M

Mechanical Computing: The Computational Complexity of Physical Devices

John H. Reif
Department of Computer Science, Duke University, Durham, NC, USA

Glossary

Computable	Capable of being worked out by calculation, especially using a computer.
Mechanism	A machine or part of a machine that performs a particular task computation; the use of a computer for calculation.
Simulation	Used to denote both the modeling of a physical system by a computer and the modeling of the operation of a computer by a mechanical system; the difference will be clear from the context.

Definition of the Subject and Its Importance

Mechanical devices for computation appear to be largely displaced by the widespread use of microprocessor-based computers that are pervading almost all aspects of our lives. Nevertheless, mechanical devices for computation are of interest for at least three reasons:

(a) *Historical*: The use of mechanical devices for computation is of central importance in the historical study of technologies, with a history dating to thousands of years and with surprising applications even in relatively recent times.
(b) *Technical and practical*: The use of mechanical devices for computation persists and has not yet been completely displaced by widespread use of microprocessor-based computers. Mechanical computers have found applications in various emerging technologies at the microscale that combine mechanical functions with computational and control functions not feasible by purely electronic processing. Mechanical computers also have been demonstrated at the molecular scale and may also provide unique capabilities at that scale. The physical designs for these modern micro- and molecular-scale mechanical computers may be based on the prior designs of the large-scale mechanical computers constructed in the past.
(c) *Impact of physical assumptions on complexity of motion planning, design, and simulation*: The study of computation done by mechanical devices is also of central importance in providing lower bounds on the computational resources such as time and/or space required to simulate a mechanical system observing

R.A. Meyers (ed.), *Encyclopedia of Complexity and Systems Science*,
DOI 10.1007/978-3-642-27737-5_325-4

given physical laws. In particular, the problem of simulating the mechanical system can be shown to be *computationally hard* if a hard computational problem can be simulated by the mechanical system. A similar approach can be used to provide lower bounds on the computational resources required to solve various motion-planning tasks that arise in the field of robotics. Typically, a robotic motion-planning task is specified by a geometric description of the robot (or collection of robots) to be moved, its initial and final positions, the obstacles it is to avoid, as well as a model for the type of feasible motion and physical laws for the movement. The problem of planning such as robotic motion-planning task can be shown to be computationally hard if a hard computational problem can be simulated by the robotic motion-planning task.

Introduction to Computational Complexity

Abstract Computing Machine Models

To gage the computational power of a family of mechanical computers, we will use a widely known abstract computational model known as the Turing machine, defined in this section.

The Turing machine. The *Turing machine* model formulated by Alan Turing (1937) was the first complete mathematical model of an abstract computing machine that possessed universal computing power. The machine model has (i) a finite-state transition control for logical control of the machine processing, (ii) a tape with a sequence of storage cells containing symbolic values, and (iii) a tape scanner for reading and writing values to and from the tape cells, which could be made to move (left and right) along the tape cells.

A machine model is *abstract* if the description of the machine transition mechanism or memory mechanism does not provide specification of the mechanical apparatus used to implement them in practice. Since Turing's description did not include any specification of the mechanical mechanism for executing the finite-state transitions, it cannot be viewed as a concrete mechanical computing machine but instead as an abstract machine. Still it is a valuable computational model, due to its simplicity and very widespread use in computational theory.

A *universal* Turing machine simulates any other Turing machine; it takes its input a pair consisting of a string providing a symbolic description of a Turing machine M and the input string x and simulates M on input x. Because of its simplicity and elegance, the Turing machine has come to be the standard computing model used for most theoretical works in computer science. Informally, the Church-Turing hypothesis states that a Turing machine model can simulate a computation by any "reasonable" computational model (we will discuss some other reasonable computational models below).

Computational problems. A *computational problem* is given an input string specified by a string over a finite alphabet; determine the Boolean answer: 1 if the answer is *yes* and otherwise 0. For simplicity, we generally will restrict the input alphabet to be the binary alphabet {0,1}. The *input size* of a computational problem is the number of input symbols, which is the number of bits of the binary specification of the input. (Note: It is more common to make these definitions in terms of language acceptance. A *language* is a set of strings over a given finite alphabet of symbols. A computational problem can be identified with the language consisting of all strings over the input alphabet where the answer is 1. For simplicity, we defined each complexity class as the corresponding class of problems.)

Recursively computable problems and undecidable problems. There is a large class of problems, known as *recursively computable* problems, that Turing machines compute in finite computations, that is, always halting in finite time with the answer. There are certain problems that are not recursively computable; these are called *undecidable* problems. The *halting problem is* given a Turing machine description and an input, output 1 if the Turing machine ever halts and else output 0. Turing

proved the halting problem is undecidable. His proof used a method known as a diagonalization method; it considered an enumeration of all Turing machines and inputs and showed that a contradiction occurs when a universal Turing machine attempts to solve the halting problem for each Turing machine and each possible input.

Computational complexity classes. *Computational complexity* (see Lewis and Papadimitriou 1997) is the amount of computational resources required to solve a given computational problem. A *complexity class* is a family of problems, generally defined in terms of limitations on the resources of the computational model. The complexity classes of interest here will be associated with restrictions on the time (number of steps until the machine halts) and/or space (the number of tape cells used in the computation) of Turing machines. There are a number of notable complexity classes:

P is the complexity class associated with efficient computations and is formally defined to be the set of problems solved by Turing machine computations running in time polynomial in the input size (typically, this is the number of bits of the binary specification of the input).

NP is the complexity class associated with combinatorial optimization problems which if solved can be easily determined to have correct solutions and is formally defined to be the set of problems solved by Turing machine computations using nondeterministic choice running in polynomial time.

PSPACE is the complexity class that is defined to be the set of problems solved by Turing machines running in space polynomial in the input size.

EXPTIME is the complexity class that is defined to be the set of problems solved by Turing machine computations running in time exponential in the input size.

NP and **PSPACE** are widely considered to have instances that are not solvable in P, and it has been proved that **EXPTIME** has problems that are not in **P**.

Polynomial time reductions. A *polynomial time reduction* from a problem Q′ to a problem Q is a polynomial time Turing machine computation that transforms any instance of the problem Q′ into an instance of the problem Q which has an answer *yes* if and only if the problem Q′ has an answer *yes*. Informally, this implies that problem Q can be used to efficiently solve the problem Q′. A problem Q is *hard* for a family F of problems if, for every problem Q′ in F, there is a polynomial time reduction from Q′ to Q. Informally, this implies that problem Q can be used to efficiently solve *any* problem in F. A problem Q is *complete* for a family F of problems if Q is in C and also hard for F.

Hardness proofs for mechanical problems. We will later consider various mechanical problems and characterize their computation power:

Undecidable mechanical problems: typically this was proven by a computable reduction from the halting problem for universal Turing machine problems to an instance of the mechanical problem; this is equivalent to showing that the mechanical problem can be viewed as a computational machine that can simulate a universal Turing machine computation.

*Mechanical problems that are hard for **NP**, **PSPACE**, **or EXPTIME***: typically this was proven by a polynomial time reduction from the problems in the appropriate complexity class to an instance of the mechanical problem; again, this is equivalent to showing that the mechanical problem can be viewed as a computational machine that can simulate a Turing machine computation in the appropriate complexity class.

The simulation proofs in either case often provide insight into the intrinsic computational power of the mechanical problem or mechanical machine.

Other Abstract Computing Machine Models

There are a number of abstract computing models discussed in this chapter that are equivalent or nearly equivalent to conventional deterministic Turing machines:

Reversible Turing machines. A computing device is *(logically) reversible* if each transition of its computation can be executed both in forward direction and in reverse direction, without loss of information. Landauer (1961) showed that irreversible computations must generate heat in the computing process and that reversible computations have the property that, if executed slowly enough, can (in the limit) consume no energy in an adiabatic computation. A *reversible Turing machine* model allows the scan head to observe three consecutive tape symbols and to execute transitions both in forward and in reverse direction. Bennett (1973) showed that any computing machine (e.g., an abstract machine such as a Turing machine) can be transformed to do only reversible computations, which implied that reversible computing devices are capable of universal computation. Bennett's reversibility construction required extra space to store information to insure reversibility, but this extra space can be reduced by increasing the time. Vitanyi (Li and Vitanyi 1996) gives trade-offs between time and space in the resulting reversible machine. Lewis and Papadimitriou (Turing 1937) showed that reversible Turing machines are equivalent in computational power to conventional Turing machines when the computations are bounded by polynomial time, and Crescenzi and Papadimitriou (1995) proved a similar result when the computations are bounded by polynomial space. This implies that the definitions of the complexity classes **P** and **PSPACE** do not depend on the Turing machines being reversible or not. Reversible Turing machines are used in many of the computational complexity proofs to be mentioned involving simulations by mechanical computing machines.

Cellular automata. These are sets of finite-state machines that are typically connected together by a grid network. There are known efficient simulations of Turing machine by cellular automata (e.g., see Wolfram (1984) for some known universal simulations). A number of the particle-based mechanical machines to be described are known to simulate cellular automata.

