In Section 7 a mechanical device in the class mentioned above is proposed for the solution of 3-SAT. Naturally, the efficient operation of this machine is highly suspect. Be careful to notice that the operation of any machine in practice is a *physics* question, not a question susceptible of ultimate mathematical demonstration. Our analysis must necessarily be based on an idealized mathematical model for the machine. However, we can take the likelihood of $P \neq NP$, plus the likelihood of Strong Church’s Thesis, as evidence that in fact such a machine cannot operate with polynomially bounded resources, whatever the particular laws of physics happen to be.

The paradigm that emerges from this line of reasoning is then the following:

> If a strongly NP-complete problem can be solved by an analog computer, and if $P \neq NP$, and if Strong Church’s Thesis is true, then the analog computer cannot operate successfully with polynomial resources.

We will then prove a restricted form of Strong Church’s Thesis, for analog computers governed by well-behaved differential equations. This suggests that any interesting analog computer should rely on some strongly nonlinear behavior, perhaps arising from quantum-mechanical mechanisms; however, the problem of establishing Strong Church’s Thesis (or even the Weak Thesis) in the case of quantum-mechanical or probabilistic laws is an open problem.

2. Some Terminology

We know what a digital computer is; Turing has laid out a model for what a well-defined digital computation must be: it uses a finite set of symbols (without loss of generality {0,1}) to store information, it can be in only one of a finite set of states, and it operates by a finite set of rules for moving from state to state. Its memory tape is not bounded in length a priori, but only a finite amount of tape can be used for any one computation. What is fundamental about the idea of a Turing Machine and digital computation in general, is that there is a perfect correspondence between the mathematical model and what happens in a reasonable working machine. Being definitely in one of two states is easily arranged in practice, and the operation of real digital computers can be (and usually is) made very reliable.

In order to discuss the application of the Turing machine model to solving computational problems, we need some additional terminology. A *problem instance* is a finite string of bits, of length L, together with an *interpretation* of the bit string that specifies the encoding of a particular computational problem. The integer L is termed the *size* of the input. It is with respect to L that the complexity of computation is measured. If a computation requires no more than L^k steps, for some fixed k, we say it is *polynomial*; otherwise we say it is *exponential*.

We now turn to the task of formulating models for analog computers and to a discussion of how analog computers are applied to solving computational problems. An analog computer is an indexed family of physical devices, parametrized by a set of problem instances for which solutions are to be obtained. Mathematical modeling of the operations of the devices depends on the mathematical representation for the underlying laws of physics, whatever those laws may be.

Some additional restrictions are assumed to hold. First, for each problem instance, the problem variables (determined from the interpretation of the bit string) are encoded within the corresponding physical device as variables whose mathematical descriptions are specifically constrained. Each physical variable is modeled by a quantity taking values in a normed, finite-dimensional, real space whose dimension does not depend on the problem instance. As an example, the value of a problem variable x may be encoded by the angular displacement of a shaft, by an electric field in 3-space, by a magnetic field strength, etc. This restriction is to be compared with the use of binary encoding of variables in digital computers. A ‘‘physical digital computer’’ would allow encoding the value n for the variable x with $k = O(\log n)$ distinct electric fields, shaft angles, etc.

A second restriction concerns the decoding process whereby the solution of each problem instance is obtained as a function of the physical variables after operation of the physical device. It is essential to model the inherent accuracy limitations of physical sensors that must be employed to ‘‘read out’’ the solution to each instance, by assuming that each analog computer has an associated *absolute precision*, ε. We require that for any problem instance, the solution obtained from the physical device does not change when the physical variables range over an ε-neighborhood (defined using the mathematical model for physical variables) of their nominal values (i.e. the values generated by the mathematical model of the device).

We want to point out a distinction related to the precision issue. All mathematical models may be regarded as idealizations of physical reality due to unmodeled and imperfectly modeled effects. In order to discuss the operation of physical devices using mathematical models, it is important to insure that the models are robust in the sense that the physical behavior predicted by the mathematical model is not more sensitive to small changes in the model than is the underlying physical system to small perturbations. However, it is a difficult task, in general, to come up with suitable quantifications of the notion of small changes in a model. We would argue that for the purposes of investigating the limitations on analog computation arising from computational complexity theory, the use of ‘‘idealized’’ analog computers whose physical operation corresponds precisely to its mathematical description is appropriate. In some cases it will be possible to incorporate some robustness in the mathematical model explicitly through the limited precision property described above.

Finally, we make a general assumption that the physical devices used for analog computation exhibit causal, deterministic behavior: given a complete description of the device (model) at any time instant t_0, the description of the device (model) at times $t > t_0$ is *uniquely* determined by the external input acting on the device (model) during the interval $[t_0, t]$. We thus rule out quantum-mechanical systems, although in Section 9 quantum mechanics is discussed by means of an example.

Now that we have a general framework in which to study analog computation through mathematical models, we need to define our notion of the *resources* used by an analog computer. Intuitively, we associate physical resources with the operating costs of the physical device. Thus the physical size, the mass, the initial stored energy, and the time interval of operation of the device are among the resources used. In addition, the mathematical model obtained from applying physical laws will involve physical variables and possibly their time and spatial derivatives. The maximum magnitudes of all such quantities will also be regarded as resource requirements. As an example, for a particle described by Newtonian mechanics, the maximum displacement, velocity, acceleration, and applied force are all resources in addition to the mass and the time of operation of the device.

3. Combinatorial vs. Numerical Problems

An input string of length L bits can encode a number as large as 2^L, and this creates a fundamental roadblock preventing the efficient solution of certain computational problems with an analog computer. To illustrate the problem, suppose we want to compare two positive integers, n_1 and n_2. We imagine the following analog comparator. Create two particles with equal charges having masses m_1 and m_2, respectively. Place them in a uniform electric field. The transit time from a fixed starting position to another fixed ending position is proportional to $\sqrt{m_i}$, so the particle with the smaller mass arrives at the goal line first. The time complexity of the computation is $T(L) = O(\min_{\{i=1,2\}} \sqrt{m_i})$.

We are left to decide how the masses are to encode the numbers. In order to obtain a machine that does not depend on the specific problem instance, the encoding should be a monotonic function. Suppose we let $\sqrt{m_i} = f(n_i)$. Then if f is a polynomial, the time $T(L)$ is exponential in L. To keep the time complexity polynomial, therefore, we should choose f to be logarithmic. But this leads to accuracy problems: for adjacent large numbers the masses will be so close together that we will have to make the physical size of the machine exponentially large to discriminate between the arrival times. (Or what is the same thing, we will need to discriminate between times that are exponentially close together.)

The difficulty is caused by the fact that the size of a physical quantity (mass in this case) is used to encode a number that is binary-encoded in the input sequence. We can state this result in general terms as follows.

