Chemical Reaction Networks and Stochastic Local Search

Erik Winfree[✉]

California Institute of Technology, Pasadena, CA, USA
winfree@caltech.edu

Abstract. Stochastic local search can be an effective method for solving a wide variety of optimization and constraint satisfaction problems. Here I show that some stochastic local search algorithms map naturally to stochastic chemical reaction networks. This connection highlights new ways in which stochasticity in chemical reaction networks can be used for search and thus for finding solutions to problems. The central example is a chemical reaction network construction for solving Boolean formula satisfiability problems. Using an efficient general-purpose stochastic chemical reaction network simulator, I show that direct simulation of the networks proposed here can be *more* efficient, in wall-clock time, than a somewhat outdated but industrial-strength commercial satisfiability solver. While not of use for practical computing, the constructions emphasize that exploiting the stochasticity inherent in chemical reaction network dynamics is not inherently inefficient – and indeed I propose that stochastic local search could be an important aspect of biological computation and should be exploited when engineering future artificial cells.

1 Introduction

Can a cell solve Sudoku? While few would take seriously the prospect of whether an individual *E. coli* could beat the puzzle page of the *Daily Mail*, this question of principle has significant implications. Sudoku, when generalized, is a particularly relatable example of an NP-complete problem, and it has been effectively used to illustrate methods to solve constraint satisfaction problems [1,2] as well as to explore neural computing architectures underlying natural intelligence [3,4]. So our real question is whether 1 cubic micron of biochemistry could efficiently implement the kinds of algorithms necessary to solve difficult problems like Sudoku, and if so, how? An answer to this more general question could be valuable for engineering a future artificial cell that makes the most of its limited computing resources to get by in the world; it may conceivably also provide new perspectives on the principles exploited by naturally evolved cells to solve the problems they encounter during their lives.

We will use the $\mathsf{P} \overset{?}{=} \mathsf{NP}$ question to frame our discussion [5]. Informally, P is the class of problems that are *solvable* in polynomial time (with respect to

C. Thachuk and Y. Liu (Eds.): DNA 25, LNCS 11648, pp. 1–20, 2019.
https://doi.org/10.1007/978-3-030-26807-7_1

the size of the problem instance). A canonical problem in P is the CircuitEvaluation problem, where a problem instance specifies a feedforward Boolean circuit and values for each input wire, and the solution is what the circuit should output on the given input. To solve such an instance requires simply evaluating each gate in order, which requires only polynomial (roughly linear) time. Informally, NP is the class of problems whose solutions are *verifiable* in polynomial time (where a solution now is the answer accompanied by a polynomial-sized certificate that explains why). A canonical problem in NP is the CircuitSatisfiability problem, where a problem instance specifies a feedforward Boolean circuit and values for each output wire, and the solution is whether there exist input values for which the circuit produces the specified output; those input values constitute a certificate that can be verified in polynomial (roughly linear) time. CircuitSatisfiability is effectively the inverse problem for CircuitEvaluation. Problems in NP always can be solved within exponential time by brute-force enumerating all possible certificates, and verifying them one by one – if any certificate checks out, then an answer has been found. The $\mathsf{P} \overset{?}{=} \mathsf{NP}$ question essentially asks whether there are better algorithms for NP problems that do something much more clever and thus are guaranteed to find solutions in polynomial time – if this is possible for all NP problems, then $\mathsf{P} = \mathsf{NP}$. But most computer scientists think the answer is "no": while clever algorithms may reduce the form of the exponential, or may provide polynomial time solutions for a subset of cases, nonetheless for the worst-case hard problem instances of NP problems, no algorithm can guarantee a polynomial time to find a solution.

What interests us here is *not* whether $\mathsf{P} = \mathsf{NP}$ or not, but rather the fact that studying this question has revealed how fundamental and natural the class NP is. Presuming that $\mathsf{P} \neq \mathsf{NP}$, then there are problems (many problems!) that are often enough very hard to solve, yet recognizing that a solution has been found is relatively easy. Problems with this character have substantial impact in the real world. For example, in cryptography, decoding a message is hard without a key, but easy with the key – and this makes internet commerce feasible. But problems with this character can be found far beyond computer science and technology. For example, in academia, a professor managing a large group is feasible only because solving research problems is hard and time-consuming (the graduate student's job) but recognizing that the research problem has been solved (the professor's job) is dramatically easier and less time-consuming. In fact, the professor is able to steer the research group in the direction she wants just by wisely posing a set of problems to be tackled, waiting for the students to solve them (by cleverness, by luck, by standard techniques, by unexpected genius, by brute force, it doesn't matter), and then verifying that each solution is genuine. More generally, hierarchical organization in science, business, and society relies on the distinction between the difficulty of doing a task and the ease of verifying that it has been done properly. The design of organizations that are effective at accomplishing their goals in turn relies on aptly defining the subtasks and the criteria for success.

Let's call this *programming by recognition*: rather than specifying all the details of an algorithm for how to *find* a solution, the programmer just provides a simpler specification for how to *recognize* a solution. This is in fact a central idea for constraint logic programming languages, such as PROLOG and CLP, where the programmer concerns himself with what (easy to check) constraints define the solution that is being sought, and standardized methods are used to find the solution [6]. Programs in such languages often don't have guarantees on run times, but when done carefully, they can often find solutions effectively.

From this perspective, CIRCUITSATISFIABILITY can be viewed as a very low-level constraint logic programming language, and along with more powerful generalizations such as satisfiability modulo theories, Boolean satisfiability solvers are now used routinely in industry [7–10]. While carefully-crafted deterministic algorithms remain the dominant general-purpose methods for solving Boolean constraint problems, for many years a broad class of hard problems were best solved by surprisingly simple stochastic algorithms that perform biased random walks through the space of potential solutions [11–14]. This observation positions stochastic local search as a viable engine to power systems designed using the programming by recognition paradigm.

There is ample evidence for programming by recognition in biology, suggesting that it provides evolutionary advantages as a system architecture. A classic example occurs during Eukaryotic cell division, when it becomes necessary to move one copy of each chromosome to polar opposite sides of the cell. Since the interior of a cell can be chaotic, this poses the problem of how to find each chromosome, prior to pushing them into place. Nature's solution is to grow microtubules bridging pairs of chromosomes, using a "search and capture" approach [15] whereby microtubules grow in all directions but are stabilized only when they recognize their proper target. The search involves stochastic exploration and energetically expensive mechanisms such as "dynamic instability" that alternate fast growth with fast depolymerization, which has been shown to be a more effective algorithm than a passive and energetically-neutral random walk of polymer length [16]. Compared to a hypothetical deterministic molecular algorithm for solving the same problem ("take a left at the endoplasmic reticulum..."), the stochastic "search and capture" algorithm presumably has the advantages that it is simpler to encode genetically, more robust, and thus more easily evolvable [17,18]. Moving to a higher level of biological organization, a second classic example is learning, which often involves a biased random walk through parameter space that recognizes when good performance has been achieved [19,20]. As before, there are advantages: organisms that learn can have smaller genomes, are more robust, and can evolve faster than organisms with hard-coded circuity [21]. Indeed, evolution itself exhibits some characteristics of programming by recognition – if survival is the ultimate form of recognition.

Our interest here is to explore programming by recognition as a paradigm for engineering circuits within artificial cells that exploit the natural stochasticity of molecular interactions to provide efficient solutions to hard problems that the cell might encounter. For simplicity, we ignore the geometric factors present in the motivating example of cell division and instead focus on well-mixed chemical reaction networks involving a finite number of species in a finite volume. This choice is attractive not only for its mathematical simplicity, but also because it can be considered a programming language for engineering molecular systems using dynamic DNA nanotechnology [22–25].

