Heuristic Search for Bounded Model Checking of Probabilistic Automata

Robert P. Goldman and **David J. Musliner** and **Michael W. Boldt**
SIFT, LLC
Minneapolis, MN USA
{rpgoldman,musliner,mboldt}@sift.net

1 Abstract

We describe a new method for bounded model-checking of probabilistic automata (PAs), based on heuristic search and integrated into the PRISM model-checker. PA models include both probabilistic transitions and nondeterministic choice transitions. Our search-based approach aims to address weaknesses in statistical and dynamic-programming approaches to checking bounded PCTL properties of probabilistic automata. To model-check properties of PA models, we must demonstrate that the property will be satisfied even in the face of an adversary that optimally resolves nondeterministic choices. We use heuristically guided AO* search to find the optimal adversary policy and estimate the probability of properties. We adapt techniques from Artificial Intelligence Planning to develop a heuristic that is based on a relaxation of the underlying probability model. This heuristic provides critical guidance to the search algorithm: without it, even very small models are unsolvable. We have implemented our algorithm in the PRISM model-checker, and show cases where it outperforms PRISM's dynamic programming methods. We also describe promising directions for future work in search-based PA model-checking, notably introducing further abstraction into the heuristic to address memory issues.

2 Introduction

Probabilistic automata (PAs) are automata that combine probabilistic and nondeterministic models of uncertainty (Sokolova and De Vink 2004). They are useful for modeling phenomena such as randomized algorithms, communications protocols, adversarial games, etc. For example, a communications protocol might combine stochastic elements (e.g., a probabilistically chosen back-off delay) with nondeterministic choice (e.g., a node can choose when to send a message). Claims about PAs are typically probabilistic. For example, the probability of a collision in 100 message interchanges is less than 0.001. Many safety claims like this one are made in terms of a time bound. In some cases, safety claims *must* be made with respect to a time bound, or they will be trivially true or false by the law of large numbers. For example, we don't require an aircraft to *never* crash, we only require that the probability of a crash is very low over a reasonable lifetime of the aircraft. For this reason, we are particularly interested in *bounded* model-checking for PAs.

Definition 1 (Probabilistic Automaton) *A probabilistic automaton is a four-tuple, $\langle s, S, A, \alpha \rangle$, where S is a set of states, $s \in S$ is the initial state, A is a set of actions, and $\alpha : S \times A \rightarrow \mathcal{D}(S)$ is the transition relation, mapping a state and an action to a distribution over successor states.*[1]

The properties we check are expressed in the temporal logic PCTL:

Definition 2 (PCTL) *Assuming a set of atomic propositions, $a_1, a_2 ... a_n \in A$, the set of* state formulas *are as follows: every atomic proposition is a state formula. If f_1 and f_2 are state formulas, so are $\neg f_1$, $f_1 \wedge f_2$, $f_1 \vee f_2$, and $f_1 \rightarrow f_2$. If f_1 and f_2 are state formulas, then $f_1 W^{\leq t} f_2$ (weak bounded until) and $f_1 U^{\leq t} f_2$ (strong bounded until) are* path formulas. *Finally, if ϕ is a path formula, $P_{(\leq,<,\geq,>)p}\phi$ are* probabilistic formulas.[2]

Note that we follow PRISM, and diverge from Hansson and Jonsson, by distinguishing between probabilistic and state formulas.

In order to evaluate a probabilistic claim about a PA, with bounded or unbounded time, we must have some way to resolve the nondeterminism in the model. The PA model does not define a single probability distribution (Markov Chain), but rather a family of distributions, depending on how the nondeterministic choices are made. In order to perform *verification*, we should resolve this nondeterminism in a worst-case manner: *i.e.*, we should resolve them as the optimal adversary would. Finding this optimal adversary reduces to the problem of finding the optimal policy for a Markov Decision Process (Parker 2011, e.g.). We seek to prove that the system description forces a win against a (nondeterministic) adversary who is trying to make ϕ occur with a probability of at least P. These model-checking problems are *extremely* computationally demanding, both in terms of time and space.

Note that the relationship between bounded and unbounded model checking in PAs is fundamentally different from the relationship that holds in conventional model checking. In conventional model checking, the ideal is to

[1] This definition is adapted from Sokolova and de Vink (2004).