Theorem 1. *Suppose an analog computer encodes an input variable x that appears in the input string in binary form by the physical quantity $f(x)$. Suppose that the number of different values that may be taken on by x is not bounded by any polynomial in L, the size of a problem instance. Then the norm of the physical variable $f(x)$ is not bounded by any polynomial in L.*

Proof. Let ε be the absolute precision associated with the analog computer, and suppose that $f(x)$ takes its values in p-dimensional space (where p is fixed over all problem instances). If the norm of $f(x)$ is bounded by the polynomial L^k, the volume of the corresponding sphere in p-dimensional space is $O(L^{pk})$. However in order to be distinguishable, each possible value of $f(x)$ must be surrounded by a sphere of diameter ε, and hence volume $O(1)$ with respect to L. Clearly, there can be only polynomially many values taken on by $f(x)$. This proves the result by contraposition. $\square$

This result shows the futility of searching for (asymptotically efficient) analog computers to solve problems involving large numbers. NP-complete problems such as the PARTITION problem and the INTEGER KNAPSACK problem fall in this class. However, it turns out that there are other NP-complete problems whose problem instances consist only of input strings corresponding to numbers that are bounded by some polynomial in L, the size of the instance. This is the class of so-called *strongly NP-complete* problems, and it contains such problems as HAMILTON CIRCUIT, 3-SAT, and others [9].

4. A Polynomial Analog Machine

Our example of comparing two integers shows that some problems that are ‘‘easy’’ for digital computers, e.g. solvable in linear time, are inherently difficult for analog computers because of the nature of the numerical representation of analog quantities. We now provide an example to show that analog computers can be found that do in fact solve some (‘‘easy’’) combinatorial problems with polynomial resources. These problems cannot have numbers encoded in the input string that get exponentially large. After that we will turn to the more interesting class of seemingly intractable problems, the strongly NP-complete problems.

Consider the following problem:

GRAPH CONNECTIVITY: Given a graph $G = (V, E)$ and two distinguished vertices $s, t \in V$, is there a path in G from s to t?

Notice that an instance of this problem can be encoded in such a way that the largest number in the problem description is only polynomially large as a function of the length of the input, L. We will call such problems *combinatorial*. This problem can be solved in polynomial time on a Turing machine, and we say that such problems are in *Digital P-time*. The amount of tape used by a Turing machine computation can be no larger than the number of time steps, and it uses no resources other than time and tape (‘‘space’’). Therefore, a problem in Digital P-time is also guaranteed to use no more than a polynomial amount of *resources* on a Turing machine. On the other hand, an analog computer can conceivably operate successfully in polynomial time but require an exponential amount of some other resource, such as torque or instantaneous current. We will therefore want to insist that a ‘‘fast’’ and well behaved analog computation use total *resources* polynomial in the input description.

It is now easy to show that GRAPH CONNECTIVITY can be solved by an analog machine with polynomial resources. Make an electrical network out of the graph, as shown in Fig. 1, putting a wire of constant resistance per unit length wherever there is an edge, and joining the wires at the nodes. Apply a voltage source of size $|V|$ volts between nodes s and t, and measure the current. If there is a path between s and t there will be a resistance of at most $|V|$ ohms between them, and a steady-state current of at least 1 ampere will flow. If there is no path, the resistance will be high and the current will ultimately go to zero.

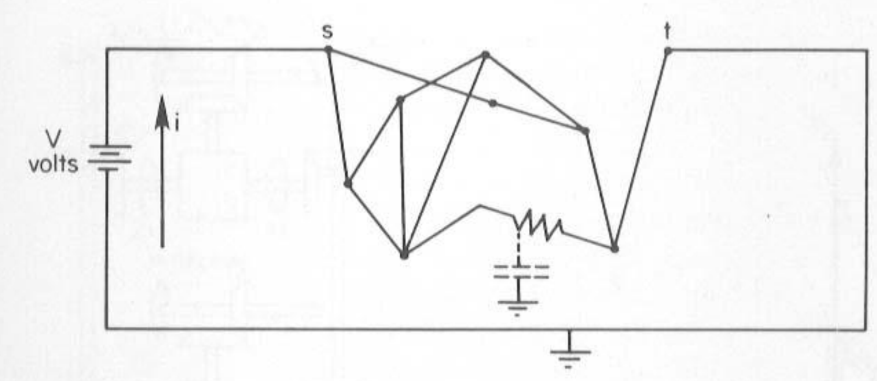


Fig. 1. An electrical network to solve GRAPH CONNECTIVITY in polynomial time.

The time required for the operation of this analog computer will depend on the parasitic capacitance of the circuit, which will determine the effective *RC* time constant of the circuit. If the wire lengths grow linearly with the number of edges $|E|$, the total capacitance seen by the voltage source will be no worse than proportional to $|E|^2$. Similarly, the total resistance will be no worse than proportional to the length of the longest wire and the number of edges, and so also $O(|E|^2)$. Hence, to distinguish between the cases where there is and is not a path (in the presence of fixed precision) takes time proportional to the *RC* time constant, which is $O(|E|^4)$. It is also clear that the total size and power consumption of the network are also polynomial in $|E|$.

We thus have at least one problem where an analog computer operates successfully with polynomial resources. A key question, then, is whether there is a strongly NP-complete combinatorial problem (nonnumerical in the sense described above) that can be solved with polynomial resources by some analog computer. After some preliminaries, we will describe a machine that ostensibly solves such a problem: 3-SAT. We are able to predict that this machine cannot operate efficiently.

5. A Class of Mathematical Programming Machines

Consider the following instance of a linear programming problem:

$$\begin{aligned} &\max \quad z = 2x_1 + x_2 \\ &w = x_1 + x_2 \le 1 \\ &x_1 \ge 0, \quad x_2 \ge 0 \end{aligned} \tag{5.1}$$

The optimum solution of (5.1) is $x_1 = 1, x_2 = 0$ (see Fig. 2).

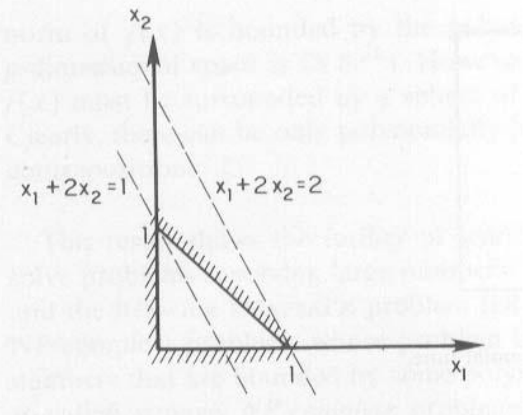


Fig. 2. An instance of LINEAR PROGRAMMING.

We propose the following analog computer for this problem. Each of the variables x_1, x_2 will be represented by the angular position of a shaft. Shaft positions can be negated by a simple 1:1 gear coupling and can be multiplied by a constant $-|c|$ by a $1{:}c$ gear coupling (see Fig. 3). Two shaft positions can be subtracted with a *differential gear* (see Fig. 4). The differential forces the angles *p*, *q*, *r* to satisfy the equation $p - q = r$. A full description of it can be found in [20]. (Differentials are used in automobile transmissions.) To preserve symmetry, we shall make the assumption that the differential *adds* the angles p and q; this can be accomplished easily by incorporating into it an inverter for the angle q.