2 Stochastic Chemical Reaction Networks

We will use the standard model for formal chemical reaction networks (CRNs) with discrete stochastic semantics for mass-action kinetics [26]. A CRN is specified by a finite set of species, e.g. $\{A, B, C, \ldots, X, Y, Z\}$ and a finite set of reactions, e.g. $\{A + B \xrightarrow{k_1} B, X + A \xrightarrow{k_2} C, Y \xrightarrow{k_3} A + C, A \xrightarrow{k_4} B\}$, where k_i are the stochastic rate constants with respect to an implicitly given reaction volume. The state of the CRN at time t is given by the discrete counts of each species, which we will refer to (with a mild abuse of notation) as A_t, B_t, etc. The propensity ρ_i of a reaction i at time t gives the instantaneous rate of firing for that reaction, and may be calculated as the product of k_i and the current counts of each reactant (if all reaction stoichiometries are 1), e.g. $\rho_1(t) = k_1 A_t B_t$ in the example above. The total propensity $\rho = \sum_i \rho_i$ is the sum over all reactions. This implicitly defines a continuous time Markov chain (CTMC) with an infinite state space (although, depending on the CRN and initial state, only a finite subset may be reachable). In the standard stochastic simulation algorithm (SSA), the time at which the next reaction occurs is chosen from an exponential distribution with mean $1/\rho$, and at this time, the reaction that occurs is reaction i with probability ρ_i/ρ. Simulation continues until a fixed time is exceeded, or a state is reached where $\rho = 0$.

Since our central thesis is that stochastic chemical kinetics allow surprisingly effective problem-solving, it is worth reviewing how stochastic chemical kinetics (appropriate for a finite volume containing discrete counts of each species) differs from the more familiar deterministic chemical kinetics (which models a hypothetical well-mixed but infinite volume where real-valued concentrations are the appropriate measure and ordinary differential equations provide the appropriate dynamics). I will offer six perspectives.

Noisy behavior. The classical view is that stochastic CRNs are just noisy versions of deterministic CRNs. There are senses in which this is true. For non-explosive CRNs [27], if the volume V of a stochastic CRN is scaled up while keeping the initial concentrations of each species constant, then for chosen time t, the concentrations in the stochastic model approach the concentrations in the deterministic model [28]. This is a Central Limit Theorem result, that is, X_t approaches a Gaussian with coefficient of variation shrinking with V. But it is not a uniform convergence: to ensure that the stochastic CRN's concentrations are likely within a certain percentage of the deterministic CRN, V may have to grow with t (depending on the specific CRN).

Extinction and reachability. For a fixed volume, stochastic CRN behavior can differ qualitatively from deterministic behavior. Most notably, species (or reactions) may become extinct in the stochastic CRN despite being active forever in the deterministic CRN. This occurs in the classical predator-prey oscillator, $\{R \to 2R, F + R \to 2F, F \to \emptyset\}$. More generally, stochastic CRNs can be limited by discrete state-space reachability constraints, reflecting deep connections to computability theory [29–31] that are not present in the continuous deterministic state space.

Perfect Boolean values. The hard distinction between *none* and *some* that is inherent to stochastic CRNs can make deterministic computation easier than in deterministic CRNs. For example, implementing feedforward Boolean logic circuit computations is straightforward with stochastic CRNs [22], where signals on wires can be represented by counts of 0 or 1 for specific species. But when signals are represented as high or low real-valued concentrations in deterministic CRNs, error-correcting non-linearities are needed to perform signal restoration [32].

Computing with counts. The count for a single chemical species can store the value of an arbitrary non-negative integer, giving stochastic CRNs the ability to perform uniform computation – in the sense that a single computing machine can process arbitrarily large input instances [33–36]. Turing-universal computation is possible with vanishingly small probability of error.

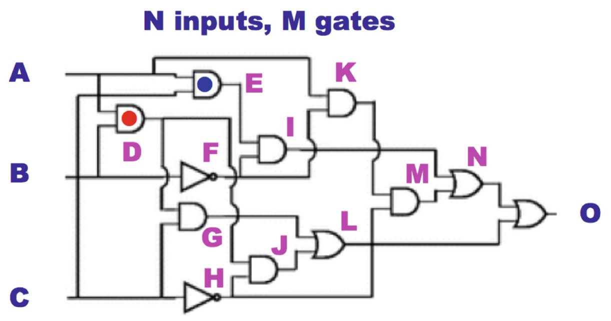
Computing with distributions. Probabilistic behavior and probabilistic reasoning is an adaptive response for living in a world where partial information is available. Unlike deterministic CRNs, where probabilities must be represented as concentrations [37,38], stochastic CRNs have the potential to directly represent probabilities via probabilities [39–41]. Although stochastic CRNs that satisfy detailed balance can only produce equilibrium distributions that are products of Poisson distributions [42], constraints of state-space reachability can shape marginals to approximate arbitrary distributions, as can non-equilibrium steady-states in CRNs that do not satisfy detailed balance [43].

That was only five. The sixth perspective – **that stochastic CRNs inherently perform stochastic local search** – is developed in the rest of this paper.

3 Evaluating and Satisfying Circuits

We begin by reviewing how stochastic CRNs can efficiently solve CircuitEvaluation problems, using methods similar those in work cited above. Given a feedforward Boolean circuit that uses 2-input unbounded fan-out gates, our goal is to construct a stochastic CRN that, presented with molecules representing any input to the circuit, will produce molecules representing the output of the circuit, and then all reactions will become extinct. Specifically, for a circuit c with N inputs and M gates, the CRN CircuitEvaluationCRN$[c]$ will employ $2(N + M)$ species and $4M$ reactions, as illustrated in Fig. 1(A) and as follows:

(A)



Species:	**A0, A1, B0, B1, C0, C1, D0, D1, E0, E1, F0, F1, G0, G1, H0, H1, I0, I1, J0, J1, K0, K1, L0, L1, M0, M1, N0, N1, O0, O1**
Initial conditions:	**1 molecule each for each variable, e.g. A1, B0, C0, D1, etc... Set the inputs to the desired values, let the others be arbitrary**
Reactions:	

$A0 + B0 + D1 \rightarrow A0 + B0 + D0$	$A0 + C0 + E1 \rightarrow A0 + C0 + E0$
$A0 + B1 + D1 \rightarrow A0 + B1 + D0$	$A0 + C1 + E1 \rightarrow A0 + C1 + E0$
$A1 + B0 + D1 \rightarrow A1 + B0 + D0$	$A1 + C0 + E1 \rightarrow A1 + C0 + E0$
$A1 + B1 + D0 \rightarrow A1 + B1 + D1$	$A1 + C1 + E0 \rightarrow A1 + C1 + E1$
...	...

(B)

$A0 + B0 + D1 \rightarrow A0 + B0 + D0$	$A0 + B0 + D1 \rightarrow A0 + B1 + D1$	$A0 + B0 + D1 \rightarrow A1 + B0 + D1$
$A0 + B1 + D1 \rightarrow A0 + B1 + D0$	$A0 + B1 + D1 \rightarrow A0 + B0 + D1$	$A0 + B1 + D1 \rightarrow A1 + B1 + D1$
$A1 + B0 + D1 \rightarrow A1 + B0 + D0$	$A1 + B0 + D1 \rightarrow A1 + B1 + D1$	$A1 + B0 + D1 \rightarrow A0 + B0 + D1$
$A1 + B1 + D0 \rightarrow A1 + B1 + D1$	$A1 + B1 + D0 \rightarrow A1 + B0 + D0$	$A1 + B1 + D0 \rightarrow A0 + B1 + D0$
...	$O1 \rightarrow O0$	...