Randomized Turing machines. The machine can make random choices in its computation. While the use of randomized choice can be very useful in many efficient algorithms, there is evidence that randomization only provides limited additional computational power above conventional deterministic Turing machines. (In particular, there are a variety of pseudorandom number generation methods proposed for producing long pseudorandom sequences from short truly random seeds, which are widely conjectured to be indistinguishable from truly random sequences by polynomial time Turning machines.) A number of the mechanical machines to be described using Brownian motion have natural sources of random numbers.

There are also a number of abstract computing machine models that appear to be more powerful than conventional deterministic Turing machines:

Real- valued Turing machines. In these machines, due to Blum et al. (1996), each storage cell or register can store any real value (which may be transcendental). Operations are extended to allow infinite precision arithmetic operations on real numbers. To our knowledge, none of the analog computers that we will describe in this chapter have this power.

Quantum computers. A *quantum superposition* is a linear superposition of basis states; it is defined by a vector of complex amplitudes whose absolute magnitudes sum to 1. In a quantum computer, the quantum superposition of basis states is transformed in each step by a unitary transformation (this is a linear mapping that is reversible and always preserves the value of the sum of the absolute magnitudes of its inputs). The outputs of a quantum computation are read by observations that project the quantum superposition to classical values; a given state is chosen with probability defined by the magnitude of the amplitude of that state

in the quantum superposition. Feynman (1982) and Benioff (1982) were the first to suggest the use of quantum mechanical principles for doing computation, and Deutsch (1985) was the first to formulate an abstract model for quantum computing and showed it was universal. Since then, there is a large body of work in quantum computing (see Gruska 1999; Nielsen and Chuang 2000) and quantum information theory (see Jaeger 2006; Reif 2009a). Some of the particle-based methods for mechanical computing described below make use of quantum phenomena but generally are not considered to have the full power of quantum computers.

The Computational Complexity of Motion Planning and Simulation of Mechanical Devices

Complexity of Motion Planning for Mechanical Devices with Articulated Joints

The first known computational complexity result involving mechanical motion or robotic motion planning was in 1979 by Reif (1979). He considers a class of mechanical systems consisting of a finite set of connected polygons with articulated joints, which are required to be moved between two configurations in three-dimensional space avoiding a finite set of fixed polygonal obstacles. To specify the movement problem (as well as the other movement problems described below unless otherwise stated), the object to be moved, as well as its initial and final positions, and the obstacles are all defined by linear inequalities with rational coefficients with a finite number of bits. He showed that this class of motion-planning problems is hard for PSPACE. Since it is widely conjectured that PSPACE contains problems that are not solvable in polynomial time, this result provided the first evidence that these robotic motion-planning problems are not solvable in time polynomial in n if the number of degrees of freedom grows with n. His proof involved simulating a reversible Turing machine with n tape cells by a mechanical device with n-articulated polygonal arms that had to be maneuvered through a set of fixed polygonal obstacles similar to the channels in Swiss cheese. These obstacles were devised to force the mechanical device to simulate transitions of the reversible Turing machine to be simulated, where the positions of the arms encoded the tape cell contents, and tape read/write operations were simulated by channels of the obstacles which forced the arms to be reconfigured appropriately. This class of movement problems can be solved by reduction to the problem of finding a path in an O (n)-dimensional space avoiding a fixed set of polynomial obstacle surfaces, which can be solved by a PSPACE algorithm due to Canny (1988). Hence, this class of movement problems is PSPACE complete. (In the case the object to be moved consists of only one rigid polygon, the problem is known as the piano mover's problem and has a polynomial time solution by Schwartz and Sharir (1983).)

Other PSPACE Completeness Results for Mechanical Devices

There were many subsequent PSPACE completeness results for mechanical devices (two of which we mention below), which generally involved *multiple degrees of freedom*:

The warehouseman' s problem. Schwartz and Sharir (Hopcroft et al. 1984) showed in 1984 that moving a set of n disconnected polygons in two dimensions from an initial position to a final position among finite set of fixed polygonal obstacles PSPACE hard.

There are two classes of mechanical dynamic systems, the ballistic machines and the Browning machines described below that can be shown to provide simulations of polynomial space Turing machine computations.

Ballistic Collision-Based Computing Machines and PSPACE

A *ballistic computer* (see Bennett 1982, 2003) is a conservative dynamical system that follows a

mechanical trajectory isomorphic to the desired computation. It has the following properties:

Trajectories of distinct ballistic computers cannot be merged.
All computational operations must be reversible.
Computations, when executed at constant velocity, require no consumption of energy.
Computations must be executed without error and need to be isolated from external noise and heat sources.

Collision-based computing (Adamatzky 2001) is computation by a set of particles, where each particle holds a finite state value, and state transformations are executed at the time of collisions between particles. Since collisions between distinct pairs of particles can be simultaneous, the model allows for parallel computation. In some cases the particles can be configured to execute cellular automata computations (Squier and Steiglitz 1994). Most proposed methods for *collision-based computing* are ballistic computers as defined above. Examples of concrete physical systems for collision-based computing are:

The billiard ball computers. Fredkin and Toffoli (1982) considered a mechanical computing model, the *billiard ball computer*, consisting spherical billiard balls with polygonal obstacles, where the billiard balls were assumed to have perfect elastic collisions with no friction. They showed in 1982 that a billiard ball computer, with an unbounded number of billiard balls, could simulate a reversible computing machine model that used reversible Boolean logical gates known as Toffoli gates. When restricted to finite set of n spherical billiard balls, their construction provides a simulation of a polynomial space reversible Turing machine.

Particle-like waves in excitable medium. Certain classes of excitable medium have discrete models that can exhibit particle-like waves that propagate through the media (Adamatzky 1996), and using this phenomenon, Adamatzky (1998a) gave a simulation of a universal Turing machine that, if restricted to n particle waves, provided a simulation of a polynomial space Turing machine.

Soliton computers. A soliton is a wave packet that maintains a self-reinforcing shape as it travels at constant speed through a nonlinear dispersive media. A soliton computer (Jakubowski et al. 1998, 2001) makes use of optical solitons to hold state, and state transformations are made by colliding solitons.

Brownian Machines and PSPACE

In a mechanical system exhibiting *fully Brownian motion*, the parts move freely and independently, up to the constraints that either link the parts together or force the parts to exert on each other. In a fully Brownian motion, the movement is entirely due to heat, and there is no other source of energy driving the movement of the system. An example of mechanical systems with fully Brownian motion is a set of particles exhibiting Browning motion with electrostatic interaction. The rate of movement of mechanical system with fully Brownian motion is determined entirely by the drift rate in the random walk of their configurations.

Other mechanical systems, known as *driven Brownian motion systems,* exhibit movement only partly due to heat; in addition, there is a source of energy driving the movement of the system. Examples of driven Brownian motion systems are:

Feynman's ratchet and pawl (Feynman 1963), which is a mechanical ratchet system that has a driving force but that can operate reversibly
Polymerase enzyme, which uses ATP as fuel to drive their average movement forward but also can operate reversibly

There is no energy consumed by fully Brownian motion devices, whereas driven Brownian motion devices require power that grows as a quadratic function of the drive rate in which operations are executed (see Bennett 2003).

Bennett (1982) provides two examples of Brownian computing machines:

An *enzymatic machine.* This is a hypothetical biochemical device that simulates a Turing machine, using polymers to store symbolic values in a manner similar to Turing machine tapes, and uses hypothetical enzymatic reactions to execute state transitions and read/write operations into the polymer memory. Shapiro (1999) also describes a mechanical Turing machine whose transitions are executed by hypothetical enzymatic reactions.

A *clockwork computer.* This is a mechanism with linked articulated joints, with a Swiss-cheese-like set of obstacles, which force the device to simulate a Turing machine. In the case where the mechanism of Bennett's clockwork computer is restricted to have a linear number of parts, it can be used to provide a simulation of PSPACE similar to that of Reif (1979).

Hardness Results for Mechanical Devices with a Constant Number of Degrees of Freedom

There were also additional computation complexity hardness results for mechanical devices, which only involved a *constant number of degrees of freedom*. These results exploited special properties of the mechanical systems to do the simulation:

Motion planning with moving obstacles. Reif and Sharir (1985) considered the problem of planning the motion of a rigid object (the robot) between two locations, avoiding a set of obstacles, some of which are rotating. They showed this problem is PSPACE hard. This result was perhaps surprising, since the number of degrees of freedom of movement of the object to be moved was constant. However, the simulation used the rotational movement of obstacles to force the robot to be moved only to a position that encoded all the tape cells of M. The simulation of a Turing machine M was made by forcing the object between such locations (which encoded the entire n tape cell contents of M) at particular times and further forced that object to move between these locations over time in a way that simulated state transitions of M.

