DNA walker circuits: computational potential, design, and verification

Frits Dannenberg[1], Marta Kwiatkowska[1],
Chris Thachuk[1], and Andrew J. Turberfield[2]

[1] University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
[2] University of Oxford, Department of Physics, Clarendon Laboratory, Parks Road, Oxford, OX1 3PU, UK

Abstract. Unlike their traditional, silicon counterparts, DNA computers have natural interfaces with both chemical and biological systems. These can be used for a number of applications, including the precise arrangement of matter at the nanoscale and the creation of smart biosensors. Like silicon circuits, DNA strand displacement systems (DSD) can evaluate non-trivial functions. However, these systems can be slow and are susceptible to errors. It has been suggested that localised hybridization reactions could overcome some of these challenges. Localised reactions occur in DNA 'walker' systems which were recently shown to be capable of navigating a programmable track tethered to an origami tile. We investigate the computational potential of these systems for evaluating Boolean functions and forming composable circuits. We find that systems of multiple walkers have severely limited potential for parallel circuit evaluation. DNA walkers, like DSDs, are also susceptible to errors. We develop a discrete stochastic model of DNA walker 'circuits' based on experimental data, and demonstrate the merit of using probabilistic model checking techniques to analyse their reliability, performance and correctness. This analysis aids in the design of reliable and efficient DNA walker circuits.

1 Introduction

The development of simple biomolecular computers is attractive for engineering and health applications that require *in vitro* or *in vivo* information processing capabilities. DNA computing models which use hybridization and strand displacement reactions to perform computation have been particularly successful. DNA strand displacement systems (DSD) have been shown experimentally to simulate logic circuits [24, 23] and are known to be Turing-universal [22]. However, computing with biomolecules creates many challenges. For example, reactions within a DSD are global in the following sense: strands which are intended to react must first encounter one another in a mixed solution. The mixing of all reactants may lead to unintended reactions between strands. These systems do not, at present, ensure the spatial locality typical of other computing models. Qian and Winfree suggested that tethering DNA based circuits to an origami tile could overcome some of these challenges [23]. This idea was explored and expanded upon by Chandran et al. [9], who investigate how such systems could be realised experimentally, give constructions of composable circuits, and propose a biophysical model for verification of tethered, hybridization-based circuits. Our work is largely inspired by theirs, but we consider another setting which also exhibits localised reactions: DNA walker systems [5, 6, 8, 11, 21, 28, 29], in particular programmable walkers [18, 19, 27]. Theoretical work on the motion of walkers from a non-computational perspective includes [25].

In the walker system considered in this work, the walker traverses a track of strands, called *anchorages*, that are tethered to a DNA origami tile [27]. Anchorages contain a domain that is complementary to the walker strand. Movement of the walker between anchorages is shown in Fig. 1. After experimental preparation, all anchorages are unblocked — they are hybridized to the origami and no other strand — with the exception of designated blocking anchorages that are initially bound in a duplex to a *blocking strand*. Anchorages and their blocking strands are addressed by means of distinct toehold sequences (shown coloured): anchorages are selectively unblocked by adding strands complementary to their blockers as *input*. Much like field programmable gate arrays, these systems are easily reconfigured. By using programmable anchorages at track junctions, Wickham et al. [27] demonstrate that a walker can be directed to any leaf in a complete two-level binary tree using input strands that unblock the intended path.

In Section 2, the computational expressiveness of such walker systems is explored — including their potential to create composable circuits — using a theoretical framework that assumes ideal conditions. We highlight significant limitations of current walker systems and motivate future work. In Section 3 we develop a probabilistic model to analyse the impact of different sources of error that arise in experiments on reliability, performance and correctness of the computation. The model can be used to support the design and verification of DNA walker *circuits*.

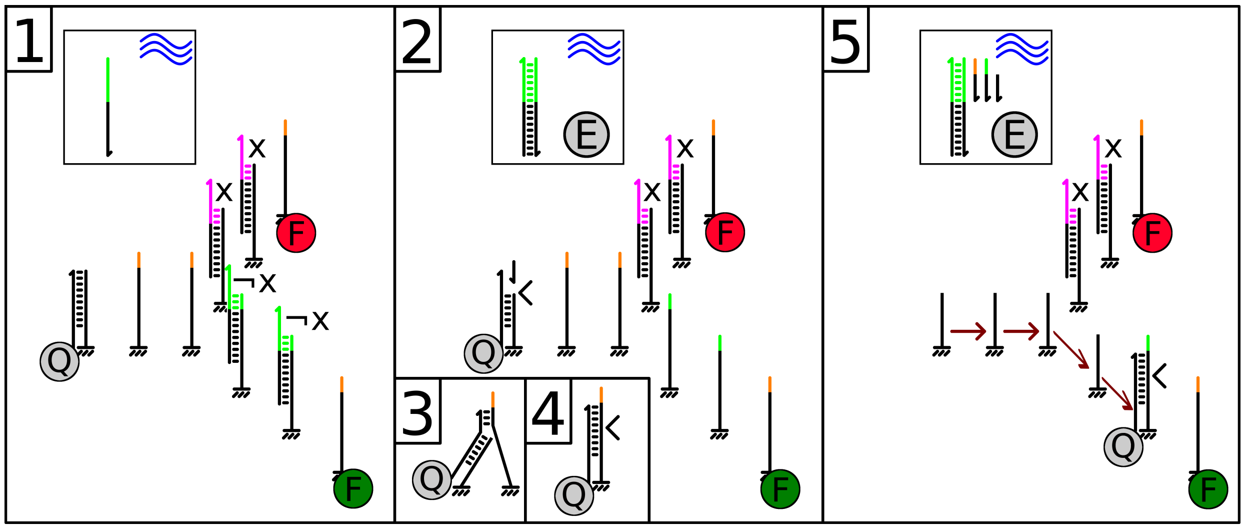


Fig. 1. (1) The walker strand carries a load (Q) that will quench fluorophores (F) when nearby. After experimental preparation, the walker is attached to the initial anchorage and blocking strands are present on designated blocking anchorages. Selected anchorages that are initially blocked can become unblocked by adding complementary unblocking strands. In this case, unblocking strands are added for the blocked anchorages that are labelled by $\neg X$. (2) Once a nicking enzyme (E) is added, it can attach to the walker-anchorage complex and cut the anchorage. The anchorage top melts away from the walker, exposing 6 nucleotides as a toehold. (3) The exposed toehold becomes attached to the next anchorage. (4) In a displacement reaction, the walker migrates to the new anchorage. The stepping is energetically favourable, because it re-forms the base pairs that were lost after the previous anchorage was cut. (5) Repeating this process, the walker arrives at a junction. The walker continues down the unblocked track, eventually reaching the final anchorage and quenching the fluorophore.

2 Computational potential of DNA walker circuits

In this section we explore the computational potential of DNA walker systems. We focus on deterministic Boolean function evaluation, and call the resulting constructions *DNA walker circuits*. We begin by defining a model of computation that makes explicit the underlying assumptions that characterize the DNA walker systems considered here. These assumptions are consistent with current published experimental systems: in particular, we do not explore the potential for multiple walkers to interact within the same circuit. However, we do consider the potential consequences for parallel computation. To simplify the presentation, some technical proofs have been moved to the appendix.

2.1 A model of computation for DNA walker circuits

A *DNA walker circuit* is composed of straight, undirected, *tracks* (consecutive anchorages), and *gates* (track junction points) that connect at most three tracks. A gate can have at most one Boolean *guard* for each track that it connects. A particular guard is implemented using one or more blocking strands that share a common toehold sequence. Distinct guards use distinct toehold sequences: a logical relationship between guards (*e.g.,* one is the negation of another) does not imply a relationship between their toehold sequences. A track adjacent to a gate is unblocked if it has a guard that evaluates to true, *i.e.,* if its unblocking strands are added to solution, and is otherwise blocked. When the system is prepared, a self-consistent set of unblocking strands is added to unblock X or $\neg X$ but not both. For example, Fig. 1 depicts a circuit of a single gate connecting three tracks. The track ending with the anchorage marked with the red fluorophore (top right of panel 1) has the Boolean guard X, while the track ending with the anchorage marked with the green fluorophore has the Boolean guard $\neg X$. Panel 2 of Fig. 1 shows that the path to the green fluorophore is unblocked when $\neg X$ evaluates to true (*i.e.,* the unblocking strands for $\neg X$ are added to solution). In this case, X evaluates to false and the path to the red fluorophore remains blocked (*i.e.,* the unblocking strands for X are *not* added to solution). This is an example of a fork gate. We define a *fork gate* as having at most one input track, and exactly two guarded output tracks. Each circuit has one *source* – a fork gate with no input track denoting the initial position of a walker. A *join gate* with an output track has two guarded input tracks. A join gate with no output track is a *sink* and has at most two (unguarded) input tracks. Each circuit has one or more *true sink*s and one or more *false sink*s.

In a circuit $\mathcal{C}$ with Boolean guards over n variables, a *variable assignment* A for $\mathcal{C}$ is a truth assignment of those n variables. Consider any DNA walker circuit $\mathcal{C}$ and variable assignment A for $\mathcal{C}$. Let $\mathcal{C}[A]$ denote the set of unblocked paths originating from the source of $\mathcal{C}$, after all guards are evaluated as blocked or unblocked, under assignment A. We say that $\mathcal{C}$ is *deterministic* under assignment A if there is exactly one unblocked path from the source to a sink in $\mathcal{C}[A]$. A fork gate is deterministic if, under any assignment that it is reachable, exactly one output track is unblocked. Similarly, a join gate is deterministic if, under any assignment that it is reachable by one input track, the other input track is blocked. Note that this definition of determinism precludes the possibility of a *deadlock*, (*i.e.,* when there is no unblocked path from the source to a sink). Let $\texttt{VALUE}\,(\mathcal{C}[A])$ be the *output value* of the circuit under assignment A (*i.e.,* whether the reachable sink is a true sink or a false sink). Circuit $\mathcal{C}$ is *deterministic* if it is deterministic under all possible variable assignments.

A *circuit set* S, consisting of one or more unconnected circuits, is deterministic if and only if $\texttt{VALUE}\,(\mathcal{C}_i[A]) = \texttt{VALUE}\,(\mathcal{C}_j[A])$, for each $\mathcal{C}_i, \mathcal{C}_j \in \mathsf{S}$, under any possible assignment A. Informally, this states that different circuit components in a deterministic circuit set cannot report different output values under the same assignment. (The purpose of circuit sets is demonstrated in the next section.) Let $\texttt{VALUE}\,(\mathsf{S}[A])$ be the value of S under assignment A. The *size* of S, denoted by $\texttt{SIZE}\,(\mathsf{S})$, is the total count of component gates.[3] We define the worst case *time* of a computation in S, denoted by $\texttt{TIME}\,(\mathsf{S})$, as the longest unblocked path from a source to a sink, under any variable assignment. This notion of time captures the ability of multiple walkers to simultaneously traverse disjoint paths (one per unconnected circuit).

