Generating Strategies from Transient Test Graphs*

Andreas Blass¹, Lev Nachmanson², and Margus Veanes²

¹ University of Michigan, Ann Arbor, MI, USA
ablass@umich.edu

² Microsoft Research, Redmond, WA, USA
levnach@microsoft.com, margus@microsoft.com

Abstract. Model based test generation for reactive systems reduces in some cases to generation of a strategy for a reachability game played on a test graph. The test graph is a finite approximation of the model. We consider an important subclass of test graphs, called transient test graphs, where the distinguished goal vertex of the graph can be reached from all vertices in the graph. Transience arises from the practical requirement that all tests must end in a distinguished goal vertex. We prove that a strategy that minimizes the expected costs of reaching the goal vertex exists if and only if the graph is transient, and we show how to find such a strategy by solving a linear program. We discuss how the problem arises in the context of model based testing of reactive systems with the SpecExplorer tool that has been developed at Microsoft Research.¹

1 Introduction

In industrial software testing it is often infeasible to specify the behavior of a large reactive system as an explicit finite transition system. To avoid explicit representation of transitions one can model the system behavior in a model program, and then invocations of methods-procedures of the model called actions represent the system transitions. This approach is taken by the SpecExplorer tool developed in the Foundations of Software Engineering group at Microsoft Research, where the model program is written in a high level specification language AsmL [17] or Spec# [5]. In order to distinguish behavior that a tester has full control over, from behavior that can only be observed about the IUT (Implementation Under Test), the actions of a model program are partitioned into controllable and observable actions. The model corresponding to a model program $P$ is a transition system that is a complete unwinding or expansion of $P$; the model is typically infinite because of unbounded data structures and objects used in $P$. Semantically our notion of models coincides with interface automata as a game model of reactive systems [10].

¹ Much of the research reported here was done while the first author was a visiting researcher at Microsoft Research.
The primary task of the SpecExplorer tool is to generate test cases from models. Typical users of SpecExplorer deal daily with models of reactive systems. For example the upcoming version of Windows will contain the Indigo system that provides a unified programming model and communications infrastructure for distributed applications [6]. Test cases are in this setting game strategies that are played according to the rules of the model, where transitions labeled by controllable actions are moves of the test tool (TT) and transitions labeled by observable actions are moves of the IUT. For large or infinite models the first phase of the test case generation task is to generate a finite approximation of the model; we call this phase FSM generation. During FSM generation various restrictions on possible model states are introduced by the tester to limit the size of the state space to a feasible range. The output of the FSM generation phase is a normalized finite (under)approximation of the model that we call a test graph. The vertices of a test graph are disjointly partitioned into states, choice points, and a distinguished goal vertex. From states, TT can make a move and from choice points IUT can make a move. All actions are associated with costs, and all edges emanating from a choice point are associated with positive probabilities that total to 1.

In [20] several algorithms are described that generate optimal length-bounded strategies from test graphs, where optimality is measured by minimizing the total cost while maximizing the probability of reaching the goal, if the goal is reachable. The problem of generating a strategy with optimal expected cost is stated as an open problem in [20]. The main result of this paper is that strategies that are optimal in this sense can be computed for test graphs if and only if the goal is reachable from every vertex. We call such test graphs transient. Transience is a very natural requirement for test graphs when used as a basis for test case generation for reactive systems. Often IUT does not have a reliable reset that can bring it to its initial state, and thus re-execution of tests is only possible from certain states from which reset is allowed. A typical example is when IUT is a multithreaded API. As a result of a move of TT (a controllable action) that starts a thread in the IUT, the thread may acquire shared resources that are later freed. The test should not be finished before the resources have been freed. FSM generation does not always produce transient test graphs, because of premature termination of state space exploration. However, there are efficient algorithms that can be used to prune the test graph to make it transient [8, 9].

The remainder of this paper is structured as follows. In Section 2 we prove the main result of the paper by formulating the problem of generating optimal strategies for transient test graphs as an optimization problem in linear programming; we also discuss an approximate solution to the problem. In Section 3 we show how the problem arises in model-based testing with SpecExplorer. In Section 4 we discuss related work, and finally we mention some open problems in Section 5.
2 Strategy calculation

We use a modification of the definition of a test graph in [20] to describe non-deterministic systems. A test graph $G$ has a set $V$ of vertices and a set $E$ of directed edges. The set of vertices $V$ consists of states ($States$), choice points ($Choices$), and one goal vertex ($g$). $States$, $Choices$ and $\{g\}$ are mutually disjoint sets. No edge exits from the goal vertex. There is a probability function $p$ mapping edges exiting from choice points to positive real numbers such that, for every choice point $u$,

$$\sum_{(u,v) \in E} p(u, v) = 1.$$  \hspace{1cm} (1)

Notice that this implies that for every choice point there is at least one edge starting from it, and we assume the same for states. Finally, there is a cost function $c$ from edges to positive reals. One can think about the cost of an edge as, for example, the time for IUT to execute the corresponding function call.

