Snodgrass, S., & Aha, D.W. (2014). *System model formulation using Markov chains* (Technical Note AIC-14-170). Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in AI.

System Model Formulation Using Markov Chains

Sam Snodgrass
Computer Science Department
Drexel University
Sam.PSnodgrass@gmail.com

David W. Aha
Navy Center for Applied Research in AI
Naval Research Laboratory, Code 5514
Washington DC
david.aha@nrl.navy.mil

Abstract

Creating formal models for systems manually is time consuming and difficult. Automating the generation and verification of these formal models can reduce the overhead of developing the models. In this paper we propose an approach (MC-MC) to verifying and generating portions of the formal model using Markov chains.

Learning Formal Models of Systems

Formal models are used to explain the required behavior of a system. However, manually creating these models is difficult and time consuming. Machine learning techniques can be used to automatically learn, generate, and verify portions of the model, lessening the burden on the designers and developers.

An important part of a formal model is a diagram showing the different states of the system, and how each state can be reached (or not reached) from another. These diagrams are similar to state machines or automata, and there is a large corpus of work related to automatic generation of automata through learning. For example, Balle et al. (Balle et al. 2013) use a spectral learning algorithm to learn weighted automata. Additionally, Cleeremans et al. (Cleeremans, Servan-Schreiber, and McClelland 1989) and Giles et al. (Giles et al. 1992) explore the use of neural networks for learning state machines. However, these approaches have been used primarily for grammar modeling, and have not been applied to constructing a formal model of a system.

Additionally, some research has been performed on automating and aiding in the design and verification of formal systems. Păsăreanu et al. (Păsăreanu et al. 2008) employ the L^* machine learning algorithm to perform system verification. We are instead interested in the problem of *model generation*, which needs to precede verification. Neema et al. (Neema et al. 2003) introduce a mixed-initiative toolset for aiding in model design, but we want to automatically generate models in order to ease the burden of the designers and developers. There has also been some work on using Markov chains to test formal models (Whittaker, Thomason, and others 1994) and on learning the transition weights for the automaton style models, given the structure (Whittaker and Poore 1993), but not on using Markov chains to learn the structure of the models themselves.

The work we present in this paper is closely related to the above work on learning state machines and their associated transition weights. We describe an application of Markov chains to an interesting domain: learning a state machine from a set of traces of system behavior in order to model a system.

Methods

In this section we discuss how the Markov chain model checker (from here on referred to as $MC-MC$) learns from a set of traces, and how it utilizes the learned information. A *trace*, in the context of this paper, is the sequence of monitored events of the system that occured during a user test. Table 1 shows a sample section of a trace from a simplified version of RESCHU[1], a simulator in which a user controls multiple vehicles using a graphical interface. The user's goal is for their vehicles to to reach as many targets as possible in an allotted time, while avoiding hazard zones.

Learning

To learn the probability of transitioning from one event to another, we employ Markov chains. Markov chains (Markov 1971) can be used to model probabilistic transitions between states (in our case, events). A Markov chain is defined by a set of states, $S = e_1, e_2, ..., e_n$, and a conditional probability distribution (CPD), $P(E_t|E_{t-1})$, where E_t is a variable representing whichever state is encountered at time t. The CPD corresponds to the probability of transitioning to a state, $E_t \in S$, from a state, $E_{t-1} \in S$.

Using Markov chains, MC-MC learns a conditional probability distribution corresponding to the probability of the transitions between events. It learns the CPD in two stages:

1. **Compute Totals**: MC-MC counts the number of times that each event type follows each other event type, in all of the input traces. This is represented by $T(e_i \mid E_{i-1})$. Table 2 shows the totals computed

[1] http://web.mit.edu/aeroastro/labs/halab/inventions.shtm

Table 1: A section of a user's trace while interacting with RESCHU.