Let $\mathsf{S}[A]$ denote the set of unblocked paths in S under assignment A (one per unconnected circuit). Given a circuit $\mathcal{C}_i \in \mathsf{S}$, we say that a gate $G \in \mathcal{C}_i$ is *reachable* in $\mathsf{C}_i[A]$ (equivalently $\mathsf{S}[A]$) if there exists an unblocked path from the source of $\mathcal{C}_i$ to G, under assignment A. Gates that are not reachable under any variable assignment are called *redundant*. An example of a fork gate that is never reachable, and therefore redundant, is shown in Fig. 2. We will reason about circuit sets where all gates are non-redundant. When this is not the case, the circuit set can be simplified to one that is logically equivalent.

2.2 Deterministic fork and join gates in DNA walker circuits

In this section, we begin by establishing the necessary and sufficient conditions for a *fork* gate to be deterministic in a DNA walker circuit. The fork gate in this model is the primitive used to branch a computation based on the values of its output guards.

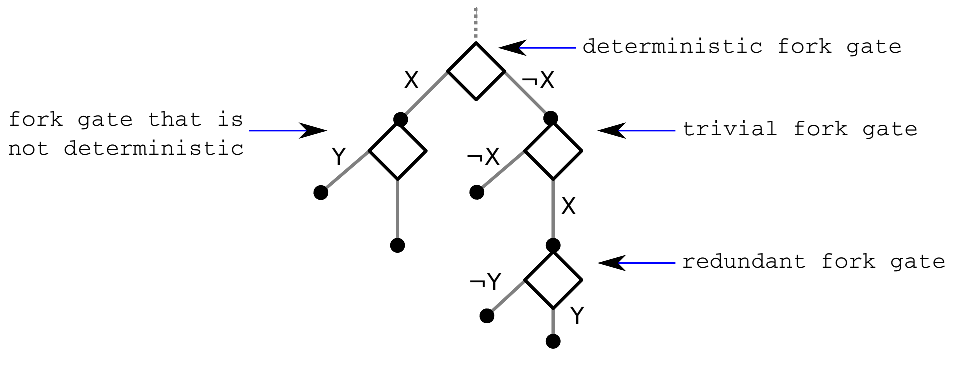


Fig. 2. Examples of different types of fork gates. See text for definitions.

Recall that a fork gate that is not reachable under any variable assignment is called *redundant*. A fork gate G with output track guards G_L and G_R is *trivial* if every path p that can reach G either traverses a track guarded by G_L and another guarded by $\neg G_R$ before reaching G, or it traverses a track guarded by $\neg G_L$ and another guarded by G_R before reaching G. Note that if G is reachable (*i.e.,* non-redundant), then it must be the case that $G_L \not\equiv G_R$. We say that gate G is trivial because any path leading to G fully dictates which output track will be traversed. The following lemma holds by definition of a trivial fork gate.

[3] We do not investigate circuit area in this paper.

Lemma 1. *A non-redundant fork gate G in a DNA walker circuit is deterministic if it is* trivial.

Note that when $G_L \equiv \neg G_R$ (*i.e.,* the guards are negations of each other), then a gate is trivial if every path p that can reach G must first traverse a track guarded by any of $G_L, G_R, \neg G_L$, or $\neg G_R$. An example of a trivial fork gate of this kind is depicted in Fig. 2. A trivial fork gate does have uses as we will see in Section 2.3.

Lemma 2. *A fork gate in a DNA walker circuit without a distinct guard on both output tracks is either redundant, trivial or not deterministic.*

The following theorem shows that non-redundant fork gates that are deterministic must be trivial or have output guards that are negations of each other.

Theorem 1. *A non-redundant fork gate in a DNA walker circuit is deterministic if and only if it is trivial or there exists some guard G such that the left output track is guarded by G and the right is guarded by $\neg G$.*

Given any Boolean function $f : \{0,1\}^n \to \{0,1\}$, there exists a deterministic DNA walker circuit set S that can evaluate f, under any assignment to its n variables, such that $\texttt{TIME}(\mathsf{S}) = O(n)$. One construction is to simply form a canonical binary decision tree over some fixed order of the n variables. However, in such a construction $\texttt{SIZE}(\mathsf{S}) = \Theta(2^n)$. It is natural to consider more space efficient representations to evaluate f, such as binary decision diagrams (BDDs) [7]. In particular, reduced ordered BDDs are capable of representing some Boolean functions in a compressed form that can be exponentially smaller than its canonical binary decision tree representation. Like walker circuits, BDDs have a unique source. Unlike general BDDs, DNA walker circuits are necessarily planar. Either we are limited to considering planar BDD representations or additional fork and join nodes must be added to a BDD representation when realising it as a walker circuit — we show how a non-planar circuit can be made planar in Section 2.3. A significant difference, however, is that BDDs form directed acyclic graphs while tracks in a DNA walker circuit are undirected. Consider the case when a walker reaches a join gate via its left input track. Unless the right input track is blocked, the walker is equally likely to continue on the right input track as it is on the output track. Additional steps are necessary to compensate for the undirected nature of tracks in walker circuits.

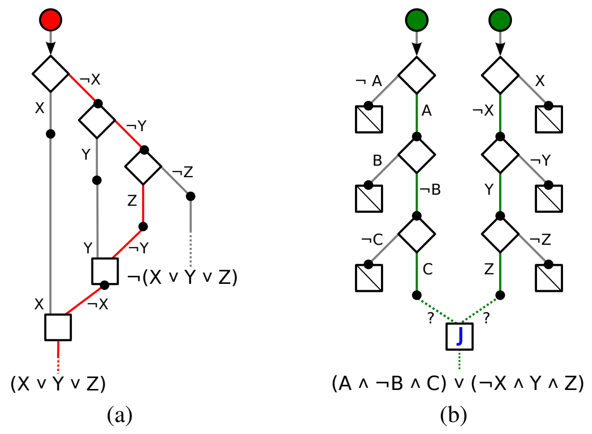


Fig. 3. (a) A connectivity graph of a DNA walker circuit to evaluate the disjunction $(X \vee Y \vee Z)$. There are two output tracks: one when the circuit evaluates to true, the other when it evaluates to false. The resulting path when $X = Y = f$ and $Z = t$ is shown highlighted. (b) Two conjunction circuits are composed into the disjunction $(A \wedge \neg B \wedge C) \vee (\neg X \wedge Y \wedge Z)$. Two source nodes (two walkers) are used to evaluate clauses in parallel. No assignment of guards to the join gate labelled J can ensure that this circuit is deterministic. This is evident when $A = C = Y = Z = t$ and $B = X = f$.

Unlike fork gates, it is not obvious whether all join gates can be made deterministic. Theorem 2 characterizes both the necessary and sufficient conditions: a deterministic join of two disjoint sets of paths, one for each input track, is only possible if they were previously "forked"[4] on some variable X (*i.e.,* in one set all paths traverse an edge guarded

[4] It is not a necessary condition that the two disjoint sets of paths reaching the join were forked by a common gate, only that they can be partitioned based on the value of some variable.

by X and in the other set all traverse an edge guarded by $\neg X$). This property is exemplified by the contrast between the disjunction circuit of Fig. 3(a) and the disjunction of two conjunctions circuit as shown in Fig. 3(b). In the latter, two walkers are used in an attempt to parallelize the evaluation. However, as the clauses do not have literals over a common variable, there are no guards that can be assigned to the join gate labeled J to ensure the circuit is deterministic. Note that this limitation is not caused by the restricted topology of walker circuits (*i.e.,* their layout on a planar surface), but rather by the property that their tracks are undirected.

Theorem 2. *A non-redundant join gate in a DNA walker circuit is deterministic if and only if it is a sink or there exists some guard G such that the left input track is guarded by G, the right by $\neg G$ and, prior to reaching those guards, all paths that can reach the left input must traverse a track guarded by G and all paths that can reach the right must traverse a track guarded by $\neg G$.*

2.3 Composable DNA walker circuits

Despite the shortcomings of join gates in current DNA walker circuits, it is not the case that Boolean formulas must be evaluated using a circuit forming a binary decision tree. Large scale combinatorial Boolean circuit design is possible because of composable gates. In this section, we demonstrate that DNA walker circuits can simulate fundamental Boolean functions that are easily composable into larger circuits. A gate is *composable* if and only if it has one input gate and two output gates — one denoting true, the other false — such that the input and output gates lie on the external face of the connectivity graph of the circuit. The last condition ensures that a track can be connected to these gates without crossing another existing track. A common design pattern is to first create gates for simple functions. In turn, these can be composed to create circuits for functions with increased complexity. Using automated verification techniques, these fundamental component circuits could be designed and optimised for a number of criteria, including reliability and expected time. (This is the focus of Section 3.)

We begin by considering composable DNA walker circuits that can function as composable gates for Boolean functions over two variables. Each DNA walker circuit can realize a number of different Boolean functions by interpreting different blocking anchorages as different Boolean guards. Consider the composable circuits over two variables, X and Y, in Fig. 4; the common circuit topology shown in the top row can realize six different Boolean functions, while the common topology shown in the middle and bottom rows can realize eight different Boolean functions. Furthermore, each instantiation is fully composable: its source can be connected to the output of a previous circuit, and both its true and false outputs can be connected to other circuits.

The common topology of the middle and bottom rows can be composed to simulate any Boolean function. This is possible as the topology is (i) composable and (ii) can function as either of two universal gates (`NAND` and `NOR`). While just two different topologies can realize fourteen different Boolean functions over two variables, note that there are in fact sixteen possible Boolean functions over two variables[5]. The two functions which cannot be realized by either of these topologies are `EQUAL` (are both inputs equal), and `XOR` (are both inputs different). The two topologies of Fig. 4 are called *simple* circuits — circuits that do not contain trivial fork gates. Recall that a fork gate with output guards G and $\neg G$ is trivial if all paths leading to the gate must first traverse a track guarded by G or $\neg G$. We next show that topological constraints preclude composable, simple circuits from evaluating certain classes of functions, such as `EQUAL` and `XOR`. An invalid topology for a simple, composable DNA walker circuit that can simulate the `EQUAL` and `XOR` functions is shown in Fig. 5.

Our strategy for identifying a class of topologies which cannot be realized as composable, simple walker circuits is to identify those which are non-planar. We use the following characterization of planar graphs due to Wagner [26]. Specifically, we will identify topologies that contain the forbidden graph minor $K_{3,3}$ — the complete bipartite graph where each partition has three vertices.