[2] This definition adapted from Hansson and Jonsson (1994).

check properties over unbounded time, but for many systems this check cannot be effectively performed, and bounded model checking provides a feasible approximation. We have already pointed out that bounded model checking can provide qualitatively different answers in the probabilistic context, because of the law of large numbers. In addition, the nature of the adversary/counterexample is qualitatively different. Under appropriate assumptions, for unbounded time, there is an optimal policy that is deterministic and stationary (Puterman 1994), and many algorithms for MDPs exploit this fact. However, for bounded horizon MDPs, the optimal policy will generally be non-stationary: *i.e.*, the optimal action choice for a given state will differ, depending on the time. For the intuition behind this, consider a problem where there is a long path to the goal that has a high probability of success, and a low probability short path. As the agent nears the time horizon, it should shift its choice from the long path to the short path, since it won't have time to complete the long path before the deadline.

The leading tool for probabilistic model checking is PRISM (`www.prismmodelchecker.org`) (Kwiatkowska, Norman, and Parker 2011). PRISM provides model checking methods (and additional analysis tools, such as simulation) for a number of classes of models, including PAs. PRISM is an open source system, readily extended by researchers outside the core team. We have implemented our heuristic search solver as an extension to PRISM.

For bounded model checking of PAs/MDPs, PRISM provides a solution based on Bellman backup (dynamic programming), using Algebraic Decision Diagrams (ADDs) (Bahar et al. 1993)[3] to efficiently represent transition matrices, etc. One weakness of dynamic programming approaches is that they can be forced to explore irrelevant (unreachable or suboptimal) parts of the state space, which may present challenges in some models, since typically the state space is exponential in the size of the input. Well-directed search methods can avoid this problem, limiting their attention to relevant portions of the search space (sampling methods provide a different means of attack on this same challenge of dimensionality; see Section 6).

In this paper, we describe preliminary results from our work applying heuristically-guided AND/OR tree search to bounded model checking of PAs. The strengths (and weaknesses) of heuristic search nicely complement those of sampling methods and dynamic programming. Unlike dynamic programming methods, when the heuristic performs well, we can avoid enumerating the full state space. Like dynamic programming, but unlike statistical methods, systematic search guarantees the correctness of the probability bounds it computes. This is particularly important when we *fail* to find a counterexample for a claim: if we report that a system is safe because it *cannot* reach a safety-violating state, s, with greater than a probability P, we can make this claim with confidence. Since sampling methods do not currently provide bounds on the quality of the policy (counterexample) they compute, they cannot currently make such safety claims with confidence.

Our heuristic search method combines AO* search with a heuristic function that is based on a relaxation of the underlying model. It has been implemented in the PRISM probabilistic model checking tool (Kwiatkowska, Norman, and Parker 2011), and we present early results on a number of probabilistic model-checking problems. The heuristic is of interest even outside of the context of systematic search, as in our AO* search. Heuristics like the one we propose can also be used in Monte Carlo Tree Search (MCTS), and we would like to use the current heuristic to guide our MCTS-based approach to PA verification (Boldt, Goldman, and Musliner 2015a; 2015b).

In the following sections we describe the search method, and then the heuristic we use to guide the search. We then present some test cases that illustrate strengths and weaknesses of the technique. We then review related work, and conclude with some promising directions for enhancement of the tool we have built.

3 AO* search for model-checking

Finding an optimal policy for an MDP is an AND/OR tree search problem. The OR nodes involve the adversary choosing an action for $\langle s, t \rangle$, a particular state at a particular time index. AO* search is the AND/OR analog of A* search, and provides the same guarantee: when guided by an *admissible* heuristic function, it is guaranteed to return the minimum cost solution (in our case, the policy providing the maximum probability of reaching a target state) as the first solution it finds (Bagchi and Mahanti 1983). Our implementation was based on the text by Edelkamp and Schrödl (2012).

In A* search, the best solution so far corresponds to a single node in the search tree and, implicitly, a path to that node. In AO* search, on the other hand, the current best solution is a *subtree* of the search tree: for the OR nodes, the single child that looks best at present, and at AND nodes, *all* of the child OR nodes. For this reason, when AO* search expands a node, it must update the value estimates, potentially all the way to the root of the tree, before proceeding. From this subtree, we collect the *non-terminal tip nodes*, and that acts as our openlist.

We have implemented AO* search as an extension to the Java-based PRISM probabilistic model checker. The AO* search explores the AND/OR tree of PA states, so we can use it to find the probability of reachability for a property in PRISM's Probabilistic LTL. By finding the maximum probability of reachability, we can check bounded-until PCTL properties. The PRISM infrastructure also translates related properties (bounded eventually, bounded always) for our solver. In addition, we made substantial modifications to the algorithm in order to make it (1) interface with the methods and data structures of the PRISM model checker, (2) exploit special features of the MDP search problems, and (3) handle problems posed by the large search spaces. We also developed a heuristic for the verification problems, which we discuss in the following section.