NP Hardness Results for Path Problems in Two and three Dimensions

Shortest path problems in fixed dimensions involve only a constant number of degrees of freedom. Nevertheless, there are a number of NP hardness results for such problems. These results also led to proofs that certain physical simulations (in particular, simulation of multi-body molecular and celestial simulations) are NP hard and therefore not likely efficiently computable with high precision:

Finding shortest paths in three dimensions. Consider the problem of finding the shortest path of a point in three dimensions (where distance is measured in the Euclidean metric) avoiding fixed polyhedral obstacles whose coordinates are described by rational numbers with a finite number of bits. This shortest path problem can be solved in PSPACE (Canny 1988), but the precise complexity of the problem is an open problem. Canny and Reif (1987) were the first to provide a hardness complexity result for this problem; they showed the problem is NP hard. Their proof used novel techniques called *free path encoding* that used 2^n homotopy equivalence classes of shortest paths. Using these techniques, they constructed exponentially many shortest path classes (with distinct homotopy) in single-source multiple-destination problems involving O (n) polygonal obstacles. They used each of these paths to encode a possible configuration of the nondeterministic Turing machine with n binary storage cells. They also provided a technique for simulating each step of the Turing machine by the use of polygonal obstacles whose edges forced a permutation of these paths that encoded the modified configuration of the Turing machine. This encoding allowed

them to prove that the single-source single-destination problem in three dimensions is NP hard. Similar free path encoding techniques were used for a number of other complexity hardness results for mechanical simulations described below.

Kinodynamic planning. *Kinodynamic planning* is the task of motion planning while subject to simultaneous kinematic and dynamics constraints. The algorithms for various classed of kinodynamic planning problems were first developed in (Canny et al. 1988). Canny and Reif (1987) also used free path encoding techniques to show that two-dimensional kinodynamic motion planning with bounded velocity is NP hard.

Shortest curvature-constrained path planning in two dimensions. We now consider *curvature-constrained shortest path problems*, which involve finding the shortest path by a point among polygonal obstacles, where there is an upper bound on the path curvature. A class of curvature-constrained shortest path problems in two dimensions was shown to be NP hard by Reif and Wang (1998), by devising a set of obstacles that forced the shortest curvature-constrained path to simulate a given nondeterministic Turing machine.

PSPACE Hard Physical Simulation Problems

Ray tracing with a rational placement and geometry. Ray tracing is given an optical system, and the position and direction of an initial light ray determine if the light ray reaches some given final position. This problem of determining the path of light ray through an optical system was first formulated by Newton in his book on optics. Ray tracing has been used for designing and analyzing optical systems. It is also used extensively in computer graphics to render scenes with complex curved objects under global illumination. Reif et al. (1990) first showed in 1990 the problem of ray tracing in various three-dimensional optical systems, where the optical devices either consist of reflective objects defined by quadratic equations or refractive objects defined by linear equations, but in either case the coefficients are restricted to be rational. They showed these ray tracing problems are PSPACE hard. Their proof used free path encoding techniques for simulating a nondeterministic linear space Turing machine, where the position of the ray as it enters a reflective or refractive optical object (such as a mirror or prism face) encodes the entire memory of the Turing machine to be simulated, and further steps of the Turing machine are simulated by optically inducing appropriate modifications in the position of the ray as it enters other reflective or refractive optical objects. This result implies that the apparently simple task of highly precise ray tracing through complex optical systems is not likely to be efficiently executed by a polynomial time computer. These results for ray tracing are another example of the use of a physical system to do powerful computations. A number of subsequent papers showed the NP hardness (recall NP is a subset of PSPACE, so NP hardness is a weaker type of hardness result PSPACE hardness) of various optical ray problems, such as the problem of determining if a light ray can reach a given position within a given time duration (Haist and Osten 2007; Oltean and Muntean 2008, 2009; Oltean 2008; Muntean and Oltean 2009), optical masks (Dolev and Fitoussi 2010), and ray tracing with multiple optical frequencies (Goliaei and Jalili 2009, 2012) (see Woods and Naughton 2009 for a survey of these and related results in optical computing). A further PSPACE hardness result for an optics problem is given in a recent paper (Goliaei and Foroughmand-Araabi 2013) concerning ray tracing with multiple optical frequencies, with an additional concentration operation.

Molecular and gravitational mechanical systems. The work of Tate and Reif (1993) on the complexity of n-body simulation provides an interesting example of the use of natural physical systems to do computation. They showed that the problem of n-body simulation is PSPACE hard and therefore not likely efficiently computable with high precision. In particular, they considered multi-body systems in three dimensions with n particles and inverse polynomial force laws between each pair of particles (e.g., molecular systems with Columbic force laws or celestial simulations with gravitational force laws). It is

quite surprising that such systems can be configured to do computation. Their hardness proof made use of free path encoding techniques similar to their proof (Reif et al. 1990) of the PSPACE-hardness of ray tracing. A single particle, which we will call the *memory-encoding particle*, is distinguished. The position of a memory-encoding particle as it crosses a plane encodes the entire memory of the given Turing machine to be simulated, and further steps of the Turing machine are simulated by inducing modifications in the trajectory of the memory-encoding particle. The modifications in the trajectory of the memory-encoding particle are made by use of other particles that have trajectories that induce force fields that essentially act like force mirrors, causing reflection-like changes in the trajectory of the memory-encoding particle. Hence, highly precise n-body molecular simulation is not likely to be efficiently executed by a polynomial time computer.

A Provably Intractable Mechanical Simulation Problem: Compliant Motion Planning with Uncertainty in Control

Next, we consider compliant motion planning with uncertainty in control. Specifically, we consider a point in three dimensions which is commanded to move in a straight line but whose actual motion may differ from the commanded motion, possibly involving sliding against obstacles. Given that the point initially lies in some start region, the problem is to find a sequence of commanded velocities that is guaranteed to move the point to the goal. This problem was shown by Canny and Reif (1987) to be nondeterministic EXPTIME hard, making it the first provably intractable problem in robotics. Their proof used free path encoding techniques that exploited the uncertainty of position to encode exponential number of memory bits in a Turing machine simulation.

Undecidable Mechanical Simulation Problems

Motion planning with friction. Consider a class of mechanical systems whose parts consist of a finite number of rigid objects defined by linear or quadratic surface patches connected by frictional contact linkages between the surfaces. (Note: This class of mechanisms is similar to the analytical engine developed by Babbage as described in the next sections, except that there are smooth frictional surfaces rather than toothed gears.) Reif and Sun (1998) proved that an arbitrary Turing machine could be simulated by a (universal) frictional mechanical system in this class consisting of a finite number of parts. The entire memory of a universal Turing machine was encoded in the rotational position of a rod. In each step, the mechanism used a construct similar to Babbage's machine to execute a state transition. The key idea in their construction is to utilize frictional clamping to allow for setting arbitrary high gear transmission. This allowed the mechanism to execute state transitions for arbitrary number of steps. Simulation of a universal Turing machine implied that the movement problem is undecidable when there are frictional linkages. (A problem is *undecidable* if there is no Turing machine that solves the problem for all inputs in finite time.) It also implied that a mechanical computer could be constructed with only a constant number of parts that has the power of an unconstrained Turing machine.

Ray tracing with nonrational positioning. Consider again the problem of ray tracing in a three-dimensional optical systems, where the optical devices again may be either consist of reflective objects defined by quadratic equations or refractive objects defined by linear equations. Reif et al. (1990) also proved that in the case where the coefficients of the defining equations are not restricted to be rational and include at least one irrational coefficient, then the resulting ray tracing problem could simulate a universal Turing machine, and so is undecidable. This ray tracing problem for reflective objects is equivalent to the problem of tracing the trajectory of a single particle bouncing between quadratic surfaces, which is also undecidable by this same result of Reif et al. (1990). An independent result of Moore (1990) also showed the undecidability of the problem of tracing the trajectory of a single particle bouncing between quadratic surfaces.

Dynamics and nonlinear mappings. Moore (Moore and Shifts 1991), Ditto (Sinha and Ditto

1999), and Munakata et al. (2002) have also given universal Turing machine simulations of various dynamical systems with nonlinear mappings.

Concrete Mechanical Computing Devices

Mechanical computers have a very extensive history; some surveys are given by Knott (1915), Hartree (1950), Engineering Research Associates (1950), Chase (1980), Martin (1992), and Davis (2000). Norman (2002) gave an overview of the literature of mechanical calculators and other historical computers, summarizing the contributions of notable manuscripts and publications on this topic.

Mechanical Devices for Storage and Sums of Numbers

Mechanical methods, such as notches on stones and bones and knots and piles of pebbles, have been used since the Neolithic period for storing and summing integer values. One example of such a device, the *abacus*, which may have been developed and invented in Babylonia approximately 5000 years ago, makes use of beads sliding on cylindrical rods to facilitate addition and subtraction calculations.