Formally we denote by $G$ the tuple

$$(V, E, States, Choices, g, p, c).$$

We suppose that for every $u, v \in V$ there is at most one edge with $u$ as a source and $v$ as a target. Thus every edge is given by its two endpoint vertices and we may assume without loss of generality that $E \subset V \times V$. It is also convenient to assume that the cost function is defined on all elements on $V \times V$ by letting $c(i, j) = 0$ for every $(i, j) \notin E$. We show later in 2.4 how to transform a graph to avoid multiple edges between two vertices.

2.1 Reachability game

Given a test graph $G$, a reachability game over $G$ is given by a start vertex $s \in V$. We denote a reachability game by $R(s)$. There are two players in the game, a testing tool ($TT$) and an implementation under test ($IUT$). The current state of the game is given by a marked vertex $v$. Initially $v = s$. If $v$ is a state then $TT$ chooses an edge exiting $v$ and the target node of the edge becomes the new $v$. If $v$ is a choice point then $IUT$ randomly picks an edge exiting $v$ according to the probability distribution given by $p$ and moves the marker from $v$ to the end of the edge. $TT$ wins if the marker reaches $g$. With every move $e$ the testing tool $TT$ pays $c(e)$ and the total amount spent by $TT$ is the game cost or it is undefined if the goal is never reached.

2.2 Strategies for a reachability game

A strategy for a test graph $G$ is a function $S$ from $States$ to $V$ such that $(u, S(u)) \in E$ for every $u \in States$.

A strategy $S$ is used by $TT$ to choose an edge $(v, S(v))$ when the marker stands on $v$. We define a reachability game $R(v, S)$ to be the game $R(v)$ where
TT follows the strategy $S$. We would like to measure reachability game strategies and compare them. To this end for every strategy $S$ we define a vector $M_S$ with coordinates indexed by $V$ such that for each $v \in V$ the value $M_S[v]$ is the expected cost of the game $R(v, S)$. If the expected cost is undefined we say that $M_S[v]$ equals $\infty$. We say that $M_S$ is defined if $M_S[v]$ is defined for all $v$. If, for example, $c$ reflects the durations of transition executions then $M_S$ reflects the expected game duration.

We call a strategy $S$ optimal if $M_S[v] \leq M_{S'}[v]$ for every strategy $S'$ and for every $v \in V$, or, writing it more concisely, $M_S \leq M_{S'}$. It is assumed here that $x \leq \infty$ for all real numbers $x$.

Imagine that we know the vector $M$ of the smallest expected costs. Then we can immediately construct the optimal strategy by choosing $S(u) = v$ such that $c(u, v) + M[v] = \min_{(u, w) \in E} \{ c(u, w) + M[w] \}$.

Our plan is to show that $M$ is an optimal solution of some linear program and to give necessary and sufficient conditions for existence of a solution. First we have establish some facts about strategies and expected costs.

Let us suppose that the test graph vertices are enumerated; $V = \{0, 1, ..., n-1\}$ and the goal vertex $g = 0$. Consider a fixed test graph $G$ and a strategy $S$ over $G$. We denote by $P_S$ the following $n \times n$ matrix of non-negative reals:

$$P_S[i, j] = \begin{cases} p(i, j), & \text{if } i \text{ is a choice point and } (i, j) \in E \\ 1, & \text{if } i \text{ is a state and } j = S(i) \\ 0, & \text{if } i \text{ is a state and } j \neq S(i) \\ 1, & \text{if } i = 0, j = 0 \\ 0, & \text{if } i = 0, j \neq 0. \end{cases}$$

(2)

One can see that $P_S[i, j]$ is the probability that the game $R(i, S)$ makes the move $(i, j)$. So $P_S$ is a probability matrix (also called a stochastic matrix) [11] since all entries are nonnegative and each row sum equals 1.

A strategy $S$ is called reasonable if for every $v \in V$ there exists a number $k$ such the probability to reach the goal state within at most $k$ steps in the game $R(v, S)$ is positive. Intuitively, a reasonable strategy may be not optimal but eventually it leads the player to the goal state.