We present pseudocode for our implementation as Appendix A. At the top we list a number of data structures used in the algorithm. As with any search algorithm, it's

[3] ADDs are also referred to as Multi-terminal BDDs.

essential that we manage memory efficiently, and these tables are critical to that. One area where we deviate from the simple, textbook algorithm is in performing AND/OR *graph* search, rather than tree search. With the help of the *dup* table, we detect repeated states rather than regenerating them. In our early development work, this was found to be critical to a usable implementation: without the duplicate table, the algorithm choked on repeated states. However, because the textbook algorithm presentations did not clearly present assumptions, invariants, pre-conditions, or post-conditions, building a correct graph-based variant caused us considerable effort.

The search algorithm falls into three major blocks. The first, between lines 15 and 29, reconstructs the best partial solution, π, and builds a set of non-terminal tip (NTT) nodes, ntt. The latter serves as our open list. In the hopes of solving, and thus garbage-collecting, search nodes more quickly, we choose NTT nodes deepest-first. Right now, we maintain a set of NTT nodes, but we could easily save only a single "next node." It's not clear that the savings would be significant.

The second block covers the actual expansion of the chosen NTT node, u, between lines 30 and 44. The expand operation finds the children of the node u and computes their heuristic function values using the heuristic, h.

Finally, probability information from the children of u must be propagated up the tree. Starting at u, we update the cost-to-go values based on information at the children. At each OR node, we must also check to see if the best child choice for that node has changed, based on the new information. This loop appears between lines 45 and 54.

One issue that we found in our development was that we were inadvertently bridging two largely disjoint components of PRISM. The first component performs dynamic programming using ADDs. In this component, states are not really first-class objects: instead, reasoning is done in terms of sets of states, implicitly represented using decision diagrams. The second component is the codebase to support performing simulation, where the first class entities are states and traces through states, rather than sets of states. We bridged these two components because our forward search explored individual states, like the PRISM simulator, but we represented our heuristic function using ADDs (see below). There were some minor awkwardnesses when we tried to use capabilities from both of these components together. For example, we had to add some methods to support looking up the entry for an individual state in an ADD. We hope our extensions will prove useful to other PRISM programmers.

4 The Heuristic

In order to get acceptable performance from AO*, we must have an *informed* and *admissible* heuristic. The heuristic must be admissible – that is, for reachability, it must provide an *upper bound* on the probability of reaching the goal. Without this property we cannot ensure that we will find the optimal adversary. The heuristic must be informed, or the search will be inefficient We have used heuristics inspired by methods that have been found effective in AI planning, in particular heuristics that are based on *relaxations* of the reachability problem that we are trying to solve. Our "metric disjunctive reachability heuristic" provides good guidance for several domains. We present it below, discuss its performance in the following section, and then discuss potential improvements in our conclusions.

Before experimenting with informed heuristics, we tested our AO* search algorithm with an uninformed admissible heuristic. This heuristic assigned a value to 1 to any state that was not a failure state (*i.e.*, a non-goal state at the time bound). All but the most trivial problems were practically unsolvable with the uninformed heuristic.

Our next step was to experiment with the use of simple reachability as a heuristic. For each state $\times$ time pair, we computed whether the goal state was reachable from that state (in the time bound), and used this zero-one heuristic to guide our search. We implemented this by computing backwards reachability from the goal state using BDDs, taking the probabilistic transition relation and abstracting it into a zero/one transition relation. To compute a labeling for each state, however, we must resolve nondeterministic choices. The way we do this is to choose the maximum reachability value over all of the action variables (in this simple reachability context, "maximum" is equivalent to "disjunction"). More formally, the operations for computing the heuristic value for t from $t+1$ is as follows:

Definition 3 (Disjunctive heuristic function)

$$h_t = G \vee (\exists(\vec{a})\, TD^{-1} \cdot h_{t+1})$$

where $\vec{a}$ are the action variables, TD is the determinized transition matrix, h_{t+1} is the (previously-computed) heuristic DD for time $t+1$, and G is the BDD for the goal state. For a time bound k, $h_k = G$.

The operations above are all provided by the CUDD ADD/BDD library (Somenzi 2001),[4] accessed through a Java interface provided by PRISM.

