Classical and Alternative Computing

William Gasarch

August 14, 2026

1 Introduction

In this document we discuss several models of computation.

We first discuss Turing machines and polynomial-time-bounded Turing machines. These are classical but set the stage for what alternative computing is trying to beat.

We then discuss models of computing over the reals, and the Siegelmann-Sontag model, which takes discrete inputs but uses rationals or reals.

We then discuss two approaches to factoring. The first uses optical computers and so can be considered alternative computing. Here *alternative computing* means models or hardware that depart substantially from ordinary digital, finite-precision computation.

For each model, we describe it informally, discuss its benefits and limitations, and consider possible physical realizations and/or physical approximations. A *physical realization* refers to hardware that implements the essential assumptions of the mathematical model, not merely software that simulates or approximates the model. A *physical approximation* refers to simulations or other implementations that do not fully capture the model.

2 Turing Machines

In 1936 Turing defined the model of computation now called *the Turing machine*. For this exposition we do not need to know the formal definition. We only need to know the following:

1. Turing machines are discrete. The input is a finite string of bits, each operation is discrete, and the output (if it has one—it might not halt) is a finite string of bits.

2. Any function that can be computed by a conventional digital computer can be computed by a Turing machine. For computability purposes you may think of a Turing machine as an idealized Java program with as much memory as it needs.

3. Ordinary digital computers can simulate Turing machines for as long as their finite memory permits.

Turing machines give a *well-defined notion of computability*. This lets us ask precise questions such as: *Are there functions that are not computable?* And the answer is yes; the following problem is undecidable.

Definition 2.1 *The Halting Problem (HALT):* Given a program M (say in Java) and an input x, will $M(x)$ halt?

Benefits Turing machines give us a rigorous way to pose the question: *Is problem X decidable?*

Limits of this model Turing machines can run a very long time, so they are not good for modeling efficient computation. We will look at time-bounded Turing machines in the next section.

Physical realization As noted above, ordinary digital computers can simulate Turing-machine computations within their memory limits.

3 Polynomial Time

Definition 3.1

1. M is a polynomial-time Turing machine (henceforth P-Turing machine) if there exists a polynomial p such that $M(x)$ halts within $p(|x|)$ steps.
2. A set A is in P (polynomial time) if there exists a P-Turing machine M such that
   - If $x \in A$ then $M(x) = YES$.
   - If $x \notin A$ then $M(x) = NO$.

Note 3.2 The class P is defined using Turing machines, but it is robust across the standard reasonable models of digital computation since these models simulate one another within polynomial overhead. This statement does not extend to every alternative model discussed later in this article.

We now consider two problems that illustrate the importance of *polynomial time*.

1. SAT: Given a Boolean formula $\phi(x_1, \ldots, x_n)$, does there exist $b_1, \ldots, b_n \in \{T, F\}$ such that $\phi(b_1, \ldots, b_n) = T$? This can clearly be solved in roughly 2^n steps, by going through all elements of $\{T, F\}^n$. If one could give an algorithm for SAT that ran in time 2^{n-10} you would think it is still essentially brute force search with some tricks to speed it up a little. However, if someone had an algorithm in time n^{1000}, even though it would not be practical, it would *not* be a brute force search. Thus P is often used as a rough mathematical stand-in for *efficiently solvable without exhaustive search*, although polynomial time does not mean practical.
2. TSP (Travelling Salesperson Problem, decision version): Given a weighted graph on n vertices, and a bound K, is there a tour visiting every city exactly once, returning to the start, whose total cost is at most K? This can clearly be solved in roughly $n!$ steps, by a brute force search through all the possible tours. Similar to SAT: if TSP is in P there is an algorithm that is *not brute force*.

The following is true:

SAT is in P if and only if TSP is in P.

Both problems are called NP-complete. We won't go into the definition here, except to say that NP-complete problems are generally believed not to be in P. Thousands of problems are known to be NP-complete.

