Model Checking by Random Walk

P@trik Haslum
Department of Computer Science, Linköping University
pahas@ida.liu.se

Abstract

While model checking algorithms are in theory efficient, they are in practice hampered by the explosive growth of system models. We show that for certain specifications the model cheking problem reduces to a question of reachability in the system state-transition graph, and apply a simple, randomized algorithm to this problem.

Introduction

The model checking approach to automated verfication of finite state systems appears the most successfull to date. At the heart of the method are efficient algorithms for deciding if a system model, described as a transition system, satisfies a specification, expressed in temporal logic (Clarke, Emerson, & Sistla 1986). While the theoretical time and space complexities of the algorithms are low order polynomial, they are in practice often time consuming, because the system model grows exponentially with the number of system components.

For certain classes of specification formulas, the model checking problem reduces to a problem of reachability in the state-transition graph. For certain classes of graphs, random walks can be used to create a polynomial algorithm for deciding reachability with siglesided, controllable probability of error. Allthough the time complexity of this algorithm is no better than that of other model checking algorithms (in fact, it is worse) two properties of the random walk algorithm make it interesting for use in verification: First, it uses extremely little space. Since verification is mostly an "off-line" activity, space, which is definitely limited, can be considered a more critical resource than time. Second, it is highly parallelizable, the expected runtime decreasing linearly with the number of parallel processes.

Basics

This section briefly introduces the model checking approach to verification, the theory of Markov chains and the random walk algorithm. For more thorough introductions, see for instance (Clarke, Grumberg, & Peled 1999), (Hoel, Port, & Stone 1972) and (Motwani & Raghavan 1995), respectively.

Markov Chains

Markov chains model discrete state, discrete time, stochastic processes. A Markov chain consists of a finite or countably infinite set of states, $\mathcal{X}$ and a sequence $X_0, X_1, \ldots$ of stochastic variables over $\mathcal{X}$. The distribution of X_i, for all $i \geqslant 1$, is such that it satisfies the *Markov property*:

$$\begin{aligned} &P(X_i = x_i \,|\, X_0 = x_0, \ldots, X_{i-1} = x_{i-1}) \\ &= P(X_i = x_i \,|\, X_{i-1} = x_{i-1}) \end{aligned} \tag{1}$$

i.e. the probability of a certain state x materializing at time i depends only on the state of the process at time $i-1$, and not on its previous history. The probabilities $P(X_i = x_i \,|\, X_{i-1} = x_{i-1})$ are called the *transition probabilities*. If

$$\begin{aligned} &P(X_i = x_i \,|\, X_{i-1} = x_{i-1}) \\ &= P(X_j = x_j \,|\, X_{j-1} = x_{j-1}) \end{aligned} \tag{2}$$

for all $i, j \geqslant 1$, *i.e.* the transition probabilities are independent of time, the chain is said to be *stationary*. We will in sequel consider only stationary chains, and can therefore abbreviate $P(X_i = x \,|\, X_{i-1} = y)$ as $P_{x,y}$. A probability distribution over the state space $\mathcal{X}$ of a Markov chain is called a distribution of the chain. A stationary Markov chain is characterized by the transition probabilities and the *initial distribution* $\pi_0(X_0 = x)$, $x \in \mathcal{X}$.

Denote by $\mathbf{E}_\pi(\cdot)$ expectation taken under the assumption that the initial distribution is π. We will also write $\mathbf{E}_x(\cdot)$ for the case when

$$\pi(x') = \begin{cases} 1 & \text{when } x' = x \\ 0 & \text{elsewhere} \end{cases}$$

i.e. the initial distribution is concentrated to a single state x. The *hitting time* of a state $x \in \mathcal{X}$ is defined as $T_x = min\{n > 0 \,|\, X_n = x\}$, *i.e.* T_x is a stochastic variable denoting the first time that the chain enters state x. The *mean return time* of x is defined as $m_x = \mathbf{E}_x(T_x)$. Let $r_{x,y} = P(T_y < \infty \,|\, X_0 = x)$, *i.e.* $r_{x,y}$ is the probability of a chain starting in state x ever reaching state y. A set of states $\mathcal{C} \subset \mathcal{X}$ is *closed* iff $r_{x,y} = 0$ for all $x \in \mathcal{C}$ and $y \notin \mathcal{C}$. A closed set $\mathcal{C}$ is *irreducible* iff $r_{x,y} > 0$ for all $x, y \in \mathcal{C}$.