The disjunctive heuristic could be efficiently computed, but was not sufficiently informative. We have replaced it with a *metric* disjunctive heuristic that computes a more refined (over)estimate of the probability of reaching the goal from each state. This heuristic is a relatively straightforward variant of the disjunctive heuristic, except that we use the original transition relation, T, instead of the determinized version, TD. Now we cannot use existential quantification to remove the action variables, but must use maximization as the quantification method (thus preserving the admissible nature of the heuristic):

Definition 4 (Metric disjunctive heuristic function)

$$h_t = \max(G, (\max(\vec{a})\, T^{-1} \cdot h_{t+1}))$$

where $\vec{a}$ are the action variables, T is the transition matrix, h_{t+1} is the (previously-computed) heuristic DD for time $t+1$, and G is the BDD for the goal state. For a time bound k, $h_k = G$.

All of the operations needed to build these heuristic functions in ADDs can conveniently be performed using CUDD,

[4] `http://vlsi.colorado.edu/~fabio/CUDD/`

Name	Backoff	States	Transitions
wlan0	0	6,063	1,0619
wlan1	1	10,978	2,0475
wlan2	2	28,598	57,332
wlan3	3	96,420	204,744
wlan4	4	345,118	762,420
wlan5	5	1,295,336	2,930,128
wlan6	6	5,007,666	11,475,916

Figure 1: Size of Wireless LAN models.

bundled with PRISM, through the Java API PRISM provides. The preimage operations are performed by conjunction (for the deterministic version), or matrix multiplication for the metric variant. Max and existential quantification are also provided by CUDD, as are the maximum and disjunction operators. In our initial implementation, we build a time-indexed vector of ADDs, each of which represents the heuristic function of a state and time index, $h(s, t)$, by looping backwards through time from terminal goal states, using the recursive definitions provided above.

As we show in the following section, this heuristic estimate provides a good compromise between information content and efficient computability. However, for larger problems, our current representation is too naive, and further enhancements will be required. We discuss this further in our concluding notes about future work.

5 Experiments

We have implemented our approach and evaluated it on several domains. Our preliminary experiments illustrate both the promise of our technique and its limitations, and will guide our future work (see Section 7). Models and property files describe in this section are available at `http://rpgoldman.goldman-tribe.org/difts15-data/index.html`

Wireless LAN The wireless LAN example shows the potential of our search based approach. This domain is from a PRISM case study of the IEEE 802.11 Wireless LAN protocol (Kwiatkowska, Norman, and Sproston 2002). The protocol involves a randomized exponential backoff for retransmission of collided packets, to avoid repeated collisions. To scale up the size of the domain, we increase the maximum backoff value from 0 to 6. See Figure 1 for the size of the models in this domain.

The property we analyze is the probability of having two collisions within 100 time steps. The underlying probability of this property is the same across the varying maximum backoff parameter. Figure 2 gives runtime comparisons for our AO* approach versus PRISM's dynamic programming approach. This shows that an informed heuristic enables a search-based approach to exploit its potential to avoid irrelevant parts of the search space. To get a sense of the importance of the heuristic, we ran `wlan0` with an uninformative heuristic (all non-goal/non-failure states have admissible heuristic value of 1.0). Instead of approximately 0.7 second to solve the problem, with the uninformative heuristic, AO* took 6012 seconds. As this was the simplest of the problems, we did not run additional tests.

Name	Probes	States	Transitions
zeroconf1	1	31,954	73,318
zeroconf2	2	89,586	207,825
zeroconf4	4	307,768	712,132
zeroconf6	6	798,471	1,833,673
zeroconf8	8	1,870,338	4,245,554

Figure 3: Size of IPv4 Zeroconf Protocol models.

IPv4 Zeroconf Protocol The IPv4 Zeroconf Protocol, another PRISM case study example, dynamically configures IP addresses in an ad-hoc network, to avoid setting up static IP addresses for each device or a DHCP server (Kwiatkowska et al. 2006). The state size is scaled up by the number of probes it sends out before claiming an IP address. The size of the models are in Figure 3.

AO* can handle these domains, but it is much slower than PRISM on them. See Figure 4: the runtime of PRISM on these problems is so fast by comparison that the plot of its runtime cannot easily be distinguished from the X axis. The overhead of building the heuristic in the current implementation overwhelms any time savings. Also, as the problems get larger, it's not clear that the heuristic is provided useful guidance. We need further investigation to understand the distinction between these two models.