Theorem 3 (Wagner [26]). *A finite graph is planar if and only if it does not contain K_5 nor $K_{3,3}$ as a minor.*

Theorem 4. *If a DNA walker circuit is* composable *and* simple *then both output gates are not reachable from two other distinct gates.*

Proof. Suppose, by contradiction, that some DNA walker circuit $\mathcal{C}$ is composable and simple and that both output gates are reachable from two other distinct gates. Label one output gate t, the other f, the single input gate s and let p

[5] A Boolean function must differ from another on at least one input. As such, there are 2^{2^n} possible Boolean functions of n variables that differ in at least one of the 2^n possible inputs. Intuitively, this is the number of unique binary strings of length 2^n.

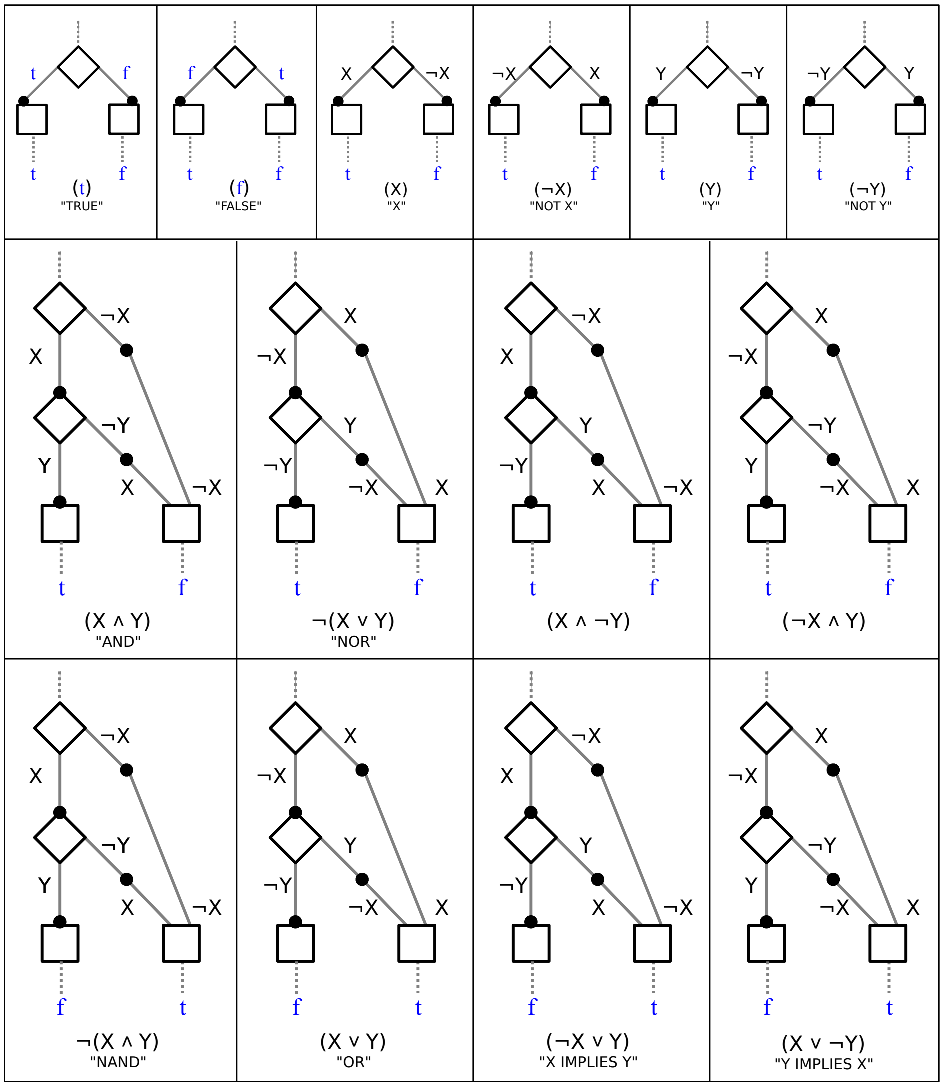


Fig. 4. There are sixteen possible Boolean functions over two variables. DNA walker circuits which are *composable* and *simple* can realize fourteen of them. Each of these fourteen circuits are composable as their input track and both of their output tracks are accessible. Note how the same circuit topology can realize a number of Boolean functions, based on the assignment of guards and the labelling of output tracks. The common topology of the top row realizes six different functions. The common topology of the middle and bottom rows realize eight different Boolean functions. The two functions not realizable as composable, simple circuits are (i) equality $(X \wedge Y) \vee (\neg X \wedge \neg Y)$ and (ii) exclusive or $(X \wedge \neg Y) \vee (\neg X \wedge Y)$.

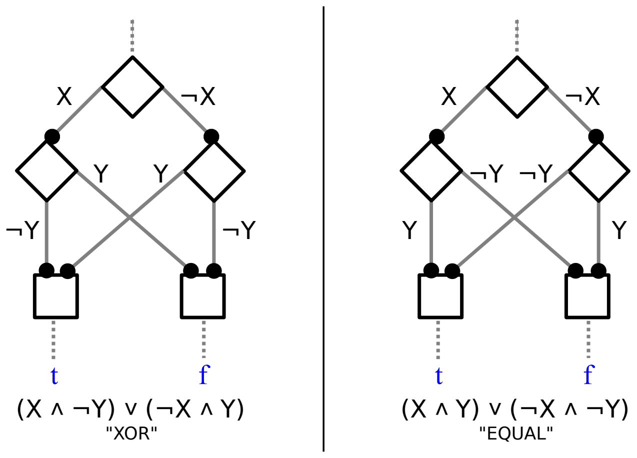


Fig. 5. Simple, composable DNA walker circuits for the `XOR` Boolean function and for the `EQUAL` Boolean function are non-planar.

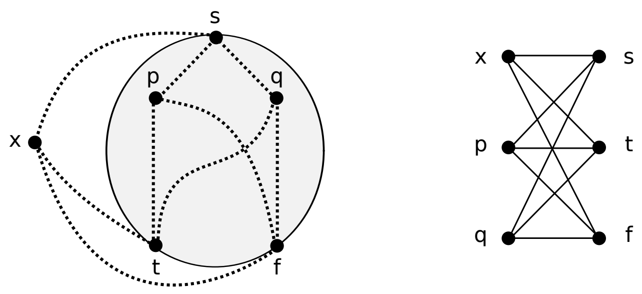


Fig. 6. An illustration used in the proof of Theorem 4 demonstrating that no composable, simple circuit is planar if both output gates are reachable from two distinct gates. If both output gates were reachable from two distinct gates, then the connectivity graph (left) must contain $K_{3,3}$ (right) as a graph minor.

and q be the labels of the two distinct gates which can reach both t and f. There must exist a path from s to p and a path from s to q. By the assumption, there is also a path from p to t, p to f, q to t and q to f. As $\mathcal{C}$ is a composable circuit, then s, t and f must be on the external (unbounded) face of the connectivity graph. To enforce this condition, add a new vertex x to the external face and connect it to s, t and f. If s, t and f did lie on an external face before adding x, then they can be connected to x without crossing any edges. The resulting topological connections are illustrated in Fig. 6 (left). However, these connected paths contain $K_{3,3}$ as a minor and therefore $\mathcal{C}$ is not planar by Theorem 3. As such, $\mathcal{C}$ is not a composable, simple walker circuit (or a valid walker circuit in general). Contradiction. $\square$

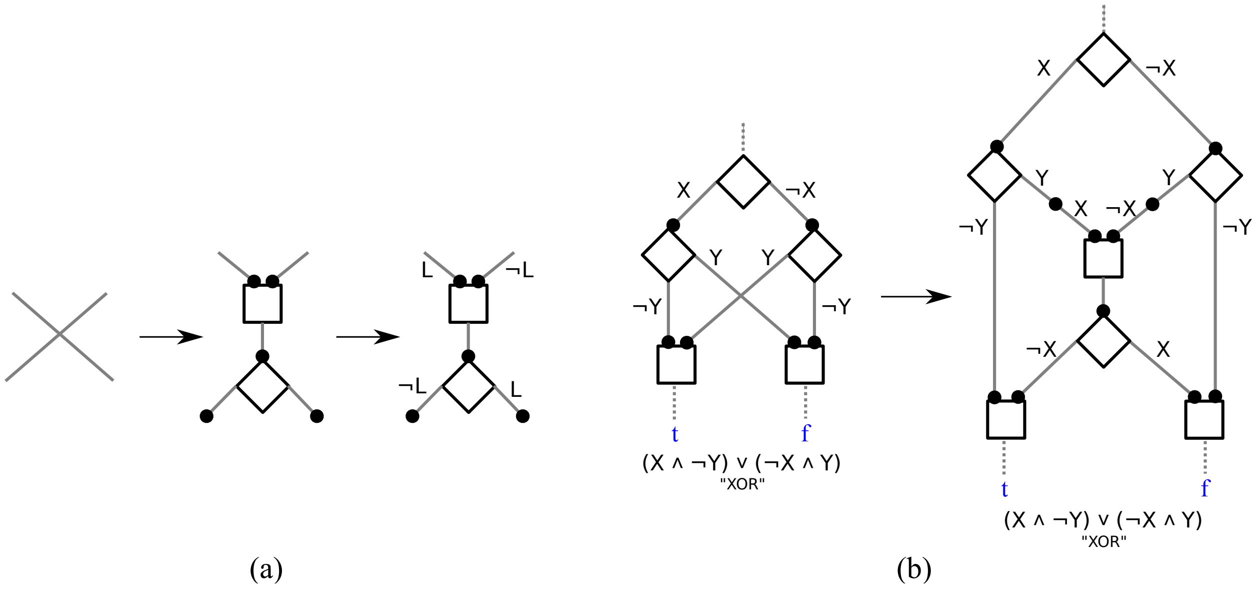


Fig. 7. (a) Non-planar topologies can be made planar by replacing the intersection of crossing tracks with a connecting gadget consisting of a join gate, connecting track, and a trivial fork gate. Details can be found in the text. (b) The non-planar, composable, simple `XOR` circuit from Fig. 5 can be made into a planar, composable (but not simple) `XOR` circuit by replacing the crossing tracks with the connecting gadget from (a).

The composable, simple circuits of Fig. 5 are invalid as their topologies are non-planar. However, composable circuits for these functions which are planar, but not simple, can be realized. A non-planar topology can be made planar by introducing additional join and trivial fork gates. The general strategy is illustrated in Fig. 7(a). Consider a pair of crossing paths (tracks). Firstly, their intersection is replaced by a simple gadget consisting of a join gate, a connecting track, and a fork gate. As each connected component of a walker circuit has a single source, then these paths must have diverged from one or more fork gates as otherwise they would be the same path. Suppose that the last fork gate where these paths diverged had output guards L and $\neg L$. Without loss of generality, suppose the path originating in the top left of the crossing traversed the output track guarded by L (of the last fork gate that the paths diverged). Then the path originating in the top right of the crossing traversed the output track guarded by $\neg L$. Secondly, add the guards L and $\neg L$ to the left and right input tracks of the new join gate, respectively, and add the guards $\neg L$ and L to the left and right output guards of the fork gate, respectively. This completes the transformation. By Theorem 2, the new join gate is deterministic as it is guarded by L and $\neg L$ — a variable on whose value the two paths were previously "forked". By Theorem 1, the fork gate is deterministic as the guards are negations of each other. Together, these gates ensure that there is a unique path through the connecting gadget for any variable assignment. An example of transforming the invalid, composable, simple walker circuit for `XOR` into a valid, composable (but not simple) walker circuit is shown in Fig. 7(b). In this case, the crossing paths last diverged at the initial fork gate guarded by X and $\neg X$. In general, this transformation necessarily results in a larger circuit — two gates, and four guards are added for every crossing path.

