

applicant: **Marc D. Riedel**, *Ph.D., Caltech*
position: Assistant Professor, Tenure-Track
department: Electrical Engineering, Computer Engineering or Computer Science

Research Statement

For most students of computer science and electrical engineering, the trend during the formative years of graduate school is one of increasing specialization. Each student carves out a niche, delving into a specific topic for his or her dissertation. And yet, moving beyond graduate school, too narrow a focus in a single domain of expertise can stunt a research career. Breakthroughs nearly always stem from the infusion of new ideas and techniques, sometimes from quite unrelated areas. I believe that a sound approach is to acquire a breadth of knowledge in a variety of fields coupled with a depth of knowledge of the underlying and unifying concepts. One should remain versatile, exploring problems of merit as they arise.

My research interests can be broadly categorized into the following areas: novel **circuit constructs**, novel **data structures and algorithms** for circuit design, and novel **computational platforms**. Of course, all three areas are intertwined. For instance, in the nascent field of *systems biology*, researchers are studying cellular signaling pathways and genetic regulatory networks from a computational perspective. Such systems are inherently noisy, operate asynchronously, and involve thousands of molecular species. Modeling them requires entirely new circuit constructs. Furthermore, custom-tailored data structures and algorithms are needed to cope with the scale and complexity of the problems.

1. Novel Circuit Constructs

In the realm of digital design, a rigid hierarchical view has become firmly entrenched: Logic gates are built from transistors; combinational circuits are built from logic gates arranged in feed-forward (i.e., *acyclic*) configurations; synchronous circuits are built from combinational circuits and memory elements arranged in feedback (i.e., *cyclic*) configurations; and, finally, digital systems are built from synchronous modules and peripheral logic. This paradigm has been tremendously successful in the design of silicon integrated circuits – so successful, in fact, that the perception exists that this is the only way circuits can be designed.

1.1 Cyclic Combinational Circuits

In my dissertation, I establish a counterintuitive construct: combinational circuits with cycles. My interest in this topic was sparked by paper of R. Rivest from 1977 [33]. I came across it in my third year of graduate studies, after I had begun writing a dissertation on a completely different topic. The setting is a familiar one: logic gates (AND, OR, NOT, etc.) computing boolean functions. However, there is a striking incongruity in one of the examples: A circuit is labeled *combinational* and yet the gates are arranged in a *cyclic* configuration.

I had learned in logic design courses that a combinational circuit – that is, a circuit with outputs that depend only on the current inputs – cannot contain feedback paths. It must be acyclic. If a circuit has feedback paths resulting in cycles, then it behaves sequentially – that is, the outputs depend on past as well as current inputs. Logic design textbooks state this fact quite plainly (e.g., [18], p. 14; [41], p. 193). However, in spite of the fact that it is cyclic, Rivest’s circuit clearly fits the definition of combinational. Was this an isolated example? Or did it hint at a general concept, perhaps one that had been overlooked? I could not find a single reference to Rivest’s paper in the literature.

Of course, the design of cyclic circuits is not self-evident since feedback can result in spurious behavior. When is a circuit with cycles valid? I require that a circuit be combinational in a rigorous sense: For every possible assignment of input values, it must produce the required output values *regardless* of the prior values on the wires and for *any* choice of delay parameters. Acyclic circuits often contain “false” paths, that is, topological paths that are never sensitized. (As early as 1978, Khrapchenko recognized that the delay of a circuit may be less than its depth if its longest path is false [19].) A cyclic circuit is valid provided that all of the cycles are false.

Consider the example in Figure 1. If $x = 0$ then g_1 produces an output of 0, because 0 is a controlling value for an AND gate. If $x = 1$ then g_4 produces a value of 1, because 1 is a controlling value for an OR gate. In both cases, the cycle is broken and the circuit produces definite outputs. Since x must assume one of these two values, we conclude that the circuit *always* produces definite outputs. In fact, it implements two functions that both depend on all five variables. Note that the computation of the two functions overlaps. If we were to implement these functions with an acyclic circuit, then we would need eight two-input gates.

In my dissertation, I explore the topic of cyclic combinational circuits in depth – both the theoretical underpinnings and the practical implications [32]. On the theoretical front, I show that the distinction between cyclic and acyclic circuits is a fundamental one. I prove that certain functions can be implemented with fewer gates using cyclic designs than is possible with acyclic designs. In fact, some functions can be implemented with *half* as many gates. On the practical front, I present a general methodology for the **analysis** and **synthesis** of cyclic circuits [29], [30]. The package that I developed, CYCLIFY, is built within the Berkeley SIS environment [35].

The approach for synthesis is conceptually very general. Cycles are introduced through the incremental application of restructuring and minimization operations, optimizing a design for area and delay. These optimizations are carried through to the decomposition and technology mapping phases. Perhaps the most salient result to report is that cyclic solutions are not a rarity; they can readily be found for most circuits of practical interest. CYCLIFY reduced the area of standard benchmark circuits by as much as 30% and the delay by as much as 25%. Such numbers immediately caught the attention of practitioners. Indeed, companies such as Altera and Intel have expressed strong interest. (Caltech has applied for a patent on the methodology.)