Robot problems Finally, we considered a parameterized set of robot navigation problems, developed by the CMU researchers as part of their work on sampling approaches to PA model checking (see Section 6). These domains are designed to be difficult for systems that use ADDs, because the transition function was intentionally made to foil the kind of abstraction that ADDs perform. Unsurprisingly, both PRISM's dynamic programming and our AO* search fail on these problems, running out of memory as they compute preimages. It will be interesting to see if AO* search, coupled with a heuristic that uses abstraction (see Section 7) can do better on these problems.

6 Related Work

Henriques, *et al.* (2012) have addressed the challenges of verifying PAs using a sampling method. Their Statistical Model Checking for Markov Decision Processes (SMCMDP) algorithm operates in two phases. The first phase resolves the PA's nondeterminism with an initial probability distribution, and then uses multiple rounds of Monte Carlo sampling and Reinforcement Learning to improve the policy for nondeterministic choices with respect to satisfying a Bounded Linear Temporal Logic (BLTL) property. The second phase uses the best learned policy to reduce an MDP to a fully probabilistic Markov chain, on which known statistical model checking methods may be applied to give an

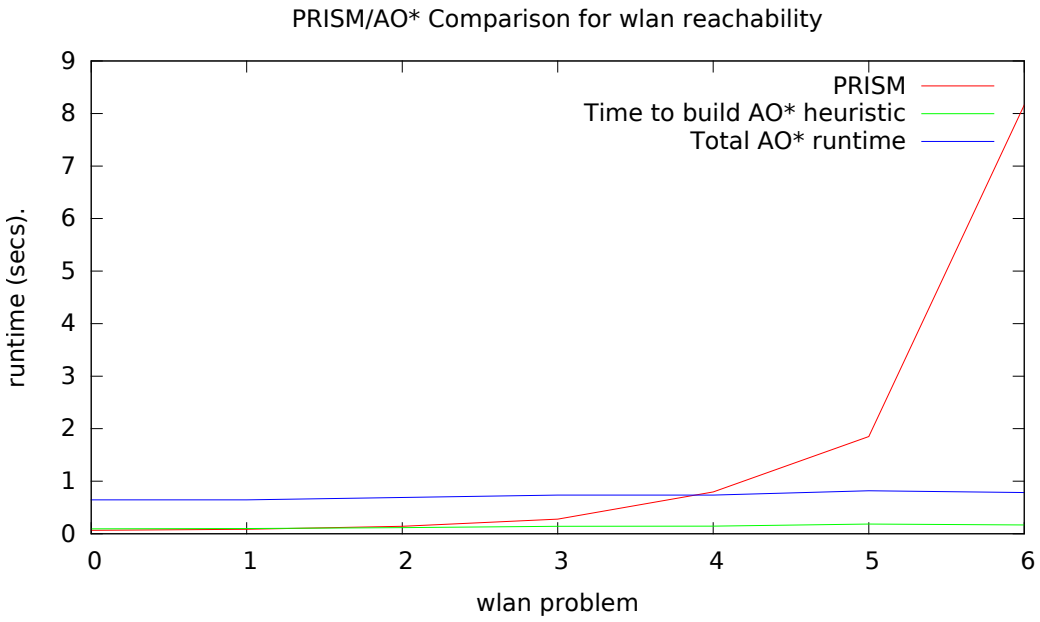

Figure 2: Comparison between AO* algorithm and PRISM's dynamic programming on scaled WLAN problems.

approximate solution to the problem of checking the probabilistic BLTL property. Unfortunately, there are a number of challenges to this approach: The Monte Carlo sampling process in SMCMDP can take a long time to converge. This problem can manifest itself either in the first phase where reinforcement learning is used to find an adversary policy, or in the second phase when, after the non-determinism has been resolved, we sample from the resulting Markov Chain to evaluate the property's worst case probability. Another problem is that much of the performance improvements of the SMCMDP technique come from the fact that it computes a stationary policy, instead of a non-stationary one, providing a substantial time and space saving. Unfortunately, we know of no way to estimate how far from optimal a stationary policy may be for a bounded reachability problem, so it's not clear how useful the output of the current SMCMDP algorithm is. Finally, it's not clear that the reinforcement learning algorithm is guaranteed to converge in this case. The proofs of convergence for such algorithms rely upon learning being applied to a Stochastic Shortest Path (SSP) problem (Bertsekas and Tsitsiklis 1996), and this reachability problem is not an SSP.