Analog Mechanical Computing Devices

Computing devices will be considered here to be *analog* (as opposed to digital) if they do not provide a method for restoring calculated values to discrete values, whereas digital devices provide restoration of calculated values to discrete values. (Note that both analog and digital computers use some kind of physical quantity to represent values that are stored and computed, so the use of physical encoding of computational values is not necessarily the distinguishing characteristic of analog computing.) Descriptions of early analog computers are given by Horsburgh (1914), Turck (1921), Svoboda (1948), Hartree (1950), Engineering Research Associates (1950), and Soroka (1954). There are a wide variety of mechanical devices used for analog computing:

Mechanical devices for astronomical and celestial calculation. While we have no sufficient space in this article to fully discuss this rich history, we note that various mechanisms for predicting lunar and solar eclipses using optical illumination of configurations of stones and monoliths (e.g., Stonehenge) appear to date to the Neolithic period. The Hellenistic civilization in the classical period of ancient history seems to develop a number of analog calculating devices for astronomy calculations. A *planisphere*, which appears to have been used in the *Tetrabiblos* by Ptolemy in the second century, is a simple analog calculator for determining for any given date and time the visible portion of a star chart and consists of two disks rotating on the same pivot. *Astrolabes* are a family of analog calculators used for solving problems in spherical astronomy, often consisting of a combination of a planisphere and a dioptra (a sighting tube). An early form of an astrolabe is attributed to Hipparchus in the mid-second century. Other more complex mechanical mechanisms for predicting lunar and solar eclipses seem to have been developed in Hellenistic period. The most impressive and sophisticated known example of an ancient gear-based mechanical device from the Hellenistic period is the *Antikythera mechanism*, and recent research (Freeth et al. 2006) provides evidence it may have used to predict celestial events such as lunar and solar eclipses by the analog calculation of arithmetic-progression cycles. Like many other intellectual heritages, some elements of the design of such sophisticated gear-based mechanical devices may have been preserved by the Arabs at the end of that Hellenistic period and then transmitted to the Europeans in the middle ages.

Planimeters. There is a considerable history of mechanical devices that integrate curves. A *planimeter* is a mechanical device that integrates the area of the region enclosed by a two-

dimensional closed curve, where the curve is presented as a function of the angle from some fixed interior point within the region. One of the first known planimeters was developed by J.A. Hermann in 1814 and improved (as the polar planimeter) by J.A. Hermann in 1856. This led to a wide variety of mechanical integrators known as wheel-and-disk integrators, whose input is the angular rotation of a rotating disk and whose output, provided by a tracking wheel, is the integral of a given function of that angle of rotation. More general mechanical integrators known as ball-and-disk integrators, whose input provided 2 degrees of freedom (the phase and amplitude of a complex function), were developed by James Thomson in 1886. There are also devices, such as the intergraph of Abdank Abakanowicz (1878) and C.V. Boys (1882), which integrate a one-variable real function of x presented as a curve $y = f(x)$ on the Cartesian plane. Mechanical integrators were later widely used in WWI and WWII military analog computers for solution of ballistics equations, artillery calculations, and target tracking. Various other integrators are described in Morin (1913).

Harmonic analyzers. A *harmonic analyzer* is a mechanical device that calculates the coefficients of the Fourier transform of a complex function of time such as a sound wave. Early harmonic analyzers were developed by Thomson (1878) and Henrici (1894) using multiple pulleys and spheres, known as ball-and-disk integrators.

Harmonic synthesizers. A *harmonic synthesizer* is a mechanical device that interpolates a function given the Fourier coefficients. Thomson (then known as Lord Kelvin) in 1886 developed (Kelvin 1878) the first known harmonic analyzer that used an array of James Thomson's (his brother) ball-and-disk integrators. Kelvin's harmonic synthesizer made use of these Fourier coefficients to reverse this process and interpolate function values, by using a wire wrapped over the wheels of the array to form a weighted sum of their angular rotations. Kelvin demonstrated that the use of these analog devices is to predict the tide heights of a port: first, his harmonic analyzer calculated the amplitude and phase of the Fourier harmonics of solar and lunar tidal movements, and then his harmonic synthesizer formed their weighted sum, to predict the tide heights over time. Many other harmonic analyzers were later developed, including one by Michelson and Stratton (1898) that performed Fourier analysis, using an array of springs. Miller (1916) gives a survey of these early harmonic analyzers. Fisher (1911) made improvements to the tide predictor, and later Doodson and Légé increase the scale of this design to a 42-wheel version that was used up to the early 1960s.

Analog equation solvers. There are various mechanical devices for calculating the solution of sets of equations. Kelvin also developed one of the first known mechanical mechanisms for equation solving, involving the motion of pulleys and tilting plate that solved sets of simultaneous linear equations specified by the physical parameters of the ropes and plates. John Wilbur in the 1930s increased the scale of Kelvin's design to solve nine simultaneous linear algebraic equations. Leonardo Torres Quevedo constructed various rotational mechanical devices, for determining real and complex roots of a polynomial. Svoboda (1948) describes the state of art in the 1940s of mechanical analog computing devices using linkages.

Differential analyzers. A *differential analyzer* is a mechanical analog device using linkages for solving ordinary differential equations. Vannevar Bush (1931) developed in 1931 the first differential analyzer at MIT that used a torque amplifier to link multiple mechanical integrators. Although it was considered a general-purpose mechanical analog computer, this device required a physical reconfiguration of the mechanical connections to specify a given mechanical problem to be solved. In subsequent differential analyzers, the reconfiguration of the mechanical connections was made automatic by resetting electronic relay connections. In addition to the military applications already mentioned above, analog

mechanical computers incorporating differential analyzers have been widely used for flight simulations and for industrial control systems.

Mechanical simulations of Physical Processes: **Crystallization and Packing**. There are a variety of macroscopic devices used for simulations of physical processes, which can be viewed as analog devices. For example, a number of approaches have been used for mechanical simulations of crystallization and packing:

Simulation using solid macroscopic ellipsoid bodies. Simulations of kinetic crystallization processes have been made by collections of macroscopic solid ellipsoidal objects – typically of diameter of a few millimeters – which model the molecules comprising the crystal. In these physical simulations, thermal energy is modeled by introducing vibrations; low level of vibration is used to model freezing and increasing the level of vibrations models melting. In simple cases, the molecule of interest is a sphere, and ball bearings or similar objects are used for the molecular simulation. For example, to simulate the dense random packing of hard spheres within a crystalline solid, Bernal (1964) and Finney (1970) used up to 4000 ball bearings on a vibrating table. In addition, to model more general ellipsoidal molecules, orzo pasta grains as well as M&M candies (Jerry Gollub at Princeton University) have been used. Also, Cheerios have been used to simulate the liquid state packing of benzene molecules. To model more complex systems, mixtures of balls of different sizes and/or composition have been used; for example, a model ionic crystal formation has been made by using a mixture of balls composed of different materials that acquired opposing electrostatic charges.

Simulations using bubble rafts (Bragg and Nye 1947; Bragg and Lomer 1948). These are the structures that assemble among equal-sized bubbles floating on water. They typically form two-dimensional hexagonal arrays and can be used for modeling the formation of close-packed crystals. Defects and dislocations can also be modeled (Corcoran et al. 1997); for example, by deliberately introducing defects in the bubble rats, they have been used to simulate crystal dislocations, vacancies, and grain boundaries. Also, impurities in crystals (both interstitial and substitutional) have been simulated by introducing bubbles of other sizes.

Reaction- Diffusion Chemical Computers. Adamatzky (Adamatzky et al. 2005; Adamatzky 1998b) described a class of analog computers where there is a chemical medium which has multiple chemical species, where the concentrations of these chemical species vary spatially and which diffuse and react in parallel. The memory values (as well as inputs and outputs) of the computer are encoded by the concentrations of these chemical species at a number of distinct locations (also known as micro-volumes). The computational operations are executed by chemical reactions whose reagents are these chemical species. Example computations (Adamatzky et al. 2005; Adamatzky 1998b) include (i) Voronoi diagram, which determines the boundaries of the regions closest to a set of points on the plane, (ii) skeleton of planar shape, and (iii) a wide variety of two-dimensional patterns periodic and aperiodic in time and space.

Digital Mechanical Devices for Arithmetic Operations

Recall that we have distinguished digital mechanical devices from the analog mechanical devices described above by their use of mechanical mechanisms for insuring the values stored and computed are discrete. Such discretization mechanisms include geometry and structure (e.g., the notches of Napier's bones described below) or cogs and spokes of wheeled calculators. Surveys of the history of some of these digital mechanical calculators are given by Knott (1915), Turck (1921), Hartree (1950), Engineering Research Associates (1950), Chase (1980), Martin (1992), Davis (2000), and Norman (2002):

Leonardo da Vinci's mechanical device and mechanical counting devices. This intriguing device, which involved a sequence of interacting wheels positioned on a rod, which appear to provide a mechanism for digital carry operations, was illustrated in 1493 in Leonardo da Vinci's Codex Madrid (1493). A working model of its possible mechanics was constructed in 1968 by Joseph Mirabella. Its function and purpose is not decisively known, but it may have been intended for counting rotations (e.g., for measuring the distance traversed by a cart). There are a variety of apparently similar mechanical devices used to measuring distances traversed by vehicles.

Napier's bones. John Napier (1614) developed in 1614 a mechanical device known as Napier's bones that allowed multiplication and division (as well as square and cube roots) to be done by addition and multiplication operations. It consists of rectilinear rods, which provided a mechanical transformation to and from logarithmic values. Wilhelm Schickard developed in 1623 a six-digit mechanical calculator that combined the use of Napier's bones using columns of sliding rods, with the use of wheels used to sum up the partial products for multiplication.