Of course, a number of implementation issues still need to be addressed. The verification and mapping routines in most CAD packages balk when given a cyclic combinational circuit. (Some routines, including those in Berkeley SIS, check a design compulsively after every transformation to see if it contains cycles. If it does, the program screeches to a halt.) These tools will have to be coaxed into accepting this new paradigm. This change will not happen overnight, but it will happen. Perhaps the ALU of the next Intel Pentium chip will have a cyclic design. At the very least, I hope that future editions of logic design textbooks will posit that combinational circuits can – and should – be designed with cycles.

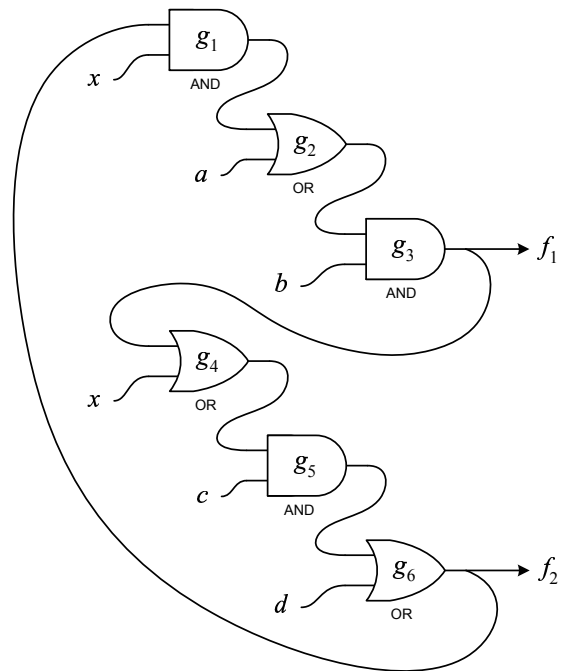


Figure 1: A cyclic combinational circuit.

$$f_1 = b(a + x(d + c)),$$

$$f_2 = d + c(x + ba).$$

(+) denotes OR, (·) denotes AND

1.2 Future Directions

The concepts in my dissertation pertain to logic design at the gate level. The circuit design process typically begins at a higher, more abstract level in the form of a behavioral specification. In behavioral-level designs, feedback is sometimes introduced during resource-sharing optimizations [38]. In such designs, there is dedicated control circuitry multiplexing the feedback paths between functional units. An exciting direction for future research is the development of a general cyclic design methodology at the behavioral level, for instance within the framework of hardware description languages such as Verilog and VHDL.

Logic gates have an inherent direction; by definition, they are feed-forward devices. Beneath the abstraction of gates, digital circuits are built from transistors. Transistors have an inherent direction in some technologies, such as bipolar-junction (BJT), but not in others, such as metal-oxide-semiconductor field-effect (MOSFET). In an idealized sense, transistors can be viewed as *switches*. Switching circuits – built from electro-mechanical relays, not transistors – were the subject of seminal papers by Claude Shannon [36], [37]. The circuits of Shannon’s day often had switches through which current flows bidirectionally. For instance, in the bridge circuit in Figure 2, current through the switch x_3 flows up in some cases and down in others. This circuit implements a function of five variables with five switches. Limiting designs to gated architectures imposes topological constraints at the switch level. (Implementing the same function as in Figure 2 with logic gates, say two-input NAND/NOR gates, would be more costly – assuming that each gate requires two transistors.) I believe that it is worth examining the interplay between cyclic gate-level designs and the resulting switch-level circuits.

Cyclic constructs may also play a role in an old topic that has found renewed interest: computing with **noisy** or **faulty logic gates**. The idea of computing with probabilistic logic gates was first proposed by von Neumann in 1956 [40]. With his scheme, each gate may have an individual probability of failure approaching $\frac{1}{2}$. Nevertheless, the circuit computes correctly with an asymptotic probability approaching 1. Research in this same vein, mostly aimed at proving lower bounds with various failure models, has been pursued by others, most notably by Pippenger [27]. Still, a truly effective framework for designing circuits with noisy/probabilistic gates has remained elusive. Feedback is the primary mechanism for coping with noise in analog computation. I am exploring the idea of using feedback in mixed-mode analog/digital circuits in order to bias noisy signals toward stable values. My goal is to combine this strategy with feedback optimizations for area and delay.