A distribution π over $\mathcal{X}$ is said to be *stationary* iff

$$\pi(y) = \Sigma_{x \in \mathcal{X}} \pi(x) P(x, y) \tag{3}$$

That is, if the state of a chain at time n, X_n, is drawn at random according to a stationary distribution π, the distribution of X_i for all $i \geq n$ will also be π. It can be shown that if $\mathcal{X}$ is finite, closed and irreducible, the chain has a unique stationary distribution and this is given by

$$\pi(x) = \frac{1}{m_x} \tag{4}$$

Random Walks on Graphs

A graph, G, consists of a (finite) set of vertices, V, and a binary edge relation E over V. For a vertex $v \in V$, the *neighbourhood* of v is defined as $N(v) = \{v' \,|\, (v_i, v') \in E\}$ and the *outdegree* of v is defined as $d_{out}(v) = |N(v)|$. The indegree of v is $d_{in}(v) = |\{v' \,|\, (v', v) \in E\}|$. When $d_{in}(v) = d_{out}(v)$ for all v we call the graph *Eulerian* and write only $d(v)$. A special class of Eulerian graphs are graphs with symetric edge relation, *i.e.* for all v, v' if $(v, v') \in E$ then also $(v', v) \in E$. Such graphs are called *undirected*. A *closed component* of G is a graph consisting of a subset of vertices $C \subseteq V$ and the edge relation restricted to C such that there exists no path[1] from any vertex $v \in C$ to any vertex $v' \notin C$. A component is *strongly connected* iff there exists a path from v to v' for every $v, v' \in C$.

A *walk* on G (of length n) is a sequence of vertices $v_0, v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E$, for $i = 0, \ldots, n-1$. The walk is *random* iff each v_{i+1} is drawn at random with equal probability from the neighbourhood of v_i. Let G be a graph consisting of a single strongly connected component, and denote by $T_{v,v'}$ the length of a random walk starting at vertex v and ending at the first time vertex v' is reached. The following result by Aleliunas *et.al.* (1979) is at the heart of the random walk algorithm.

$$\mathbf{E}(T_{v,v}) = \frac{2|E|}{d(v)} \tag{5}$$

[1] A *path* is a walk without repeated vertices.

A random walk on G forms a Markov chain with state space V and transition probabilities $P(v, v') = \frac{1}{d(v)}$ when $v' \in N(v)$, and $P(v, v') = 0$ elsewhere. It is easily verified that the distribution

$$\pi(v) = \frac{d(v)}{2|E|}$$

satifies equation (3), that is, is a stationary distribution of the chain. Because the state space is finite, closed and irreducible, the stationary distribution is unique and therefore

$$m_v = \frac{1}{\pi(v)} = \frac{2|E|}{d(v)}$$

which, since $m_v = \mathbf{E}_v(T_v)$ is only another way of saying $\mathbf{E}(T_{v,v})$, yields (5). Furthermore, for any pair of vertices, v, v' in G,

$$\mathbf{E}(T_{v,v'}) \leqslant |V||E| \tag{6}$$

Because G is strongly connected, there is a path $v_0, v_1, \ldots, v_n$ from v to v', and because a path contains no repeated vertices, $n \leqslant |V|$. Whenever the walk is in a vertex v_i which lies on this path, the next step in the walk will with probability $\frac{1}{d(v_i)}$ be a step "in the right direction", *i.e.* to the next vertex in the path. If any other neighbour of v_i is chosen, the walk will after on average $\frac{2|E|}{d(v_i)}$ steps return once more to v' and try again, and after on average $\frac{1}{2}d(v_i)$ tries it will chose the "right" neighbour of v_i. Thus, the expected number of steps needed to take one step along the path is $|E|$, and (6) follows.

We can now describe the random walk algoritm: Let G be a graph consisting of closed, strongly connected components[2] and let v and v' be two vertices in G. Choose the probability of error, $0 \leq \epsilon \leq 1$, and make a random walk of length

$$\frac{1}{\epsilon}|V||E|$$

on G, starting in vertex v[3]. Then,

(*i*) if the random walk reaches v', then there exists a path from v to v';

(*ii*) if the random walk does not reach v', then with probability at least $1 - \epsilon$ there is no path from v to v'.