Slide rules. Edmund Gunter devised in 1620 a method for calculation that used a single log scale with dividers along a linear scale; this anticipated key elements of the first slide rule described by William Oughtred (1632) in 1632. A very large variety of slide machines were later constructed.

Pascaline: **Pascal's wheeled calculator**. Blaise Pascal (1645) developed in 1642 a calculator known as the Pascaline that could calculate all four arithmetic operations (addition, subtraction, multiplication, and division) on up to eight digits. A wide variety of mechanical devices were then developed that used revolving drums or wheels (cogwheels or pinwheels) to do various arithmetical calculations.

Stepped drum calculators. Gottfried Wilhelm von Leibniz developed in 1671 an improved calculator known as the stepped reckoner, which used a cylinder known as a *stepped drum* with nine teeth of different lengths that increase in equal amounts around the drum. The stepped drum mechanism allowed the use of moving slide for specifying a number to be inputted to the machine and made use of the revolving drums to do the arithmetic calculations. Charles Xavier Thomas de Colbrar developed in 1820 a widely used arithmetic mechanical calculator based on the stepped drum known as the arithmometer. Other stepped drum calculating devices included Otto Shweiger's millionaire calculator (1893) and Curt Herzstark's curta (early 1940s).

Pinwheel calculators. Another class of calculators, independently invented by Frank S. Baldwin and W. T. Odhner in the 1870s, is known as pinwheel calculators; they used a pinwheel for specifying a number input to the machine and use revolving wheels to do the arithmetic calculations. Pinwheel calculators were widely used up to the 1950s, for example, in William S. Burroughs's calculator/printer and the German Brunsviga.

Digital Mechanical Devices for Mathematical Tables and Functions

Babbage's difference engine. Charles Babbage (1822, 1825) in 1820 invented a mechanical device known as the *difference engine* for calculation of tables of an analytical function (such as the logarithm) that summed the change in values of the function when a small difference is made in the argument. That difference calculation required for each table entry involved a small number of simple arithmetic computations. The device made use of columns of cogwheels to store digits of numerical values. Babbage planned to store 1000 variables, each with 50 digits, where each digit was stored by a unique cogwheel. It used cogwheels in registers for the required arithmetical calculations and also made use of a rod-based control mechanism specialized for control of these arithmetic calculations. The design and operation of the mechanisms of the device were described by a symbolic scheme developed by Babbage

(1826). He also conceived of a printing mechanism for the device. In 1801, Joseph-Marie Jacquard invented an automatic loom that made use of punched cards for the specification of fabric patterns woven by his loom, and Charles Babbage proposed the use of similar punched cards for providing inputs to his machines. He demonstrated over a number of years certain key portions of the mechanics of the device but never completed a complete function device.

Other difference engines. In 1832 Ludgate (1909) independently designed, but did not construct, a mechanical computing machine similar but smaller in scale to Babbage's analytical engine. In 1853 Pehr and Edvard Scheutz (Lindgren 1990) constructed in Sweden a cogwheel mechanical calculating device (similar to the difference engine originally conceived by Babbage) known as the tabulating machine for computing and printing out tables of mathematical functions. This (and a later construction of Babbage's difference engine by Doron Swade (1991) of the London Science Museum) demonstrated the feasibility of Babbage's difference engine.

Babbage's analytical engine. Babbage further conceived (but did not attempt to construct) a mechanical computer known as the analytical engine to solve more general mathematical problems. Lovelace's extended description of Babbage's analytical engine (Lovelace 1843) (translation of "Sketch of the Analytical Engine" by L. F. Menabrea) describes, in addition to arithmetic operations, also mechanisms for looping and memory addressing. However, the existing descriptions of Babbage's analytical engine lack the ability to execute a full repertory of logical and/or finite state transition operations required for general computation. Babbage's background was very strong in analytic mathematics, but he (and the architects of similar cogwheel-based mechanical computing devices at that date) seemed to have lacked knowledge of sequential logic and its Boolean logical basis, required for controlling the sequence of complex computations. This (and his propensity for changing designs prior to the completion of the machine construction) might have been the real reason for the lack of complete development of a universal mechanical digital computing device in the early 1800s.

Subsequent electromechanical digital computing devices with cogwheels. Other electromechanical computing digital devices (see Engineering Research Associates Staff 1950) developed in the late 1940s and 1950s that contain cogwheels included Howard Aiken's Mark 1 (Cohen and Welch 1999) constructed at Harvard University and Konrad Zuse's Z series computer constructed in Germany.

Mechanical Devices for Timing, Sequencing, and Logical Control

We will use the term *mechanical automata* here to denote mechanical devices that exhibit autonomous control of their movements. These can require sophisticated mechanical mechanisms for timing, sequencing, and logical control:

Mechanisms used for timing control. Mechanical clocks and other mechanical devices for measuring time have a very long history and include a very wide variety of designs, including the flow of liquids (e.g., water clocks) or sands (e.g., sand clocks), and more conventional pendulum-and-gear-based clock mechanisms. A wide variety of mechanical automata and other control devices make use of mechanical timing mechanisms to control the order and duration of events automatically executed (e.g., mechanical slot machines dating up to the 1970s made use of such mechanical clock mechanisms to control the sequence of operations used for payout of winnings). As a consequence, there is an interwoven history in the development of mechanical devices for measuring time and the development of devices for the control of mechanical automata.

Logical control of computations. A critical step in the history of computing machines was the development in the middle 1800s of Boolean logic by George Boole (1847, 1854). Boole's innovation was to assign values to logical propositions: 1 for true propositions and 0 for

false propositions. He introduced the use of Boolean variables which are assigned to these values, as well as the use of Boolean connectives (and and or), for expressing symbolic Boolean logic formulas. Boole's symbolic logic is the basis for the logical control used in modern computers. Shannon (1938) was the first to make use of Boole's symbolic logic to analyze relay circuits (these relays were used to control an analog computer, namely, MIts Differential Equalizer).

The Jevons' logic piano: a **mechanical logical inference machine**. In 1870 William Stanley Jevons (who also significantly contributed to the development of symbolic logic) constructed a mechanical device (Jevons 1870, 1873) for the inference of logical proposition that used a piano keyboard for inputs. This *mechanical inference machine* is less widely known than it should be, since it may have had impact in the subsequent development of logical control mechanisms for machines.

Mechanical logical devices used to play games. Mechanical computing devices have also been constructed for executing the logical operations for playing games. For example, in 1975, a group of MIT undergraduates including Danny Hillis and Brian Silverman constructed a computing machine made of Tinkertoys that plays a perfect game of tic-tac-toe.

Mechanical Devices Used in Cryptography

Mechanical cipher devices using cogwheels. Mechanical computing devices that used cogwheels were also developed for a wide variety of other purposes beyond merely arithmetic. A wide variety of mechanical computing devices were developed for the encryption and decryption of secret messages. Some of these (most notably the family of German electromechanical cipher devices known as *Enigma Machines* (Hamer et al. 1998) developed in the early 1920s for commercial use and refined in the late 1920s and 1930s for military use) made use of sets of cogwheels to permute the symbols of text message streams. Similar (but somewhat more advanced) electromechanical cipher devices were used by the USSR up to the 1970s.

Electromechanical computing devices used in breaking ciphers. In 1934 Marian Rejewski and a team including Alan Turing constructed an electrical/mechanical computing device known as the bomb, which had an architecture similar to the abstract Turing machine described below and which was used to decrypt ciphers made by the German Enigma cipher device mentioned above.

Mechanical and Electrooptical Devices for Integer Factorization

Lehmer's number sieve computer. In 1926 Derrick Lehmer (1928) constructed a mechanical device called the number sieve computer for various mathematical problems in number theory including factorization of small integers and solution of Diophantine equations. The device made use of multiple bicycle chains that rotated at distinct periods to discover solutions (such as integer factors) to these number theoretic problems.

Shamir's TWINKLE. Adi Shamir (n.d., 1999; Lenstra and Shamir 2000) proposed a design for an optical/electric device known as TWINKLE for factoring integers, with the goal of breaking the RSA public-key cryptosystem. This was unique among mechanical computing devices in that it used time durations between optical pulses to encode possible solution values. In particular, LEDs were made to flash at certain intervals of time (where each LED is assigned a distinct period and delay) at a very high clock rate so as to execute a sieve-based integer factoring algorithm.

Mechanical Computation at the Microscale: MEMS Computing Devices. Mechanical computers can have advantages over electronic computation at certain scales; they are already having widespread use at the microscale. Microelectromechanical systems (MEMSs) are manufactured by lithographic etching methods similar in nature to the processes microelectronics are manufactured and have a similar microscale.

A wide variety of MEMS devices (Madou 2002) have been constructed for sensors and actuators, including accelerometers used in automobile safety devices and disk readers, and many of these MEMS devices execute mechanical computation to do their task. Perhaps the MEMS device most similar in architecture to the mechanical calculators described above is the recodable locking device (Plummer et al. 1999) constructed in 1998 at Sandia Labs, which made use of microscopic gears that acted as a mechanical lock and which was intended for mechanically locking strategic weapons.