We let $P'_S$ denote the minor of $P_S$ obtained by crossing out from $P_S$ the row number 0 and the column number 0. Since $S$ is reasonable, no subset $U$ of $V - \{g\}$ is closed under the game $R(u, S)$, for any $u \in U$. A closed subset is one such that the game never leaves it after starting at any vertex of it. The only closed subset of $V$ is $\{g\}$. This property is used to establish the following facts.

**Lemma 1.** Let $S$ be a reasonable strategy. Then

$$\lim_{k \to \infty} P'_S^k = 0$$

(3)
and
\[\sum_{k=0}^{\infty} P_S^k = (I - P_S')^{-1}.\]  
(4)

**Proof.** This follows from [11, Proposition M.3] but we present the proof here for the sake of completeness. The top row of \( P_S \) has 1 as the first element followed by a sequence of zeroes. Therefore for each \( k \geq 0 \) the power \( P_S^k \) equals the minor of \( P_S^k \) obtained by removal of row 0 and column 0. The element \( i, j \) of the matrix \( P_S^k \) is the probability to get from \( i \) to \( j \) in exactly \( k \) moves. Since the strategy \( S \) is reasonable, for every \( i \) there exists an integer \( k \) such that \( P_S^k[i, 0] > 0 \) and therefore the sum of the \( i \)-th row of \( P_S^k \) < 1. The same is true for \( P_S^n \) for any \( n > k \), that can be proved by induction. Therefore there exists an integer \( k \) and a positive real number \( \alpha < 1 \) such that the sum of every row of \( P_S^k \) is at most \( \alpha \). But then \( P_S^{i+1} \) has row sums at most \( \alpha^m \) for any \( m > 0 \) and \( i \geq mk \) which proves the convergence in (3) and also existence of the sum \( \sum_{i=0}^{\infty} P_S^i \).

Now we can prove (4). For any \( j > 0 \) we have the equality \((I - P_S') \sum_{i=0}^{j} P_S^i = (\sum_{i=0}^{j} P_S^i)(I - P_S') = I - P_S^{j+1} \). Upon taking the limit as \( j \to \infty \) we get \((I - P_S') \sum_{i=0}^{\infty} P_S^i = (\sum_{i=0}^{\infty} P_S^i)(I - P_S') = I \), which proves (4).

Reasonable strategies can be characterized in terms of their cost vectors as follows.

**Lemma 2.** A strategy \( S \) is reasonable if and only if \( M_S \) is defined.

**Proof.** We first prove the direction \((\Rightarrow)\). Assume \( S \) is a reasonable strategy and let us show that \( M_S \) exists. By using reasonableness of \( S \) we can find a natural number \( k \) such that for any vertex \( v \in V \) the probability to finish the game \( R(v, S) \) within \( k \) steps is positive and greater than some positive real number \( a \). Let \( b \) be the largest cost of any edge. Let us consider an arbitrary \( v \in V - \{0\} \). For every natural number \( m \) let us denote by \( A_m \) the event that the game \( R(v, S) \) ends within \( km \) steps but not within \( k(m - 1) \) steps, and for every integer \( l \geq 0 \) let \( B_l \) be the event that the game does not end within \( kl \) steps. If \( P \) is the function of probabilities of events then, obviously, \( P(A_m) \leq P(B_{m-1}) \) for \( m > 0 \), and \( P(B_l) \leq (1 - a)^l \) for \( l \geq 0 \). Now we can estimate \( M_S[v] \) from above as follows: \( M_S[v] \leq \sum_{m=1}^{\infty} P(A_m)km \leq kb \sum_{l=0}^{\infty} P(B_l)(l + 1) \leq kb \sum_{l=0}^{\infty} (l + 1)(1 - a)^l = kb/a^2 \), which is finite. So we have proved that \( M_S[v] \) is defined for every \( v \in V - \{0\} \) and, of course, \( M_S[0] = 0 \) so \( M_S \) is defined. This is enough for the proof of \((\Rightarrow)\) but we need more information about \( M_S \).

Let us define the vector \( b \) over \( V \) by
\[b[i] = \sum_{j \in V} P_S[i, j]c(i, j) \quad (\forall i \in V),\]  
(5)
and let \( b' \) be the projection of \( b \) onto the set \( \{1, ..., n - 1\} \). Let \( M \) be another vector over \( V \), defined by \( M[0] = 0 \) and
\[M' = (I - P_S')^{-1}b',\]  
(6)
where $M'$ is the projection of $M$ onto the set $\{1, \ldots, n-1\}$. $M'$ is defined because of Lemma 1. By transforming equation (6) we get $M' = b' + P_S' M'$, and from this one can conclude, taking into account the fact that $M[0] = 0$, that $M$ is really equal to $M_S$.