Benefits

1. The notions of P and NP-completeness allow us to discuss *efficiency* rigorously. This has two effects: (1) If a problem is NP-complete, one generally should not expect to solve every instance of the general problem exactly and efficiently. Instead, one can look at approximations or restricted subcases. (2) If a problem is NP-complete, then one may ask whether a genuinely different model of computation changes its complexity.

2. Once a problem is in P, we know that there is a polynomial-time algorithm for it. This algorithm need not be practical; however, it might be a starting point for a practical algorithm.

Limits

1. A problem that is in time (say) n^{10} may be impractical. As noted under *Benefits* one can often use the core ideas to get a truly efficient algorithm; however, this is not guaranteed.

2. The definition of P does not, by itself, tell us how to find efficient algorithms.

Physical realization As noted above, ordinary digital computers can simulate polynomial-time Turing-machine computations within memory limits.

4 Can Analog Computers Solve NP-Complete Problems?

Vergis, Steiglitz, and Dickinson [**?**] asked whether an analog computer could solve an NP-complete problem in polynomial time assuming P $\neq$ NP. They formulated what they call the *Strong Church's Thesis*: any finite analog computer can be simulated by a digital computer with only polynomial overhead in the resources used by the analog computer. If this thesis and P $\neq$ NP are both true, then an analog computer cannot solve an NP-complete problem using only polynomial resources.

An important issue is what counts as a resource. An analog device might appear to solve a problem quickly while requiring exponentially growing high-precision, energy, physical size, or some other resource. The authors illustrate this by designing an idealized mechanical device, using shafts, gears, and cams, for an optimization problem related to 3-SAT. They argue that if the following hold:

- the idealized device works as intended,
- P $\neq$ NP,
- their Strong Church's Thesis holds, and
- an additional physical assumption they call *the Downhill Principle*

then some physical resource required by the device must grow superpolynomial.

They do not prove the Strong Church's Thesis for all possible analog computers. However, they do prove it for an important class of analog systems described by sufficiently well-behaved ordinary differential equations.

5 Computing Over the Reals: The Pour-El & Richards Model

Pour-El & Richards [**?**] developed a framework in computable analysis for asking when real numbers, functions, and solutions of analytic problems are computable. A real number is accessed through arbitrarily accurate rational approximations rather than treated as an exact atomic object.

Benefits

1. If a problem is noncomputable in this framework, then no algorithm acting on these approximation-representations of reals can solve it. Such a result is a rigorous obstruction within computable analysis and may be viewed as evidence of a more general computability barrier. It is not, by itself, a theorem ruling out every conceivable model of physical computation.

Limits

1. The framework, as described here, has no bound on how much time a computation takes. Hence it cannot be used to model efficient algorithms.

2. The framework is not primarily a method for discovering algorithms; one of its principal uses is to determine whether mathematical objects and processes are computable from effective approximations.

Physical approximations

Computable-analysis algorithms can be carried out on ordinary digital computers by working with finite rational approximations of the required precision. A physical machine, however, can use only finite resources and finite precision at any one time.

6 Computing Over the Reals: The Blum-Shub-Smale Model

Blum, Shub, and Smale [**?**] (see also [**?**], [**?**]) defined a model of computation over the reals. In this model (1) a real fits in one memory cell, and (2) $+, -, \times$, division when defined, and comparisons take one step.

They developed analogs of P, NP, and NP-completeness over the reals which they denote P_{R}, NP_{R}, and NP_{R}-complete. They proved several problems about computation over the reals are NP_{R}-complete. We discuss the implications of this under *Benefits*.

Example 6.1 They showed that the problem of, given a polynomial $f : \mathsf{R}^n \to \mathsf{R}$ of degree 4, determining whether there is an $a_1, \ldots, a_n \in \mathsf{R}$ such that $f(a_1, \ldots, a_n) = 0$ is NP_{R}-complete. Thus, assuming $\mathrm{P}_{\mathsf{R}} \neq \mathrm{NP}_{\mathsf{R}}$, this problem does not have a polynomial-time BSS algorithm.

There are some problems that can be solved quickly in the BSS model. We list two.