[2] It is in fact sufficient that the component to which v belongs is closed and strongly connected.

[3] To be exact, we should in place of $|V|$ and $|E|$ have the number of vertices and edges, respectively, in the component in which the walk begins.

Claim (i) is trivial; if the random walk reaches v', then there has to exist a path from v to v'.

To show claim (ii), assume that there exists a path from v to v'. Recall that $T_{v,v'}$ denotes the stochastic variable telling us how many steps the random walk will take before reaching v'. Because $T_{v,v'}$ is a random variable taking only non-negative values it holds for any $k > 0$ that

$$P(T_{v,v'} \geq k\mathbf{E}(T_{v,v'})) \leq \frac{1}{k} \tag{7}$$

(this is known as Markovs inequality). Therefore

$$P(T_{v,v'} \geq \frac{1}{\epsilon}\mathbf{E}(T_{v,v'})) \leq \epsilon$$

and from equation (6) we have that

$$P(T_{v,v'} \geq \frac{1}{\epsilon}|V||E|) \leq \epsilon.$$

That is, the probability that the random walk must take more than $\frac{1}{\epsilon}|V||E|$ steps to reach v' is less than ϵ. Thus, if the random walk does not reach v', there is with probability at least $1 - \epsilon$ no path from v to v'.

Lastly, we need to characterise the class of graphs to which the algorithm is applicable. The following theorem provides only a sufficient condition, but one more readily checked than the requirement that the closed component containing the initial vertex is strongly connected.

Theorem 1
A Eulerian graph G consists of closed, strongly connected components.

Proof: By induction on the number of vertices, $|V|$. Note first that an edge (v, v) (a *loop*) adds one to both $d_{in}(v)$ and $d_{out}(v)$, and thus the presence or absence of loops does not affect whether the graph is Eulerian or not. Neither can a loop affect whether a component is strongly connected or closed. We therefore can assume that G is free of loops.

For $|V| = 1$, G consists obviously of a single, strongly connected component. Assume any Eulerian graph with $|V| = n$ vertices is divisible into k closed, strongly connected components, and consider a graph G' with $|V| = n + 1$. We can think of G' as being constructed from the previous graph by adding a vertex v_{n+1} and a number of edges. If no edges are added, v_{n+1} is closed and the graph consists of $k + 1$ closed, strongly connected components.

If we add one edge to v_{n+1} from one of the components, say C_i, we must also add an edge from v_{n+1}, since the graph should be Eulerian. If this edge goes back to C_i, $C_i \cup \{v_{n+1}\}$ still forms a closed and strongly connected component. If the edge goes to another component, C_j, this will cause an imbalance in degrees. For, the sum

$$\Sigma_{v \in C_i} d_{out}(v) - d_{in}(v)$$

was 0, the graph being Eulerian, but is now 1 since we have added an edge from C_i. Similarly, the corresponding sum for C_j will now be -1. The only way to right the imbalance is to add an edge from C_j to C_i. Adding an edge to C_i from another component, or from C_j to another component, only moves the excess, or shortage, elsewhere (where we can of course resolve it by adding an edge between those two components, or by connecting the edges to C_i and from C_j to the same component). In the resulting graph, $C_i \cup C_j \cup \{v_{n+1}\}$ (and possibly other components as well) form an closed, strongly connected component. □

Model Checking

The system model employed in model checking is a *transition system* (also often described as a Kripke structure). A transition system consists of a set of states, $\mathcal{S}$, a binary *transition relation*, R, on $\mathcal{S}$, and a labeling function, L, which maps each state to a set of atomic propositions that hold in the state. A subset of states, $S_0 \subseteq \mathcal{S}$ are designated as *initial states*. Since a transition system is, in fact, a labeled graph, we sometimes refer to it as a state-transition graph.

The specification language is most commonly some form of temporal logic. We describe briefly *Computation Tree Logic* (CTL), which is used by the system described in the next section.