Fig. 1. (A) Evaluating feedforward Boolean circuits with stochastic CRNs. Red reactions correspond to the gate with the red dot; blue reactions correspond to the gate with the blue dot. **(B) Solving Circuit-SAT with stochastic CRNs.** Red reactions correspond to the gate with the red dot. Clamping reactions set the output to the desired value. In this example, $N = 3$ and $M = 12$. (Color figure online)

- Each wire in the circuit is named, and for each wire X we use two species, $X0$ and $X1$.
- Initially, and for all time thereafter, there will be one molecule per wire, i.e. $X0_t + X1_t = 1$.
- For each gate $Z = g(X, Y)$ where g is a Boolean function, the four reactions are

$$\begin{aligned}
X0 + Y0 + Z(1-v) &\rightarrow X0 + Y0 + Z(v) && \text{with } v = g(0,0)\\
X0 + Y1 + Z(1-v) &\rightarrow X0 + Y1 + Z(v) && \text{with } v = g(0,1)\\
X1 + Y0 + Z(1-v) &\rightarrow X1 + Y0 + Z(v) && \text{with } v = g(1,0)\\
X1 + Y1 + Z(1-v) &\rightarrow X1 + Y1 + Z(v) && \text{with } v = g(1,1)
\end{aligned}$$

where $Z(0) \equiv Z0$, $Z(1) \equiv Z1$, and all rate constants are identical, say k.

In other words, for each input case for the gate, there are reactions that catalytically convert an incorrect output species into a correct output species. It will be convenient to associate full states of the CRN, in which there are exactly one molecule for each wire, with assignments of Boolean values to each wire, in the obvious way. We can generalize this to partial assignments, so that e.g. for circuit input x, $\textsc{Wires}[x]$ refers to any full state of the CRN such that the species representing input wires have the specified values, while other species have arbitrary counts consistent with the one-molecule-per wire constraint. Similarly, for circuit output y, $\textsc{Wires}[y]$ refers to any full state of the CRN such that the output species have the specified values, and other species are unconstrained.

Theorem 1 (Feedforward circuit evaluation). *For feedforward Boolean circuit c, input x, and stochastic CRN* $\textsc{CircuitEvaluation}\text{CRN}[c]$*, any initial state* $\textsc{Wires}[x]$ *will evolve to a state* $\textsc{Wires}[c(x)]$ *at which point all reactions will be extinct.*

Proof. Since c is feedforward, we can order the gates such that their inputs are always either prior gates or inputs to the circuit. Because each reaction is catalytic in the gate inputs, once all wires prior to a gate are correct, they will never change again. That gate will become correct with expected time $1/k$. Thus, all M wires will be correct within expected time $O(M/k)$. □

Feedforward circuits compute from input to output in polynomial time. To reverse the information flow from output to input, thus solving a circuit satisfaction problem, requires some guesswork or brute-force enumeration, and (if $\mathsf{P} \neq \mathsf{NP}$) requires more than polynomial time, e.g. exponential time. Surprisingly, the stochastic CRN for solving this problem is not significantly larger, in terms of the number of reactions.

The idea remains that each CRN reaction will detect a violation of circuit correctness, such that if all reactions go extinct, then the circuit represents a valid solution to the problem. There are two new ingredients. First, unlike how we set the inputs for feedforward evaluation using the initial state, here we will allow arbitrary initial state, but treat the output species having the wrong value as a violation of circuit correctness. Therefore, we introduce clamping reactions that detect if the output is wrong, and if so, fix it. E.g. if output wire Y should be 0, we include a reaction $Y1 \rightarrow Y0$. Second, when a gate's output is inconsistent with its input, we no longer know whether the problem is that the output is incorrect, or the input is incorrect – so we also include reactions that detect the gate's inconsistency and change one of the inputs, instead of changing the output. These reactions are illustrated in Fig. 1(B). Which reaction fires first is random, according to stochastic CRN semantics. Altogether, for circuit c and output y, this construction results in a circuit $\textsc{Circuit}\text{SATCRN}[c, y]$ with $2(N + M)$ species and $12M + L$ reactions, where there are L output wires. Note that the circuit no longer needs to be feedforward, and the clamped target values do not need to be output wires; our comments below can be trivially extended to this generalization.

Theorem 2 (Solving circuit satisfiability). *For satisfiable Boolean circuit c, output y, and stochastic CRN* CircuitSATCRN$[c, y]$*, any initial state will evolve to a state* Wires$[x, y]$ *where $y = c(x)$, at which point all reactions will be extinct.*

Proof. Since the set of possible local violations of correctness correspond exactly to the set of reaction reactants, it follows that if all reactions go extinct, then a solution has been found. Similarly, if the current CRN state does not correspond to a solution, then at least one reaction in the CRN will be poised to fire. What remains to be shown is that, from any initial state, a state representing a valid solution can be reached. Let s be a CRN state corresponding to a valid solution, and let s' be the current state, which is not a valid solution. Suppose s' deviates from s in n wires. If s' is not valid because of an incorrect output wire, then we can take a reaction that corrects the output wire, leading to a state s'' that has $n-1$ deviations from s. If s' is not valid because of an incorrect gate, then either an input or the output of that gate deviates from s. One of the 12 reactions for that gate corrects this deviation, leading to a state s'' that has $n-1$ deviations from s. Either way, if s'' is not itself a valid solution, then we recursively show that there is a sequence of reactions that leads either to s or to another valid solution. $\square$

Thus, when CircuitSATCRN$[c, y]$ is started in any one-molecule-per-wire state, it will eventually find a solution x if one exists. However, if no solution exists – i.e. the circuit is *not* satisfiable – then the CRN will continue exploring the state space forever. And even when a solution does exist, we have no useful bound on how long the CRN will take to find it[1].

4 Formula Satisfiability

The computer science community has converged not on circuit satisfiability problems, but on formula satisfiability problems as the standard for Boolean logic constraint solvers. Circuit satisfiability problems can be translated to formula satisfiability problems with little overhead; in fact, even when limited to clauses with no more than three literals each – i.e. 3-SAT – the problem is still NP-complete. Such simple building blocks facilitate analysis and constructions. Figure 2(A) illustrates how to construct stochastic CRN FormulaSATCRN$[f]$ that solves 3-SAT formula f using the same principles as for circuit satisfiability, but now using just $2N$ species and $3M$ reactions, where f has N variables and M clauses. A sample run is shown in Fig. 2(B). This CRN has similar merits and demerits as the circuit satisfiability CRN: in theory, it is guaranteed to eventually find a solution if one exists; in practice, you're not likely to have the patience to watch it try to solve a hard problem with 100 variables.

[1] The sequence of reactions identified in the proof will have a positive probability of occurring next, specifically, at least $(12M+L)^{-(N+M)}$. This provides an exponential bound on the expected time to halt, to wit, less than $(12M+L)^{N+M}/k+(N+M)/k$. But is that useful?