Future Directions

Mechanical Self-Assembly Processes

Most of the mechanical devices discussed in this chapter have been assumed to be constructed top-down; that is, they are designed and then assembled by other mechanisms generally of large scale. However, a future direction to consider is bottom-up processes for assembly and control of devices. Self-assembly is a basic bottom-up process found in many natural processes and in particular in all living systems:

Domino tiling problems. The theoretical basis for self-assembly has its roots in domino tiling problems (also known as Wang tilings) as defined by Wang (1963) (also see the comprehensive text of Grunbaum et al. (1987)). The input is a finite set of unit size square tiles, each of whose sides is labeled with symbols over a finite alphabet. Additional restrictions may include the initial placement of a subset of these tiles and the dimensions of the region where tiles must be placed. Assuming an arbitrarily large supply of each tile, the problem is to place the tiles, without rotation (a criterion that cannot apply to physical tiles), to completely fill the given region so that each pair of abutting tiles has identical symbols on their contacting sides.

Turing-universal and NP complete self-assemblies. Domino tiling problems over an infinite domain with only a constant number of tiles were first proved by Berger (1966) to be undecidable. Lewis and Papadimitriou (1981) showed the problem of tiling a given finite region is NP complete.

Theoretical models of tiling self-Assembly processes. Domino tiling problems do not presume or require a specific process for tiling. Winfree (Winfree et al. 1996) proposed kinetic models for self-assembly processes. The sides of the tiles are assumed to have some methodology for selective affinity, which we call pads. Pads function as programmable binding domains, which hold together the tiles. Each pair of pads has specified binding strengths (a real number on the range [0,1] where 0 denotes no binding and 1 denotes perfect binding). The self-assembly process is initiated by a singleton tile (the seed tile) and proceeds by tiles binding together at their pads to form aggregates known as tiling assemblies. The preferential matching of tile pads facilitates the further assembly into tiling assemblies.

Pad binding mechanisms. These provide a mechanism for the preferential matching of tile sides can be provided by various methods:

Magnetic attraction, e.g., pads with magnetic orientations (these can be constructed by curing ferrite materials (e.g., PDMS polymer/ferrite composites) in the presence of strong magnet fields) and also pads with patterned strips of magnetic orientations

Capillary force, using hydrophobic/hydrophilic (capillary) effects at surface boundaries that generate lateral forces

Shape matching (also known as *shape complementarity* or *conformational affinity*), using the shape of the tile sides to hold them together

Also see the sections below *discussion of the used of molecular affinity* for pad binding.

Materials for tiles. There are a variety of distinct materials for tiles, at a variety of scales: Whitesides (see Xia and Whitesides (1998) and http://www-chem.harvard.edu/GeorgeWhitesides.html) has developed and tested multiple technologies for mesoscale self-assembly, using capillary forces, shape complementarity, and magnetic forces. Rothemund (Rothemund 2000) experimentally demonstrated some of the most complex known mesoscale tiling

self-assemblies using polymer tiles on fluid boundaries with pads that use hydrophobic/hydrophilic forces. A material science group at the University of Wisconsin (http://mrsec.wisc.edu/edetc/selfassembly) has also tested mesoscale self-assembly using magnetic tiles.

Mesoscale tile assemblies. Mesoscale tiling assemblies have tiles of size a few millimeters up to a few centimeters. They have been experimentally demonstrated by a number of methods, such as placement of tiles on a liquid surface interface (e.g., at the interface between two liquids of distinct density or on the surface of an air/liquid interface) and use of mechanical agitation with shakers to provide a heat source for the assembly kinetics (i.e., a temperature setting is made by fixing the rate and intensity of shaker agitation).

Applications of mesoscale assemblies. There are a number of applications, including:

Simulation of the thermodynamics and kinetics of molecular-scale self-assemblies

For placement of a variety of microelectronics and MEMS parts

Computation at the Molecular Scale: **DNA Computing Devices**. Due to the difficulty of constructing electrical circuits at the molecular scale, alternative methods for computation, and in particular mechanical methods, may provide unique opportunities for computing at the molecular scale. In particular the bottom-up self-assembly processes described above have unique applications at the molecular scale:

Self-assembled DNA nanostructures. Molecular-scale structures known as *DNA nanostructures* (see surveys by Seeman 2004; Reif et al. (Reif and LaBean 2009)) can be made to self-assemble from individual synthetic strands of DNA. When added to a test tube with the appropriate buffer solution and the test tube is cooled, the strands self-assemble into DNA nanostructures. This self-assembly of DNA nanostructures can be viewed as a mechanical process and in fact can be used to do computation. The first known example of a computation by using DNA was by Adleman (1994, 1998) in 1994; he used the self-assembly of DNA strands to solve a small instance of a combinatorial optimization problem known as the Hamiltonian path problem.

DNA tiling assemblies. The Wang tiling (1963) paradigm for self-assembly was the basis for scalable and programmable approach proposed by Winfree et al. (1998) for doing molecular computation using DNA. First, a number of distinct DNA nanostructures known as DNA tiles are self-assembled. End portions of the tiles, known as pads, are designed to allow the tiles to bind together a programmable manner similar to Wang tiling but in this case use the molecular affinity for pad binding due to hydrogen bonding of complementary DNA bases. This programmable control of the binding together of DNA tiles provides a capability for doing computation at the molecular scale. When the temperature of the test tube containing these tiles is further lowered, the DNA tiles bind together to form complex patterned tiling lattices that correspond to computations.

Assembling patterned DNA tiling assemblies. Programmed patterning at the molecular scale can be produced by the use of strands of DNA that encode the patterns; this was first done by Yan et al. (2003a) in the form of bar-cord-striped patterns, and more recently Rothemund (2006) self-assembled complex 2D molecular patterns and shapes using a technique known as DNA origami. Another method for molecular patterning of DNA tiles is via computation done during the assembly, as described below.

Computational DNA tiling assemblies. The first experimental demonstration of computation via the self-assembly of DNA tiles was in 2000 by Mao et al. (2000, Yan et al. 2003b), which provided a one-dimensional computation of a binary-carry computation (known as prefix sum) associated with binary adders. Rothemund et al. (2004) in 2004 demonstrated a two-dimensional computational assemblies of tiles displaying a pattern known as the Sierpinski triangle, which is the modulo 2 version of Pascal's triangle.

Other autonomous DNA devices. DNA nanostructures can also be made to make sequences of movement, and a demonstration of an autonomous moving DNA robotic device that moved without outside mediation across DNA nanostructures was given by Yin et al. (2004). The design of an autonomous DNA device that moves under programmed control is described in (Reif and Sahu 2008). Surveys of DNA autonomous devices are given in Reif and LaBean (2007), Chandran et al. (2013), and Bath and Turberfield (2007).

Analog Computation by Chemical Reactions

Chemical reaction systems have a set of reactants, whose concentrations evolve by a set of differential equations determined by chemical reactions. Magnasco (1997) showed that a chemical reaction can be designed to simulate any given digital circuit. Soloveichik et al. (2008) showed that a chemical reaction can be designed to simulate a universal Turing machine, and Soloveichik et al. (2010) showed that this can be done by a class of chemical reactions involving only DNA hybridization reactions. Senum and Riedel (2011) gave detailed design rules for chemical reactions that executed various analog computational operations such as addition, multipliers, and logarithm calculations.

Analog Computation by Bacterial Genetic Circuits

Sarpeshkar et al. have developed analog transistor models for the concentrations of various reactants within bacterial genetic circuits (Danial et al. 2011) and then used these models to experimentally demonstrate various computations, such as square root calculation, within living bacteria cells (Daniel et al. 2013).

References

Adamatzky AI (1996) On the particle-like waves in the discrete model of excitable medium. Neural Netw World 1:3–10

Adamatzky AI (1998a) Universal dynamical computation in multidimensional excitable lattices. Int J Theory Phys 37:3069–3108

Adamatzky AI (1998b) Chemical processor for computation of voronoi diagram. Adv Mater Opt Electron 6(4):191–196

Adamatzky A (ed) (2001) Collision-based computing. Springer, London

Adamatzky A, De Lacy CB, Asai T (2005) Reaction-diffusion computers. Elsevier, New York. isbn:0444520422

Adleman LM (1994) Molecular computation of solutions to combinatorial problems. Science 266(11): 1021–1024

Adleman L (1998) Computing with DNA. Sci Am 279(2):34–41

Babbage C (1822) On machinery for calculating and printing mathematical tables. Edinb Philos J. In: Jameson R, Brewster D (eds) vol VII. Archibald Constable, Edinburgh, pp 274–281

Babbage C (1825) Observations on the application of machinery to the computation of mathematical tables. Phil Mag J LXV:311–314. London: Richard Taylor

Babbage C (1826) On a method of expressing by signs the action of machinery. Philos Trans R Soc Lond 116(Part III):250–265

Bath J, Turberfield AJ (2007) DNA nanomachines. Nat Nanotechnol 2:275–284

Benioff P (1982) Quantum mechanical models of Turing machines that dissipate no energy. Phys Rev Lett 48:1581