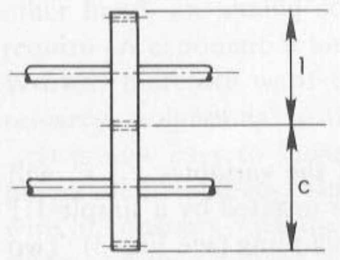


Fig. 3. A 1:c gear coupling.

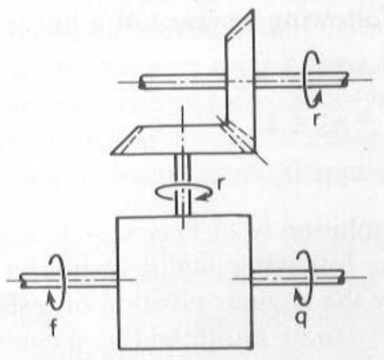


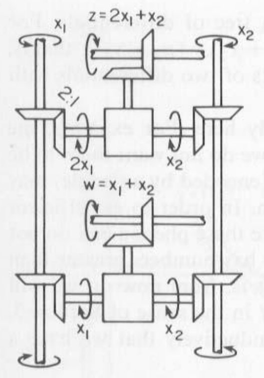
Fig. 4. A differential.

We use the above primitives for multiplication by a constant and summation to solve (5.1), as shown in Fig. 5. We have four shafts whose angular positions represent the variables x_1, x_2, w and z. Their angular positions are not independent; the differentials and gear couplings enforce the relationships: $z = 2x_1 + x_2$, $w = x_1 + x_2$. Hence we have two degrees of freedom. We can set the angular positions of any two shafts to any desired values, and this will fix the angular positions of the other two shafts. The constraint $x_1 + x_2 \leq 1$ can be imposed by putting a stop at position 1 of the shaft representing $w = x_1 + x_2$. The constraints $x_1 \geq 0$, $x_2 \geq 0$ can be imposed similarly, by putting stops at positions 0 of the shafts representing x_1 and x_2.

Suppose that we start the machine at the feasible state $x_1 = 0$, $x_2 = 0$. Then we can maximize $z = 2x_1 + x_2$ (under the constraints of (5.1)), by simply rotating the shaft representing z towards increasing values, as far as possible. Since the angular positions x_1 and x_2 always satisfy the constraints imposed by the stops, the maximum angular position of the z shaft will be the optimum solution of (5.1).

Now consider the dynamics of the machine. As we start rotating the z shaft towards increasing values of angular position, $2x_1 + x_2$ will increase from 0 to 2. Since the x_1, x_2, and w shafts are left alone (except for the stops), we are basically using only one degree of freedom. Thus there are many feasible paths from the initial point $(x_1, x_2) = (0, 0)$ to the final point $(x_1, x_2) = (1, 0)$; the one followed will be determined by the relative values of the various friction coefficients inside the machine. For example, assume that the shaft representing x_1 is much harder to turn than the shaft representing x_2. Then, as we are increasing z, it is possible that x_1 remains at position 0, and $x_2 = z$. That is, the differential enforcing $z = 2x_1 + x_2$ ‘‘chooses’’ to distribute the angle z as: $x_2 = z$, $x_1 = 0$. However, when w reaches 1, w cannot increase further because of the stop at position $w = 1$. At this point,

x_2 cannot increase any more, but if the force applied to the z shaft is large enough to overcome the resistance of the x_1 shaft, x_1 will start increasing. Since $w = x_1 + x_2 \le 1$, x_2 will decrease until it reaches the stop at position $x_2 = 0$. This way, the path p_1 shown in Fig. 6 will be followed.



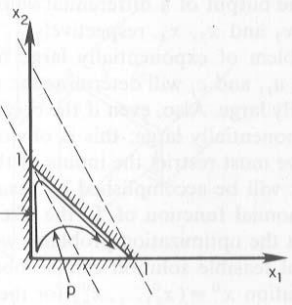


Fig. 5. A machine that solves an instance of LINEAR PROGRAMMING.

Fig. 6. Two feasible paths directed towards increasing values of the objective function.

In general, a path like p will be followed; p has the property that it is directed towards increasing values of z. The actual path p will be determined by the machine's preferred direction in the state space at each state. This is determined by the relative friction coefficients inside the machine. We ensure that the directions towards increasing values of z are achieved by forcing z forward, with a force greater than the total frictional resistance.

The above can obviously be extended to the general instance of linear programming:

$$\min \quad z = \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1, 2, \ldots, m$$

$$x_j \ge 0, \quad j=1, 2, \ldots, n.$$

(Without loss of generality we can assume that c_j, a_{ij}, and b_i are integers.)

The summation of more than two variables can be done using a tree of differentials. For example $y = x_1 + x_2 + x_3 + x_4$ can be enforced as implied by $y = [(x_1 + x_2) + (x_3 + x_4)]$; that is, y will be the output of a differential whose two inputs are the outputs of two differentials with inputs x_1, x_2 and x_3, x_4, respectively.

The problem of exponentially large numbers appears quite vividly here. For example, the coefficients a_{ij} and c_j will determine the ratios of the gear couplings; we do not want them to be exponentially large. Also, even if the coefficients are small, a variable (encoded by an angle) may become exponentially large; this is obviously an undesirable situation. In order to get efficient solutions, we must restrict the inputs of the machine to instances where these phenomena do not occur.

That will be accomplished by assuming that an instance never has numbers greater than some polynomial function of L, the size of the input encoding. That is, from now on we will assume that the optimization problems we deal with are *combinatorial* in the sense of Section 3.

An initial feasible solution can be obtained as follows: Assume inductively that we have a feasible solution $x^0 = (x_1^0, \ldots, x_n^0)$ for the first k constraints

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, 2, \ldots, k.$$

If

$$\sum_{j=1}^{n} a_{k+1,j} x_j^0 \leq b_{k+1}$$

then x^0 is a feasible solution for the first $k+1$ constraints; otherwise minimize

$$\sum_{j=1}^{n} a_{k+1,j} x_j$$

subject to the first k constraints with initial feasible solution x^0. If its minimum value is $\leq b_{k+1}$, the value of $x = (x_1, \ldots, x_n)$ that does this is a feasible solution for the first $k+1$ constraints; otherwise the problem is infeasible.

This technique can be further extended to the mathematical programming problem

$$\begin{aligned} &\min \quad z = f(x) \\ &g_i(x) \leq 0, \quad i = 1, 2, \ldots, m \end{aligned} \tag{5.2}$$

assuming, of course, that we have the devices that can enforce the relations $y_i = g_i(x_1, \ldots, x_n)$ and $z = f(x_1, \ldots, x_n)$. Then a local optimum x^* will be found; the machine will move (in the state space) from its initial state to x^* along a path that is directed towards decreasing values of z. In Fig. 7 we depict a device that realizes a smooth function f.