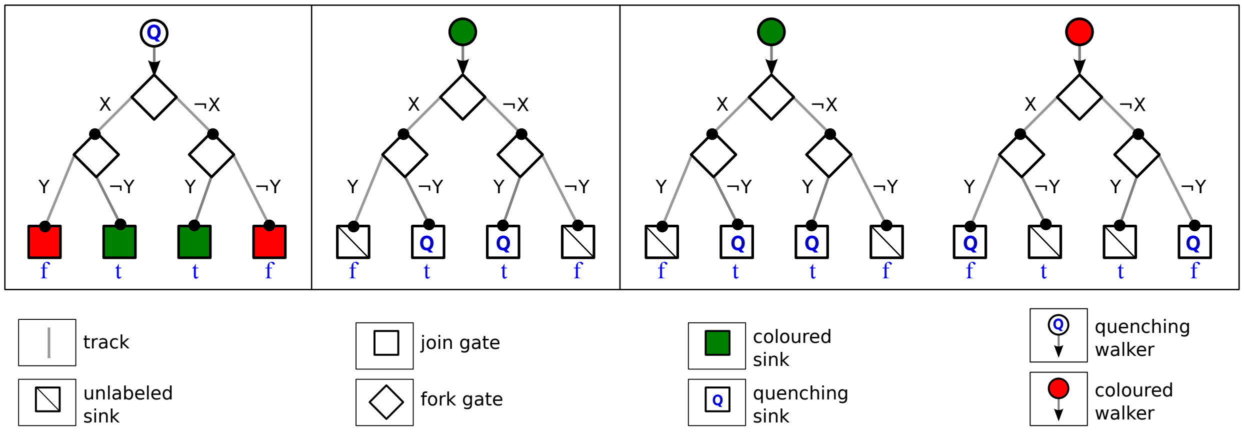


Fig. 8. Reporting Boolean decisions with DNA walker circuits. (Left) A quenching walker with red fluorophores labelling false sinks and green fluorophores labelling true sinks. A drop in signal for one colour indicates the truth value of the circuit. However, the signal drop is inversely proportional to the number of sinks of the same colour. (Center) A green coloured walker and quenching true sinks. When the circuit evaluates to true the green signal is fully suppressed. However, the fluorescence output from this circuit cannot distinguish between an incomplete computation and a false one. (Right) Two parallel copies of the circuit, with different fluorophores labelling the walkers and with quenching true sinks in one and quenching false sinks in the other: the computation is complete and unambiguously reported when one colour is suppressed.

2.4 Reporting output in DNA walker circuits

Output of a DNA walker circuit can be reported with the use of different coloured (spectrally resolvable) fluorophores and also quenchers. If a walker carries a quencher cargo, then it has the potential to decrease one of a number of different fluorescent signals from fluorophores positioned at the circuit sinks. This scenario is illustrated in Fig. 8 (Left). In a circuit that decides a Boolean function, a single, quenching, walker can only decrease the signal of a particular colour (corresponding to a particular fluorophore) by an amount that is inversely proportional to the number of sinks labelled with that same colour. Accurate output reporting could be problematic in larger circuits with many sinks. We will therefore focus only on reporting strategies that fully suppress a particular colour. Rather than carrying a quencher, a walker instead carries a fluorophore of a single colour and either all true sinks or all false sinks are labelled with quenchers. An example with quenching true sinks is shown in Fig. 8 (Center). This circuit can fully suppress the fluorophore signal when it evaluates to true, regardless of its size. However, this is a one-sided reporting strategy as one cannot distinguish between the case of an incomplete computation or one evaluating to false. As illustrated in Fig. 8 (Right), this shortcoming can be addressed by using two circuits in parallel, with each using a one-sided reporting strategy. Each of the two (otherwise identical) circuits uses a different coloured walker: one has quenching false sinks and the other quenching true sinks. In this circuit set, one colour will be fully suppressed when it is true, the other when it is false, and neither will be suppressed until the computation completes.

2.5 Evaluating any Boolean formula with a DNA walker circuit

Any Boolean formula can be represented in one of its canonical forms. In this section, we focus on conjunctive normal form (CNF) which is a single conjunction of clauses, where each clause is a disjunction over literals. A formula in CNF is said to be k-CNF if the largest clause has size k. Using a standard transformation, a Boolean formula in k-CNF with at most l total literals can be converted to an equisatisfiable 3-CNF formula over $O(l)$ variables, with at most $O(l)$ clauses (each having at most 3 literals) [13]. As such, we will reason exclusively about circuits to evaluate 3-CNF formulas.

Constructing a walker circuit to represent a formula in 3-CNF with m clauses is straightforward. Each clause can be represented by the disjunction circuit of Fig. 3(a). The source of the circuit will be the first fork gate of the first clause. The output track signalling the i-th clause is satisfied is connected to the input track of clause $i + 1$. Thus, the walker will only reach the single true sink of the circuit (output from clause m) if the formula is satisfied for that particular variable assignment. To ensure that both true and false signals can be reported deterministically, we use the reporting strategy depicted in Fig. 8 (Right) which employs two parallel copies of the circuit, each using different coloured walkers and different quenching sinks.

Theorem 5. *Let $\mathcal{F}$ be any 3-CNF Boolean formula with m clauses. There exists a DNA walker circuit set* S*, with* $\texttt{SIZE}(\mathsf{S}) = \Theta(m)$ *and* $\texttt{TIME}(\mathsf{S}) = O(m)$*, such that given any variable assignment A for $\mathcal{F}$,* $\texttt{VALUE}\,(\mathsf{S}[A])$ *is the truth value of $\mathcal{F}$ under assignment A.*

Proof. The construction is described in the preceding paragraph. It is easy to see that the circuit is deterministic and that it correctly reports the truth value of $\mathcal{F}$ under assignment A. What remains is to bound the circuit size and worst case time. The construction uses a set of two circuits: $\mathcal{C}_T$ and $\mathcal{C}_F$. Consider the circuit $\mathcal{C}_T$ used to evaluate if $\mathcal{F}$ is true under assignment A. There are m clauses and each is simulated by a disjunction circuit of size $O(1)$. These circuits are composed in series to form $\mathcal{C}_T$. Therefore, $\texttt{SIZE}(\mathcal{C}_T) = \Theta(m)$ and $\texttt{TIME}(\mathcal{C}_T) = O(m)$. The arguments are the same for circuit $\mathcal{C}_F$ and, as both are evaluated in parallel, the claim follows. □

While the construction of Theorem 5 can represent any Boolean formula, and some in exponentially less space than a binary decision tree, the resulting circuit set is formula specific. Given the effort of creating DNA walker circuits, a more uniform circuit — one capable of evaluating many Boolean functions — is worth exploring. As with silicon circuits, we can construct a uniform circuit to evaluate any 3-CNF formula, under any variable assignment, up to some bound on the number of variables. Each variable can be present in a clause as either a positive or negative literal, but not both. (The circuit can be modified to handle this case if necessary.) Therefore, there are at most $2^3\binom{n}{3}$ unique clauses in any 3-CNF Boolean formula over n variables. This bound also holds true for any formula over m variables, where $m \leq n$. In this general circuit, we supplement each possible clause with an initial fork gate guarded on the condition of the clause being active or inactive in the particular formula being evaluated. If it is inactive, the walker can pass through to the output track denoting true, without traversing guards for the literals of the clause. Note that this only increases the size of each clause by a constant.

Corollary 1. *There exists a DNA walker circuit set* S*, with* $\texttt{SIZE}(\mathsf{S}) = O(n^3)$ *and* $\texttt{TIME}(\mathsf{S}) = O(n^3)$*, that can evaluate any 3-CNF Boolean formula over $m \leq n$ variables under any variable assignment.*

A 3-CNF formula with m clauses can be evaluated in polylogarithmic time (in m) using a silicon circuit in a straightforward manner: each clause can be evaluated in parallel and those results can be combined using a binary reduction tree of height $O(\log m)$—only if all clauses are satisfied will the root of the reduction tree output true. Is the same possible in DNA walker circuits? Unfortunately, this is not the case in general. Such a circuit would require a new kind of join gate, outside of our current model of computation, to perform a conjunction of multiple walkers — one walker leaves the gate only after all input walkers have arrived. Parallel evaluation of circuits representing formulas in disjunctive normal form (DNF) does not fare better. Consider the case of a DNF formula with m clauses where clause $m-1$ and clause m have no literals over a common variable. By Theorem 2, a join gate connecting the circuits for these clauses cannot be deterministic. An example of this situation is given in Fig. 3(b).

3 Design and verification of DNA walker circuits

We have so far assumed DNA walker circuits to work perfectly. In a real experiment various errors can occur, for example, the walker may release from a track, or a blockade can fail to block an anchorage. In this section, we analyse the reliability and performance of DNA walker circuits using probabilistic model checking. We develop a continuous-time Markov chain (CTMC) model, based on DNA walker experiments [6, 27, 28], and analyse it against quantitative properties such as the probability of the computation terminating or the expected number of steps until termination. We use the PRISM model checker [17], which accepts properties in the form of temporal logic. The stepping behaviour of the walker is modelled using rate constants that were estimated in previous work [28]. The predictions of the model match the experiments in Fig. 9 well, and we use the rate constants without further fitting. We do, however, fit a *failure rate* for blockades using single-junction tracks (Fig. 10). The quality of the model is evaluated by comparison with experimental results for the double-junction circuit. While doing so, we identify a leakage transition within that circuit, which can be removed by changing the layout of the circuit. As an example, we optimise a large circuit composed of three circuits that only differ in blockade guards, and state design principles that aim to minimize leakage reactions.