Bennett CH (1973) Logical reversibility of computation. IBM J Res Dev 17(6):525–532

Bennett CH (1982) The thermodynamics of computation – a review. Int J Theor Phys 21(12):905–940

Bennett CH (2003) Notes on Landauer's principle, reversible computation, and Maxwell's demon. Stud Hist Philos Mod Phys 34:501–510. eprint physics/0210005

Berger R (1966) The undecidability of the domino problem. Mem Am Math Soc 66:1–72

Bernal JD (1964) The structure of liquids. Proc R Soc Lond Ser A 280:299

Blum L, Cucker F, Shub M, Smale S (1996) Complexity and real computation: a manifesto. Int J Bifurc Chaos 6(1):3–26. World Scientific, Singapore

Boole G (1847) Mathematical analysis of logic: the mathematical analysis of logic: Being an essay towards a calculus of deductive reasoning, pamphlet

Boole G (1854) An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities. Macmillan, London

Bragg L, Lomer WM (1948) A dynamical model of a crystal structure II. Proc R Soc A 196:171–181

Bragg L, Nye JF (1947) A dynamical model of a crystal structure. Proc R Soc A 190:474–481

Bush V (1931) The differential analyzer: a new machine for solving differential equations. J Frankl Inst 212:447

Canny J (1988) Some algebraic and geometric computations in PSPACE. In: Cole R (ed) Proceedings of the 20th annual ACM symposium on the theory of computing. ACM Press, Chicago, pp 460–467

Canny J, Reif JH (1987) New lower bound techniques for robot motion planning problems. In: 28th annual IEEE symposium on foundations of computer science, Los Angeles, pp 49–60

Canny J, Donald B, Reif JH, Xavier P (1988) On the complexity of kinodynamic planning. In: 29th annual IEEE symposium on foundations of computer science, White Plains, pp 306–316. Published as Kinodynamic motion planning J ACM 40(5): 1048–1066 (1993)

Chandran H, Gopalkrishnan N, Reif J (2013) In: Mavroidis C, Ferreira A (eds) DNA nanoRobotics, chapter, nanorobotics: current approaches and techniques. Springer, New York, pp 355–382. ISBN 13 **:** 9781461421184, ISBN 10 **:** 1461421187

Chase GC (1980) History of mechanical computing machinery. IEEE Ann Hist Comput 2(3):198–226

Cohen IB, Welch GW (1999) Makin' numbers: Howard Aiken and the computer. MIT Press, Cambridge, MA

Corcoran SG, Colton RJ, Lilleodden ET, Gerberich WW (1997) Phys Rev B 190:474

Crescenzi P, Papadimitriou CH (1995) Reversible simulation of space-bounded computations. Theor Comput Sci 143(1):159–165

Danial R, Woo SS, Turicchia L, Sarpeshkar R (2011) Analog transistor models of bacterial genetic circuits. In: Proceedings of the 2011 I.E. biological circuits and systems (BioCAS) conference, San Diego, pp 333–336

Daniel R, Rubens J, Sarpeshkar R, Lu T (2013) Synthetic analog computation in living cells. Nature. doi:10.1038/nature12148

Davis M (2000) The universal computer: the road from Leibniz to Turing. Norton Press, Norton

Deutsch D (1985) Quantum theory, the church-Turing principle and the universal quantum computer. Proc R Soc Lond A400:97–117

Dolev S, Fitoussi H (2010) Masking traveling beams: optical solutions for NP-complete problems, trading space for time. Theor Comput Sci 411:837–853

Engineering Research Associates Staff (1950) High-speed computing devices. McGraw-Hill Book, New York City

Feynman RP (1963) "ratchet and pawl", chapter 46. In: Feynman RP, Leighton RB, Sands M (eds) The Feynman lectures on physics, vol 1. Addison-Wesley, Reading

Feynman RP (1982) Simulating physics with computers. Int J Theor Phys 21(6/7):467–488

Finney JL (1970) Random packings and the structure of simple liquids. I The geometry of random close packing Proc R Soc Lond A Math Phys Sci 319(1539):479–493

Fisher EG (1911) Tide-predicting machine. Eng News 66:69–73

Fredkin E, Toffoli T (1982) Conservative logic. Int J Theory Phys 21:219–253

Freeth T, Bitsakis Y, Moussas X, Seiradakis JH, Tselikas A, Mangou H, Zafeiropoulou M, Hadland R, Bate D, Ramsey A, Allen M, Crawley A, Hockley P, Malzbender T, Gelb D, Ambrisco W, Edmunds MG (2006) Decoding the ancient Greek astronomical calculator known as the Antikythera mechanism. Nature 444:587–591

Goliaei S, Foroughmand-Araabi M (2013) Light ray concentration reduces the complexity of the wavelength-based machine on PSPACE languages, unpublished manuscript

Goliaei S, Jalili S (2009) An optical wavelength-based solution to the 3-SAT problem. In: Dolev S, Oltean M (eds) Optical supercomputing, Lecture notes in computer science, vol, vol 5882, pp 77–85

Goliaei S, Jalili S (2012) An optical wavelength-based computational machine. Unconventional computation and natural computation lecture notes in computer science, vol 7445, pp 94–105. Also, Int J Unconv Comput (in press)

Grunbaum S, Branko SGC (1987) Tilings and patterns, chapter 11. H Freeman, San Francisco

Gruska J (1999) Quantum computing. McGraw-Hill, New York

Haist T, Osten W (2007) An optical solution for the traveling salesman problem. Opt Express 15(16):10473–10482

Hamer D, Sullivan G, Weierud F (1998) Enigma variations: an extended family of machines. Cryptologia 22(3):211–229

Hartree DR (1950) Calculating instruments and machines. Cambridge University Press, London

Henrici (1894) On a new harmonic analyzer, Phil Mag 38:110

Hopcroft JE, Schwartz JT, Sharir M (1984) On the complexity of motion planning for multiple independent objects: PSPACE hardness of the warehouseman's problem. Int J Robot Res 3(4):76–88

Horsburgh EM (1914) Modern instruments of calculation. G. Bell & Sons, London, p 223

Jaeger G (2006) Quantum information: an overview. Springer, Berlin

Jakubowski MH, Steiglitz K, Squier R (1998) State transformations of colliding optical solitons and possible application to computation in bulk media. Phys Rev E58:6752–6758

Jakubowski MH, Steiglitz K, Squier R (2001) Computing with solitons: a review and prospectus, collision-based computing. Springer, London, pp 277–297

Jevons WS (1870) On the mechanical performance of logical inference. Philos Trans R Soc 160(Part II): 497–518

Jevons SW (1873) The principles of science; a treatise on logic and scientific method. Macmillan, London

Kelvin L (1878) Harmonic analyzer and synthesizer. Proc R Soc 27:371

Knott CG (ed) (1915) Napier tercentenary memorial volume. Published for the Royal Society of Edinburgh by Longmans, Green, London

Landauer R (1961) Irreversibility and heat generation in the computing process. IBM J Res Dev 5:183

Lehmer DH (1928) The mechanical combination of linear forms. Am Math Mon 35:114–121

Lenstra AK, Shamir A (2000) Analysis and optimization of the TWINKLE factoring device, proc. Eurocrypt 2000, LNCS 1807. Springer, Heidelberg, pp 35–52

Lewis HR, Papadimitriou CH (1981) Elements of the theory of computation. Prentice-Hall, Upper Saddle River, pp 296–300. and 345–348

Lewis HR, Papadimitriou CH (1997) Elements of the theory of computation, 2nd edn. Prentice Hall, Upper Saddle River

Li M, Vitanyi P (1996) Reversibility and adiabatic computation: trading time and space for energy. Proc R Soc Lond Ser A 452:769–789. (Online preprint quant-ph/9703022)

Lindgren M (1990) Glory and failure: difference engines of Johann Muller, Charles Babbage and Georg and Edvard Scheutz. MIT Press, Cambridge, MA

Lovelace A, translation of "Sketch of the Analytical Engine" by L. F. Menabrea with Ada's notes and extensive commentary. Ada Lovelace (1843) Sketch of the analytical engine invented by Charles Babbage. Esq Scientific Memoirs 3:666–731

Ludgate P (1909–1910) On a proposed analytical engine. Sci Proc Roy Dublin Soc 12:77–91

Madou MJ (2002) Fundamentals of microfabrication: the science of miniaturization, 2nd edn. CRC Publishers, Boca Raton

Magnasco MO (1997) Chemical kinetics is Turing universal. Phys Rev Lett 78:1190–1193

Mao C, LaBean TH, Reif JH, Seeman NC (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 407:493–495

Martin E (1992) The calculating machines. The MIT Press, Cambridge, MA

Miller D (1916) The Henrici harmonic analyzer and devices for extending and facilitating its use. J Franklin Inst 181:51–81 and 182:285–322

Moore C (1990) Undecidability and unpredictability in dynamical systems. Phys Rev Lett 64:2354–2357

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

de Morin H (1913) Les appareils d'intégration: intégrateurs simples et composés; planimètres; intégromètres; intégraphes et courbes intégrales; analyse harmonique et analyseurs. Gauthier-Villars Publishers, Paris