(A)

Given: a boolean formula

(A or L or N) and (A or !K or !N) and (!J or !P or !R) and (Q or E or I) and (D or !A or !K) and (B or T or C) and (O or I or !Q) and (G or !O or !Q) and (K or D or F) and (T or !M or !B) and (M or !S or !B) and (R or C or !T) and (R or !P or !C) and (S or F or !D) and (L or I or !E) and (I or !L or !E) and (M or S or !F) and (H or !O or !G) and (P or !J or !H) and (J or !G or !H) and (L or E or !I) and (E or !L or !I) and (L or !E or !I) and (!L or !E or !I)

Construct: a chemical reaction network that stops when it finds a solution to the formula

Species: **A0, A1, B0, B1, C0, C1, D0, D1, E0, E1, F0, F1, G0, G1, H0, H1, I0, I1, J0, J1, K0, K1, L0, L1, M0, M1, N0, N1, O0, O1, P0, P1, Q0, Q1, R0, R1, S0, S1, T0, T1**

Initial conditions: **1 molecule each for each variable, e.g. A1, B0, C0, D1, etc...**

Reactions:

$$A0 + L0 + N0 \rightarrow A1 + L0 + N0 \qquad A0 + K1 + N1 \rightarrow A1 + K1 + N1$$
$$A0 + L0 + N0 \rightarrow A0 + L1 + N0 \qquad A0 + K1 + N1 \rightarrow A0 + K0 + N1$$
$$A0 + L0 + N0 \rightarrow A0 + L0 + N1 \qquad A0 + K1 + N1 \rightarrow A0 + K1 + N0$$

$$J1 + P1 + R1 \rightarrow J0 + P1 + R1$$
$$J1 + P1 + R1 \rightarrow J1 + P0 + R1 \qquad \ldots$$
$$J1 + P1 + R1 \rightarrow J1 + P1 + R0$$

(B)

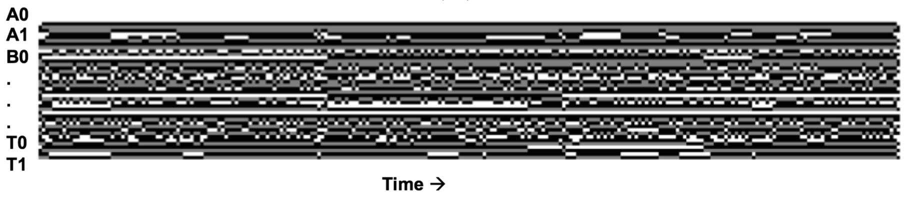


Fig. 2. (A) Solving 3-SAT problems with stochastic CRNs. Red reactions correspond to the red clause; blue reactions correspond to the blue clause; green reaction correspond to the green clause. The ($N = 20, M = 24$) formula illustrated here was once solved on a DNA computer [44], although variables have been renamed. **(B) Space-time history of a 3-SAT run.** Species are arranged vertically; each column of 40 pixels corresponds to the state of the CRN before or after each reaction fires; black if the species count is 0, grey if the species count is 1 but that disagrees with the solution eventually found, white if the species count is 1 and it agrees with the solution eventually found. At the end of time, the CRN has gone extinct. (Color figure online)

Why is our formula satisfiability CRN so ineffective? One intuition is that when a clause is not satisfied, there are three possible ways to try to fix it – flip one of the three variables – but some of those actions will cause other clauses to break. The CRN makes no distinctions, so in a situation where there is one way to make things better, and two ways to make things worse, it is more likely to make things worse. Thus, as the CRN performs a stochastic local search of the state space, the projection onto the number of conflicted clauses does a random walk that hovers around some average number, making occasional excursions toward more or toward fewer.

Similar problems have been encountered in conventional SAT-solvers based on stochastic local search, and effective – though heuristic – countermeasures have been devised [11–14]. The basic idea is to bias the choice of clause and variable to flip so as to favor moves that reduce the number of conflicts. A particularly simple and effective incarnation of this idea is WalkSAT [12]. The core algorithm is just this:

1. Start with a random assignment of values to variables.
2. At random choose an unsatisfied clause.
3. Flip a variable from that clause, thus making it satisfied, such that the fewest other clauses become unsatisfied.
4. With some low probability, flip a random variable.
5. If the formula is not satisfied, go back to step 2.

Variants of this algorithm dominated the "random" category in the international SAT competitions for over a decade, until 2014 [9]. (For formulas generated randomly with a certain ratio of clauses to variables, $\alpha = M/N$, there is a critical value of $\alpha \approx 4.26$ below which problems are likely to be satisfiable and "easy" to solve, and above which problems are likely to be unsatisfiable; near the threshold, problems become quite hard [14,45,46].)

A similar kind of bias can be introduced into our CRN SAT-solver, yielding a new construction WalkSATCRN$[f]$, which now has $4N$ species and $4M + 2N$ reactions. What seems simplest to implement in a CRN is to reject attempts to flip bits that, if changed, would cause many clauses to become unsatisfied. Our specific construction is illustrated in Fig. 3(A). If there is an unsatisfied clause, e.g. $(A$ or L or $N)$, then the corresponding species will all attempt to flip, e.g. via the reaction $A0{+}L0{+}N0 \rightarrow tryA1{+}tryL1{+}tryN1$. While they are trying to flip, these variables cannot be involved in additional conflict detection events, thus somewhat limiting the number of variables that simultaneously try to flip. However, at the beginning of the run, a "wound area" of variables trying to flip will quickly emerge, possibly until all clauses either have some flipping variable or are already satisfied. Now there is a competition for "healing" the wound: at a slow rate, variables trying to flip will successfully solidify their choice via reactions such as $tryA1 \rightarrow A1$; while at a faster rate flips will be rejected if solidifying would have introduced a conflict, e.g. via the rejection reaction $D0 + tryA1 + K1 \rightarrow D0 + A0 + K1$ associated with clause $(D$ or $!A$ or $!K)$. Thus, the healing wound will result in changed values preferentially for variables that introduce no new conflicts, or few of them – with the preference being more strongly against changes that introduce more conflicts, because the rate of rejecting an attempted flip will be proportional to the number of reactions trying to reject it. Nonetheless, occasionally changes will increase the number of conflicts, since which reaction occurs next is probabilistic in the SSA.

Whereas a simple argument sufficed to show that CircuitSATCRN$[c, y]$ and FormulaSATCRN$[f]$ halt with a valid solution if and only if their problem is satisfiable, the fact that WalkSATCRN$[f]$ attempts to flip three variables simultaneously has so far confounded my attempts to prove its correctness[2]. Moreover, we have no theoretical guarantees for the effectiveness of the stochastic local search bias for finding solutions quickly.

Therefore, without further analysis and sticking with the first fast-to-slow reaction rate ratio that seemed to work OK, we evaluated the effectiveness of WalkSATCRN[f] on random satisfiable formulas near the critical threshold. Formulas with 100 variables were reliably solved in just a few seconds, while formulas with 500 variables were reliably solved in under an hour ($10^{3.56}$ seconds) and typically much faster. In no case did we encounter a satisfiable formula that the CRN failed solve. Several comments are in order. First, how fast the CRN runs, in wall-clock time, depends on what CRN simulator you use. Second, it would be useful to have a point of comparison for how hard it is to solve the random formula instances.