1. Exact matrix multiplication over arbitrary real entries is polynomial-time in the BSS model.

2. Ordinary polynomial-time discrete computations can be simulated in the BSS model (with an appropriate encoding of bits).

Benefits This model makes the notion of computing over the reals rigorous. If one proves a problem is NP_{R}-complete, assuming $\mathrm{P}_{\mathsf{R}} \neq \mathrm{NP}_{\mathsf{R}}$, then no BSS machine solves it in polynomial time. Since BSS machines are granted exact real-number operations that are stronger in important respects than those available on ordinary finite-precision hardware, hardness in the BSS model can reasonably be viewed as evidence that the problem will remain hard in more realistic models. However, it is not a model-independent theorem that no conceivable machine can solve the problem efficiently.

Similarly, if a problem is proved undecidable in the BSS model, that may be viewed as evidence of a broader computability barrier, but it is not by itself a model-independent impossibility theorem.

Limits See the next item.
Physical realizations and approximations No physical device is known to realize the full idealized BSS model exactly, with arbitrary real numbers stored exactly and the model's primitive arithmetic operations performed at unit cost. We list obstacles to finding such a realization, and a comment on practical approximations.

1. Obstacle: Physically preparing and reading an arbitrary real number exactly appears problematic.

2. Obstacle: Treating the BSS model's primitive operations on reals as having unit cost is unrealistic.

3. Results and principles from physics, such as the holographic principle [**?**] and the Bekenstein bound [**?**], place or suggest finite upper bounds on the information content of a bounded physical system under appropriate energy or size assumptions. These results do not constitute a theorem that BSS machines are physically impossible, but they are suggestive obstacles to storing an arbitrary real number exactly in a finite physical device.

4. Practical approximation: Python and Java can support software packages that compute with very high, user-selected finite precision. This can approximate some BSS computations, but it does not supply exact arbitrary real numbers or unit-cost arithmetic.

7 The Siegelmann-Sontag Model: Computing Discrete Functions Using Reals

In 1991 Siegelmann and Sontag [**?**] (see also [**?**]) defined the *Analog Recurrent Neural Net (ARNN)* model of computation which is a recurrent neural network whose connections have numerical weights. We consider two versions of the model, according to which weights are allowed.

1. If all weights are rational, then the standard Siegelmann-Sontag networks have essentially the computational power of Turing machines; under the standard conventions the two models can simulate one another with polynomial overhead.

2. If arbitrary real weights are allowed, then the idealized model can compute more, including noncomputable functions. The extra power comes from the information that can be encoded in an arbitrary real weight.

Benefits: The model gives a rigorous way to study recurrent neural networks as computers, and recurrent networks are natural for sequential and streaming computation.

Limits: The arbitrary-real version requires infinite precision and is therefore not physically realistic. Even the rational-weight mathematical model assumes exact numerical states that physical hardware can only approximate.

Physical approximations Neuromorphic chips, memristor circuits, and other analog neural hardware can implement recurrent neural dynamics and therefore resemble the ARNN model. They should be regarded as finite-precision physical approximations or related hardware, not as exact implementations of the idealized arbitrary-real model.

8 Factoring With an Optical Computer

The following is the decision version of factoring: given (n, m) with $2 \leq m < n$, determine whether n has a nontrivial factor less than m. We do this because decision problems are the standard setting for P, NP, and NP-completeness.

In this section and the next, when we refer to *factoring* we mean the decision version.

Many important problems in NP are known either to be in P or to be NP-complete. The decision version of factoring is a prominent problem in NP that is not known to be in either category. There are reasons from complexity theory to think factoring is not NP-complete. However, as of 2026, no classical polynomial-time factoring algorithm is known.

We discuss both the asymptotic runtime and the real-world runtime of the Number Field Sieve (NFS), the fastest known classical general-purpose factoring method for large integers.

1. Let L be the bit-length of the number to be factored. The heuristic asymptotic running time of the NFS grows roughly like $\exp(cL^{1/3}(\log L)^{2/3})$ for an appropriate constant c. This is much better than exponential time in L, but still grows rapidly.