2. Novel Data Structures and Algorithms

By some measures, modern integrated circuits are among the most complex devices ever designed by humankind. To wit, Intel Corporation is planning on delivering chips with 1.7 billion transistors this year [15]. The true marvel of the technology is not in its *physical* complexity, although the challenge of manufacturing devices on the scale of 45 nanometers is mind-boggling. Rather it is the scale of the *computation* that should astonish us. Consider a combinational circuit, such as a digital signal processing chip, say with 128 boolean inputs. It performs mappings from a space of 2^{128} input values to boolean output values. A truth table representing 2^{128} input assignments is inconceivable; *verifying* the behavior of such a circuit, let alone designing it, would seem like an intractable problem.

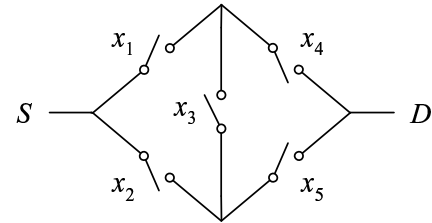


Figure 2: A switching circuit with a cyclic topology.

Variables control the switches: If a variable is valued 1, the corresponding switch is closed; if it is valued 0, the switch is open. The circuit evaluates to 1 if there is a closed path between S and D , and to 0 otherwise. It implements the function

$$g = x_1x_4 + x_2x_5 + x_3(x_1x_5 + x_2x_4).$$

2.1 Symbolic Analysis

Circuit designers have succeeded in their endeavor largely as a result of innovations in the data structures used to represent boolean functions. So-called symbolic techniques, based on efficient data structures such as *binary decision diagrams* (BDDs), have revolutionized the business. First proposed in 1959 by Lee [22], BDDs were popularized in 1986 through a seminal paper by Bryant [8]. Although comparable in size to a truth table in the worst case, BDDs are surprisingly compact for most of the functions encountered in practice. Functions of thousands of variables can often be represented with ease. The strength of the representation resides in the fact that it is canonical and that it can be manipulated efficiently. Most logical operations have linear complexity in the number of variables.

A BDD consists of a directed graph. Each node either has an associated input variable or is designated as a constant (“0” or “1”). Variable nodes have two outgoing edges, one designated as “0” and the other designated as “1”. The two constant nodes have out-degree zero. To evaluate a function represented by a BDD, one begins at a designated source node and follows a path dictated by the values of the variables until one arrives at one of the two constant nodes. It specifies the value of the function.

The accepted wisdom is that BDDs must be directed *acyclic* graphs (DAGs). Indeed, if a BDD contains cycles, then one might loop indefinitely when evaluating a function. However, this need not be the case. Consider the pair of BDDs shown in Figure 3. The BDD for f_1 depends on that for f_2 , and vice-versa. Nevertheless, for any assignment of values to the variables, both functions are defined. (To see this, note that f_1 only depends on f_2 if $x = 1$ and f_2 only depends on f_1 if $x = 0$.) I have constructed examples of cyclic BDDs with provably fewer nodes than is possible with acyclic BDDs representing the same functions [32]. In future work, I plan on developing a comprehensive framework for BDDs with cycles.

Symbolic analysis with BDDs is an integral part of the design methodology that I propose in my Ph.D. dissertation. When introducing cyclic dependencies in combinational logic, I validate and rank potential solutions through *exact* (or *input-dependent*) timing analysis. I compute the true delay through symbolic event propagation with BDDs [31]. Through this work I have acquired an expertise in symbolic data structures.

The development and application of sophisticated representations for boolean functions continues to be a fertile topic of research, in circuit design as well as other domains. So-called SAT-based techniques, based on heuristic solutions to the boolean satisfiability problem, have been highly successful [20], [24]. Spectral transformations have been studied by the theoretical community, with applications in learning theory and artificial intelligence [7], [16], [21]; this topic deserves further attention by the logic synthesis community [9]. I am convinced that novel data representations, including multivalued representations over mixed discrete/continuous domains, will be key to the solution of problems in new areas, such as systems biology (see Section 3).

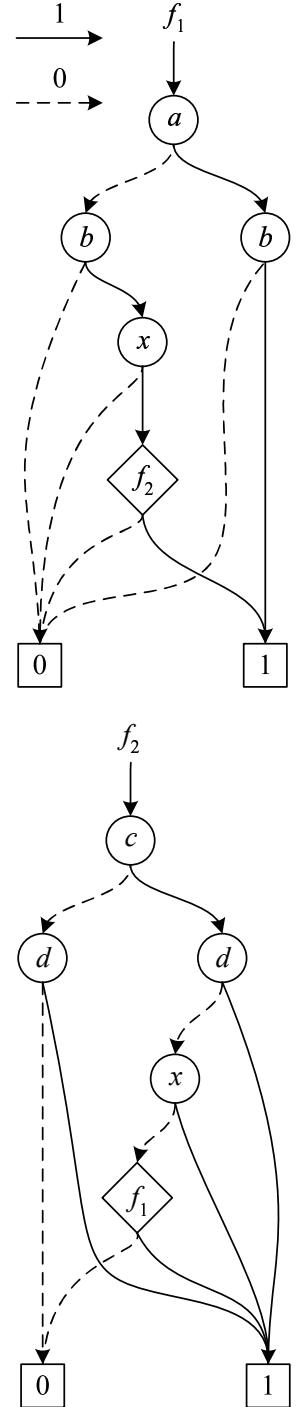


Figure 3: Binary decision diagrams for the functions f_1 and f_2 of Figure 1. Notice the cyclic dependency: The BDD for f_1 depends on that for f_2 , and vice-versa.