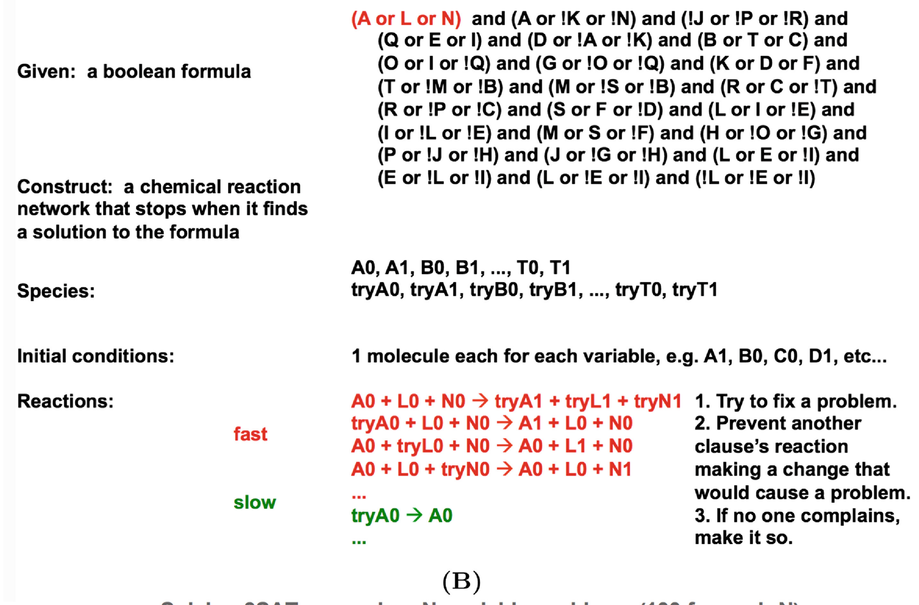
For the comparison, we used Mathematica's built-in command **SatisfiableQ**, which makes use of the MiniSAT 1.4 library [47]. MiniSAT is a deterministic algorithm that can both solve satisfiable problems as well as declare problems to be unsatisfiable. MiniSAT and its variants have been perennial winners in the international SAT competition, although in recent competitions the improved MiniSAT 2.2 has merely been the baseline that the winning algorithms soundly surpass [9].

Regarding the simulator, I used a general-purpose Mathematica-based CRN simulator originally developed by Soloveichik [48] that I extended to be more efficient on large CRNs by using data structures similar to those in prior work [49–51]. Specifically, the simulator pre-computes a fixed binary tree of reactions in which all reaction propensities are stored, along with a list Δ_i, for each reaction i, indicating which other reactions' propensities will be changed when reaction i fires – i.e. those whose reactants have their counts changed by the given reaction. Thus, for a CRN with R reactions, each step of the SSA involves $\lg R$ choices through the binary tree to navigate to the selected reaction i, followed by recalculation of the propensities of only the reactions in Δ_i, followed by $|\Delta_i| \lg R$ hierarchical updates of propensities within the binary tree. This avoids much of the redundant calculations in naive implementations of SSA. For further speed, the inner loop of the SSA uses Mathematica's **Compile** function, which effectively compiles to C. Thus, the CRN simulator we use here is reasonably fast, but has no optimizations that are specific to the SAT-solving CRN constructions – it is a general-purpose CRN simulator.

Figure 3(B) compares the wall-clock time for running the CRN simulator on WalkSATCRN[f] for random hard formulas f (selected by **SatisfiableQ** to be satisfiable) versus the time taken by **SatisfiableQ** itself to solve the same problem. For a given problem size (i.e. dot color, labeled by N for $M = 4.2N$),

[2] A straightforward adaptation of the previous argument works for a closely related CRN that is identical to WalkSATCRN[f] except that species $tryX0$ and $tryX1$ are conflated as $tryX$ for each variable X. This CRN should work similarly, as the main difference is merely that a variable being flipped now might spontaneously revert. But it is not the CRN that I simulated.

(A)



Given: a boolean formula

(A or L or N) and (A or !K or !N) and (!J or !P or !R) and (Q or E or I) and (D or !A or !K) and (B or T or C) and (O or I or !Q) and (G or !O or !Q) and (K or D or F) and (T or !M or !B) and (M or !S or !B) and (R or C or !T) and (R or !P or !C) and (S or F or !D) and (L or I or !E) and (I or !L or !E) and (M or S or !F) and (H or !O or !G) and (P or !J or !H) and (J or !G or !H) and (L or E or !I) and (E or !L or !I) and (L or !E or !I) and (!L or !E or !I)

Construct: a chemical reaction network that stops when it finds a solution to the formula

Species:

A0, A1, B0, B1, ..., T0, T1
tryA0, tryA1, tryB0, tryB1, ..., tryT0, tryT1

Initial conditions: 1 molecule each for each variable, e.g. A1, B0, C0, D1, etc...

Reactions:

fast

A0 + L0 + N0 → tryA1 + tryL1 + tryN1
tryA0 + L0 + N0 → A1 + L0 + N0
A0 + tryL0 + N0 → A0 + L1 + N0
A0 + L0 + tryN0 → A0 + L0 + N1
...

slow

tryA0 → A0
...

1. Try to fix a problem.
2. Prevent another clause's reaction making a change that would cause a problem.
3. If no one complains, make it so.

(B)

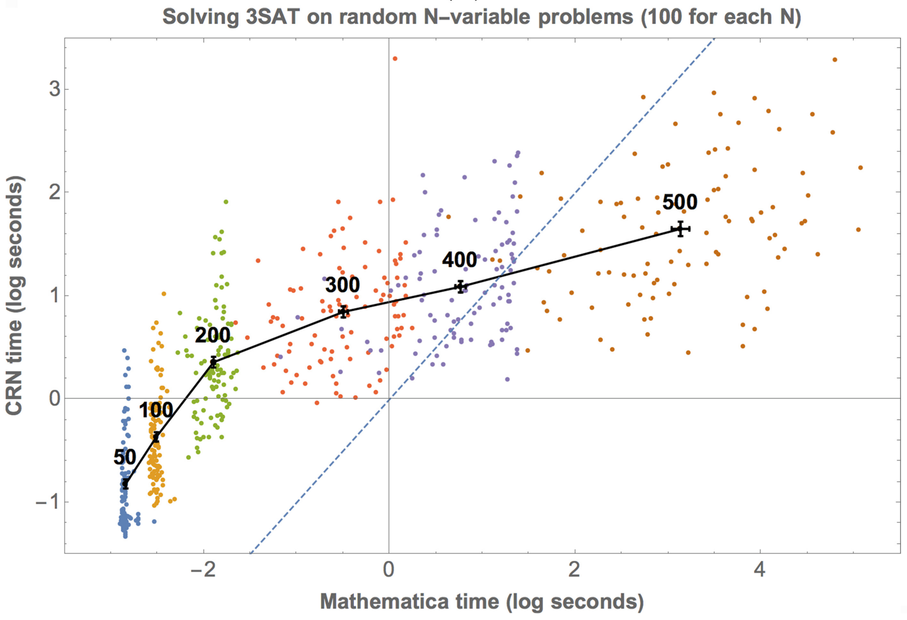


Fig. 3. (A) A WalkSAT-inspired CRN for solving 3-SAT problems with stochastic CRNs. Fast reactions have rate constant 1.0, while slow reactions have rate constant 0.1. **(B) Wall-clock time comparison for solving random 3-SAT problems.** Each dot represents a different random 3-SAT formula with $M = 4.2N$ and the indicated number of variables. Times are reported by their logarithm base 10. Blue dashed line indicates where Mathematica's **SatisfiableQ** takes the same time as simulating the CRN. A 2.6 GHz Macbook Pro (2013) was used. (Color figure online)

there is considerable scatter in the times taken to solve different instances. The vertical scatter (the CRN time) is due not only to the variability in the problem instance, but also due to variability in the CRN's stochastic local search process – solving the same large instance multiple times reveals a timing standard deviation nearly equal to the mean. (In contrast, Mathematica's solver exhibited only 3% variation when re-solving the same problem, which was presumably due to background processes in the operating system.) The CRN simulator clearly has a substantial overhead: for small ($N = 50$) problems, the average CRN time is about a third of a second, while Mathematica solves most problems in little more than a millisecond. Despite this, the CRN becomes more efficient for larger problems, eventually out-performing Mathematica on the largest ($N = 500$) instances. Here, the CRN is taking on average about 5 minutes, while Mathematica is averaging at over two hours. For this N, the CRN has roughly 10,000 reactions.