We now prove the direction ($\Leftarrow$) by contraposition. Assume that $S$ is not reasonable. We need to show that, for some $v \in V$, $M_S[v] = \infty$. Indeed, let $v$ be such a vertex that for every $k > 0$ the probability to reach the goal vertex in $k$ moves in the game $R(v, S)$ is zero. Let $A \subset V$ be the set of vertices that can be reached in the game $R(v, S)$. Let $\beta = \min_{u,w \in A, (u,w) \in E} c(u, w)$. Then $\beta > 0$ and any run of the game of length $l$ will have cost at least $\beta l$. Now starting from $v$ all runs are infinite, hence have infinite cost. So $M_S[v]$ is undefined which is a contradiction.

A vertex $v$ of a test graph is called transient if the goal state is reachable from $v$. We say that a test graph is transient if all its non-goal vertices are transient. There is a close connection between transient graphs and reasonable strategies.

**Lemma 3.** A test graph is transient if and only if it has a reasonable strategy.

**Proof.** Let $G$ be a transient test graph. We construct a reasonable strategy $T$. Using transience of $G$, we fix, for every $v \in V$, a shortest path $P_v$ to 0 (shortest in terms of number of edges). We can arrange also that if $w$ is a state that occurs in $P_v$, then $P_w$ is a suffix of $P_v$. For each state $v$ define $T(v)$ as the immediate successor of $v$ in $P_v$. We show that $T$ is a reasonable strategy. Let $v$ be a vertex in $V$. We need to show that there exists a number $k$ such that the probability to reach the goal state within at most $k$ steps in the game $R(v, T)$ is positive. Let $P_v$ be the sequence $(v_1, v_2, \ldots, v_k)$, where $k$ is the length of $P_v$, $v_1 = v$, and $v_k = 0$. If $v_i$ is a state then $v_{i+1} = T(v_i)$ and the probability $p_i$ of going from $v_i$ to $v_{i+1}$ is 1. If $v_i$ is a choice point then the edge $(v_i, v_{i+1})$ in $E$ has probability $p_i > 0$. The probability that $R(v, T)$ follows the sequence $P_v$ is the product of all the $p_i$, so it is positive.

To prove the other direction assume that $G$ has a reasonable strategy $S$. Then for each $v \in V$ the game $R(v, S)$ eventually moves the marked vertex to the goal vertex thus creating a path from $v$ to $g$.

### 2.3 Computing strategies

Ultimately, our goal is to get a procedure for finding optimal strategies for a given test graph $G$. We start by formulating the properties of the expected cost vector $M$ as the following optimization problem. Let $d$ be the constant row vector $(1, \ldots, 1)$ of length $|V| = n$.

**LP:** Maximize $dM$, i.e. $\sum_{i \in V} M(i)$, subject to $M \geq 0$ and

$$
\text{LP: } \text{Maximize } dM, \text{ i.e. } \sum_{i \in V} M(i), \text{ subject to } M \geq 0 \text{ and }
$$
Let us write the inequalities above as a family \( \{ e_i \}_{i \in I} \) over an index set \( I \), where

\[
I = \{0\} \cup \{(i, j) \in E : i \in \text{States}\} \cup \text{Choices}.
\]

It is convenient to assume that the indices in \( I \) are ordered in some fixed order as \((i_0, i_1, \ldots, i_{k-1})\) such that \( i_0 = 0 \) and we refer to the position of \( i \in I \) in this order by \( \bar{i} \). The inequalities can be written in normalized form as follows.

\[
e_i \equiv \sum_{j=0}^{n-1} A(\bar{i}, j) M(j) \leq b(\bar{i}),
\]

where \( A \) is a \( k \times n \) matrix and \( b \) is a column vector of length \( k \). The inequalities \( \text{LP} \) can be written in matrix form as \( AM \leq b \) and we can formulate the dual optimization problem as follows.

**DP:** Minimize \( Xb \), subject to \( XA \geq d \) and \( X \geq 0 \).

We will use DP in the proof of Lemma 4. It is helpful in understanding the notions to consider a simple example first.