2.2 Synthesis Algorithms

In the steady march toward design automation, the research community has grappled with a succession of problems classified as NP-hard (or harder!). Progress has come in fits and starts with the development of heuristic algorithms for problems such as rectangle covering and graph partitioning. Comparatively little research has appeared on logic synthesis in recent years, as the emphasis in electronic design automation has shifted to higher-level topics, such as behavioral compilers, and lower-level topics, such as routing and signal integrity. And yet, logic synthesis remains a pivotal step. Optimizations for different criteria at the logic level – area, delay, power, reliability – translate directly into improvements at the silicon level.

The accepted paradigm for the synthesis of combinational logic is that championed by the UC Berkeley Synthesis Group. In the early 1980s, the group developed highly successful heuristics for *two-level* logic minimization [5]. Then, in the late 1980s and early 1990s, they proposed a methodology for *multi-level* synthesis, consisting of the ad-hoc application of minimization, decomposition, and restructuring operations [6]. Minimization in the multi-level case is accomplished by feeding appropriate “don’t care” conditions into two-level minimization algorithms. The algorithms and tools from this era assumed a *sum-of-products* (SOP) representation for boolean functions. While the SOP representation might be appropriate for circuits built from primitive gates (e.g., NAND and NOR gates), it is ill-suited for other technologies, such as reconfigurable hardware based on memory cells. When mapping a logic function onto a memory cell, the only relevant factor is the size of the function’s support set (i.e., how many variables it depends upon).

With the advent of modern data structures such as BDDs, many of the general problems in logic synthesis have been formulated in symbolic terms. And yet, somewhat surprisingly, certain fundamental operations relating to **functional decomposition** have not been explored within a BDD framework. I am interested in the following open problem: Suppose that we have *external* variables $x_1, \dots, x_m$, and *target* functions $f_1, \dots, f_n$, each of which depends on a subset of the external variables. How can we partition the computation into *blocks* $g_1, \dots, g_n$, such that g_i implements f_i for $i = 1, \dots, n$, and each block has the *fewest* possible inputs (whether external variables or outputs from other blocks)?

Consider the target functions f_1 and f_2 of Figure 1. How can we implement f_1 and f_2 with blocks g_1 and g_2 , respectively, such that g_1 depends on the fewest possible inputs in $\{a, b, c, d, x, f_2\}$ and g_2 depends on the fewest possible inputs in $\{a, b, c, d, x, f_1\}$? The solution is sketched in Figure 4. Both blocks can be expressed with only four inputs. (Indeed, this was apparent from the BDDs given in Figure 3.) Note that this decomposition has cyclic dependencies. In general, cycles can result in spurious computation. A cyclic decomposition must be validated with the techniques discussed in the previous section.

The problem of functional decomposition, in its most general form, is NP-complete. However, it may admit effective heuristic solutions. Note that in a decomposition, outputs from blocks cannot be treated as independent variables. One can say that the output from a block g_i has “don’t care” conditions corresponding to the *complement* of the function f_i that g_i implements. A possible strategy is to tackle the decomposition problem with “don’t care” minimization techniques. Such techniques are well-developed for SOP representations. There has been some work addressing this problem for BDD representations [14], [26]. However, in my opinion, this line of research deserves further attention.

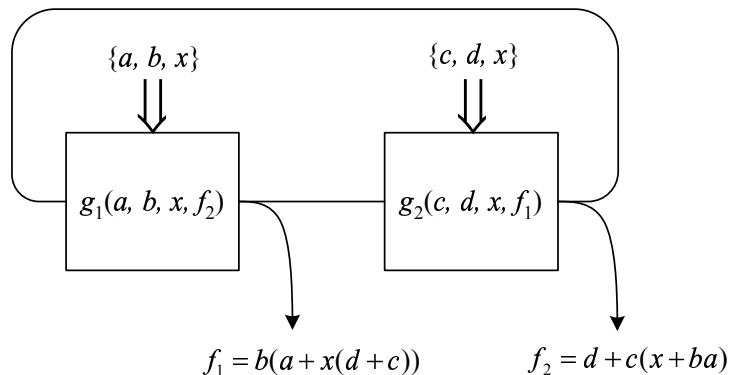


Figure 4: Partitioning the computation of the functions f_1 and f_2 from Figure 1.