3.1 Probabilistic model checking with PRISM

Probabilistic model checking is a formal verification technique for the modelling and analysis of discrete stochastic models, such as those arising from biochemical reactions [14, 16]. We use the probabilistic model checker PRISM [17],

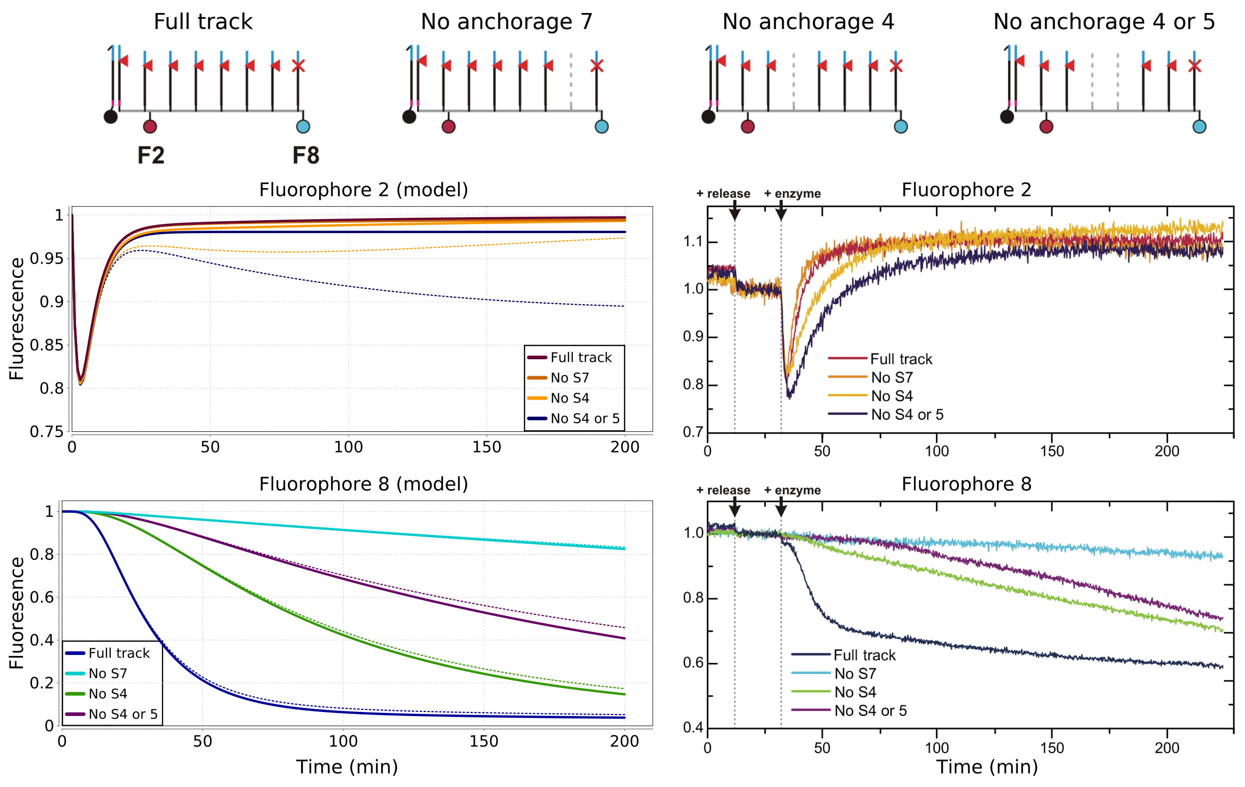


Fig. 9. Top: A small track of 8 anchorages with fluorophores on both the second and last anchorage. Experiments were performed with one or more anchorages omitted [28]. Right: Normalized fluorophore response (reproduced with permission from the authors). The walker hardly reaches the final anchorage when anchorage 7 is removed, due to the double penalty of a longer final step *and* the mismatch in the final anchorage. Left: Linear interpolation of the fluorophore luminosity in the model, evaluated at $T = 0, 1, 2, ..$ using PRISM's standard uniformisation method using default settings for the hybrid engine. Dotted lines: Alternative model where the walker can step onto already-cut anchorages with rate $k_b = k/500$, where k is the normal stepping rate.

which supports numerical (uniformisation) and statistical (Monte Carlo) methods, to compute various properties of DNA walker circuits. PRISM employs the Continuous Stochastic Logic (CSL) [3, 4], endowed with the reward operator introduced in [15], over which the properties are specified. A path is an alternating sequence of states and residence times, where states are labelled with atomic propositions, e.g., we can label all states of the model where a walker quenches any fluorophore by "finished". Temporal logic properties allow us to reason about the order of events in a path. For example, the formula "F finished" (eventually) is true for a path if there exists a state in the path labelled "finished", and the time-bounded formula "$F^{[0,T]}$ finished" (eventually by time T) is true if there exists a state in the path labelled "finished" that is reached in up to T time units. More generally, given state formulas Φ and Ψ, the (until) formula "$\Phi\ U^{[T,T']}\ \Psi$" is true for a path if Ψ becomes true at a time point in the interval $[T, T']$, and Φ remains true until then. The unbounded variant "$\Phi\ U\ \Psi$" is defined similarly. The formula "$F\ \Phi$" is equivalent to "true Φ".

The set of paths of a CTMC is endowed with a probability measure [4, 14], and we can use the CSL probability operator P, which takes a path formula as an argument, to reason about the probability of events occurring. For example, the formula $P_{=?}[\,F^{[T,T]}\text{ finished}\,]$ yields the probability of all paths that satisfy "$F^{[T,T]}$ finished" (in other words, the probability of the event of reaching a state where a walker has quenched a fluorophore at time T). PRISM also provides the reward operator R, which computes the expected reward over paths. The model is enhanced with a reward structure, which assigns nonnegative real numbers to states and transitions. For example, the formula $R_{\{\text{“steps”}=?\}}[C^{\leq T}]$ denotes the expected number of walker steps accumulated until time T (C stands for cumulative reward), assuming the reward structure "steps" assigns the value 1 to each transition that represents a walker step and 0 to all states.

To facilitate the generation of walker circuit models, we develop a custom tool to generate PRISM models with matching track-layout graphs. PRISM then builds an internal representation of the model and computes the probability and reward values for the given formulas. There are several methods provided to compute these [14, 17], including numerical computation performed to a specified precision based on uniformisation, and simulation-based methods (also known as approximate model checking). For more details concerning probabilistic model checking of molecular networks the reader is invited to consult [14, 16].

3.2 Model

In the model, each transition corresponds to the walker changing its location from one anchorage to the next, skipping over any intermediate steps (Fig. 1). Anchorages are constrained to lie on a triangular lattice [27, 28]. Experiments show that the walker can step onto anchorages that are fixed as far away as 19 nm. We assume non-zero rates for the walker to step onto any intact anchorage within 24 nm distance. This range was chosen by taking into account the lengths of the empty anchorage and walker-anchorage complex, estimated around 15 nm and 11 nm respectively.

Assume that the stepping rate k depends on distance d between anchorages and some base stepping rate k_s. Denote by $d_a = 6.2$ nm the average distance between anchorages in the experiment shown in Fig. 9. Denote by $d_M = 24$ nm the maximal interaction distance discussed earlier. Based on previous experimental estimates of [28], we approximate the stepping rate k as

$$k = \begin{cases} k_s = 0.009\text{s}^{-1} & \text{when } d \leq 1.5d_a \\ k_s/50 & \text{when } 1.5d_a < d \leq 2.5d_a \\ k_s/100 & \text{when } 2.5d_a < d \leq d_M \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

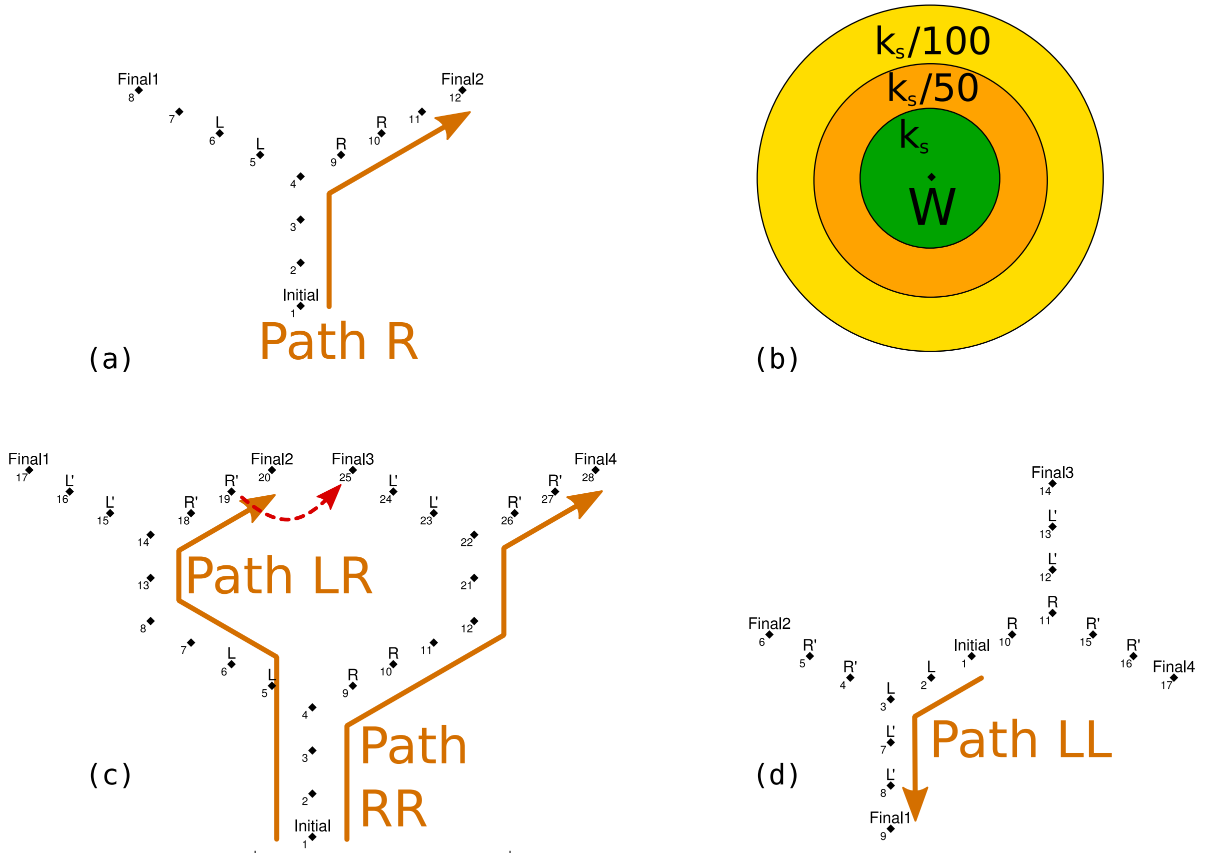
These rates define a sphere of reach around the walker-anchorage complex, allowing the walker to step onto an uncut anchorage when it is nearby. In Fig. 10(b) the sphere of reach is depicted to scale with walker circuits. Because both the initial anchorage and the absorbing anchorages are slightly different from regular anchorages, we allow two exceptions to the stepping rate. Firstly, the domain complementary to the walker on the initial anchorage is two bases longer than the corresponding domain of a regular anchorage. Stepping from the initial anchorage was reported to happen $3\times$ more slowly: this is incorporated in the model. Secondly, absorbing anchorages include a mismatched base that prevents cutting by the nicking enzyme. Based on the experimental data, we fit a tenfold reduction for the rate of stepping *onto* the final absorbing anchorage (Fig. 9).