A CTL formula is built from the atomic propositions of the system model, propositional connectives and the temporal operators **AX**, **EX**, **AF**, **EF**, **AG**, **EG**, **AU** and **EU**. A CTL formula is evaluated with respect to a transition system, M, and a state $s \in \mathcal{S}$, as follows:

- A formula ψ not containing any temporal operator is evaluated as a normal propositional logic formula in $L(s)$.
- A formula of the form $\mathbf{EX}\psi$ is true iff ψ holds in some state s' such that $(s, s') \in R$.
- A formula of the form $\mathbf{AG}\psi$ is true iff ψ holds in every state of every walk on the state-transition graph of M that starts from s.
- A formula of the form $\mathbf{E}(\varphi\mathbf{U}\psi)$[4] holds iff there is a path from s to some state s' in which ψ holds, such that φ holds in every state along the path up to s'.

[4]The **AU** and **EU** operators are written in this notation.

The remaining operators can be expressed in terms of these three by means of equalities. In particular, $\mathbf{EF}\psi \equiv \mathbf{E}(\text{TRUE}\mathbf{U}\psi)$.

The model checking problem is the problem of deciding if the specification formula holds in every initial state $s \in S_0$ of the system model. The time, and space, complexity of the best known model checking algorithm for CTL is linear in the product of the sizes of the system model and the specification. The use of symbolic representation techniques (Burch *et al.* 1992) can reduce the space required, but in the worst case it is still linear.

Model Checking by Random Walk

From the definition of the interpretation of CTL formulas, it is clear that the model checking problem for certain formulas reduces to a question of reachability in the state-transition graph.

(*i*) A formula $\mathbf{EF}\psi$, where ψ contains no temporal operators, holds in state s if *some* state s' in which ψ holds is reachable from s.

(*ii*) A formula $\mathbf{AG}\psi$, where ψ contains no temporal operators, holds in state s if *no* state s' in which ψ holds is reachable from s.

(*iii*) A formula $\mathbf{E}(\varphi\mathbf{U}\psi)$, where neither φ nor ψ contains temporal operators, holds in state s if *some* state s' in which ψ holds is reachable from s in the restriction of the state-transition graph consisting only of states in which φ holds.

When the state-transition graph is Eulerian, questions of reachability can be decided by the random walk algorithm. Systems with a non-singleton set of initial states can be dealt with by extending the transition relaition with a complete set of edges on the set of initial states. Clearly then, a target state is reachable from some initial state if and only if it is reachable from any initial state.

Representation and Complexity

For even moderately sized system models, it is infeasible to represent the state-transition graph explicitly. Therefore, we assume that the state set, $\mathcal{S}$, is the set of interpretations over a collection of p propositions, which also gives a natural definition of L, and that the transition relation, R, is represented by a set of functions $t_i(s)$, $i = 1 \ldots k$, from states to successor states (another common representation of R is as a logic formula $\tau(s, s')$ which holds when and only when s' is a successor of s).

The random walk algorithm requires knowledge of $|V|$ and $|E|$. With the assumed representation, we have that $|V| = 2^p$, that $|E| \leqslant |V|^2$ and that $|E| \leqslant k|V|$ (k is typically much smaller than $|V|$). Because these are overestimates of $|V|$ and $|E|$, we can apply them in the random walk algorithm without increasing the probability of error. This makes the time complexity of the algorithm

$$\frac{1}{1-\epsilon} k 2^{p+1}$$

The space requirement is only what is required to store a single state, p, and $O(\epsilon p)$ bits to represent a counter.

Verifying Symmetry

The random walk algorithm is applicable when the state-transition graph is Eulerian, but not necessarily when the graph is not. Deciding if a state-transition graph, in the representation we have assumed, is Eulerian requires time on the order of $k|V|^2$ (to determine the indegree of a state s we have to enumerate all states and count the number of them that have s as a successor).

However, deciding if the state-transition graph is undirected can be done in the course of the random walk, with a time overhead of only k. Whenever a successor $t_i(s)$ of s is chosen to be the next state in the walk, check that for some $1 \leqslant j \leqslant k$, $t_j(t_i(s)) = s$.

Experiments

We have implemented a model checker based on the random walk algorithm. It accepts a system description in a restricted form of the syntax used by the SMV model checker (SMV) and a specification formula in CTL of one of the three forms listed in section . We have used it for comparisons with SMV, and to explore some properties of random walks.