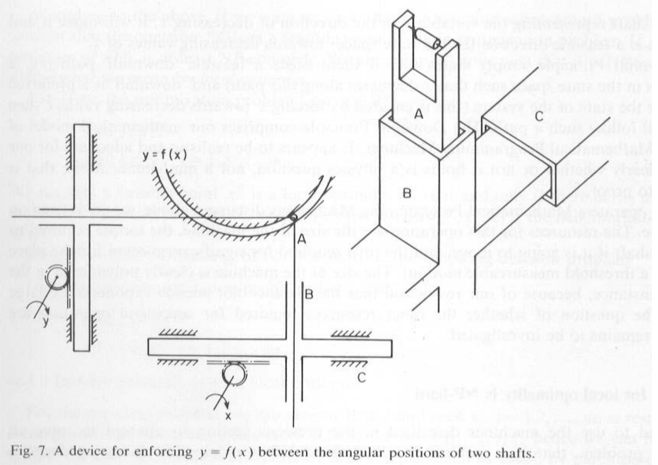


Fig. 7. A device for enforcing $y = f(x)$ between the angular positions of two shafts.

For example, if f is the function of one variable $z = x^2$, this device, called a ‘‘squarer,’’ enforces $z = x^2$ between the angular positions of two shafts. More complex relationships can be enforced. For instance $z = x \cdot y$ can be enforced with two squarers, three differentials and a 1:4 gear coupling, as implied by the formula $z = [(x + y)^2 - (x - y)^2]/4 = x \cdot y$.

We call such devices ‘‘Mathematical Programming Machines.’’ With them, the feasible space can be mapped out by simply rotating the $x_1, \ldots, x_n$ shafts; each combination of angular positions of these shafts corresponds to a point in the feasible space. When, in our attempt to find a local optimum, we rotate the z shaft towards decreasing values, we are tracing out some path in the feasible space.

An important restriction is that these machines can only find *local* optima. Consider for instance an optimization problem with a nonconvex feasible space, for example

$$\min \quad z$$

under the constraint

$$z \le h(x) .$$

Assume that x^0 is a local minimum of h (but not a global minimum). Put $z^0 = h(x^0)$. Then, if we initialize the machine to (x^0, z^0) and try to decrease the value of the position of the z shaft, this shaft will not move. The reason is that there is no way for it to move in the feasible space from point (x^0, z^0) to a point (x^*, z^*), where $z^* < z^0$, without first passing through a higher value of z. The machine will tend to move (in the feasible space) to a new state (x', z') such that $z' < z^0$; it tends to move along a direction such that the projection of the gradient of the objective function on this direction is negative (since we are trying to decrease the value of the objective function). We formalize this intuitive notion by the following principle.

Downhill Principle. Let S be a Mathematical Programming Machine whose shaft positions (state variables) x_i satisfy the set of relations (5.2). Then if we start it at a feasible state x^0 and apply a force to the shaft representing the variable z in the direction of decreasing z, it will move if and only if there is a feasible direction (in the state space) towards decreasing values of z.

The Downhill Principle simply states that if there exists a feasible ‘‘downhill’’ path (i.e. a feasible path in the state space such that z decreases along this path) and ‘‘downhill’’ is a preferred direction for the state of the system (this is ensured by forcing z towards decreasing values), then the state will follow such a path. The Downhill Principle comprises our mathematical model of physics for Mathematical Programming Machines. It appears to be realistic and adequate for our purposes; clearly whether or not it holds is a physics question, not a mathematical one that is susceptible to proof.

When we operate a Mathematical Programming Machine as discussed above, we are relying on this principle. The resources required for this operation are the size of the machine, the torque required to move the z shaft if it is going to move, and the time required for significant motion to take place (in terms of a threshold measurable motion). The size of the machine is clearly polynomial in the size of the instance, because of our restriction that the instance not encode exponentially large numbers. The question of whether the other resources required for successful operation are polynomial remains to be investigated.

6. Checking for Local Optimality is NP-hard

We intend to use the machines described in the previous section to attempt to solve an ‘‘interesting’’ problem, that is a problem that is at least NP-complete. The following decision problem is NP-hard [9]:

QUADRATIC PROGRAMMING: Given a linear objective function, constraints involving quadratic functions, and a constant K, does there exist a vector that makes the value of the objective function less than or equal to K, while satisfying the constraints?

We could try to solve QUADRATIC PROGRAMMING with the machines described in the previous section. However, since these machines can only find local optima, for certain initial conditions (i.e. certain initial values of the variables), the machine would get stuck at a local optimum. Then we would have to rotate back the z shaft (i.e. the shaft representing the value of the objective function), change the values of the other shafts, and try again. Obviously we cannot assert that this procedure requires polynomial time. (It is interesting to note that an electrical network for solving quadratic programming problems is proposed in [5], but it is based on sufficiency of the Kuhn-Tucker conditions, and depends for its operation on positive definiteness of the quadratic form. At first this might seem to be another candidate for an analog machine that solves an NP-complete problem, but in fact Quadratic programming with a positive-definite matrix can be solved in Digital P-time with a variant of the ellipsoid algorithm [19, Chapter 15, Problem 16].)

If, however, we were asked only to find a local optimum, we would need to try only once, rotating the z shaft towards decreasing values until it gets stuck. This point must be a local optimum by the Downhill Principle. There is of course the question of how long we would need to rotate it, which depends on how far the local optimum is from the initial state. But this problem would also disappear if we knew what this ‘‘candidate’’ local optimum is; we could initialize the machine to this point and then just try to rotate the z shaft. If it does not move then this point is a local optimum; if it does move, then it is not.

According to the above, if we want to show that our machine solves a hard problem, we need to show that the question, ‘‘Given a feasible point x^0 of an optimization problem Π, is x^0 a local optimum of Π?’’ is at least NP-complete. We are going to prove next that it is NP-hard. First we define what we mean by *local optimum*.

Let Π be the optimization problem: for $x \in R^n$,

$$\min \quad f(x)$$

$$g_i(x) \le 0, \quad i = 1, 2, \ldots, m$$

We say that a feasible point x^0 is a local optimum of Π if and only if there exists an $\varepsilon > 0$ such that for every feasible $x \in N(x^0, \varepsilon)$, $f(x) \ge f(x^0)$, where $N(x^0, \varepsilon)$ is the neighborhood $\|x - x^0\| \le \varepsilon$.

Local Optimality Checking, or LOC for short, is the following decision problem:

LOC: Given an optimization problem: for $x \in R^n$,

$$\min \quad f(x)$$

$$g_i(x) \le 0, \quad i = 1, 2, \ldots, m \tag{6.1}$$

and a feasible point x^0, is x^0 a local optimum?

For the encoding problem, we can assume that f and each g_i, $i = 1, 2, \ldots, m$, is restricted to be a composition of functions taken from a fixed set $S = \{ f_1, \ldots, f_k \}$; hence we can have a fixed symbol for each f_i. Function composition can be represented as usual, by parentheses.

Theorem 2. *LOC is NP-hard for optimization problem (6.1) even if the functions involved are (a) linear and piecewise linear or (b) linear and quadratic.*

Hence the search for local optima in nonlinear optimization is a very hard problem indeed. Even if we have a candidate point we cannot decide in polynomial time if it is or isn't a local optimum (assuming P $\neq$ NP).