Today's best SAT-solvers can beat random 3-SAT problems with $\alpha = 4.267$ and $N > 5000$ in under an hour [9]. I have not tried it, but I don't presume the WalkSATCRN[f] simulation would be anywhere close to being competitive. The take-home message is not that simulating a CRN is a good way to solve SAT problems – I don't think it is – but rather that stochastic local search comes fairly naturally to CRNs, and they are not incredibly bad at it. For a given SAT problem, the size of the CRN is linear in the number of variables and clauses (specifically, $4M + 2N$) and the volume required to store one molecule per wire is just $O(N)$. (Note, however, that for a DNA strand displacement implementation, one DNA fuel complex would be required for each time a reaction fires, entailing a ridiculously unrealistic overhead since solving SAT problems of the scale shown in Fig. 3(B) involves millions or billions of reaction events. For fuel-efficient SAT-solving CRNs, see the beautiful papers by Thachuk and colleagues [52–54]; their CRNs effectively implement a brute-force enumeration method, but using reversible reactions within polynomial volume.)

5 Recognizing and Generating Patterns

We are tempted to think of SAT-solving by stochastic local search as a general-purpose mechanism for programming by recognition in molecular systems. The hard SAT instances may not represent the most interesting or useful cases; the value might be in the flexibility and robustness of the computing paradigm.

Consider the problem of making inferences about complex environmental signals. Formulas, circuits, or most generally, networks of relations [55] can be used to represent knowledge about a domain in terms of constraints. In the absence of information from the environment, there may be an enormous number of possible solutions; perhaps we only need one. When additional information is available, in the form of certain variables being True or False, all we need are clamping reactions (such as $A1 \rightarrow A0$ if the environment dictates that A is False), and the stochastic local search will be forced to find a solution that also satisfies the environmental constraints. Depending on the structure of the network and which variables are clamped, this may correspond to evaluating a circuit in the feedforward direction (easy!) or solving a circuit satisfiability problem (hard!) or something in between. It is a general form for potentially omnidirectional computing in CRNs by clamping. (There are striking similarities to the Chemical Boltzmann Machine [40], which allows for similarly omnidirectional inferences to be made by clamping species representing known information, but in that work the inference is probabilistic and computes conditional probabilities.)

A classic paradigm for omnidirectional computation is the associative recall of memories in the Hopfield model [56]. The task here is to store a finite set of "memories" (i.e. binary vectors) into the weights of a neural network, such that when the network is initialized with a test vector, the dynamics of neural updates will bring the system to the known memory that is "closest" (in some sense) and halt there. For example, suppose the memories are 17×17 binary images of Alan Turing, George Boole, Ludwig Boltzmann, Rosalind Franklin, Ada Lovelace, and Warren McCulloch. If the network can "see" only the top half of an image – i.e. the neurons corresponding to those pixels are clamped to the correct values for one of the memories, while the rest of the neurons continue to be updated – then it will reconstruct the rest of the image by "associative recall". It can do the same if it sees any sufficiently large subset of pixels. If the pixels that it sees are partially corrupted, then when the clamping is released, the errors will be corrected and the stored memory established – at least most of the time, with some caveats, and if not too many memories are stored.

A very similar associative memory task can be accomplished by a Boolean SAT solver performing stochastic local search. To "store" the memories, one needs to construct a formula (or circuit) that has several valid solutions – one for each memory. Now, if enough variables are clamped such that only one memory is compatible with the clamped variables, the SAT solver is guaranteed to find that unique solution, thus performing associative recall. If, instead, the variables are initialized (but not clamped) to a pattern similar to one of the memorized patterns, then the stochastic local search is likely to reach that solution first. We demonstrate this idea by constructing a formula whose only solutions are exactly those shown in Fig. 4(A), using the Exclusion Network approach [55]. Specifically, we randomly choose n triples of variables, for each triple make a subformula that allows only the combinations that occur in the set of target memories, and create a formula that is the conjunction of all n subformulas. This formula is guaranteed to have the target memories as valid solutions, and as n increases, it excludes more and more alternative solutions (if there are not too many target memories). After algebraically converting to conjunctive normal form for 3-SAT, we can build the CRN SAT-solver.

The constraints imposed by a SAT formula may be used to define not only pattern recognition processes, but also pattern generation processes. To illustrate the use of SAT solving for a morphogenesis application, we consider a case where, rather than having a unique solution or a small number of chosen solutions as in the associative memory discussed above, the SAT constraints define a combinatorial set of solutions with meaningful properties in common. Figure 4(B) shows several solutions of a SAT formula that imposes only local restrictions that

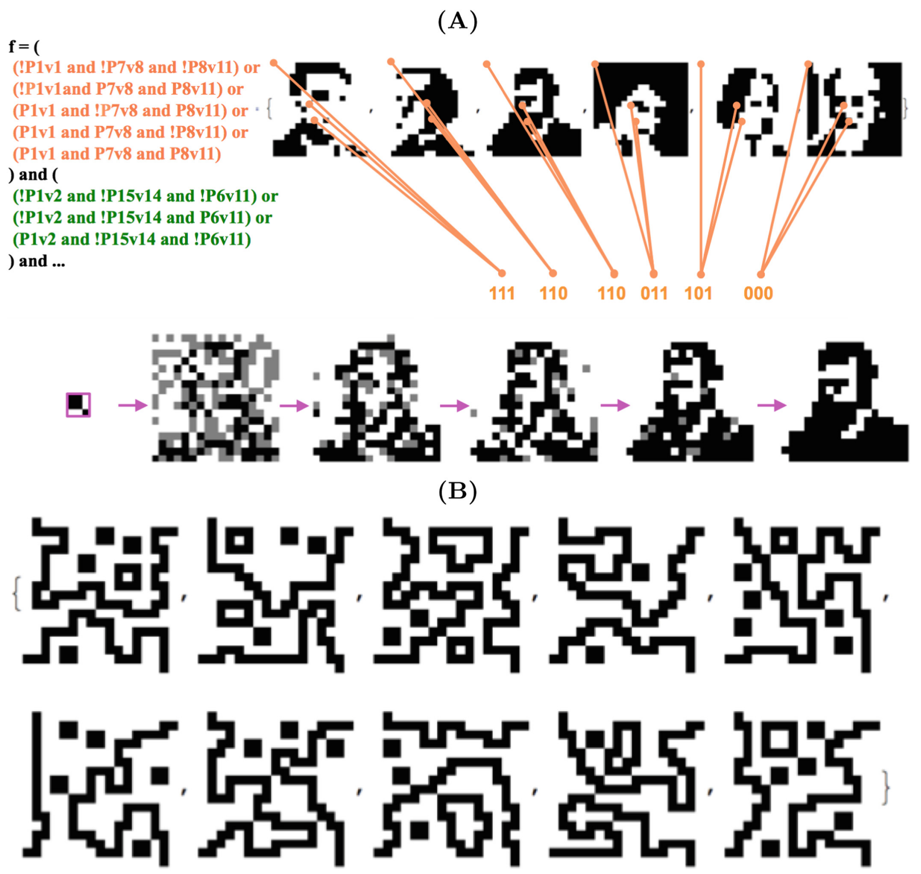
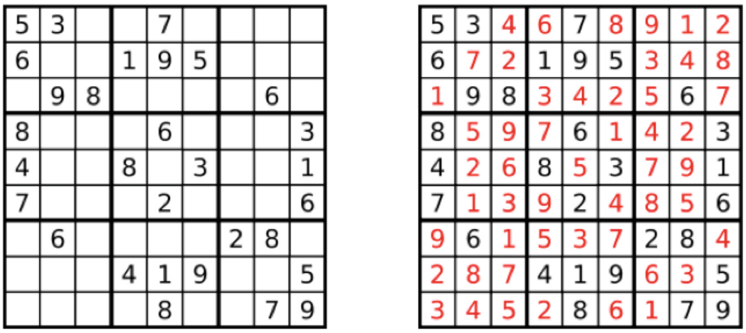