Embedded memory has become increasingly inexpensive in custom hardware. One can readily obtain a large array of programmable memory cells where each cell is, say, a megabit. This has given rise to many interesting research problems, sometimes classified under the general heading of hardware/software co-design [10]. For instance, instead of computing numerical functions iteratively through software, one can implement them directly in hardware by precomputing tables of coefficients. In an abstract sense, memory cells in reconfigurable hardware can be viewed as complex programmable “gates”, sometimes referred to as look-up tables (LUTs). For instance, the blocks g_1 and g_2 in Figure 4 could each be implemented with four-bit LUTs. I have explored the topic of computing numerical functions with LUT arrays through BDD-based decomposition [34]. Currently, I am pursuing the concept of *cyclic* arrangements of LUTs, for instance ring, torus, and hypercube topologies.

3. Novel Computational Platforms

Recently, there has been a groundswell of interest in novel devices and platforms for computation. Much of it has been spurred by research in material science. For instance, constructs such as carbon nanotubes and semiconducting nanowires promise to push circuit manufacturing past the technological limits imposed by photolithography [28]. Research on biomolecular systems for computation remains speculative. Some researchers have contemplated using DNA as a computational medium. As early as 1994, Adelman demonstrated that DNA could be applied to combinatorial optimization problems, such as the Traveling Salesman Problem [1]. Such schemes boast a dizzying degree of parallelism: A DNA-based computer can process up to a trillion problem instances per microliter. However, the computation is inherently slow, with each *step* in the algorithm requiring on the order of seconds or minutes to complete. For this reason, it is not clear that biomolecular systems will ever compete with semiconductors as computational devices, *per se*.

The impetus to study biological systems from a computational perspective is, in fact, much broader. Although designed by evolution, biological systems – from single cells to higher organisms – have remarkable parallels to computing systems engineered by humans. They exhibit modular designs, with separate sensory and information-processing units; they have well-defined inputs and outputs, whether chemical or neurological; and they have elaborate synchronization and error-correction mechanisms. Clearly, the knowledge accumulated over the decades in computer engineering may provide valuable insights into the workings of biological systems. Beyond analysis, there is the hope that one day biologists could engineer a form of “logical” control over biological processes, designing pathways that produce specific outputs in response to different combinations of inputs. This approach could have applications in biochemical sensing as well as in the medical diagnosis and treatment of diseases [4].

What sort of specific expertise can circuit designers bring to the table? One of the great successes of the electrical engineering community has been in *abstracting* and *scaling* the design problem. The physical behavior of transistors is understood in terms of differential equations – say, with models found in tools such as SPICE [25]. However, the design of integrated circuits proceeds at a more abstract level – in terms of gates, modules, and components. In principle, one could write down a gargantuan system of coupled differential equations describing a circuit in its entirety, at the transistor level. However, one could never hope to compute solutions to such a system, and there would be little gained by doing so.

For a variety of cellular processes, biologists have developed a fine-grained understanding of the molecular mechanisms involved – the equivalent of a transistor model. Often the knowledge can be encapsulated in the form of a coupled set of chemical reactions. Consider the reactions



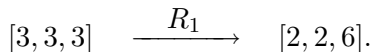
Here X_1 , X_2 , and X_3 are different types of molecules. Each reaction specifies how the types of molecules can combine. For instance, in reaction R_1 , one molecule of type X_1 combines with one molecule of type X_2 to produce three molecules of type X_3 . The types of molecules in a reaction that are consumed are referred to as the *reactants*, whereas those that are created are referred to as the *products*. Roughly speaking, the rate of occurrence of each reaction is proportional to the quantity of its reactants present. A set of reactions will often be annotated with physical constraints: chemical gradients, the localization of molecules, cell growth, etc.

In spite of the fact that precise models exist, methods for the *quantitative* analysis of biological systems remain elusive. The most common strategy has been to translate a set of reactions for a biological pathway into the language of physical chemistry, in the form of a system of coupled ordinary differential equations [13], [39]. However, as with integrated circuits, computing solutions to such a system is a formidable challenge for all but the simplest examples. Furthermore, this approach assumes that (i) the reactions take place in a spatially *uniform* environment, and (ii) the number of molecules of each type is a *continuous* quantity that varies *deterministically* over time. Typical biological systems, viewed at a molecular level, break both of these assumptions. Cellular processes often take place on the surface of membranes or in highly confined areas, in the presence of chemical gradients. The molecules involved are generally large and complex (e.g., proteins and RNA), and the quantities present in individual cells are relatively small (on the order of tens, hundreds, or thousands of molecules of each type). At this scale, individual reactions matter.