The Semaphore Example

In the semaphore example, adapted from (McMillan 1992), the system consists of two concurrent processes, `P1` and `P2`, each of which can be in one of the four states `idle`, `entering`, `critical` and `exiting`, and a boolean variable `semaphore`. Each process is initially `idle`, and can change from `idle` to `entering`, and from `entering` to `critical`, but only if `semaphore` is `false`. When a process changes to state `critical`, it also changes `semaphore` to `true`. From state `critical` each process can change to `exiting`, and from `exiting` back to `idle`, at the same time changing `semaphore` back to `false`. The state-transition graph is shown in figure 1. The graph is not Eulerian, but the closed component that contians the initial state is strongly connected.

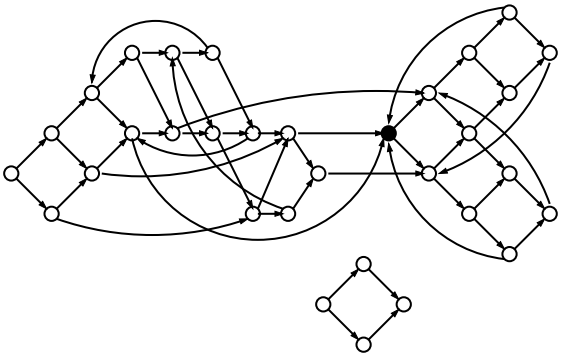

Figure 1: State-Transition graph for the semaphore system. The initial state is drawn solid.

We ran the random walk checker and SMV on the specification formulas

$$\varphi_1 \quad : \quad \mathbf{AG}\neg(\texttt{P1.critical} \wedge \texttt{P2.critical})$$

$$\varphi_2 \quad : \quad \mathbf{AG}\neg((\texttt{P1.entering} \wedge \texttt{P2.exiting}) \vee (\texttt{P1.exiting} \wedge \texttt{P2.entering}))$$

The first formula holds in the initial state. For a probability of error $\epsilon = 0.1$, the estimated number of steps required is 51200, which the random walk checker completes in slightly less than a second[5]. The second formula is false in the initial state, and the system repeatedly finds a counterexample in time too small to measure. SMV also solves both problems in a fraction of a second.

The Tile Puzzle

The tile puzzle, more commonly known as the "8-puzzle" or "15-puzzle" for the special cases $m = n = 3$ and $m = n = 4$ respectively, is a standard example in AI. The puzzle consists of $m \times n$ squares ($m, n \geqslant 2$) and $mn - 1$ labeled tiles positioned on the squares. Thus, one square is left empty, and a tile in an adjacent square can be moved into the empty square (horizontally or vertically, not diagonally). While the puzzle is a "toy example", it happens to have some convenient properties:

(*i*) The state-transition graph is undirected and contains $mn!$ states, divided into two closed components.

(*ii*) There exists an effective procedure to decide whether two given states belong to the same component.

We ran the example with $m = n = 3$, a fixed initial state and a specification formula of the form

$$\mathbf{EF}(\texttt{square}_1 = t_1 \wedge \ldots \wedge \texttt{square}_9 = t_9)$$

[5]The experiments where carried out on a SUN Ultra 10

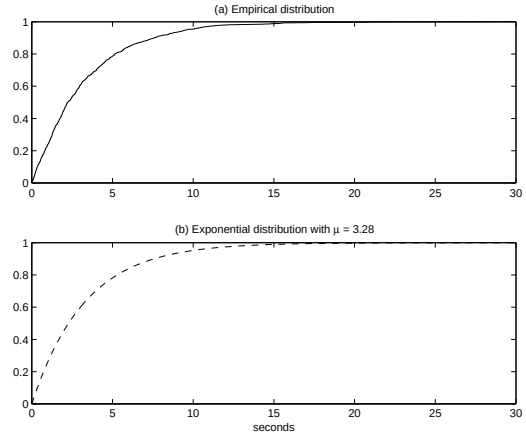

Figure 2: (a) Empirical distribution of 1000 sample runtimes (b) Exponential distribution with best-fit parameter

where, $t_1, \ldots, t_9$ is some permutation of 0..8 (the tile labels, 0 representing "blank"), *i.e.* with the question of whether a particular state is reachable. Figure 2(a) shows the empirical distribution of 1000 samples of the random walk checkers runtime for a true specification. For $\epsilon = 0.1$, the estimated number of steps required is $3.04{\cdot}10^{21}$. The system completes approximately $1.3{\cdot}10^6$ steps each second, which would place the runtime for a false specification around $2.34{\cdot}10^{15}$ seconds (around 30 million years). SMV, by comparison, solves the problem in 30 minutes and 20 seconds, regardless of whether the specification holds or not.