(A)
f = (
(!P1v1 and !P7v8 and !P8v11) or
(!P1v1and P7v8 and P8v11) or
(P1v1 and !P7v8 and P8v11) or
(P1v1 and P7v8 and !P8v11) or
(P1v1 and P7v8 and P8v11)
) and (
(!P1v2 and !P15v14 and !P6v11) or
(!P1v2 and !P15v14 and P6v11) or
(P1v2 and !P15v14 and !P6v11)
) and ...
111
110
110
011
101
000
(B)

Fig. 4. (A) The WalkSAT-inspired CRN performing as an associative memory. Top: The principle for constructing a Boolean formula encoding the memories. Bottom: Clamping a 9-pixel portion of Boltzmann leads to recall of Boltzmann. Grey pixels are trying to flip. **(B) The WalkSAT-inspired CRN maintaining a membrane-like structure.** The Boolean formula only enforces that solutions have black in the corners and all interior cells are either white or have exactly two immediate neighbors that are black. Ten representative solutions are shown.

amount to insisting that black pixels form uninterrupted lines that connect to the corners. Thus, these patterns are fixed points for the SAT-solving CRN, but if disturbed by external perturbations – or additional internal constraints – the CRN will perform stochastic local search to re-establish the connectivity of the membrane-like patterns. Here we see that stochastic local search simultaneously forms the pattern and provides a self-healing capability.

These examples are intended to highlight the connection between stochastic local search SAT solvers and more biological notions of memory, inference, morphogenesis, self-healing, homeostasis, and robustness – how a biological organism self-regulates to restore and maintain critical structures, concentrations, and functions [57].



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Fig. 5. Sudoku. Left: The puzzle instance. Given the black digits, the remaining cells must be filled in such that every 3×3 square has all digits $1 \dots 9$, as does every row and every column. Right: The solution to this puzzle, in red. (Color figure online)

6 Sudoku

This paper wouldn't be complete without a return to the original question of whether a cell could solve Sudoku. To take a first stab at answering that question, note that the constraints of Sudoku can be expressed as a SAT formula [1,2]. Unfortunately, at least for the approach I used, the resulting formulas could not be easily solved by WalkSATCRN[f] simulations. This is perhaps not too surprising; while the classical WalkSAT algorithm is effective for hard random problems and many easy problems, on most "structured" problems, deterministic algorithms that perform clever depth-first search, like MiniSAT, perform much better [14].

Can stochastic CRNs also perform efficient depth-first search? It would be easier – and nearly as effective – to repeatedly take random dives through the search tree to a leaf, rather than to deterministically traverse each node exactly once, which would require extra bookkeeping. Such a stochastic depth-first search would bear some similarity to dynamic instability in microtubules' search for chromosomes [15,16]: the microtubule quickly grows in a random direction, making random choices as it goes; if it is unsuccessful finding a chromosome, it eventually suffers a "catastophe" and rapidly contracts, then starts over... until it finds a chromosome.

Based on this vision, we can construct a stochastic CRN to solve an arbitrary Sudoku puzzle (Fig. 5). There are 9^3 species whose presence indicates that a specific digit may possibly be in a specific cell; another 9^3 that indicate it is definitely not; another 9^2 that indicate that a cell's value is known (i.e. only one digit may be there); more that indicate it is unknown; and some additional bookkeeping species, including *Forward* and *Reverse*. When *Forward* is present, a set of reactions quickly enforce logical constraints among the species; more slowly, a cell with few options will spontaneously choose one, thus descending the search tree. If a contradiction is noted, a reaction will convert *Forward* to *Reverse*, and the logic bookkeeping will dissolve... to be rebuilt when *Reverse* switches to

Forward again. The CRN has roughly 35, 000 reactions, and solves all available Sudoku problems in under half an hour, including the hardest ones on Gordon Royle's list [58].

The number of reactions in the Sudoku CRN is within an order of magnitude of existing whole-cell models of bacteria [59]. So maybe we could conceive of a bacterial-sized artificial cell that implemented that many reactions. However, successfully solving hard Sudoku puzzles involves many dives into the search tree and many millions or billions of reaction events. Even assuming rate constants on the order of 1 per second, that would take many days or years. *E. coli* reproduces in 20 min. So no, it seems that cells are unlikely to be successful solving Sudoku.

7 Discussion

Nonetheless, we may have learned something during this somewhat stochastic exploration of ideas. Foremost in my mind is that the stochasticity inherent in CRNs provides a natural engine for stochastic local search and thus programming by recognition – the hallmark algorithmic architecture defining NP problems. The architecture is robust and flexible, pivoting seamlessly from efficient solution of easy problems to relatively efficient solution of hard problems, naturally accommodating memory, inference, self-healing, and homeostasis with respect to constraints.

Software. Mathematica packages and notebooks for the CRN simulator, SAT-solving CRN constructions, and Sudoku solver can be found on the author's website [60].

Acknowledgements. This work was supported in part by National Science Foundation (NSF) grant 1317694 – The Molecular Programming Project. Thanks to Matt Cook, David Soloveichik, Chris Thachuk, William Poole, Lulu Qian, Grzegorz Rozenberg, Moshe Vardi, Tony Rojko, and Henry Lester for stimulating questions, comments, and encouragement.

References

1. Simonis, H.: Sudoku as a constraint problem. In: Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, vol. 12, pp. 13–27 (2005)
2. Lynce, I., Ouaknine, J.: Sudoku as a SAT problem. In: Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics (AIMATH) (2006)
3. Hopfield, J.J.: Searching for memories, Sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. Neural Comput. **20**, 1119–1164 (2008)
4. Jonke, Z., Habenschuss, S., Maass, W.: Solving constraint satisfaction problems with networks of spiking neurons. Front. Neurosci. **10**, 118 (2016)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Company, New York (1979)