**Example 1.** Consider the test graph \( G \) in Figure 1.a. The LP associated to \( G \) has the following inequalities:

\[
\begin{align*}
M(0) &\leq 0 \\
M(i) &\leq c(i, j) + M(j) \\
&\text{for } i \in \text{States} \text{ and } (i, j) \in E \\
M(i) &\leq \sum_{(i, j) \in E} \{ p(i, j)(c(i, j) + M(j)) \} \\
&\text{for } i \in \text{Choices}
\end{align*}
\]

The LP in matrix form looks like:

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & -1 \\
-1 & 1 & 0 \\
-2/3 & -1 & 1
\end{bmatrix}
\begin{bmatrix}
M(0) \\
M(1) \\
M(2) \\
M(3)
\end{bmatrix}
\leq
\begin{bmatrix}
0 \\
c(1, 2) \\
c(1, 0) \\
\frac{2}{3} c(2, 0) + \frac{1}{3} c(2, 1)
\end{bmatrix}
\]
The dual problem is to minimize \( Xb \), subject to \( X \geq 0 \) and

\[
\begin{pmatrix}
X
\end{pmatrix} \begin{pmatrix}
x_0 \\
x_{(1,2)} \\
x_{(1,0)} \\
x_2
\end{pmatrix} \begin{pmatrix}
A \\
d
\end{pmatrix} \geq \begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}
\]

We can write it in the form of inequalities:

\[
\begin{align*}
x_0 & \geq 1 + x_{(1,0)} + \frac{2}{3}x_2 \\
x_{(1,2)} + x_{(1,0)} & \geq 1 + \frac{1}{3}x_2 \\
x_2 & \geq 1 + x_{(1,2)}
\end{align*}
\]

Intuitively, \( x_e \) can be understood as a flow where the strategy follows an edge with a greater flow.

We say that a solution \( M \) of LP is tight for \( e_i \) if the left hand side and the right hand side of \( e_i \) are equal, i.e., there is no slackness in the solution of \( e_i \). We will use the following lemma.

**Lemma 4.** If LP has an optimal solution \( M \) then for all states \( i \) there is an edge \( (i, j) \in E \) such that \( M \) is tight for \( e_{(i,j)} \), and for all choice points \( i \), \( M \) is tight for \( e_i \).

**Proof.** Assume LP has an optimal solution \( M \). By the Duality Theorem (see e.g. [11, Proposition G.8]), DP has an optimal solution \( X \). By expanding \( XA \geq d \) we get for each state \( i \) an inequality of the form

\[
\sum_{(i,j) \in E} X((i,j)) - \sum_{j \in I} a_{i,j}X(j) \geq 1,
\]

where all \( a_{i,j} \geq 0 \), and for each choice point \( i \) an inequality of the form

\[
X(i) - \sum_{j \in I} a_{i,j}X(j) \geq 1,
\]

where all \( a_{i,j} \geq 0 \). Let \( i \) be a state. From (7) follows that some \( X((i,j)) > 0 \), which by Complementary Slackness [11, Proposition G.9] implies that \( M \) is tight for the corresponding \( e_{(i,j)} \). Let \( i \) be a choice point. From (8) follows that \( X(i) > 0 \), which by Complementary Slackness implies that \( M \) is tight for the corresponding \( e_i \).

The following characterization of transient test graphs is the main result of the paper.

**Theorem 1.** The following statements are equivalent for all test graphs \( G \).

(a) \( G \) is transient.
(b) \( G \) has a reasonable strategy.
(c) (LP) for \( G \) has a unique optimal solution \( M, M = M_S \) for some strategy \( S \) and the strategy \( S \) is optimal.
Proof. The equivalence of (a) and (b) is Lemma 3. We prove that (b) implies (c). Assume \( G \) has a reasonable strategy \( S \). To see that (LP) is feasible, note that \( M = 0 \) is a feasible solution of (LP). Let \( M \) be any feasible solution of (LP). Let \( M' \) be the projection of \( M \) onto the set \( \{1, \ldots, n-1\} \). The inequalities of (LP) ensure that in particular

\[
M[i] \leq \sum_{j \in \{1, \ldots, n\}} P_S[i, j](c(i, j) + M[j]) \quad (\forall i \in \{1, \ldots, n\}).
\]

or in matrix form \( M' \leq b_S' + P'_S M' \) which is equivalent to \( (I - P'_S)M' \leq b_S' \) where \( b_S' \) and \( P'_S \) are defined as above. By Lemma 1 the inverse matrix \( (I - P'_S)^{-1} \) exists and all its entries are positive. So our inequality will be preserved if we multiply it by \( (I - P'_S)^{-1} \) on the left. The result is \( M' \leq (I - P'_S)^{-1}b_S' \). We use equation (6) to obtain

\[
M' \leq (I - P'_S)^{-1}b_S' = M_S' \tag{9}
\]

Thus (LP) is feasible and bounded and has therefore an optimal solution \( M \). Lemma 4 ensures that there exists a strategy \( S^* \) such that

\[
M[i] = c(i, S^*(i)) + M[S^*(i)] \text{ for } i \in \text{States},
\]

\[
M[i] = \sum_{(i,j) \in E} \{p(i, j)(c(i, j) + M[j])\} \text{ for } i \in \text{Choices},
\]

which by definition of expected cost means that \( M = M_{S^*} \). From (9) follows that \( M_{S^*} \leq M_T \) for any strategy \( T \). Thus \( S^* \) is optimal.