Three types of interaction that are known to occur are omitted from the model: all three could be incorporated in future. Firstly, a rate of $k_s/5000$ is reported [28] for transfer of the walker between separate tracks built on different DNA origami tiles. Transfer between tiles could be eliminated by binding the tiles to a surface, thus keeping them apart. Secondly, the walker can move between *intact* anchorages, with a rate of $\sim k_s/13$ [28]. We assume that the enzymatic activity is high, so that an anchorage is nicked as soon as the walker steps onto it. Thirdly, the walker can step backward onto cut anchorages. This requires a blunt-end strand-displacement reaction which is known to be slow relative to toehold-mediated displacement [30]. A variant of the model with a backward rate $k_b = k/500$ is shown in dotted lines in Fig. 9 (Left). In this case the model predicts significant quenching of fluorophore F2 at late times by walkers whose forward motion is obstructed by omission of one or more anchorages: this does not match experimental data.

The time-dependent responses of fluorescent probes F2 and F8 shown in Fig. 9 (Top) are predicted by the Markov chain model using the rate parameters discussed above without any further fitting: they correspond well to the experimental data.

An additional parameter is needed to model branched circuits (Fig. 10(a)). We add a possibility of failure of the blocking mechanism, such that, before the walker starts the computation, each blocked anchorage may unblock spontaneously. The probability of unblocking is uniform for all blocked anchorages. The walker can step onto such spontaneously unblocked anchorages, and may thus divert from the intended path. This may delay the walker, or, worse, it may even direct the walker to reach a different end-node, leading to the computation returning the wrong result. We infer a failure rate of 30% by fitting to the results of the single-junction experiment [27], see Fig. 10.

Additional sources of error may exist in the system, for example, an anchorage or the walker itself could be missing from the track, or the reporting mechanism could fail. These have not been considered, but could be added to the model using our approach.



%	Single-junction						Double-junction											
	Experiment			Model			Experiment				Model				Optimised (model)			
	R	R^2	L/R	R	R^2	L/R	RR	RL	LR	LL	RR	RL	LR	LL	RR	RL	LR	LL
Finishes	65	56	56	97	96	92	33	40	22	33	90	89	89	90	94	92	94	92
Correct	76	87	50	78	85	50	70	65	55	76	77	74	74	77	78	78	78	78
Deadlock				.084	.16	.063					1.0	1.7	1.7	1.1	0.0	0.0	0.0	0.0
Steps				7.1	7.0	6.6					11.7	11.8	11.8	11.7	5.1	5.1	5.1	5.1

Fig. 10. Top: Circuit layout for single-junction (a) and double-junction (c,d) decision circuits. Circuit (d) is an improved version of circuit (c). Initial indicates the initial anchorage, Final indicates absorbing anchorages, and L, L', R and R' indicate anchorages that can be blocked by input. Coloured circles (b) indicate the range of interaction of the walker to scale. Bottom: Experimental results [27] compared with results from the model. Single-junction circuit: R means a single blockade on the left, R^2 means a two-anchorage blockade on the left, L/R means single blockades on both the left and right. Double-junction circuit: LR means anchorages labelled L and R' are unblocked, so that the walker goes left on the first decision, and right on the second. Each blockade is of two consecutive anchorages. All properties are given at time $T = 200$ min. Finishes is the probability that a walker quenches any fluorophore at time T, given by property $P_{=?}[F^{[T,T]}$ finished $]$. Correct is the probability that a finished walker quenches the correct fluorophore at time T (conditional probability), expressed as the ratio of $P_{=?}[F^{[T,T]}(\text{finished-correct})]$ and $P_{=?}[F^{[T,T]}(\text{finished})]$. Deadlock is the probability for the walker to get stuck prematurely by time T, with no intact anchorage within reach (see Fig.13), given by property $P_{=?}[F^{[T,T]}$ deadlock $]$. Steps indicates the expected number of steps taken by time T, given by property $R_{\{\text{"steps"}\}=?}[C^{\leq T}]$, where "steps" is a reward structure that assigns 1 to each step transition. The red dotted line indicates a leakage transition. The results for the single-junction circuit are obtained using PRISM's fast adaptive uniformisation [10] method to an absolute error of at most 10^{-6}. The results for the dual-junction circuit are generated by checking at least 10^5 paths against the property.

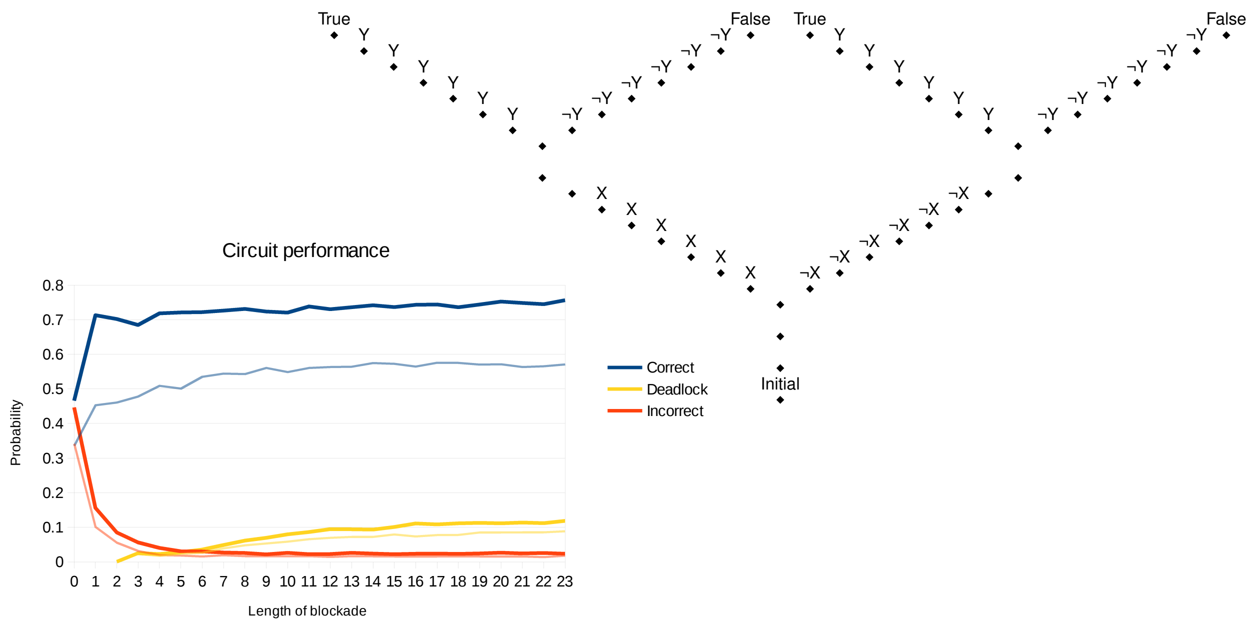


Fig. 11. Probability of reaching an absorbing anchorage by time T, for the double-junction circuit (Fig. 10) with variable blockade length. Shown here is a circuit where the guards of the output tracks of the fork gates are formed by six consecutive blocking anchorages. Deeper circuits are allowed more time to complete the property. Here the depth of the circuit is 19 steps, and we evaluate the properties at time $T = 8$ min $\times$ depth for the input $X =$ true, $Y =$ false. Faint lines are the same properties given at time $T = 4$ min $\times$ depth. For each property we simulate 10^4 paths.

3.3 Model results

Having used experiments on straight tracks and the single-junction circuit to determine the parameters of the model, we use the double-junction circuit shown in Fig. 10(c) to evaluate its quality. The model captures essential features of the walker behaviour and is reasonably well aligned with experimental data. In the model, not all walkers reach an absorbing anchorage by time $T = 200$ min, although the predicted quenching is much higher than observed. The reason for this discrepancy is not easily determined and motivates further study.

We exercise the model by model checking against temporal logic queries aimed at quantifying the reliability and performance of the computation. We note that not all the walkers that finish actually do quench the intended fluorophore. In both the model and the experiments we can identify a difference between paths that follow the side of the circuit (paths LL and RR in Fig. 10), and paths that enter the interior (paths RL and LR): the probability of a correct outcome for the side paths is greater. This is explained by *leakage transitions* between neighbouring paths; an example leakage transition is indicated by a red dotted line in Fig. 10(c). Walkers on an interior path can leak to both sides, but a path that follows the exterior can only leak to one side. This effect can also be shown by inspecting paths. The property

$$P_{=?}[\, \text{correct-path } U^{\leq T} \text{ finished-correct} \,] \tag{2}$$

denotes the probability that a walker stays on the intended path until it quenches the correct fluorophore, by time T. By using this property we can reason about the probability of the walker deviating from the intended path.

For the double-junction circuit in Fig. 10(c), we infer that the probability of staying on the intended path *and* reaching the absorbing anchorage within 200 minutes is 55% for paths LR and RL, and 58% for paths LL and RR. This shows that walkers on interior paths are indeed more likely to deviate from the intended path than walkers on paths that follow the exterior of the circuit.

The double-junction circuit can be improved by reducing the probability of leakage from the intended path. By decreasing the proximity of off-path anchorages and reducing the track length, both the proportion of walkers finishing and correctness are increased (see Fig. 10(d)). The asymmetry between paths (LL, RR vs. LR, RL) also disappears.