Proof. To prove Theorem 2, we reduce 3-SAT to LOC. For reference, 3-SAT is the following problem, which is one of the earliest known NP-complete problems, and which is strongly NP-complete as well [9].

3-SAT: Given a set of Boolean variables $X_1, \cdots, X_n$, and given B, a Boolean expression in conjunctive normal form with exactly 3 literals per clause:

$$B = (Z_{11} + Z_{12} + Z_{13})(Z_{21} + Z_{22} + Z_{23}) \cdots (Z_{m1} + Z_{m2} + Z_{m3})$$

where each literal Z_{jk} is either some variable X_i or its negation $\bar{X}_i$, is there a truth assignment for the variables X_i which makes B *TRUE* ?

For each instance of 3-SAT, we will construct an instance of a problem in real variables x_i. By example, for each clause in B that looks like

$$(X_1 + \bar{X}_3 + X_7)$$

we write an inequality of the form

$$x_1 + \bar{x}_3 + x_7 \geq x_0 \tag{6.2}$$

on real variables $x_0, x_i, \bar{x}_i, i = 0, 1, \ldots, n$. Also, we add the constraints

$$\bar{x}_i \leq f(x_i), \quad i = 1, 2, \ldots, n \tag{6.3}$$

where f is the piecewise linear function

$$f(x) = (4|x| - 5x)/3$$

In general, if the literal X_i appears in the clause, we include the term x_i in the l.h.s. of (6.2); if the literal $\bar{X}_i$ appears in the clause, we include the term $\bar{x}_i$.

The optimization problem is

$$\max \quad x_0$$

subject to the constraints of the form (6.2) (there will be m such constraints if B has m clauses) and (6.3).

Call this problem REAL 3-SAT, and let B' be the instance of REAL 3-SAT corresponding to B.

Claim 1. *For each satisfying assignment S of B, there exists a direction* $(d_1, \ldots, d_n) \in R^n$ *and a direction* $(\bar{d}_1, \ldots, \bar{d}_n) \in R^n$

where

$$(d_i, \bar{d}_i) = (3, -1) \quad \text{if} \quad X_i = \mathit{TRUE},$$

and

$$(d_i, \bar{d}_i) = (-1, 3) \quad \text{if} \quad X_i = \mathit{FALSE}$$

in A, such that

$$x_0 = \theta,$$
$$(x_1, \ldots, x_n) = (d_1, \ldots, d_n)\cdot\theta,$$
$$(\bar{x}_1, \ldots, \bar{x}_n) = (\bar{d}_1, \ldots, \bar{d}_n)\cdot\theta,$$

is a feasible solution of B' for any $\theta \geq 0$.

Proof. These values of the real variables satisfy the inequalities (6.2) because each l.h.s. has at least one term that equals 3θ (at least one literal is *TRUE*). The sum of the other two terms can be no less than -2θ, since each of them is either 3θ or $-\theta$. Also, the constraints (6.3) are satisfied, since if $x_i = 3\cdot\theta$ then $f(x_i) = -\theta$ and if $x_i = -\theta$ then $f(x_i) = 3\cdot\theta$; in both cases $\bar{x}_i \leq f(x_i)$ □

Claim 2. *If B' has a feasible solution with $x_0 > 0$, then B is satisfiable.*

Proof. If $x_0 > 0$ then each l.h.s. of (6.2) must have a positive term. If x_i is positive, put $X_i = \mathit{TRUE}$; if $\bar{x}_i$ is positive, put $X_i = \mathit{FALSE}$. No variable can be set both *TRUE* and *FALSE* by that rule, since we cannot have both x_i and $\bar{x}_i$ positive; this follows directly from (6.3). Thus every clause has a true literal and B is satisfiable. □

We have:

B is satisfiable $\Longrightarrow$ (by claim 1) $x^0 = (0, \ldots, 0)$ is not a local optimum of $B' \Longrightarrow$
B' has a feasible solution with $x_0 > 0 \Longrightarrow$ (by claim 2) B is satisfiable.

Therefore B is satisfiable if and only if the feasible point $x^0 = (0, \ldots, 0)$ is not a local optimum of B'.

To obtain a reduction to LOC when the functions involved are only linear and quadratic, we write (6.3) as

$$\bar{x}_i \leq (-5x_i + 4|x_i|)/3, \tag{6.4}$$

which is equivalent to

$$\bar{x}_i \leq (-5x_i + 4y_i)/3,$$

$$y_i \le |x_i| .$$

This is equivalent to

$$\bar{x}_i \le (-5x_i + 4y_i)/3,$$
$$y_i^2 \le x_i^2 \tag{6.5}$$
$$y_i \ge 0. \quad \square$$

7. A 3-SAT Machine

We could attempt to solve the ‘‘piecewise-linear’’ version of REAL 3-SAT with a Mathematical Programming Machine if we could find a machine that realizes the piecewise linear function $f(x)$. However, such a machine probably cannot be realized, because of the discontinuity of the derivative of f at $x = 0$. (Consider what happens if we move the shaft representing x with constant velocity past the point $x = 0$: the velocity of the shaft representing $y = f(x)$ will be discontinuous.) However, in the ‘‘quadratic’’ version of REAL 3-SAT, only smooth functions are involved. We can construct a machine implementing REAL 3-SAT using differentials, gear couplings and a squarer. The squarer can be implemented by a device like the one shown in Fig. 7, with $f(x) = x^2$.

However, we choose not to try to implement (6.5), in order to avoid the introduction of the n new variables y_i, $i = 1, 2, \ldots, n$. Instead, we write (6.4) as

$$\bar{x}_i + 5x_i/3 \le 4|x_i|/3 \quad \Longleftrightarrow \quad (3\bar{x}_i + 5x_i)/4 \le |x_i| . \tag{7.1}$$

We denote by SQ^+ the set of continuously differentiable functions $F(x)$ satisfying

$$F(x) = x^2 \quad \text{if} \quad x \ge 0$$
$$F(x) \le 0 \quad \text{if} \quad x \le 0.$$

If $F(x)$ is a function in SQ^+, it can be implemented with a device similar to the squarer, and (7.1) is equivalent to

$$F((3\bar{x}_i + 5x_i)/4) \le x_i^2 .$$

Hence, we will try to implement the optimization problem

$$\max \quad x_0 \tag{7.2}$$

under m constraints of the form

$$x_1 + \bar{x}_3 + x_7 \ge x_0 \tag{7.3}$$

and under n constraints of the form

$$F((3\bar{x}_i + 5x_i)/4) \le x_i^2 . \tag{7.4}$$

This optimization problem is equivalent to the two optimization problems discussed in the previous section.

As discussed in Section 5, we can find out if the point $x^0 = (0, \ldots, 0)$ is a local optimum by initializing the machine to x^0 and then applying a torque to the shaft representing x_0. If we accept the Downhill Principle, x^0 is not a local optimum if and only if the shaft representing x_0 moves. A way to interpret this conclusion intuitively is to say that if ‘‘infinitely accurate’’ analog devices could be built, then they could be used to solve 3-SAT arbitrarily fast.