Finally, note that (b) follows from (c) by using Lemma 2 since \( M_S \) is defined.

Notice that, even though an optimal strategy \( S \) yields a unique cost vector \( M_S \), \( S \) itself is not necessarily unique. Consider for example the graph in Figure 1.b where all vertices are states and the edges are labeled by costs; clearly both of the two possible strategies are optimal.

2.4 Graph transformation

We made the assumption that for each two vertices in the graph there is at most one edge connecting them. Let us show that we did not lose any generality by assuming this. For a state \( u \) and for any \( v \in V \) let us choose an edge leading from \( u \) to \( v \) with the smallest cost and discard all other edges between \( u \) and \( v \). In the case of a choice point \( u \) we can replace the set of multiple edges \( D \) between \( u \) and \( v \) with one edge \( e \) such that \( p(e) = \sum_{e' \in D} p(e') \) and \( c(e) = (\sum_{e' \in D} p(e')c(e'))/p(e) \). This merging of multiple edges does not change the expected cost of one step from \( u \). The graph modifications have the following impact on LP. With removal of the edges exiting from states we drop the corresponding redundant inequalities. The introduction of one edge for a choice point with changed \( c \) and \( p \) functions in fact does not change the coefficients before \( M_i \) in LP in the inequality corresponding to the choice point and therefore does not change the solution of LP.
2.5 Approximate solutions

When the number of vertices in the graph is big it is practical to search for strategies which are close to optimal in some sense, since the optimal strategy calculation becomes too expensive. One such strategy is built in the proof of Lemma 3. Another approach is presented in [20] where several methods are described that produce strategies that are optimal, in some sense, for games with a finite number of steps.

We describe here yet another approach for calculating a good strategy. We do not provide all proofs here but hope that the heuristic works.

Let $M$ be a vector with coordinates indexed by $V$ and initially $M[i] = 0$ for every state $i$ and $M[0] = 0$. Let us consider the system of linear equations

$$M[i] = \sum_{(i,j) \in E} \{p(i,j)(c(i,j) + M[j])\}, \quad i \in \text{Choices.} \quad (10)$$

where the unknown variables are $M[i], i \in \text{Choices}$. By solving (10) we obtain some values for the unknowns. We can now update the values for $M[i], i \in \text{States}$ by letting

$$M[i] := \min_{(i,j) \in E} (c(i,j) + M[j]). \quad (11)$$

Notice that $M$ is still a feasible solution of LP after this step. We can iterate this process several times; the vector $M$ would not become less and would remain a feasible solution of LP. We can stop iterating, for example, when the maximal growth of $M$ will be less than $\min_{i,j \in V, M[i] \neq M[j]} \text{abs}((M[i] - M[j])$.

After we are done with the iterations we can define $S(i)$ for a state $i$ so that $c(i, S(i)) + M[S(i)] = \min_{(i,j) \in E} (c(i,j) + M[j])$ which we believe is a good strategy. The system of equations (10) has a solution. Indeed, this follows from Theorem 1 applied to a modification of our graph. Specifically, modify the test graph so that for any state $i$ there is only one exiting edge, this edge points to the goal vertex, and the cost of this edge equals $M[i]$. Then the condition (a) of Theorem 1 holds and the solution of LP restricted to choice points solves (10).

The iterative process, generally speaking, does not reach a fixed point. Let us suppose that in Figure 1.a the cost of edge $(1, 0)$ is 3 and that all other edges cost 1. It is not difficult to see that the sequence of values for $M[1]$ calculated by the algorithm would yield the infinite sequence $0, 2, 2 + \frac{2}{3}, 2 + \frac{2}{3} + \frac{2}{9}, \ldots$. Notice, however, that non-optimal costs can yield an optimal strategy. Furthermore, it may be satisfactory for practical purposes to stop at a strategy which is not optimal but whose expected costs are close to the optimum.

Notice the special case when the graph does not contain any choice points. In that case just the step (11) is being iterated and upon termination the obtained strategy provides an approximation of a shortest path to the goal state.