On the one hand, analysis of biological systems at the *macroscopic* level – in terms of differential equations with continuous variables – provides, at best, a crude approximation. On the other hand, analysis at the *microscopic* level – in terms of the molecular dynamics – is an intractable proposition. There is simply insufficient information or computational power to assign and track the position and velocity of individual molecules. The appropriate level is an intermediate one, sometimes called the *mesoscopic*. The state of the system is described in terms of **discrete** (integer) quantities of molecules and its evolution is analyzed **probabilistically**. For the set of reactions above, let the state be

$$S = [x_1, x_2, x_3],$$

where x_1, x_2 , and x_3 are integer variables, assuming non-negative values corresponding to the number of molecules of type X_1 , X_2 , and X_3 , respectively. When a reaction occurs, there is a corresponding state transition. For instance, suppose that the system is in the state $[3, 3, 3]$ and reaction R_1 occurs:



At this level of abstraction, the model corresponds to what is known as a *vector-addition system* in theoretical computer science [17]. It is also related to a formalism known as a *Petri net* [2].

In principle, the state space can be impossibly large. For a chemical system with n types of molecules, each of which can be present in quantities of up to m molecules, there are n^{m+1} possible states. A typical system might have $n = 50$ and $m = 1000$. In practice, spatial and temporal constraints can be exploited in the analysis. I am using symbolic data structures – specifically, multi-valued decision diagrams — to represent the state information compactly. Also, I am exploring the application of fuzzy quantization techniques to the state variables.

Fixing environmental variables, such as temperature and external chemical gradients, one can assume that a biological system behaves as a *Markov* process: The probability of future events depends only on the present state, not on the past sequence of events. Indeed, at each point in time, the probability of a given reaction occurring is a function of the current state only. In some cases, these transition probabilities can be derived from chemical principles. In others, they can be estimated based on experimental data. Provided that one can compute the transition probabilities, then one can track the evolution of the system stochastically.

The existing approach for analysis, proposed by Gillespie, consists of *Monte Carlo* simulation [12]. Starting from a given initial state, reaction sequences are generated at random, based on probability distributions computed from chemical principles. Figure 5 illustrates such a randomly chosen reaction sequence. By generating many reaction sequences and averaging the results, estimates can be formed for the probability distribution of the possible states of the system at a given point in time. Gibson and Bruck refined and extended Gillespie’s algorithm [11]. Lok and Brent have used it as the basis for a sophisticated simulation tool [23].

In general, a Monte Carlo procedure is a strategy of last resort. Randomized simulation provides an *implicit* estimate of the solution when there is insufficient information or knowledge about the mechanisms for an *explicit* calculation. The drawback of a Monte Carlo approach is that it requires a prohibitive number of trials: To reduce the uncertainty by a factor of $1/f$, one has to run f^2 as many trials. In Gillespie’s algorithm, the transition probabilities are, in fact, computed explicitly. However, once computed, these probabilities are simply used to make random choices and then discarded. In my opinion, one should use the intermediate calculations directly to determine the final probability distributions.

I have been collaborating with researchers at the Molecular Sciences Institute (MSI) in Berkeley on the development of an **analytic** framework for cellular signaling pathways. Instead of generating trajectories at random, my approach consists of a state space exploration through a branch-and-bound search. The algorithm is illustrated – albeit in a cursory way – in Figure 6. For the initial state and each subsequent state that is visited, separate branches are formed for each possible reaction. Along each branch, the probabilities are multiplicative. When two branches merge, the probabilities are additive. The search is neither strictly *depth*-wise, nor strictly *breadth*-wise. Rather, at each step, the highest probability branch is explored next. The search terminates when the sum of the probabilities of the unexplored branches is less than the desired amount of uncertainty. For a broad class of problems, this analysis achieves the same accuracy as a Monte Carlo simulation with provably less computation. More importantly, since each branch is labeled with a probability value, the analysis provides highly structured information about the state space. Through informed queries on the analysis results, one can study aspects of the system such as its boundary states and bifurcation points.

As with all interdisciplinary research, the key to success in this endeavor is to listen carefully to the practitioners. Biologists have accumulated a wealth of knowledge – both quantitative and qualitative – about biochemical networks. Beyond the reaction models, they can provide higher-level insights into the workings of cellular processes. Also, they can provide experimental data to calibrate the analysis and to validate the results. I am investigating a number of real biological systems that have been modeled in considerable detail. For instance, I am analyzing a well-known model of a virus called the *lambda phage*, consisting of 75 reactions with 57 types of molecules [3]. Also, I am studying a model for the pheromone-response pathway in diploid yeast, developed by the researchers at MSI, consisting of 96 reactions with 273 types of molecules.

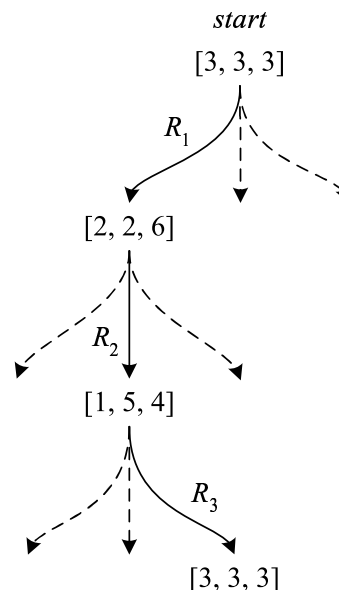


Figure 5: A randomly chosen reaction sequence in a Monte Carlo simulation. Beginning from the state $[3, 3, 3]$, R_1 is chosen, followed by R_2 , followed by R_3 .