Increasing the number of consecutive blockades that form a track guard also results in better performance. Fig. 11 shows a redesign of the circuit from Fig. 10(c); the number of consecutive blockades that constitute a guard have been

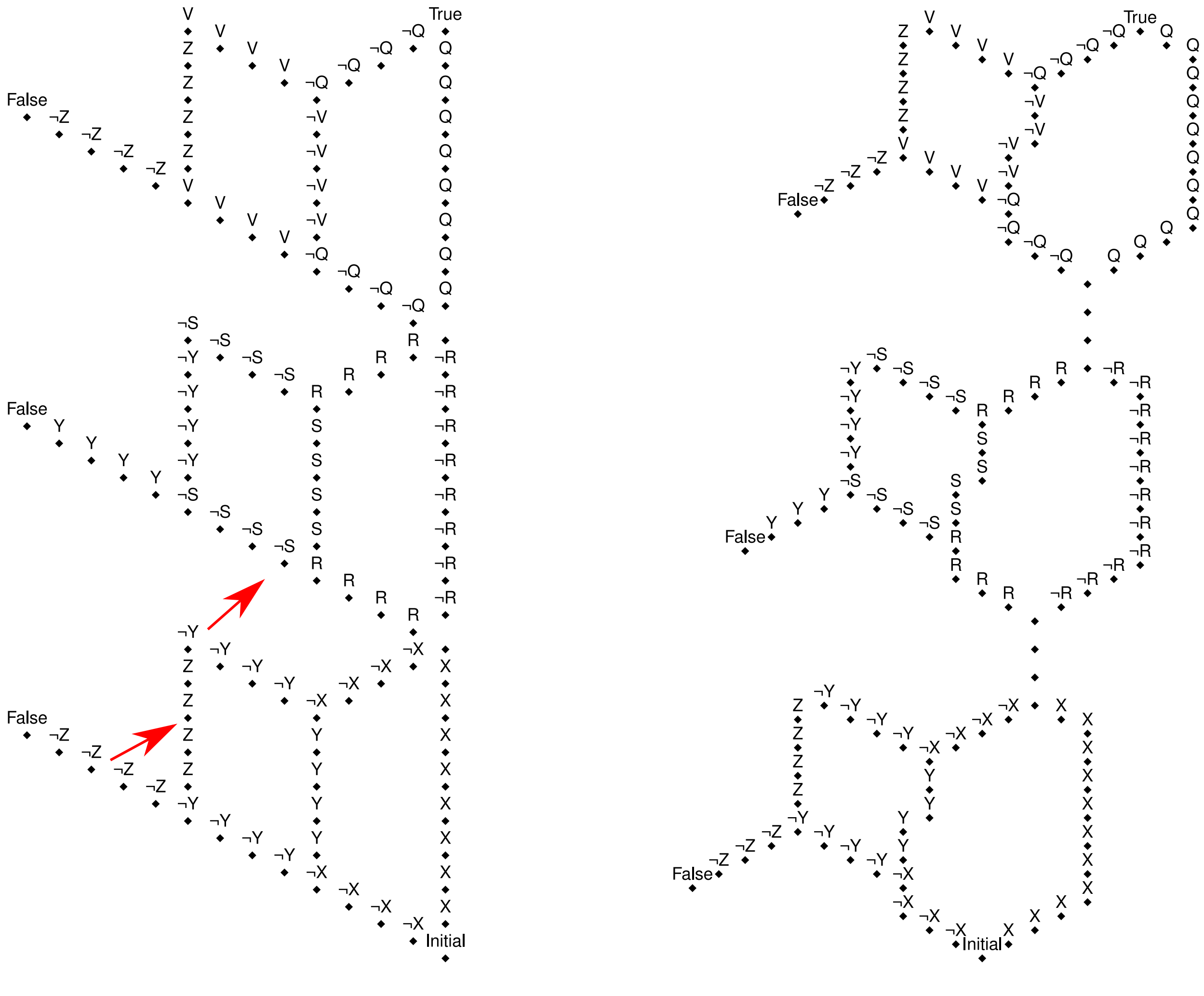


Fig. 12. Disjunction circuits evaluating $(X \vee Y \vee Z) \wedge (\neg R \vee S \vee \neg Y) \wedge (Q \vee \neg V \vee Z)$. Left: Regular version. Right: Improved version. Red arrows indicate leakage transitions.

increased from two to six. Additional guards beyond the second blockage are decreasingly effective at improving the probability that the walker arrives at the correct end-node, while the probability of deadlock increases with the depth of the circuit. Deadlock occurs when a walker is isolated on a non-absorbing anchorage with no intact anchorage in range, which can happen when the walker switches direction after stepping over an intact anchorage, as in Fig. 13. From a computational standpoint deadlock is undesirable, as it is impossible to differentiate a deadlocked process from a live process. Note that the leakage rate from the original double-junction circuit (see red arrow in Fig. 10(c)) is still present in this circuit.

The expected waiting time before the walker steps is, given the model, equal to $\frac{1}{k}$, where k is the rate of stepping. This means that, for unblocked anchorages on the origami that are immediately adjacent to the walker-anchorage complex, the expected waiting time is $\frac{1}{0.009} \approx 2$ minutes. Fig. 11 shows the probability for the walker to finish the computation (correct or incorrect) and the probability for it to deadlock. If the walker would always be able to take a step with the base stepping rate $k_s = 0.009$, then we would expect nearly all walkers to finish computation or deadlock at the given time bound.

The results show a different picture, which is due to the fact that the walker can also step onto anchorages that are further away (*i.e.,* not immediately adjacent on the origami). It is therefore possible to reach a state where all anchorages that are adjacent to the walker on the origami are cut, but one or more uncut (non-adjacent) anchorages are within range of the walker. In this case, the walker is not (yet) deadlocked; however, the stepping rate to these non-adjacent anchorages is either $k_s/50$ or $k_s/100$, depending on their range.

Therefore the expected waiting time before a jump occurs can be as high as $\frac{1}{k_s/100} \approx 185$ minutes. By using the property

$$P_{=?}[F^{[T,T]}(\text{finished} \lor \text{deadlock})] \tag{3}$$

we can compute the probability for the walker to either finish the computation or deadlock at time T. For the circuit in Fig. 11, that probability at time $T = 4$ min $\times$ depth of the circuit $= 76$ min is equal to 57%. If we *remove* from the model the ability for the walker to step onto an anchorage that is not adjacent on the origami, then the walker is much more likely to finish or deadlock, as the value now becomes 90%. This shows that the ability of the walker to step onto anchorages that are not adjacent on the origami degrades the performance of the walker. It is unclear at this point whether the actual walker also suffers from this type of behaviour.

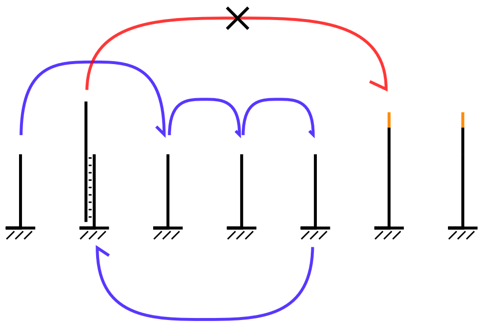


Fig. 13. Walker deadlock. No intact anchorages are within reach.

The performance of PRISM [17] depends on the model checking method. For small tracks, as in Fig. 9, PRISM computes the value of a property to a precision of 10^{-6} within 2 s, using standard uniformisation and the default options for the hybrid engine on common hardware [1]. Properties for the single-junction circuit in Fig. 10 were model checked within 3 s to a precision of 10^{-6} using fast adaptive uniformisation [10]. For large circuits, for which uniformisation may become infeasible, we may use simulation-based methods that estimate the true value of the probability to a given confidence interval by checking a given number of simulated execution paths against the property [12]. For the dual-junction circuit in Fig. 10, single-threaded simulation of 10^5 paths takes 27 s. For the probability of the walker making it to any absorbing anchorage, PRISM reports the 95%-confidence interval to be $90.2\% \pm 0.2\%$ given the input LL. More details about the molecular walkers case study and the analysis methods used can be found at [2].

3.4 Design principles for walker circuits

In this section we show how a composable circuit can be optimised, and establish some design principles that increase the reliability of the circuits. We consider a 3-CNF circuit with three clauses where each clause is a disjunction over three literals.

The first circuit layout in Fig. 12 (Left) is a straightforward embedding of the topology of the circuit. However, this layout results in potential leak reactions, some of which are indicated by red arrows in Fig. 12 (Left). Such transitions should be removed as much as possible from the design. To minimize the probability of leak reactions, we apply the following design principles for the hand optimised embedding in Fig. 12 (Right).

- (Principle 1) Increasing the distance between tracks reduces the probability for the walker to deviate from the intended path. Increasing the distance between tracks can be accomplished by using junctions that are equiangular and by elongating connecting tracks.
- (Principle 2) Increasing the length of the tracks increases the probability for the walker to deadlock. In addition, it increases the expected amount of time until the walker finishes the computation. Therefore, long tracks should be avoided if possible.

Note that these two principles can conflict as increasing the distance between tracks can require that at least one be elongated, thus increasing the rate of deadlock. A pragmatic approach was applied to the design of the layout in Fig. 12 (Right). The output tracks leading to the false terminals of each clause were made equiangular to two other tracks incident to the common fork gate (principle 1). Furthermore, the connecting tracks between clauses were elongated (principle 1), but only modestly in order to avoid an unnecessary increase in the probability of deadlock (principle 2).

We emphasise that the stated design principles are guidelines based on our understanding of the model, and that our software tool was used as a computer aided design (CAD) tool that provides the user with feedback on the performance of circuits. The manually improved circuit is not necessarily optimal, and it is possible that other designs would perform even better. Optimising the circuit does improve its performance, as demonstrated in Table 1.

$T =$	4 min $\times$ depth		8 min $\times$ depth	
%	Normal	Optimised	Normal	Optimised
Correct	42.8	46.7	49.1	54.3
Incorrect	12.9	11.3	18.2	16.7
Deadlock	9.8	7.8	14.0	11.1

Table 1. Performance for the circuits in Fig. 12. In these tests, the time at which the property is evaluated is different for the two circuits and proportional to the size of the circuit. For each of the 128 possible inputs, we simulate 200 paths per property. In this case the depth of the circuit corresponds to the longest possible path, under any variable assignment.

4 Conclusions

The capability for an autonomous DNA walker to navigate a programmable track has been recently demonstrated [27]. We have considered the potential for this system to implement DNA walker 'circuits'. Working from experimental observations, we have developed a simple model that explains the influence of track architecture, blockade failure and stepping characteristics on the reliability and performance of walker circuits. The model can be further extended as more detailed experimental measurements become available. Model checking enables analysis of path properties and quantitative measures such as the expected number of steps, which cannot be established using traditional ODE frameworks. A major advantage of our approach is that circuit designs can be manipulated to study the properties of variant architectures. We have demonstrated the utility of this approach by manually redesigning circuit layouts to decrease the potential for leak reactions to occur. It is not clear if an optimal layout can be determined by an automated algorithm that is also efficient. However, this may not be necessary. As demonstrated in Sections 2 and 3, it is possible to design DNA walker circuits that are easily composable and which compute fundamental Boolean functions. By determining optimal layouts for composable circuits, larger circuits can be made more reliable in a principled manner. An example of this design principle was demonstrated in Section 3 by first optimising a composable 3-CNF clause gadget that resulted in a more reliable circuit overall. An interesting future direction for research would be to explore whether one can efficiently synthesise or evolve DNA circuit layouts.

We have shown that walker circuits can be designed to evaluate any Boolean function. We have also demonstrated the potential for large DNA walker circuits to be built from composable components. One motivation for implementing circuits with a DNA walker system, instead of a DNA strand displacement system (DSD), is the potential for faster reaction times due to spatial locality. However, the walker system we have considered has severely limited potential for parallel circuit evaluation using multiple walkers. As this is not an issue in a DSD, it is the case that this walker system requires exponentially more time to compute certain Boolean functions than a corresponding DSD. As an example, an arbitrary 3-CNF formula of m clauses requires $O(m)$ time to be evaluated by this walker system, whereas the same formula could be evaluated by a DSD cascade in $O(\log m)$ parallel time (*e.g.,* by forming a DSD cascade of a binary reduction tree of the formula). This is not necessarily true of all walker systems. The problem arises in the system under consideration due to the undirected nature of the tracks that are traversed by a walker. For the same reason, it is not clear if circuit redundancy techniques [20] can be used in these walker systems in order to improve their reliability. Other autonomous walker systems with directed tracks have been demonstrated [11, 18, 21, 29], including mechanisms that do not modify the track [21, 29]. It would be interesting to explore the potential of reusable circuits, the information processing capabilities of DNA walkers beyond circuit evaluation, and the potential for multiple interacting walkers. Finally, it would be interesting to explore systems where circuits could be evaluated efficiently by many walkers in parallel, and are amenable to well established design techniques to improve overall circuit reliability [20].