6. Jaffar, J., Maher, M.J.: Constraint logic programming: a survey. J. Logic Program. **19**, 503–581 (1994)
7. Vardi, M.Y.: Boolean satisfiability theory and engineering. Commun. ACM **57**, 5 (2014)
8. Järvisalo, M., Le Berre, D., Roussel, O., Simon, L.: The international SAT solver competitions. AI Mag. **33**, 89–92 (2012)
9. Balyo, T., Heule, M.J.H., Jarvisalo, M.: SAT competition 2016: recent developments. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
10. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
11. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. Cliques, Color. Satisf. **26**, 521–532 (1993)
12. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence, vol. 94, pp. 337–343. MIT Press (1994)
13. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Elsevier, Amsterdam (2004)
14. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. Foundations of Artificial Intelligence **3**, 89–134 (2008)
15. Kirschner, M., Mitchison, T.: Beyond self-assembly: from microtubules to morphogenesis. Cell **45**, 329–342 (1986)
16. Holy, T.E., Leibler, S.: Dynamic instability of microtubules as an efficient way to search in space. Proc. Nat. Acad. Sci. **91**, 5682–5685 (1994)
17. Gerhart, J., Kirschner, M.: Cells, Embryos, and Evolution: Toward a Cellular and Developmental Understanding of Phenotypic Variation and Evolutionary Adaptability. Blackwell Science, Malden (1997)
18. Kirschner, M., Gerhart, J.: Evolvability. Proc. Nat. Acad. Sci. **95**, 8420–8427 (1998)
19. Cauwenberghs, G.: A fast stochastic error-descent algorithm for supervised learning and optimization. In: Advances in Neural Information Processing Systems, pp. 244–251 (1993)
20. Sebastian Seung, H.: Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. Neuron **40**, 1063–1073 (2003)
21. Hinton, G.E., Nowlan, S.J.: How learning can guide evolution. Complex Syst. **1**, 495–502 (1987)
22. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Condon, A., Harel, D., Kok, J., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses. Natural Computing Series, pp. 543–584. Springer, Heidelberg (2009)
23. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Nat. Acad. Sci. **107**, 5393–5398 (2010)
24. Chen, Y.-J.: Programmable chemical controllers made from DNA. Nat. Nanotechnol. **8**, 755 (2013)
25. Srinivas, N., Parkin, J., Seelig, G., Winfree, E., Soloveichik, D.: Enzyme-free nucleic acid dynamical systems. Science **358**, eaal2052 (2017)
26. Gillespie, D.T.: Stochastic simulation of chemical kinetics. Ann. Rev. Phys. Chem. **58**, 35–55 (2007)
27. Anderson, D.F., Cappelletti, D., Koyama, M., Kurtz, T.G.: Non-explosivity of stochastically modeled reaction networks that are complex balanced. Bull. Math. Biol. **80**, 2561–2579 (2018)

28. Kurtz, T.G.: The relationship between stochastic and deterministic models for chemical reactions. J. Chem. Phys. **57**, 2976–2978 (1972)
29. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**, 147–195 (1969)
30. Johnson, R.F., Dong, Q., Winfree, E.: Verifying chemical reaction network implementations: a bisimulation approach. Theor. Comput. Sci. **765**, 3–46 (2019)
31. Doty, D., Zhu, S.: Computational complexity of atomic chemical reaction networks. Natural Comput. **17**, 677–691 (2018)
32. Magnasco, M.O.: Chemical kinetics is turing universal. Phys. Rev. Lett. **78**, 1190–1193 (1997)
33. Liekens, A.M.L., Fernando, C.T.: Turing complete catalytic particle computers. In: Almeida e Costa, F., Rocha, L.M., Costa, E., Harvey, I., Coutinho, A. (eds.) ECAL 2007. LNCS (LNAI), vol. 4648, pp. 1202–1211. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74913-4_120
34. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comput. **7**, 615–633 (2008)
35. Chen, H.-L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. Natural Comput. **13**, 517–534 (2014)
36. Cummings, R., Doty, D., Soloveichik, D.: Probability 1 computation with chemical reaction networks. Natural Comput. **15**, 245–261 (2016)
37. Napp, N.E., Adams, R.P.: Message passing inference with chemical reaction networks. In: Advances in Neural Information Processing Systems, pp. 2247–2255 (2013)
38. Gopalkrishnan, M.: A scheme for molecular computation of maximum likelihood estimators for log-linear models. In: Rondelez, Y., Woods, D. (eds.) DNA 2016. LNCS, vol. 9818, pp. 3–18. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43994-5_1
39. Fett, B., Bruck, J., Riedel, M.D.: Synthesizing stochasticity in biochemical systems. In: Proceedings of the 44th Annual Design Automation Conference, pp. 640–645. ACM (2007)
40. Poole, W., et al.: Chemical boltzmann machines. In: Brijder, R., Qian, L. (eds.) DNA 2017. LNCS, vol. 10467, pp. 210–231. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66799-7_14
41. Cardelli, L., Kwiatkowska, M., Laurenti, L.: Programming discrete distributions with chemical reaction networks. Nat. Comput. **17**, 131–145 (2018)
42. Anderson, D.F., Craciun, G., Kurtz, T.G.: Product-form stationary distributions for deficiency zero chemical reaction networks. Bull. Math. Biol. **72**, 1947–1970 (2010)
43. Cappelletti, D., Ortiz-Munoz, A., Anderson, D., Winfree, E.: Stochastic chemical reaction networks for robustly approximating arbitrary probability distributions. arXiv preprint arXiv:1810.02854 (2018)
44. Braich, R.S., Chelyapov, N., Johnson, C., Rothemund, P.W.K., Adleman, L.: Solution of a 20-variable 3-SAT problem on a DNA computer. Science **296**, 499–502 (2002)
45. Kirkpatrick, S., Selman, B.: Critical behavior in the satisfiability of random Boolean expressions. Science **264**, 1297–1301 (1994)
46. Selman, B., Kirkpatrick, S.: Critical behavior in the computational cost of satisfiability testing. Artif. Intell. **81**, 273–295 (1996)
47. Strzebonski, A.: Mathematica SatisfiabilityQ uses MiniSAT 1.14. StackExchange and personal communication (2016, 2019). https://mathematica.stackexchange.com/questions/103726/why-is-satisfiabilitycount-faster-than-satisfiableq

48. Soloveichik, D.: CRNSimulatorSSA Mathematica Package. Personal Web Site (2016). http://users.ece.utexas.edu/~soloveichik/crnsimulator.html
49. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. J. Phys. Chem. A **104**, 1876–1889 (2000)
50. Mauch, S., Stalzer, M.: Efficient formulations for exact stochastic simulation of chemical systems. IEEE/ACM Trans. Comput. Biol. Bioinf. **8**, 27–35 (2011)
51. Thanh, V.H., Zunino, R.: Adaptive tree-based search for stochastic simulation algorithm. Int. J. Comput. Biol. Drug Des. **7**, 341–357 (2014)
52. Condon, A., Hu, A.J., Maňuch, J., Thachuk, C.: Less haste, less waste: on recycling and its limits in strand displacement systems. Interface Focus **2**, 512–521 (2012)
53. Thachuk, C., Condon, A.: Space and energy efficient computation with DNA strand displacement systems. In: Stefanovic, D., Turberfield, A. (eds.) DNA 2012. LNCS, vol. 7433, pp. 135–149. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32208-2_11
54. Condon, A., Thachuk, C.: Towards space-and energy-efficient computations. In: Kempes, C., Grochow, J., Stadler, P., Wolpert, D. (eds.) The Energetics of Computing in Life and Machines, chapter 9, pp. 209–232. The Sante Fe Institute Press, Sante Fe (2019)
55. Matthew M Cook. Networks of Relations. PhD thesis, California Institute of Technology, 2005
56. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Nat. Acad. Sci. **79**, 2554–2558 (1982)
57. Kitano, H.: Towards a theory of biological robustness. Mol. Syst. Biol. **3**, 137 (2007)
58. Reich, E.S.: Mathematician claims breakthrough in Sudoku puzzle. Nature (2012). https://doi.org/10.1038/nature.2012.9751
59. Karr, J.R., Sanghvi, J.C., Macklin, D.N., Arora, A., Covert, M.W.: WholeCellKB model organism databases for comprehensive whole-cell models. Nucleic Acids Res. **41**, D787–D792 (2012)
60. Winfree, E.: Mathematica Notebooks for CRN SAT Solvers. Personal Web Site (2019). http://www.dna.caltech.edu/SupplementaryMaterial/CRNSAT/