2. In 2020 RSA-250, an 829-bit semiprime, was factored using the NFS. The computation took roughly 2700 physical CPU core-years. This illustrates both how powerful NFS is and how rapidly the cost grows as the input gets larger.

Since factoring is an important problem in cryptography, and no classical polynomial-time algorithm is known, it is of interest to see if some alternative computing device can solve it.

In 1999 Shamir [**?**] proposed a way to do the most time-consuming part of NFS, the sieving, with an optical device that he named TWINKLE.

Benefits

1. Shamir projected in 1999 that, if TWINKLE could be built, it would increase the size of integers for which the sieving stage is feasible by roughly 100 to 200 bits.

2. Shamir's paper gives a fairly concrete proposed design for TWINKLE.

Limits

1. TWINKLE only speeds up the sieving part. Other parts of the factorization would still require substantial conventional computing resources.
2. A TWINKLE device is specialized for a particular parameter range (e.g., 512-bit RSA); substantially different parameters, and potentially different hardware, would be required for much larger inputs (e.g., 1024-bit RSA).
3. Modern cryptography is already transitioning towards post-quantum schemes designed to resist known quantum attacks. Prominent examples are based on lattices, hash functions, and error-correcting codes rather than integer factoring.

Proposed physical implementation In the end, TWINKLE was never built.

9 Factoring With a Massively Parallel Computer

In 2003, building on ideas from TWINKLE, Shamir and Tromer [**?**] proposed TWIRL. TWIRL uses massively parallel special-purpose hardware. Because TWIRL uses conventional electronics and VLSI technology, rather than TWINKLE's optical approach, it avoids some of TWINKLE's implementation difficulties.

Benefits

1. Using early-2000s VLSI assumptions, the designers estimated that the sieving required for a 1024-bit RSA modulus could be completed in less than one year with specialized hardware costing roughly $10 million. This was a design estimate.
2. The Shamir-Tromer paper gives a fairly concrete proposed design for TWIRL.

Limits Like TWINKLE, TWIRL accelerates the sieving stage rather than replacing the entire NFS computation, and it was a specialized proposed device.

Proposed physical implementation Unlike TWINKLE, it uses conventional electronics and VLSI technology. Even so, in the end, TWIRL was never built.

References

[1] J. D. Bekenstein. Universal upper bounds on the entropy-to-energy ratio for bounded systems. *Physical Review D*, 23(2):287–298, 1981.

[2] L. Blum. Computing over the reals: Where Turing meets Newton. *Notices of the American Mathematical Society*, 51(9):1024–1034, 2004.
`https://www.cs.cmu.edu/~lblum/PAPERS/TuringMeetsNewton.pdf`.

[3] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, 1997.

[4] L. Blum, M. Shub, and S. Smale. On a theory of computation over the real numbers; NP completeness, recursive functions and universal machines (extended abstract). In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 387–397. IEEE Computer Society, 1988. https://doi.org/10.1109/SFCS.1988.21955.

[5] R. Bousso. The holographic principle. *Reviews of Modern Physics*, 74(3):825–874, 2002.

[6] M. Pour-El and I. Richards. *Computability in analysis and physics*. Springer, New York, Heidelberg, Berlin, 1989.

[7] A. Shamir. Factoring large numbers with the TWINKLE device. In *Cryptographic Hardware and Embedded Systems*, pages 2–12. Springer, 1999. http://www.lia.deis.unibo.it/Courses/TecnologieSicurezzaAK/twinkle.pdf.

[8] A. Shamir and E. Tromer. Factoring large numbers with the TWIRL device. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2003. https://cs-people.bu.edu/tromer/papers/twirl.pdf.

[9] H. T. Siegelmann. *Neural networks and analog computation: Beyond the Turing limit*. Progress in theoretical computer science. Birkhäuser, 1999.

[10] H. T. Siegelmann and E. D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[11] A. Vergis, K. Steiglitz, and B. Dickinson. The complexity of analog computation. *Mathematics and Computers in Simulation*, 28(2):91–113, 1986. https://www.cs.princeton.edu/~ken/MCS86.pdf.