Reducing Expectations

Figure 2 easily leads to the conjecture that the runtime, and hence also the number of steps, $T_{v,v'}$, is exponentially distributed. The exponential distribution function is

$$F(x) = 1 - e^{-\frac{x}{\mu}} \tag{8}$$

where μ is a parameter of the distribution. From a set of 1000 samples of $T_{v,v'}$ (using a somewhat different system model), we compute a 99% confidence interval, $3.9{\cdot}10^5 - 4.6{\cdot}10^5$, and a most likely estimate of $4.2{\cdot}10^5$ for μ. This means that with probability .99, the true value of μ lies in this interval and that the most likely value is $4.2{\cdot}10^5$, assuming the variable is in fact exponentially distributed.

The expected value of an exponentially distributed variable is μ. From the estimate of $\mu = 4.2{\cdot}10^5$, we compute an expected 90th percentile of $9.7{\cdot}10^5$. This means that if we run the random walk checker on instances of the 3×3 tile puzzle with the initial and goal states in the same component, and a step limit of

$9.7{\cdot}10^5$ (which takes a little over a second to walk), it should fail to find a solution in about 10% of the cases.

By property (*ii*) above, we can randomly generate solvable problem instances in a way which is independent of any bias the random walk may have. We ran the system on five batches of 1000 such instances each, with a step limit of 970000. The result was failure in $11\% - 13\%$ of the cases, indicating that the estimate of μ is a little low. A possible explanation for this is that the estimate is based on samples from only one problem instance, which may have happened to be an easy one.

Discussion

The discrepancy between the computed number of steps needed for a random walk to reach a specific state and the empirical results indicates that the derived expectation is an overestimate. This is not only because we overestimate $|V|$ and $|E|$; even if we apply the correct values for the 3×3 tile puzzle, $|V| = 181440$ and $|E| = 483840$, the step limit for $\epsilon = 0.1$ is $8.7{\cdot}10^{11}$. Equation (6) is the expected number of steps for walking *one* particular path; it assumes the walk does not in its wanderings encounter the sought vertex "by accident".

Denote by $H_n(x, y)$ the expected number of visits to state y in the first n steps of a Markov chain starting in state x. For a recurrent state y, it can be shown that

$$\lim_{n\to\infty} \frac{H_n(x,y)}{n} = \frac{r_{x,y}}{m_y} \tag{9}$$

Since in the random walk chain, $r_{v,v'} = 1$ for all vertices v, v', this places the expected number of steps until the probability of encountering any reachable vertex v is $\frac{1}{2}$ at m_v, *i.e.* $\frac{2|E|}{d(v)}$, which is a significantly lower number. Unfortunately, this expectation is asymptotic. We have no indication of how large n must be for it to hold with probability ϵ. The estimate of $4.2{\cdot}10^5$ derived above is far greater than $\frac{2|E|}{d(v)} = 120960$ (the degree of the sought vertex is 4).

Parallel Walks

Suppose, for a given graph G and vertices v, v', we run r random walks, all starting in v and ending when v' is reached. Let $T^i_{v,v'}$ denote the number of steps taken by the ith walk, and let $T^{\min}_{v,v'}$ denote the minimum of $T^i_{v,v'}$, for $i = 1 \ldots r$. Assume each $T^i_{v,v'}$ is exponentially distributed, with expectation μ. Then,

$$P(T^i_{v,v'} > x) = e^{-\frac{x}{\mu}}$$

and therefore

$$P(T^{\min}_{v,v'} > x) = (e^{-\frac{x}{\mu}})^r = e^{-\frac{x}{\frac{\mu}{r}}}$$

i.e. $T^{\min}_{v,v'}$ is exponentially distributed with an expected value of $\frac{\mu}{r}$.