Acknowledgements We thank Masami Hagiya, Jonathan Bath, Alex Lucas, Andrew Phillips and Will Evans for useful discussions. The authors are supported by a Microsoft Research scholarship (FD), ERC AdG VERIWARE, EPSRC Grant EP/G037930/1, NSF grant CCF/HCC-1213127, a Banting Fellowship (CT), a Royal Society - Wolfson Research Merit Award (AJT), and Oxford Martin School.

References

1. Intel i5-2520M, Fedora 3.8.4-102.fc17.x86_64, OpenJDK RE-1.7, PRISM 4.0.3
2. www.prismmodelchecker.org/casestudies/dna_walkers.php

3. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time markov chains. pp. 269–276. Springer (1996)
4. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.: Model-checking algorithms for continuous-time markov chains. IEEE Transactions on Software Engineering 29, 524–541 (2003)
5. Bath, J., Green, S.J., Allen, K.E., Turberfield, A.J.: Mechanism for a directional, processive, and reversible dna motor. Small 5, 1513–1516 (2009)
6. Bath, J., Green, S.J., Turberfield, A.J.: A free-running DNA motor powered by a nicking enzyme. Angewandte Chemie 44, 4358–61 (2005)
7. Bryant, R.E.: Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys 24, 293–318 (1992)
8. Cha, T.G., Pan, J., Chen, H., Salgado, J., Li, X., Mao, C., Choi, J.H.: A synthetic DNA motor that transports nanoparticles along carbon nanotubes. Nature nanotechnology 9, 39–43 (2014)
9. Chandran, H., Gopalkrishnan, N., Phillips, A., Reif, J.: Localized hybridization circuits. DNA Computing and Molecular Programming 6937, 64–83 (2011)
10. Dannenberg, F., Hahn, E.M., Kwiatkowska, M.: Computing cumulative rewards using fast adaptive uniformisation. In: Proc. 11th Conference on Computational Methods in Systems Biology (CMSB'13) (2013)
11. Green, S., Bath, J., Turberfield, A.: Coordinated chemomechanical cycles: a mechanism for autonomous molecular motion. Physical review letters 101, 238101 (2008)
12. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI. Lecture Notes in Computer Science, vol. 2937, pp. 73–84. Springer (2004)
13. Karp, R.M.: Reducibility among combinatorial problems. Springer (1972)
14. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) Formal methods for the design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07). LNCS (Tutorial Volume), vol. 4486, pp. 220–270. Springer (2007)
15. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07). LNCS (Tutorial Volume), vol. 4486, pp. 220–270. Springer (2007)
16. Kwiatkowska, M., Norman, G., Parker, D.: Symbolic Systems Biology, chap. Probabilistic Model Checking for Systems Biology, pp. 31–59. Jones and Bartlett (2010)
17. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. Computer Aided Verification 6806, 585–591 (2011)
18. Muscat, R.A., Bath, J., Turberfield, A.J.: A programmable molecular robot. Nano Letters 11, 982–987 (2011)
19. Muscat, R.A., Bath, J., Turberfield, A.J.: Small molecule signals that direct the route of a molecular cargo. Small 8, 3593–3597 (2012)
20. von Neumann, J.: Probabilistic logics and synthesis of reliable organisms from unreliable components. In: Shannon, C., McCarthy, J. (eds.) Automata Studies. pp. 43–98. Princeton University Press (1956)
21. Omabegho, T., Sha, R., Seeman, N.C.: A bipedal DNA Brownian motor with coordinated legs. Science 324, 67–71 (2009)
22. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-universal computation with DNA polymers. DNA Computing and Molecular Programming pp. 123–140 (2010)
23. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332, 1196–1201 (2011)
24. Seelig, G., Soloveichik, D., Zhang, D., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314, 1585–1588 (2006)
25. Semenov, O., Olah, M.J., Stefanovic, D.: Mechanism of diffusive transport in molecular spider models. Phys. Rev. E 83, 021117 (2011)
26. Wagner, K.: Über eine Eigenschaft der ebenen Komplexe. Mathematische Annalen 114, 570–590 (1937)
27. Wickham, S.F.J., Bath, J., Katsuda, Y., Endo, M., Hidaka, K., Sugiyama, H., Turberfield, A.J.: A DNA-based molecular motor that can navigate a network of tracks. Nature nanotechnology 7, 169–73 (2012)
28. Wickham, S.F.J., Endo, M., Katsuda, Y., Hidaka, K., Bath, J., Sugiyama, H., Turberfield, A.J.: Direct observation of stepwise movement of a synthetic molecular transporter. Nature nanotechnology 6, 166–9 (2011)
29. Yin, P., Yan, H., Daniell, X.G., Turberfield, A.J., Reif, J.H.: A unidirectional DNA walker that moves autonomously along a track. Angewandte Chemie International Edition 43, 4906–4911 (2004)
30. Zhang, D.Y., Winfree, E.: Control of DNA strand displacement kinetics using toehold exchange. Journal of the American Chemical Society 131, 17303–17314 (2009)

Appendix

Proof of Lemma 2

Proof. Let G be any fork gate that is reachable, otherwise it is redundant. When both the left and right output tracks are unguarded, then either track is accessible simultaneously and thus G is not deterministic. Suppose only one output track of G is guarded. If G is only reachable when the single guard evaluates to false, then that guarded output track is never taken and G is trivial (and therefore deterministic by Lemma 1). Otherwise, if G is reachable only when the guard evaluates to true, then both output tracks from G are accessible simultaneously and, as before, G is not deterministic. Finally, suppose the left and right output tracks of G have the same guard. Since G is reachable, then there are only two possibilities: the common guard evaluates to true (both output tracks are accessible simultaneously), or the common guard evaluates to false, resulting in a deadlock. In either case, G is not deterministic. □

Proof of Thereom 1

Proof. *(if — sufficiency)* If G is trivial it is deterministic by Lemma 1. Suppose it is not trivial. Therefore, by assumption, the output tracks have guards which are negations of each other. Thus, there is always exactly one unblocked path from G and it is therefore deterministic.

(only if — necessity) Suppose, by contradiction, that G is deterministic, not trivial, and does not have output track guards that are negations of each other. Since, by assumption, G is not trivial, then by Lemma 2, G cannot be deterministic unless it has *distinct* guards for its left *and* right output tracks. Without loss of generality, let these be G_L and G_R, respectively ($G_L \not\equiv G_R$). Let p be any path that can reach the gate.

Case 1 (p has no guards in common with an output track of G). Consider any variable assignment where p reaches G and set $G_L = G_R = false$. The gate G is still reachable via p since they do not share a guard. However, reaching G results in a deadlock. Contradiction.

Case 2 (p has a guard or its negation in common with one output track of G). Suppose p contains the guard G_L. Consider any variable assignment where p reaches G and therefore $G_L = true$. By assigning $G_R = true$, both output tracks of G are accessible. Contradiction. The case when p contains the guard G_R is symmetric. Similarly, it can be shown that variable assignments exist to ensure G will result in a deadlock, and thus a contradiction, when p contains $\neg G_L$ or $\neg G_R$.

Case 3 (p has a guard or its negation in common with both output tracks of G). Since G is not trivial then (i) p is not guarded by both G_L and $\neg G_R$, and (ii) p is not guarded by both $\neg G_L$ and G_R. By condition (i) and (ii) p must either contain the guards G_L and G_R or the guards $\neg G_L$ and $\neg G_R$. When p reaches G and the former is true, then two output tracks are accessible, and when the latter is true, a deadlock occurs. Both cases result in a contradiction. □

Proof of Thereom 2

Proof. *(if — sufficiency)* If the non-redundant gate is a sink then any path that reaches it from its left input track cannot be extended via its right input track, and vice versa; it is therefore deterministic. Suppose the gate is not a sink and its left input track is guarded by G, the right by $\neg G$ and, prior to reaching those guards, all paths that can reach the left input must traverse a track guarded by G and all paths that can reach the right must traverse a track guarded by $\neg G$. There are two cases to consider. Suppose G evaluates to true. Then, no path can reach the right input since, by the assumption, those paths must traverse a track guarded by $\neg G$ prior to reaching the gate. It follows that all paths that can reach the gate when G evaluates to true must be to the left input. Furthermore, as the right input is guarded by $\neg G$, those paths can only be extended via the output of the gate. The other case (G evaluates to false) is symmetric. Furthermore, as the guards are negations of each other, they cannot simultaneously evaluate to false and cause a potential deadlock.

(only if — necessity) A non-redundant sink is deterministic, so consider the case when the gate is not a sink. By definition of a join gate that is not a sink, it must have two guarded input tracks. Let G_L and G_R be the guards of the left and right inputs, respectively. First, consider all paths that can reach the left input, guarded by G_L. It must

simultaneously be true that (i) none of those paths traverse a track guarded by $\neg G_L$ and (ii) all of those paths traverse a track guarded by $\neg G_R$. If condition (i) is not satisfied, then there would exist a path that traverses a track guarded by $\neg G_L$ and, to extend past the join gate, must traverse another guarded by G_L. As this is not possible, the path would end in a deadlock and the gate would not be deterministic. If condition (ii) is not satisfied then there would exist some path p that does not traverse a track guarded by $\neg G_R$, but may possibly traverse a track guarded by G_R. In this case, there exists a variable assignment where G_R, and all other guards on path p, evaluate to true. With such a variable assignment, path p could be extended via the output track or the right input track. Thus, condition (ii) must also be satisfied, as otherwise the gate would not be deterministic. The conditions (and the argument that both are necessary) when considering all paths that can initially reach the right input, guarded by G_R, are symmetric.

The sufficiency argument shows the gate is deterministic when $G_L \equiv \neg G_R$. It remains to show it is not deterministic otherwise. First, consider the consequence when both G_L and G_R evaluate to true. By condition (ii) all paths leading to the left (right) input traverse a track guarded by $\neg G_R$ ($\neg G_L$). In this case, no paths can reach the gate. Thus, consider when both G_L and G_R evaluate to false. The conditions permit that paths can reach the gate; however, if any path does it will deadlock as both inputs to the gate are blocked. Thus, for all paths that can reach the gate, it will be deterministic only when $G_L \equiv \neg G_R$. $\square$