Time	Event Name	UAV	UAV_x	UAV_y	Hazard	$Hazard_x$	$Hazard_y$	$Target_x$	$Target_y$
0	Hazard Moved	X	X	X	$Hazard_3$	259	147	X	X
0	UAV Unsafe	UAV_0	361	300	$Hazard_3$	259	147	X	X
123	Add Waypoint	UAV_0	283	210	X	X	X	171	137
123	UAV Safe	UAV_0	283	210	X	X	X	X	X
221	Target Moved	X	X	X	X	X	X	73	391
270	Hazard Moved	X	X	X	$Hazard_5$	160	80	X	X
270	UAV Unsafe	UAV_0	170	136	$Hazard_3$	160	80	X	X
280	Add Waypoint	UAV_0	167	127	X	X	X	132	87
285	Target Moved	X	X	X	X	X	X	30	11

from the section of a trace in Table 1. Note that these totals are being computed from a very short trace, and therefore not all transitions are encountered.

2. **Compute Probabilities**: After finding the totals, it computes the conditional probability distribution (CPD) describing the Markov chain as follows:

$$P(e_i|E_{i-1}) = \frac{T(e_i|E_{i-1})}{\sum_{j=0}^{n} T(e_j|E_{i-1})},$$

which means that the probability of transitioning from the event represented by E_{i-1} to e_i is equal to the number of times e_i following the event represented by E_{i-1} has been observed, divided by the total number of times any event following the event represented by E_{i-1}, has been observed. Table 3 shows the probability distribution computed from the totals in Table 2. Note that this distribution is computed from a very short trace, and as such is not as complex or complete as a typical CPD.

MC-MC does not apply any smoothing techniques, because we are interested in which scenarios (or sequences of events) are possible, as observed through the traces, and smoothing would obscure this information. However, common smoothing techniques, such as Laplace smoothing, could easily be introduced (Chen and Goodman 1996).

Applying

In this section we discuss how MC-MC applies the learned CPD and *Totals* matrix. We explore three different uses for the learned knowledge: transition diagram generation, scenario generation, and scenario detection.

Transition Diagram Generation MC-MC can generate weighted graphs corresponding to the transitions described in the conditional probability distribution. Weighted transition diagrams can be useful for identifying common transitions, or for finding transitions that are not supposed to occur, but do.

First, MC-MC creates a node representing the first event in the CPD. Next, it creates nodes representing any event in the CPD that the first event has a probability greater than some threshold of transitioning to. Then, it draws directed edges, labelled with the transition probability, from the first event to the transition event. The preceding steps are repeated for each row in the CPD, only adding new nodes when an event is not already represented. Figures 1 and 2 show transition diagrams generated using no threshold and a threshold of 10%, respectively.

Notice that, in Figure 2, all of the events are still reachable, but many of the transitions that are present in Figure 1 are missing. By removing the lower probability transitions, we get a more concise (albeit incomplete) description of the behavior of the system. Pruning low probability transitions can be helpful when trying to understand the general behavior of the system, or when trying to find the more probable behavior.

Scenario Generation Generating scenarios can help to fill in the gaps of the traces. That is, by generating scenarios we can encounter scenarios that are evidenced by the traces, but that do not necessarily occur within them. These generated scenarios can then be used to explore the space of scenarios, and determine if there is any undesired behavior in the system.

MC-MC employs a scenario generation algorithm adaptated from the recursive depth-limited search algorithm (Russell and Norvig 1995). This algorithm explores the space of possible scenarios in a depth-first manner. The search tree's root node is the start event, and the goal nodes are the end event. Each node's children are the events to which the node's event has a probability of transitioning, which is greater than some threshold. The transition probabilities are taken directly from the learned conditional probability distribution. Each time a goal node, corresponding to a desired