To summarize the discussion so far, we have arrived at a Mathematical Programming Machine that solves 3-SAT by testing the local optimality of the origin in the derived problem, REAL 3-SAT. The machine uses only gears and smooth cams and is only polynomially large. If we grant Strong Church’s Thesis, that P ≠ NP, and the Downhill Principle, we must conclude that such a machine takes exponential resources.

There are two ways to follow up on this finding. The first is an experimental study to test the hypothesis that the 3-SAT machine requires exponential resources as predicted by our theory. Alternately, we can replace the Downhill Principle with a more detailed mathematical model for the operation of the machine, for instance one based on classical Newtonian dynamical equations; this model can be examined in detail with a view toward an analytical verification of the hypothesis for an ‘‘ideal’’ analog computer. We have carried out a lengthy study to confirm that the prediction of exponential resource complexity is not altered by taking into account the precision of gear ratios, the precision of the numbers 3/4 and 5/4 in (7.4), and the precision of the initial shaft positions. One speculation about the machine is that for polynomially bounded input torque, the time for operation is exponential, with ergodicity somehow playing a significant role.

In the next section we show how to simulate efficiently analog computers described by a class of ordinary differential equations. Therefore, we can conclude that no machine that can be modeled by (7.2) - (7.4) can be described by such differential equations.

8. SCT for Analog Computers Described by a Class of Ordinary Differential Equations

Our interest in this section will be directed towards a class of ‘‘general purpose’’ analog computers, as opposed to specialized devices designed for solving particular combinatorial problems. We will introduce a class of differential equations, together with an interpretation of their computational processes, which corresponds to a model for analog computers of the Bush Differential Analyzer type [4]. The first steps toward a theory of analog computation were taken by C. Shannon, who showed that an interconnection of primitive devices — adders, scalar multipliers, and integrators — constrained by some natural conditions to ensure well-posedness, generates functions solving ordinary differential equations of a particular form [26]. Pour-El [22] added some necessary elaborations concerning existence of unique solutions. This work derived the following form for the differential equations corresponding to a (Bush) analog computer:

$$A(Y(t))\,\frac{dY(t)}{dt} = b(Y(t)) \;,\;\; Y(t_0) = Y_0 \tag{8.1}$$

where the matrix A and vector b have entries composed of linear combinations of the component functions of $Y(t) = (1,t,y_1(t),\cdots,y_n(t))'$. Here $t \in I$, a compact interval of the real line.

The following example shows that the notion of analog computation considered by Shannon and Pour-El does not account for the limitations that are inherent in the general model of analog computer developed in previous sections. From work of Plaisted [21] it is known that solution of the NP-complete PARTITION problem is equivalent to the evaluation of a particular definite integral. The integral may be computed as the solution to a differential equation of the form (8.1). However, PARTITION is not strongly NP-complete, and a specialization of the arguments used in Section 3 shows that the analog variables associated with the differential equation are not polynomially bounded.

(Certain derivatives grow exponentially, which cannot be remedied by scaling the independent variable $t \rightarrow t/\tau$, while keeping the time interval of interest polynomially bounded.)

From our perspective, the Shannon/Pour-El work, with its emphasis on computability, concerns the weak form of Church's Thesis. The PARTITION example suggests that further restrictions and interpretations of the differential equation model are necessary in order to capture the inherent accuracy limitations of analog computation or to make a meaningful statement about complexity.

Our restricted differential equation model is derived from the form

$$\frac{dY(t)}{dt} = C(Y(t)) \ , \ \ Y(t_0) = Y_0 \tag{8.2}$$

where $C(Y(t))$ is a vector of rational functions of the elements of $Y(t) = (1, t, y_1(t), \cdots, y_n(t))'$. This form is equivalent to (8.1) when A is nonsingular. We assume that the analog computer has a precise (e.g. external) clock so that the first two components of $Y(t)$ are redundant and t may be used to parametrize the functions $(y_1(t), \cdots, y_n(t))' = \mathbf{y}(t)$. Then (8.2) may be replaced by the equivalent form

$$\frac{\mathbf{dy}(t)}{dt} = f(\mathbf{y}(t), t) \ , \ \ \mathbf{y}(t_0) = \mathbf{y}_0 \tag{8.3}$$

We make the usual assumption for well-posedness of the model, namely that f obeys a uniform Lipschitz condition:

$$\| f(\mathbf{y}_1, t) - f(\mathbf{y}_2, t) \| \leq \lambda \| \mathbf{y}_1 - \mathbf{y}_2 \| , \quad t \in I ,$$

where λ does not depend on t.

Our interpretation of the computation carried out by (8.3) is a variation on the standard initial value problem of computing the value of $\mathbf{y}(t_f)$ given $[t_0, t_f] \subseteq I$ and $\mathbf{y}_0 \in \mathbf{R}^n$. Here $\mathbf{y}_0$ is the ''input'' to the analog computer which then operates over a fixed time interval to generate its ''output.'' We make the following qualification to incorporate the absolute precision, ε, associated with the use of the differential equation to represent the operation of an analog computer. The value that is provided as an ''output'' is any $\mathbf{y}^*$ that approximates $\mathbf{y}(t_f)$ in the sense that $\| \mathbf{y}(t_f) - \mathbf{y}^* \| \leq \varepsilon$.

We adopt as our measure of the resources used by this analog computation

$$R = \max_{t_0 \leq t \leq t_f} \| \ddot{\mathbf{y}}(t) \| \ .$$

This assumes that (8.3) admits a solution whose second derivative exists and is continuous. Since the derivative $\dot{\mathbf{y}}$ cannot grow large in a fixed time interval without $\ddot{\mathbf{y}}$ being large, this is a conservative measure. (In a typical electronic analog computer, for example, the signal $\dot{\mathbf{y}}$ appears as a physical (voltage) signal at the integrating amplifier input. Since a real integrator has finite bandwidth, the time derivative of its input must be bounded to assure accurate integration.)

We summarize this model for analog computation. It consists of the differential equation (8.3), with the associated Lipschitz constant λ and absolute precision constant ε. The solution at time t_f when the initial condition at time t_0 is $\mathbf{y}_0$ and the precision constant ε determine an equivalence class of ''output'' vectors. The maximum magnitude of the second derivative of the solution vector over the interval $[t_0, t_f]$, is used as a measure of the resources required by the computation.

Our result is that these analog computations can be efficiently simulated by a digital computer, the strong form of Church's Thesis.

Theorem 3. *The differential equation model described above can be simulated by a digital computer in a number of steps bounded by a polynomial in both R and* $1/\varepsilon$.