Munakata T, Sinha S, Ditto WL (2002) Chaos computing: implementation of fundamental logical gates by chaotic elements. IEEE Trans Circ Syst-I Fundam Theory Appl 49(11):1629–1633

Muntean O, Oltean M (2009) Deciding whether a linear diophantine equation has solutions by using a light-based device. J Optoelectron Adv Mater 11(11): 1728–1734

Napier J (1614) Mirifici logarithmorum canonis descriptio (the description of the wonderful canon of logarithms). Hart, Edinburgh

Nielsen M, Chuang I (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge

Norman JM (ed) (2002) The origins of cyberspace: from Gutenberg to the internet: a sourcebook on the history of information technology. Norman Publishing, Novato

Oltean M (2008) Solving the Hamiltonian path problem with a light-based computer. Nat Comput 6(1):57–70

Oltean M, Muntean O (2008) Exact cover with light. New Gener Comput 26(4):329–346

Oltean M, Muntean O (2009) Solving the subset-sum problem with a light-based device. Nat Comput 8(2):321–331

Oughtred W (1632) Circles of proportion and the horizontal instrument. Translated and Published by William Forster, London

Pascal E (1645) Lettre dédicatoire à Monseigneur le Chancelier sur le sujet de la machine nouvellement inventée par le sieur B. P pour faire toutes sortes d'opérations d'arithmétique par un mouvement réglé sans plume ni jetons, suivie d'un avis nécessaire à ceux qui auront curiosité de voir ladite machine et de s'en servir

Plummer D, Dalton LJ, Peter F (1999) The recodable locking device. Commun ACM 42(7):83–87

Reif JH (1979) Complexity of the mover's problem and generalizations. In: 20th Annual IEEE symposium on foundations of computer science, San Juan, Puerto Rico, pp 421–427. Also appearing in Chapter 11 in Planning, geometry and complexity of robot motion. Schwartz J (ed) Ablex Pub, Norwood, pp 267–281 (1987)

Reif JH (2009a) Quantum information processing: algorithms, technologies and challenges, invited chapter. In: Eshaghian-Wilner MM (ed) Nano-scale and bio-inspired computing. Wiley, Hoboken

Reif JH (2009b) Mechanical computation: it's computational complexity and technologies, invited chapter. In: Meyers RA (ed) Encyclopedia of complexity and system science. Unconventional Computing (Section Editor: Andrew Adamatzky). Springer, New York, ISBN: 978-0-387-75888-6

Reif JH, LaBean TH (2007) Autonomous programmable biomolecular devices using self-assembled DNA nanostructures, communications of the ACM (CACM), Special Section entitled "New Computing Paradigms (edited by Toshinori Munakata)

Reif JH, LaBean TH (2009) Nanostructures and autonomous devices assembled from DNA. Invited chapter. In: Eshaghian-Wilner MM (ed) Nano-scale and bio-inspired computing. Wiley, Hoboken

Reif JH, Sahu S (2008) Autonomous programmable DNA nanorobotic devices using DNAzymes, 13th international meeting on DNA computing (DNA 13), Memphis, June 4–8, 2007. In: Garzon M, Yan H (eds) DNA computing: DNA13, Springer-Verlag lecture notes for computer science (LNCS), vol 4848. Springer, Berlin, pp 66–78. Published in Special Journal Issue on Self-Assembly, Theoretical Computer Science (TCS) 410(15):1428–1439 (2009)

Reif JH, Sharir M (1985) Motion planning in the presence of moving obstacles. In: 26th annual IEEE symposium on foundations of computer science, Portland, pp 144–154. Published in Journal of the ACM (JACM) 41:4, pp 764–790 (1994)

Reif JH, Sun Z (1998) The computational power of frictional mechanical systems. In: Third international workshop on algorithmic foundations of robotics, (WAFR98), Pub. by A. K. Peters Ltd, Houston, pp 223–236. Published as On frictional mechanical systems and their computational power, SIAM Journal of Computing(SICOMP) 32(6):1449–1474 (2003)

Reif JH, Wang H (1998) The complexity of the two dimensional curvature-constrained shortest-path problem. In: Third international workshop on algorithmic foundations of robotics (WAFR98), Pub. by A. K. Peters Ltd, Houston, pp 49–57

Reif JH, Tygar D, Yoshida A (1990) The computability and complexity of optical beam tracing. 31st annual IEEE symposium on foundations of computer science, St. Louis, pp 106–114. Published as The computability and complexity of ray tracing in discrete & computational geometry 11:265–287 (1994)

Reif J, Chandran H, Gopalkrishnan N, LaBean T (2012) Self-assembled DNA nanostructures and DNA devices. Invited chapter 14. In: Cabrini S, Kawata S (eds) Nanofabrication handbook. CRC Press, Taylor and Francis Group, New York, pp 299–328. isbn13:9781420090529, isbn10: 1420090526

Rothemund PWK (2000) Using lateral capillary forces to compute by self-assembly. Proc Natl Acad Sci USA 97:984–989

Rothemund PWK (2006) Folding DNA to create nanoscale shapes and patterns. Nature 440:297–302

Rothemund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2(12): electronic pub. e424. doi:10.1371/journal.pbio.0020424

Schwartz JT, Sharir M (1983) On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. Commun Pure Appl Math 36:345–398

Seeman NC (2004) Nanotechnology and the double helix. Sci Am 290(6):64–75

Senum P, Riedel M (2011) Rate-independent constructs for chemical computation. PLoS One 6(6):e21414

Shamir A (1999) Factoring large numbers with the TWINKLE device, cryptographic hardware and embedded systems (CHES) 1999, LNCS 1717, 2-12. Springer, Heidelberg

Shamir A (n.d.) Method and apparatus for factoring large numbers with optoelectronic devices, patent 475920, filed 12/30/1999 and awarded 08/05/2003

Shannon C (1938) A symbolic analysis of relay and switching circuits. Trans Am Inst Electr Eng 57:713–719

Shapiro E (1999) A mechanical turing machine: blueprint for a biomolecular computer. In: Fifth international meeting on DNA-based computers at the Massachusetts Institute of Technology, Proc. DNA Based Computers V: Cambridge

Sinha S, Ditto W (1999) Computing with distributed chaos. Phys Rev E Stat Phys Plasmas Fluids Relat Interdiscip Top 60(1):363–377

Soloveichik D, Cook M, Winfree E, Bruck J (2008) Computation with finite stochastic chemical reaction networks. Nat Comput 7(4):615–633

Soloveichik D, Seelig G, Winfree E (2010) DNA as a universal substrate for chemical kinetics. Proc Natl Acad Sci 107:5393–5398

Soroka WA (1954) Analog methods in computation and simulation. McGraw-Hill, New York

Squier R, Steiglitz K (1994) Programmable parallel arithmetic in cellular automata using a particle model. Complex Syst 8:311–323

Svoboda A (1948) Computing mechanisms and linkages. McGraw-Hill, New York

Swade D (1991) Charles Babbage and his calculating engines. Michigan State University Press, East Lansing

Tate SR, Reif JH (1993) The complexity of N-body simulation. In: Proceedings of the 20th annual colloquium on automata, languages and programming (ICALP'93), Lund, pp 162–176

Thomson W (later known as Lord Kelvin) (1878) Harmonic analyzer. Proc R Soc Lond 27:371–373

Turck JAV (1921) Origin of modern calculating machines. The Western Society of Engineers, Chicago

Turing A (1937) On computable numbers, with an application to the Entscheidungs problem. In: Proceedings of the London mathematical society, second series, vol 42, London, pp 230–265. Erratum in vol 43, pp 544–546

da Vinci L (1493) Codex Madrid I

Wang H (1963) Dominoes and the AEA case of the decision problem. In: Fox J (ed) Mathematical theory of automata. Polytechnic Press, Brooklyn, pp 23–55

Winfree E, Yang X, Seeman NC (1996) Universal computation via self-assembly of DNA: some theory and experiments, DNA based computers II, volume 44 of DIMACS. American Mathematical Society, Providence, pp 191–213

Winfree E, Liu F, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394:539–544

Wolfram S (1984) Universality and complexity in cellular automata. Physica D10:1–35

Woods D, Naughton TJ (2009) Optical computing. Appl Math Comput 215(4):1417–1430

Xia Y, Whitesides GM (1998) Soft lithography. Ann Rev Mater Sci 28:153–184

Yan H, LaBean TH, Feng L, Reif JH (2003a) Directed nucleation assembly of barcode patterned DNA lattices. Proc Natl Acad Sci U S A 100(14):8103–8108

Yan H, Feng L, LaBean TH, Reif J (2003b) DNA nanotubes, parallel molecular computations of pairwise exclusive-or (XOR) using DNA "string tile" self-assembly. J Am Chem Soc (JACS) 125(47): 14246–14247

Yin P, Yan H, Daniel XG, Turberfield AJ, Reif JH (2004) A unidirectional DNA walker moving autonomously along a linear track. Angew Chem Int Ed 43(37):4906–4911