Thus, if we run several walks in parallel, stopping when the first reaches v', the expected runtime decreases linearly with the number of parallel walks. In other words, we can trade processors for time without loss.

Related Work

The application of random walks to model checking is not new. For instance the SPIN model checking system (SPIN) can perform "random simulation" of the system model, but does not seem to use such simulations for actual verification. On the theoretical side, Mihail and Papadimitrou (1995) have shown a severely restricted class of system models to have state-transition graphs that are so called "rapid expanders". Such graphs are particularly well suited for exploration by random walks.

The above sketched approach to model checking seems to have more in common with statistical methods used in software testing (Miller *et al.* 1992). Inputs to the program are chosen at random according to a distribution that corresponds to normal use of the program and from the number of successfully executed test cases, a probability of program failure can be estimated. In the case of reactive programs, the kind normally dealt with in model checking, a "program input" corresponds to a combination of state and event, which can lead to a prohibitively large input space. Markov chains have also been used to model the "expected use" of the system, *i.e.* to describe the test case distribution (Whittaker & Thomason 1994).

It should also be mentioned that the random walk algorithm is far from the only method of deciding reachability in graphs. Reachability questions for arbitrary graphs can be decided in space $O(\log^2 n)$ by Savitch's algorithm (1970), and for undirected graphs in space $O(\log^{1.5} n)$ (Nisan, Szemerédi, & Widgerson 1992). Search algorithms such as BFS and IDA* have lower time complexities, but require more space.

Conclusions and Questions for Further Research

The preliminary experiments we have reported indicate that the random walk algorithm may, in cases where it is applicable, be an efficient alternative to deterministic model checking algorithms. There are, however, two central obstacles to practical application of the method:

First, it manages only certain kinds of specifications. This restriction may be possible to circumvent, since verification of any LTL formula can be reduced to a question of reachability, using a tableuax construction (Gerth *et al.* 1995). More troublesome is the fact that it applies only to certain classes of system models. Whether realistic verification problems can be made to fit within these restrictions is an open question.

Second, the theoretical bound on the expected value of $T_{v,v'}$, that is, the number of steps needed to reach a target state from the initial state, has to be brought closer to the actual value. The experiments show this to be far lower than the derived bound of $|V||E|$, though not as low as equation (9) promises. Also, since the expectation depends only on the closed component in which the walk starts, methods for estimating the size of this component could aid in reducing the bound.

Lastly, the implemented random walk model checker is available at `http://www.ida.liu.se/~pahas/stocplan.html`.

References

Aleliunas, R.; Karp, R.; Lipton, R.; Lovasz, L.; and Rackoff, C. 1979. Random walks, universal traversal sequences and the complexity of maze problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, 218 – 223.

Burch, J.; Clarke, E.; McMillan, K.; Dill, D.; and Hwang, L. 1992. Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98(2):142 – 170.

Clarke, E.; Emerson, E.; and Sistla, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244 – 263.

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model checking.* MIT Press.

Gerth, R.; Peled, D.; Vardi, M.; and Wolper, P. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th Workshop on Protocol Specification, Testing and Verification.* North-Holland.

Hoel, P.; Port, S.; and Stone, C. 1972. *Introduction to stochastic processes.* Houghton Mifflin.

McMillan, K. 1992. The SMV system. Included in the SMV distribution.

Mihail, M., and Papadimitrou, C. 1995. On the random walk method for protocol testing. In *Computer Aided Verification '95.*

Miller, K.; Morell, L.; Noonan, R.; Park, S.; Nicol, D.; Murrill, B.; and Voas, J. 1992. Estimating the probability of failure when testing reveals no failures. *IEEE Transactions on Software Engineering* 18(1).

Motwani, R., and Raghavan, P. 1995. *Randomized Algorithms.* Cambridge University Press.

Nisan, N.; Szemerédi, E.; and Widgerson, A. 1992. Undirected connectivity in $\wr(\log^{1.5} n)$ space. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, 24 – 29.

Savitch, W. 1970. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* 4:177 – 192.

SMV homepage. `http://www.cs.cmu.edu/~modelcheck/smv.html`.

SPIN homepage. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

Whittaker, J., and Thomason, M. 1994. A markov chain model for statistical software testing. *IEEE Transactions on Software Engineering* 20(10).