Proof We give a constructive proof based on the Euler method of numerical integration for the system (8.3). (Our approach is based on standard techniques in numerical analysis; see [11].) For $0 \le m \le N$, set $t_m = t_0 + mh$ where $h = (t_f - t_0)/N$. Then we take $\gamma_0 = \mathbf{y}_0$ and

$$\gamma_{m+1} = \gamma_m + h\, f(\gamma_m, t_m) \quad , \qquad 0 \le m \le N-1$$

Supposing that γ_m is computed without roundoff error, the discretization error, $\mathbf{e}_m = \gamma_m - \mathbf{y}(t_m)$, satisfies

$$\mathbf{e}_{m+1} = \mathbf{e}_m + h[f(\gamma_m, t_m) - f(\mathbf{y}(t_m), t_m)] - h^2\, \ddot{\mathbf{y}}(\xi)/2$$

for some ξ, $t_m < \xi < t_{m+1}$. Using the Lipschitz condition gives

$$\|\mathbf{e}_{m+1}\| \le \|\mathbf{e}_m\|(1 + h\lambda) + h^2 R/2$$

which leads to the bound

$$\|\mathbf{e}_N\| \le h\, R[\exp(t_f - t_0)\lambda - 1]/2\lambda$$

When fixed point numerical calculations are used, we have a roundoff error sequence $\mathbf{r}_m$ due to finite precision computation. The numerical approximation is

$$\gamma^*_{m+1} = \gamma^*_m + [h f^*(\gamma^*_m, t_m)]^*$$

where the * denotes rounded value. We define the local roundoff error by

$$\delta_m = [h f^*(\gamma^*_m, t_m)]^* - h f(\gamma^*_m, t_m)$$

so that we may write

$$\gamma^*_{m+1} = \gamma^*_m + h f(\gamma^*_m, t_m) + \delta_m$$

If we assume that $\|\delta_m\| < \sigma$, then by an argument similar to the one used for discretization error, we obtain

$$\|\mathbf{r}_N\| \le \sigma[\exp(t_f - t_0)\lambda - 1]/h\lambda$$

Using the triangle inequality, we may combine these bounds to obtain a total error bound

$$\|\mathbf{y}(t_f) - \gamma^*_N\| \le h\left[\frac{R}{2} + \frac{\sigma}{h^2}\right][\exp(t_f - t_0)\lambda - 1]/\lambda \qquad (8.4)$$

In order to obtain a solution to the same accuracy as the differential equation model for analog computation, we must choose σ and h so that this bound is no larger than ε. We have the relationship $h = (t_f - t_0)/N$ and we take $\sigma = 1/N^2$ for convenience. Then it is clear from (8.4) that the number of discretization steps may be chosen to be proportional to R and to $1/\varepsilon$. Since f is rational, each step involves additions, multiplications, and divisions in order to evaluate the approximate solution value. To obtain the bound on local roundoff error of σ requires $|\log_2(\sigma)|$ bits of accuracy, and so the effort in evaluating each approximate value is proportional to $\log_2^2(\sigma)$. With the choice of $\sigma = 1/N^2$, the number of steps required by a digital computer to simulate the analog computer is $O(N \log^2 N)$, which is bounded by a polynomial in R and $1/\varepsilon$ as was to be shown. This completes the proof.

What remains is to describe how such an analog computer may be used to solve combinatorial problems of the type described earlier; we first have the implicit encoding of each combinatorial variable as an analog variable (possibly vector-valued but with *fixed* dimension) which appears as part of the vector $\mathbf{y}(t)$. The initial value $\mathbf{y}(t_0)$ encodes the values of the combinatorial input quantities. Finally, we assume that the solution of the combinatorial problem may be obtained unambiguously from the ''output,'' which is to say that function assigning the value of the combinatorial solution is constant on the sets (quantization bins) $Q(\mathbf{y}_0) = \{\mathbf{y}^* : \|\mathbf{y}(t_f) - \mathbf{y}^*\| \leq \varepsilon\}$.

In view of the theorem above, this interpretation implies the following result.

Corollary. *If a combinatorial problem can be solved on an analog computer of the type described above, then it can be solved in polynomial time on a Turing machine.*

In view of the proof of the theorem, it is possible to allow certain of the parameters of the differential equation to depend on the input in addition to the initial value $\mathbf{y}_0$. This may be important in applications because the number of variables, and hence the dimension of $\mathbf{y}(t)$, n, will generally depend on the input. We may allow n and the complexity of the rational function components of f to grow polynomially in the length of the input, and we may allow the corresponding Lipschitz constant, λ, to grow logarithmically in the length of the input.

Recently, Hopfield and Tank [12] have discussed solving the strongly NP-complete TRAVELING SALESMAN PROBLEM (TSP) with an analog electrical network whose description is given by a coupled set of differential equations. Given a problem instance of TSP, they propose to design the network in such a way that an associated potential (or Lyapunov) function achieves its global minimum value at an equilibrium point of the network corresponding to the TSP solution path. Then, if the initial conditions of the analog variables lie in the region of attraction of this particular equilibrium point, the steady-state solution provides the desired solution to the problem instance. Empirical studies of 10-city and 30-city problem instances are given in [12]; they indicate that while this analog approach does not provide a method for obtaining exact (i.e. optimal) solutions to TSP instances, it does offer a systematic technique for consistently generating good suboptimal solutions.

We view these empirical results as evidence for the validity of SCT in this context, which differs in some details from that considered in Theorem 3 and its corollary. The analysis and interpretation in [12] reinforces this view, especially in regard to two difficulties with constructing and operating the network. First, the network and its associated potential function depend on some free parameters (amplifier gains, etc.) that must be chosen by empirical means in order to ''tune'' the analog encoding of TSP. Second, the choice of unbiased initial conditions is apparently difficult because of the symmetry in the network arising from multiple encodings of TSP solution paths without regard to equivalence of tours (with respect to starting city and orientation).

Other points made by Hopfield and Tank suggest topics for further research in analog complexity theory. These include a study of the suboptimal solutions obtained for the TSP, a study of the ''fail-soft'' fault-tolerance properties of analog computers, a study of the use of penalty function techniques to obtain constraint satisfaction in the underlying combinatorial problem, and a detailed examination of proposed analog networks for the solution of other combinatorial problems, including ones in Digital P-time. Examples of the latter are found in [13,27].

9. The Spin Glass Computer

Recent work on the properties of certain magnetic alloys called *spin glasses* has led to an interesting connection between physics and combinatorial optimization [1,15,17], and in fact suggests a physical device for solving an NP-complete problem. The problem of finding a minimum-energy spin configuration (the *ground state*) in a regular lattice model can be expressed as the following problem, which is NP-complete [15].

GROUND STATE OF A SPIN GLASS: Given an $H \times L \times W$ rectangular lattice graph (H, L, W positive integers) with edges between vertices that are adjacent in any of the three directions, an integer interaction weight $J(e)$ for each edge e, and an integer K, the *spin energy*, is there an assignment of a *spin* $s(v) \in \{-1, +1\}$ for each vertex v such that

$$- \sum_{all\ edges\ (u,\ v)} J(u,\ v) s(u) s(v) \leq K \ ?$$

The problem remains NP-complete when J is restricted to the values $\{-1,\ 0,\ +1\}$, so the problem is strongly NP-complete. We can then view a piece of spin glass as a candidate (at least in theory) for a computer that solves a strongly NP-complete problem, just as with our 3-SAT machine. (We leave aside the problem of initializing the material with the problem ‘‘inputs.’’ If there is any way at all of setting an interaction weight, the total time for preparing a piece of material should be no more than polynomial in the number of lattice points. Similarly for reading output spin values.)