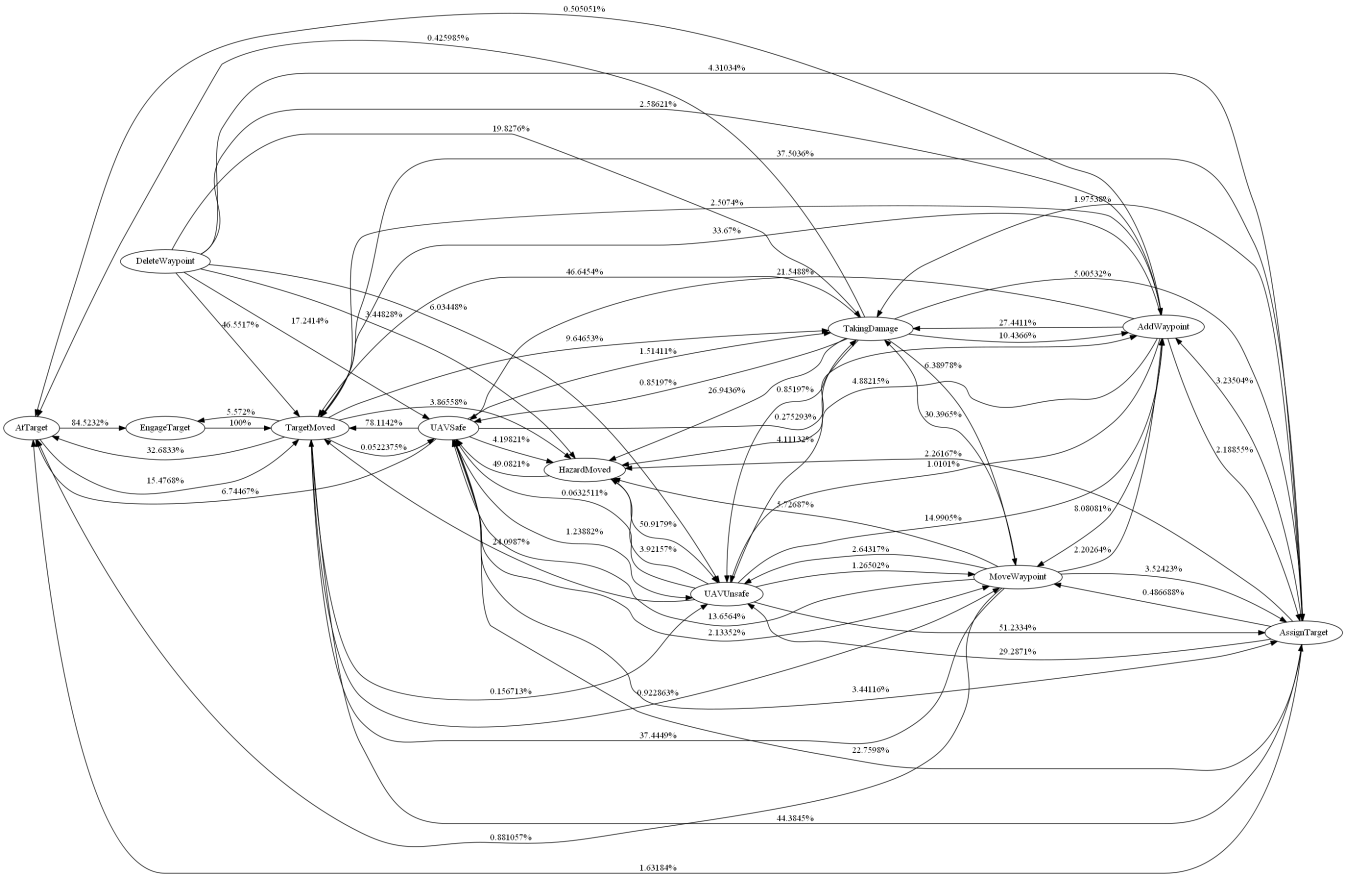

Figure 1: Transition diagram with all transitions with probability greater than 0% generated from the conditional probability distribution computed using our Markov chain implementation.