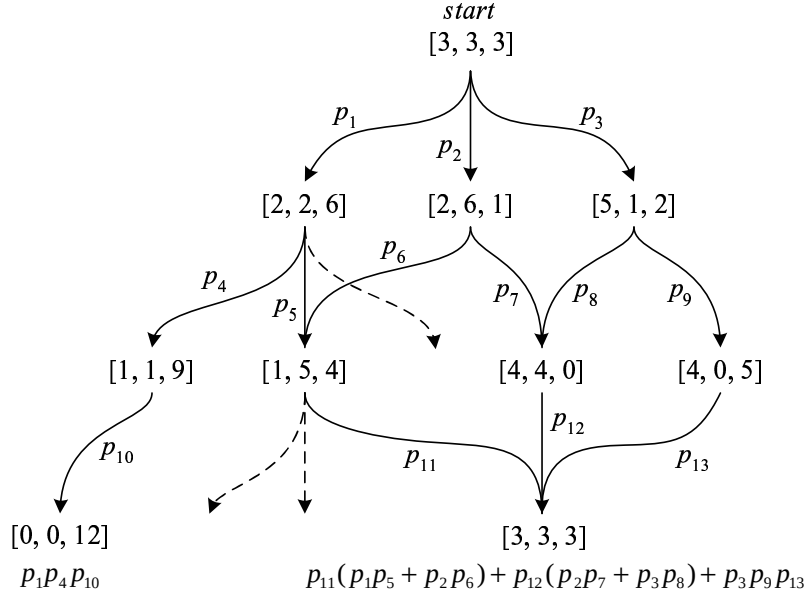


Figure 6: An illustration of the analytic method for the reactions in Eqs. 1, 2, and 3. Beginning from the state $[3, 3, 3]$, sequences of reactions are explored. Each reaction is labeled with a probability value. Along each branch, the probabilities are multiplicative; when two branches merge, the probabilities are additive. Only branches leading to two specific states after three reactions are shown: those leading to the *boundary* state $[0, 0, 12]$ and those returning to the initial state. The computation of the probabilities for these states is shown.

The great mathematician John von Neumann articulated the view that research should always be guided by applications that can have a positive and meaningful influence on humankind. This view was not based solely on his altruistic concerns, although these were deeply-rooted; rather, in his judgment, real-world applications give rise to the most *interesting* problems in mathematics. Given my background, research in computational biology may seem a bit off the beaten path. However, I believe that my strength in disparate fields such as discrete mathematics, complexity theory, and circuit design can be brought to bear on this line of research. As I embark on a career in academia, I plan to pursue a variety of topics – in logic design, distributed computing, and systems biology. I will continue to seek topics that satisfy von Neumann’s criteria: They must be conceptually deep and have the potential for a significant impact.