It can be proven that the convergence of the process is exponentially fast. It is possible to modify the process to avoid having to solve the system of linear equations (10), replacing it by a faster calculation. We suspect that the resulting process is more efficient for the task than Simplex Method but we have not investigated it yet.
3 Application to model-based testing

The problem of generating strategies from transient test graphs came up in the context of testing reactive systems with the model-based testing tool SpecExplorer that has been developed in the Foundations of Software Engineering group at Microsoft Research. A recent overview of SpecExplorer is given in [14]. We explain here how the problem arises in this context; a lot of details that fall outside the scope of the paper are omitted here.

Model-based testing, e.g. as described in [4], is a way to test a software system using a specification or model that predicts the (correct) behavior of the system under test. In systems with large state spaces, in particular in software testing, specifying a finite transition system statically is often infeasible. To overcome the limitations of a statically-described finite transition system, one can encode the transition system in terms of guarded update rules called actions of a model program.

Using SpecExplorer In SpecExplorer the user writes models in AsmL [17] or in Spec# [5], which is an extension of C# with high-level data structures and contracts in form of pre-conditions, post-conditions, and invariants. The semantics of AsmL programs is given by Abstract State Machines (ASMs) [16]. The ASM semantics of the core of AsmL can be found in [12]. A model program P induces a transition system $R_P$ that is typically infinite, because of unbounded data structures and objects. More precisely, a model state is a mapping of variables to values, i.e. a first-order structure. The initial model state of $R_P$ is given by the initial values of variables in $P$. An action is a method of $P$ with 0 or more parameters. A precondition of an action $a$ is a Boolean expression using parameters of $a$ and state variables. An action $a$ is enabled in a state $s$ if its precondition is true in $s$. There is a transition $(s, a, t)$ from model state $s$ to model state $t$ labeled by action $a$ in $R_P$ if $a$ is enabled in $s$ and invoking $a$ in $s$ yields $t$. Thus $R_P$ denotes a complete unwinding of $P$ and is typically infinite.

Prior to test case generation, a model program $P$ is algorithmically unwound using a process we call “FSM generation” to produce a finite (under)approximation of $R_P$. During FSM generation various restrictions on possible model states are introduced by the user that limit the size of the state space to a feasible range since $R_P$ may be infinite.

For modeling reactive systems with SpecExplorer, actions are partitioned into observable and controllable ones. Intuitively, a controllable action is an action that a tester has “control” over, whereas an observable action is a spontaneous reaction from the system under test that can only be “observed” by the tester. A model state where only controllable actions are enabled is called stable, it is called unstable otherwise. The FSM generation algorithm creates a timeout transition $(s, \delta, \hat{s})$ from every unstable model state $s$ to a timeout state $\hat{s}$ that denotes $s$ after timeout $\delta$, where $\delta$ is a time-span that is in SpecExplorer specified by a state based expression. Intuitively, $\delta$ specifies the amount of time to wait

\footnote{The terms ‘specification’ and ‘model’ are often used as synonyms in this context.}
in a given model state for any of the observable actions to occur; $\delta$ may be 0.

For unstable states $s$, transitions for any of the enabled controllable actions are created only from $\hat{s}$. The output of FSM generation is a test graph $G$, where the choice points of $G$ represent the unstable model states and the states of $G$ represent the stable model states and the timeout states. In the presence of observable actions it is essential that tests always terminate in certain accepting model states only. In $G$ we can assume, without loss of generality, that there is a final action leading from any state corresponding to an accepting model state to the goal state 0.

**Elimination of dead vertices** The FSM generation process is guided by various user-defined heuristics to limit the search. One such technique for FSM generation is the usage of state based expressions that partition the state space into a finite number of equivalence classes and limit the number of concrete representatives per equivalence class. A basic version of this technique for non-reactive systems is described in [13]. Quite often the FSM generation process may terminate prematurely and not continue the state space exploration from a state because of such restrictions. Consequently the generated test graph is not necessarily transient. If a test graph is not transient then we have to prune it to a transient subgraph, prior to the optimal strategy calculation, in order to guarantee that an optimal strategy exists. This step is achieved by “elimination of dead vertices”. We are considering two possible definitions of dead vertices. A vertex is **strongly dead** if no path from it leads to the goal vertex; a vertex is **weakly dead** if, when $IUT$ is no longer required to move randomly but is allowed to move maliciously, it has a strategy ensuring that the goal vertex is never reached. One can see that every strongly dead vertex is also weakly dead.

Elimination of strongly dead vertices is easily done by checking which vertices are reachable from the goal vertex by backward reachability analysis.