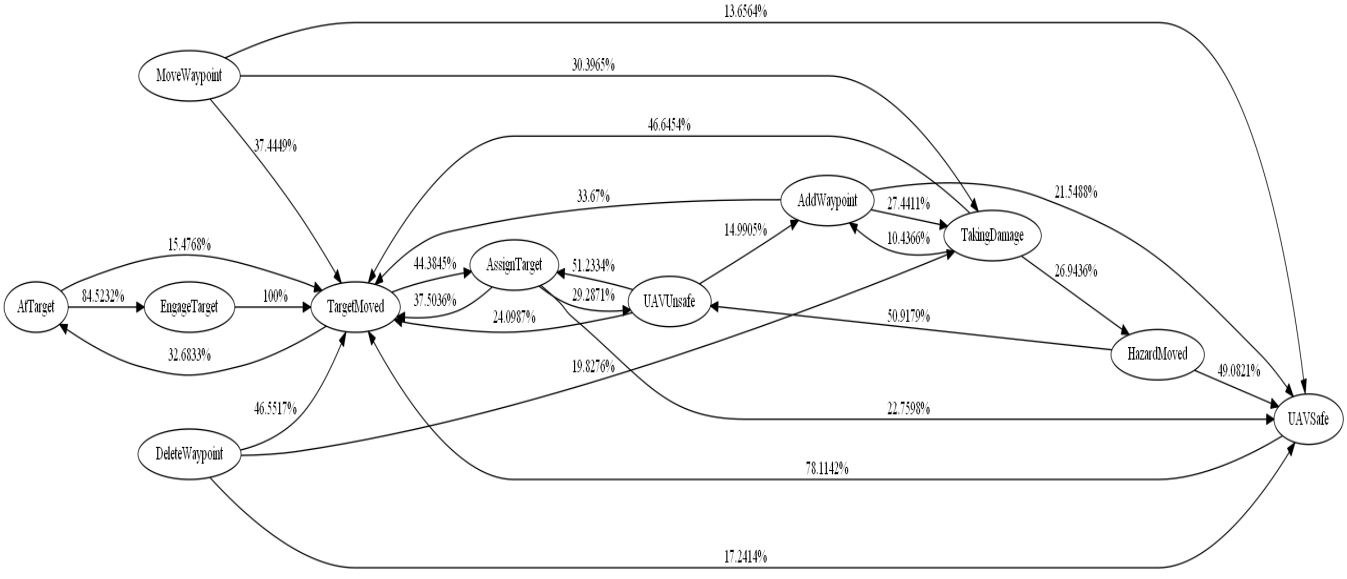

Figure 2: Transition diagram with all transitions with probability greater than 10% generated from the conditional probability distribution computed using our Markov chain implementation.

Table 2: The totals corresponding to the trace section in Table 1. The events listed in the first column correspond to the starting events, and events listed in the first row correspond to events being tansitioned to.

Events	Hazard Moved	UAV Unsafe	Add Waypoint	UAV Safe	Target Moved	Totals
Hazard Moved	0	2	0	0	0	2
UAV Unsafe	0	0	2	0	0	2
Add Waypoint	0	0	0	1	1	2
UAV Safe	0	0	0	0	1	1
Target Moved	1	0	0	0	0	1

Table 3: The conditional probability distribution (CPD) corresponding to the totals in Table 2. Each cell corresponds to the probability of transitioning to the column label event from the row label event.

Events	Hazard Moved	UAV Unsafe	Add Waypoint	UAV Safe	Target Moved
Hazard Moved	0	1	0	0	0
UAV Unsafe	0	0	1	0	0
Add Waypoint	0	0	0	.5	.5
UAV Safe	0	0	0	0	1
Target Moved	1	0	0	0	0

Table 4: The totals corresponding to the trace section in Table 1, using an order three Markov chain. Note that only the final column representing the total number of the times the configuration is seen is shown, for conciseness.

Configurations	Totals
Hazard Moved→UAV Unsafe→Add Waypoint	2
UAV Unsafe→Add Waypoint→UAV Safe	1
UAV Unsafe→Add Waypoint→Target Moved	1
Add Waypoint→UAV Safe→Target Moved	1
UAV Safe→Target Moved→Hazard Moved	1
Target Moved→Hazard Moved→UAV Unsafe	1

end event, is reached, the path to that node is stored or output, and the probability of the path is calculated. The total probability of the path is calculated by

$$\prod_{e \in Path} P(e_i | E_{i-1}),$$

which is the product of all the individual transition probabilities of the path. The algorithm then backtracks to the previous node and resumes its search. If the algorithm reaches the maximum allowed depth, as provided by the user, then the algorithm backtracks to the previous node.

Although this algorithm is designed to return all desired paths up to a certain length, it could be modified to return only the n most probable paths, instead of outputting every detected scenario, by maintaining a list of scenarios and their probabilities, and replacing the scenario with the lowest probability with a new scenario with higher probability, if any is encountered.