References

- [1] L. Adelman, “Molecular Computation of Solutions to Combinatorial Problems,” *Science*, Vol. 266, pp. 1021–1024, 1994.
- [2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, “Modelling with Generalized Stochastic Petri Nets,” Wiley, 1994.
- [3] A. Arkin, J. Ross, and H. McAdams, “Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage Lambda-Infected *E. Coli* Cells,” *Genetics*, Vol. 149, No. 1633, 1998.
- [4] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, “An Autonomous Molecular Computer for Logical Control of Gene Expression,” *Nature*, Vol. 429, pp. 423–429, 2004.
- [5] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, “Logic Minimization Algorithms for VLSI Synthesis,” Kluwer, 1984.
- [6] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, “Multilevel Logic Synthesis,” *Proceedings of the IEEE*, Vol. 78, No. 2, pp. 264–300, 1990.
- [7] J. Bruck, “Harmonic Analysis of Polynomial Threshold Functions,” *SIAM Journal on Discrete Mathematics*, Vol. 3, No. 2, pp. 168–177, 1990.
- [8] R. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation,” *IEEE Trans. on Computers*, Vol. C-35, No. 6, pp. 677–691, 1986.
- [9] E. Clarke, K. McMillan, X. Zhao, M. Fujita, and J. Yang, “Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping,” *Design Automation Conference*, pp. 54–60, 1993.
- [10] G. De Micheli and M. Sami, “Hardware/Software Co-Design,” Kluwer, 1995.
- [11] M. Gibson and J. Bruck, “Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels,” *Journal of Physical Chemistry A*, No. 104, pp. 1876–1889, 2000.
- [12] D. Gillespie, “Exact Stochastic Simulation of Coupled Chemical Reactions,” *Journal of Physical Chemistry*, No. 81, pp. 2340–2361, 1977.
- [13] R. Heinrich and S. Shuster, “The Regulation of Cellular Systems,” Chapman and Hall, 1996.
- [14] Y. Hong, P. Beerel, J. Burch, and K. McMillan, “Safe BDD Minimization Using Don’t Cares,” *Design Automation Conference*, pp. 208–213, 1997.
- [15] Intel Corporation, Press Release, July 13, 2004.
- [16] J. Jackson, E. Shamir, and C. Schwartzman, “Learning with Queries Corrupted by Classification Noise,” *Discrete Applied Mathematics*, Vol. 92, No. 2-3, pp. 157–175, 1999.
- [17] R. Karp and R. Miller, “Parallel Program Schemata,” *Journal of Computer and System Sciences*, Vol. 3, pp. 147–195, 1969.
- [18] R. Katz, “Contemporary Logic Design,” Benjamin/Cummings, 1992.
- [19] V. Khrapchenko, “Depth and Delay in a Network,” *Soviet Math. Dokl.*, No. 19, pp. 1006–1009, 1978.
- [20] T. Larrabee, “Test Pattern Generation Using Boolean Satisfiability,” *IEEE Trans. on Computer-Aided Design*, Vol. 11, No. 1, pp. 4–15, 1992.
- [21] R. Lechner, “Harmonic Analysis of Switching Functions,” in *Recent Developments in Switching Theory*, A. Mukhopadhyay, Ed., Academic Press, pp. 121–228, 1971.
- [22] C. Lee, “Representation of Switching Circuits by Binary-Decision Programs,” *Bell System Technical Journal*, Vol. 38, pp. 985–999, 1959.
- [23] L. Lok and R. Brent, “Automatic Generation of Cellular Reaction Networks with Molecuizer 1.0,” *Nature Biotechnology*, Vol. 23, No. 1, pp. 131–136, 2005.

- [24] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an Efficient SAT Solver,” *Design Automation Conference*, pp. 530–535, 2001.
- [25] L. Nagel and D. Pederson, “Simulation Program with Integrated Circuit Emphasis,” *Midwest Symposium on Circuit Theory*, 1973.
- [26] A. Olivera, L. Carloni, T. Villa, and A. Sangiovanni-Vincentelli, “Exact Minimization of Binary Decision Diagrams Using Implicit Techniques,” *IEEE Trans. on Computers*, Vol. 47, No. 11, pp. 1282–1296, 1998.
- [27] N. Pippenger, “On Networks of Noisy Gates,” *Symposium on the Foundations of Computer Science*, pp. 30–36, 1985.
- [28] M. Reed and T. Lee, Eds., “Molecular Nanoelectronics,” American Scientific, 2003.
- [29] M. Riedel and J. Bruck, “Cyclic Combinational Circuits: Analysis for Synthesis,” *Int’l Workshop Logic and Synthesis*, pp. 105–112, 2003.
- [30] M. Riedel and J. Bruck, “The Synthesis of Cyclic Combinational Circuits,” *Design Automation Conference*, pp. 163–168, 2003.
- [31] M. Riedel and J. Bruck, “Timing Analysis of Cyclic Combinational Circuits,” *Int’l Workshop Logic and Synthesis*, pp. 105–112, 2004.
- [32] M. Riedel, “Cyclic Combinational Circuits,” Ph.D. Dissertation, California Institute of Technology, 2004, available at <http://www.paradise.caltech.edu/riedel/research>.
- [33] R. Rivest, “The Necessity of Feedback in Minimal Monotone Combinational Circuits,” *IEEE Trans. on Computers*, Vol. C-26, No. 6, pp. 606–607, 1977.
- [34] T. Sasao, J. Butler, and M. Riedel, “Application of LUT Cascades to Numerical Function Generators,” *Workshop on Synthesis and System Integration of Mixed Information Technologies*, 2004.
- [35] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, “SIS: A System for Sequential Circuit Synthesis,” Tech. Rep., UCB/ERL M92/41, Electronics Research Lab, University of California, Berkeley, 1992.
- [36] C. Shannon, “A Symbolic Analysis of Relay and Switching Circuits,” *Trans. of the AIEE*, Vol. 57, pp. 713–723, 1938.
- [37] C. Shannon, “The Synthesis of Two Terminal Switching Circuits,” *Bell System Technical Journal*, Vol. 28, pp. 59–98, 1949.
- [38] L. Stok, “False Loops Through Resource Sharing,” *Int’l Conference Computer-Aided Design*, pp. 345–348, 1992.
- [39] E. Voit, “Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists,” Cambridge Univ. Press, 2000.
- [40] J. von Neumann, “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components,” in *Automata Studies*, C.E. Shannon and J. McCarthy (Eds.), Princeton Univ. Press, pp. 329–378, 1956.
- [41] J. Wakerly, “Digital Design: Principles and Practices,” 3rd. Ed., Prentice-Hall, 2000.