Let us motivate the notion of weakly dead vertices. Suppose that a path on the graph is considered to be correct for testing only if it ends in the goal vertex. As we explained above, our test graph is an under-approximation of $R_P$. Therefore we could have a choice point which is weakly dead only because FSM generation stopped too early and as a result of this, for any strategy of $TT$, some path from the choice point leads to a “trap” - a strongly dead state. We have two options: we can remove the edges which make the choice point weakly dead, or we can remove the choice point itself. Removal of choice edges is in general undesirable, since it may not correspond to the $IUT$ behavior and could violate the conformance relation. In this case we have to purge out weakly dead vertices. An efficient algorithm for achieving this is given in [8, Section 4].

A simpler but less efficient algorithm [9, Algorithm 1], a version of which has been implemented in SpecExplorer, works as follows. First, it finds the set of vertices $R$ from which the goal is reachable. The set of the rest of the vertices $Tr = V - R$ is a trap for $TT$. Then the set of vertices $W$, from which $IUT$ has a strategy to reach $Tr$, is computed. The set $W$ consists of weakly dead vertices, but does not necessarily contain all of them! $W$ is removed from the graph and the process is repeated until no new weakly dead states are found. Each iteration takes $O(n)$
time where $n = |V|$, so the whole algorithm takes $O(n^2)$ time. The algorithm in [8] takes $O(m\sqrt{m})$ time where $m$ is the number of edges.

4 Related work

The results in Section 2 are strongly influenced by [11]. The proof of Theorem 1 closely follows the scheme of the proof of [11, Theorem 2.3.1], which states that an optimal strategy for Markov decision process with discounted rewards is a solution of the corresponding linear program. However we don’t see any direct connection between the theorems; that is we don’t see how the expected cost of a reachability game can be expressed in terms of discounted rewards in a Markov decision process. At the same time, we are not able to reduce our problem to average reward stochastic games which are also discussed in [11].

Transient games defined in [11, 4.2] have a lot of similar properties with reachability games on transient test graphs. For example, for a transient game the total expected game value can be calculated and corresponds to existence of expected game cost in our case. But again, the setting is different. A transient stochastic game is a game between two players that will stop with probability 1 no matter which strategies the players choose. However, for some transient test graphs $TT$ can choose such a strategy that the game will never end with a positive probability.

The game view of testing reactive systems was addressed in [20], where the problem solved in Section 2 was left open. For solving verification problems of reactive and open systems, games have been studied extensively in the literature. In [10] the game view is proposed as a general framework for dealing with system refinement and composition. Model programs with controllable and observable actions, viewed as transition systems in Section 3, correspond to interface automata in [10], and the conformance relation used in SpecExplorer is a variation of alternating refinement [2, 10]. Intuitively, the IUT must accept all controllable actions enabled in a given state, and conversely, the model must accept all observable actions enabled in a given state. The IUT may however accept additional controllable actions that are not specified in the model; this situation arises naturally in testing of multithreaded or distributed systems, where only some aspect of the IUT, and not all of its behavior, is being tested.

Extension of the FSM-based testing theory to nondeterministic and probabilistic FSMs [1, 15, 22], in particular for testing protocols, got some attention ten years ago. However with the advent of multithreaded and distributed systems, it recently got more attention. In this regard, the recent paper [23] is related to our work but it does not use games (see [20] for comparison).

Model-based testing has recently received a lot of attention in the testing community and there are several projects and model-based testing tools that build upon FSM-based testing theory or Label Transition Systems based (IOCO) testing theory [3, 18, 19, 21]. In IOCO, test cases are trees and deal naturally with non-determinism [7, 21]. Typically, goal oriented testing is provided through model-checking support that produces counterexamples that may serve
as test sequences. The basic assumption in those cases is that the system under consideration is either deterministic, or that the non-determinism can be resolved by the tester (i.e. the systems are made deterministic from the tester’s point of view). To the best of our knowledge, SpecExplorer is currently alone in supporting goal oriented game strategies for testing.

5 Concluding remarks

Solving linear equations for generating test strategies from transient test graphs, may require a specialized algorithm. From our experiments with SpecExplorer, standard techniques, using the simplex method, do not seem to scale for large models. When the number of states exceeds 500 we fall back to an approximate solution described in Section 2.5.

For testing large reactive multithreaded or distributed systems, it is often not even feasible to first generate a finite test graph and then generate tests from it. On-the-fly is a testing technique in which test derivation from a model program and test execution are combined into a single algorithm. It can also be called behavioral stress testing or model-based online testing, to distinguish it from offline test generation as a separate process. On-the-fly testing is supported by SpecExplorer and other model-based testing tools like TorX [21]. The specification of test purposes and measurement of coverage are two important open problems here.

6 Acknowledgements

We are grateful to Yuri Gurevich for his valuable comments on the paper.

References