Scenario Detection Generating evidenced scenarios is helpful, but we may only want to concern ourselves with scenarios that have actually occurred in the given traces. Detecting a particular scenario in a set of traces, and counting its occurrences can provide insight into the actual behavior of the system (and of the users interacting with the system) as opposed to the evidenced and potential behavior.

It is possible to check whether a certain scenario occurs in the given traces using a higher order Markov chain. Suppose we have a scenario consisting of n

events, where n is a finite natural number. To check if this scenario occurs in the traces, MC-MC performs the *Compute Totals* portion of CPD learning algorithm, for a Markov chain of length n. Next, it simply locates the row containing the event configuration prescribed by the given scenario, and checks the final column in that row, which holds the total number of times that configuration has been encountered in the traces. Thus, the total number of times the desired scenario appears in the traces has been computed.

For example, suppose we are using the section of a trace from Table 1 as our trace, and we want to find the scenario composed of the events *Hazard Moved*, *UAV Unsafe*, and *Add Waypoint*, in that order. Then, MC-MC simply generates the *Totals* matrix using an order three Markov chain, as seen in Table 4, locates the row corresponding to the desired scenario (first row), and checks the final column for the total number of times the scenario appears in the trace. This shows us that the scenario in question occurs twice in the given trace.

Future Work

We would like to extend MC-MC to operate at run-time, which would allow us to apply it in other domains. For instance, if operating at run-time, it could then be used for partial plan recognition and plan prediction. This could have applications in programs that assist the users, or in games as a way of improving the enemy AI with regards to reacting to player behaviors and trying to thwart player plans. We would also like to conduct an empirical study of MC-MC. In this study, we would compare how well MC-MC performs against other approaches to state machine learning, transition weight learning, and event prediction. Alongside the study, we would like to reason about MC-MC and the data used to see if we can make any guarantees about our approach. Lastly, we want to employ hidden Markov models to learn the modes of the system. Modes are an abstraction of system states into equivalence classes (Heitmeyer et al.). Mode diagrams are used to model transitions between different modes (system states), where each mode has some number of transitions (possibly self loops). More generally, mode diagrams model the high level behavior of the system itself. We believe that the equivalence classes could be likened to the states of a hidden Markov model. Learning the modes and mode diagram would help to automate the software design process further, and allow the requirements engineer to either verify that the model matches his expectations of the system, or realize that there is something wrong with the system or his expectations.

References

Balle, B.; Carreras, X.; Luque, F. M.; and Quattoni, A. 2013. Spectral learning of weighted automata. *Machine Learning* 1–31.

Chen, S. F., and Goodman, J. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, 310–318. Association for Computational Linguistics.

Cleeremans, A.; Servan-Schreiber, D.; and McClelland, J. L. 1989. Finite state automata and simple recurrent networks. *Neural computation* 1(3):372–381.

Giles, C. L.; Miller, C. B.; Chen, D.; Chen, H.-H.; Sun, G.-Z.; and Lee, Y.-C. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation* 4(3):393–405.

Heitmeyer, C. L.; Pickett, M.; Leonard, E. I.; Ray, I.; Aha, D. W.; Trafton, J. G.; and Archer, M. M. Building high assurance human-centric decision systems.

Markov, A. 1971. Extension of the limit theorems of probability theory to a sum of variables connected in a chain.

Neema, S.; Sztipanovits, J.; Karsai, G.; and Butts, K. 2003. Constraint-based design-space exploration and model synthesis. In *Embedded Software*, 290–305. Springer.

Păsăreanu, C. S.; Giannakopoulou, D.; Bobaru, M. G.; Cobleigh, J. M.; and Barringer, H. 2008. Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design* 32(3):175–205.

Russell, S., and Norvig, P. 1995. *Arificial Intelligence: A Modern Approach*. Citeseer, 3rd edition. 87–88.

Whittaker, J. A., and Poore, J. H. 1993. Markov analysis of software specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2(1):93–106.

Whittaker, J. A.; Thomason, M.; et al. 1994. A markov chain model for statistical software testing. *Software Engineering, IEEE Transactions on* 20(10):812–824.