The actual operation of such a spin glass computer is similar to the operation of a mathematical programming machine; it minimizes a multivariate function. By a natural extension of the point of view discussed in this paper, suitably formalized to account for quantum mechanical models of systems such as the spin glass system, we are led to a definite conclusion about the time it takes for our hypothetical piece of spin glass to reach the ground state: if Strong Church’s Thesis is true, and if P ≠ NP, then it must take an exponential amount of time. Note, however, that this result is *worst case* over all inputs (interaction weights J). Thus it may be that almost all pieces of spin glass of a given size will cool to the ground state fast, and this prospect supports the use of ‘‘simulated annealing’’ as a heuristic for combinatorial optimization problems [17]. It is worth pointing out that an implementation of this heuristic requires a source of independent random variables, and so lies outside the realm of complexity theory as discussed here. Still, a proof of convergence for this kind of stochastic relaxation algorithm has only been obtained under the assumption of an annealing schedule involving logarithmically decreasing temperature [10]; the resulting algorithm requires exponential time which is in agreement with our prediction based on assuming Strong Church’s Thesis and P ≠ NP.

10. Discussion

The question of how efficiently we can compute with general, non-digital, devices appears to be difficult indeed. It touches on problems in both mathematics and physics. We have tried in this paper to establish some link between the mathematical complexity theory of NP-completeness and classical physics, but we have not dealt with quantum mechanics, or the problem of probabilistic behavior. We have shown that the likely hypotheses P ≠ NP and Strong Church’s Thesis lead to the conclusion that analog (non-digital) computers are no more efficient than digital computers, at least in the worst-case over problem inputs, and asymptotically with the problem size. Of course these two hypotheses are important open questions, but we have been able to prove a restricted form of Strong Church’s Thesis and perhaps more general results will be forthcoming.

In recent work of Bennett [2,3], there have been discussions that are germane to analog computation and efficient simulation. He has suggested that efficient simulation of physical systems up to the errors induced by uncontrollable (environmental) influences should be possible. It is our view that the effects of uncontrollable influences must be incorporated into the mathematical model of a physical process, given for example by a system of differential equations, as a fundamental part of the description of the corresponding analog computational process.

As shown by our differential equation model, certain smoothness properties of the mathematical model can provide a natural measure for the resources used. In particular it is the second derivative of the analog variables that appears as the natural measure. It is interesting to note that in the work of Pour-El and Richards [23,24], where it was shown that the three-dimensional wave equation can transform computable initial data into noncomputable solution values, imposition of continuity conditions on the second derivative will prevent this phenomenon.

Acknowledgments

We thank Karl Lieberherr, Richard Lipton, Jacobo Valdes, and Stephen Wolfram for valuable discussions about the subject of this paper.

References

[1] F. Barahona, ‘‘On the computational complexity of Ising spin glass models,’’ *J. Phys. A 15* (1982) 3241-3253.

[2] C.H. Bennett, ‘‘The thermodynamics of computation - a review,’’ *Internat. J. Theoret. Phys. 21* (1982) 905-940.

[3] C.H. Bennett, ‘‘On the logical ‘depth’ of sequences and their reducibilities to random sequences,’’ preprint; to appear in *Inform. and Control*.

[4] V. Bush, ‘‘The differential analyzer,’’ *J. Franklin Inst. 212 (1931) 447-488.*

[5] L.O. Chua and G-N. Lin, ‘‘Nonlinear programming without computation,’’ *IEEE Trans. Circuits and Systems CAS-31* (1984) 182-188.

[6] A. Church, ‘‘An unsolvable problem of elementary number theory,’’ *Amer. J. Math. 58* (1936) 345-363. (Reprinted in [7].)

[7] M. Davis, *The Undecidable*, (Raven Press, Hewlett, NY, 1965).

[8] R.P. Feynman, ‘‘Simulating physics with computers,’’ *Internat. J. Theoret. Phys. 21* (1982) 467-488.

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (W. H. Freeman and Company, San Francisco, CA, 1979).

[10] S. Geman and D. Geman, ‘‘Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images,’’ *IEEE Trans. Pattern Analysis Machine Intell. PAMI-6 (1984) 721-741.*

[11] P. Henrici, *Discrete Variable Methods in Ordinary Differential Equations*, (John Wiley, New York, NY, 1962).

[12] J.J. Hopfield and D.W. Tank, ‘‘ ‘Neural’ computation of decisions in optimization problems,’’ *Biol. Cybernet. 52* (1985) 1-12.

[13] J.J. Hopfield and D.W. Tank, ‘‘Collective computation with continuous variables,’’ preprint; to appear in *Disordered Systems and Biological Organization.*

[14] A. Jackson, *Analog Computation*, (McGraw-Hill, New York, NY, 1960).

[15] D.S. Johnson, ‘‘The NP-completeness column: an ongoing guide,’’ *J. Algorithms 4* (1983) 87-100.

[16] W. Karplus and W. Soroka, *Analog Methods*, 2nd. Ed., (McGraw-Hill, New York, NY, 1959).

[17] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, ‘‘Optimization by simulated annealing,’’ *Science 220* (1983) 671-680.

[18] W. Miehle, ‘‘Link-length minimization in networks,’’ *Oper. Res. 6* (1958) 232-243.

[19] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[20] R.M. Phelan, *Fundamentals of Mechanical Design*, 3rd. Ed., (McGraw-Hill, New York, NY, 1970) 433-434.

[21] D. Plaisted, ‘‘Some polynomial and integer divisibility problems are NP-hard,’’ *Proc. 17th Ann. Symp. on Foundations of Computer Science* (1976) 264-267.

[22] 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.

[23] M.B. Pour-El and I. Richards, ‘‘The wave equation with computable initial data such that its unique solution is not computable,’’ *Adv. in Math. 39* (1981) 215-239.

[24] M.B. Pour-El and I. Richards, ‘‘Noncomputability in models of physical phenomena,’’ *Internat. J. Theoret. Phys. 21* (1982) 553-555.

[25] I.B. Pyne, ‘‘Linear programming on an electronic analogue computer,’’ *Trans. AIEE, Part I, 75* (1956) 139-143.

[26] C.E. Shannon, ‘‘Mathematical theory of the differential analyzer,’’ *J. Math. Phys. 20* (1941) 337-354.

[27] D.W. Tank and J.J. Hopfield, ‘‘Simple ‘neural’ optimization networks: an A/D converter, signal decision circuit and a linear programming circuit,’’ preprint, July 8, 1985.

[28] A. M. Turing, ‘‘On computable numbers, with an application to the Entscheidungsproblem,’’ *Proc. London Math. Soc., Series 2 42* (1936-7) 230-265; corrections *ibid 43* (1937) 544-546. (Reprinted in [7].)