Our interest in the issues raised by SMCMDP led us to investigate other algorithms that addressed convergence issues for non-SSPs (Kolobov, Mausam, and Weld 2012; Kolobov et al. 2011). These algorithms addressed non-SSPs in the context of probabilistic AI planning. They did not directly address our bounded model-checking problem, because they addressed *indefinite* horizon problems. These are problems that would terminate when a goal state was reached, but otherwise did not terminate. They also had cost models that did not fit our reachability problems.

Our work was inspired by the paradigm of *guided model-checking*, which has aimed to bring together model-checking and heuristic search, particularly heuristic search techniques from AI planning (Edelkamp et al. 2008). However, we are not aware of other work applying guided model-checking to probabilistic automata checking. There is a rich vein of related work on probabilistic planning, however. Bonet and Geffner have identified a very abstract paradigm for solving such problems, which they call "find and revise," which unifies dynamic programming and search-based algorithms for problems such as MDPs (Bonet and Geffner 2003). Brázdil, *et al.*have used such techniques for unbounded model checking of MDPs (Brázdil et al. 2014).

7 Conclusions

Our paper provides a preliminary report on the use of heuristic search to model check bounded reachability problems for probabilistic automata. Early results show that the technique is promising. However, we must make improvements to the implementation in order to realize this promise for a wider class of problems.

We are considering a number of improvements to the implementation of the heuristic function, and some variations of the search method. The current implementation of the heuristic function is too naive for many difficult problems. In the worst case (where the decision diagram does not efficiently represent the problem structure), computing our heuristic function can be as bad as solving the original problem with dynamic programming. Part of this is a problem of our current implementation, which does not efficiently represent the heuristic functions across the time indices. However, for very difficult problems, more drastic improvements

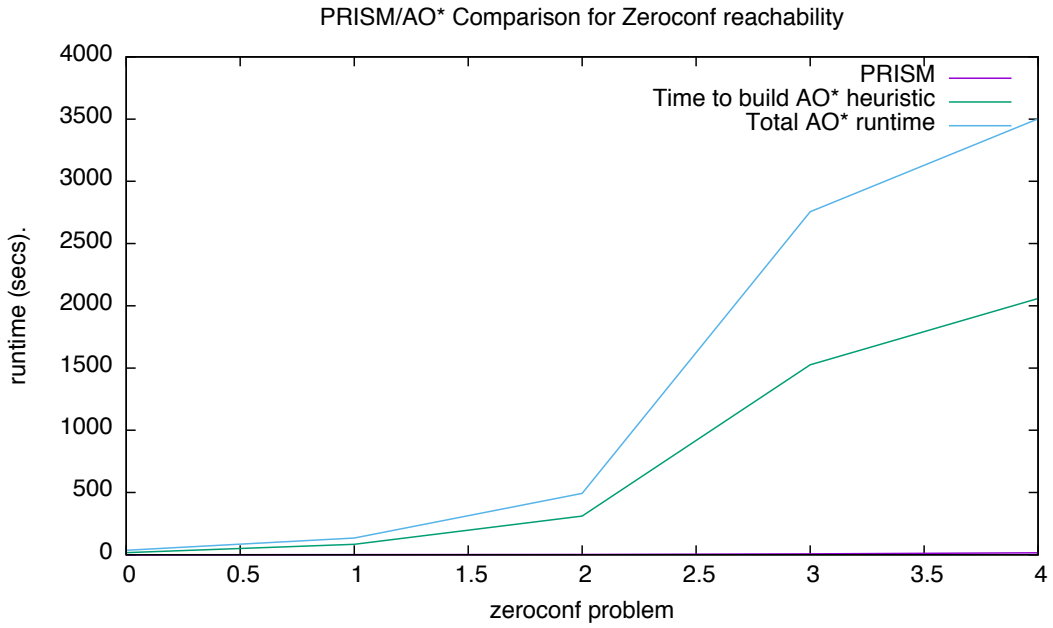

Figure 4: Comparison between AO* algorithm and PRISM's dynamic programming on scaled Zeroconf problems.

will be needed.

We believe further abstraction may provide a solution. We have a partial implementation of an abstraction-based approach adapted from the MDP planner APRICODD (St-Aubin, Hoey, and Boutilier 2000). APRICODD collapses together ADD entries for similar values. Unfortunately, while the abstraction itself is effective, the algorithm provided by APRICODD does not efficiently detect the entries for merging. We are working to improve this. An alternative abstraction method is to abstract the feature space of the problem, by eliminating some of the state variables, and using the resulting, simplified and small transition relation, in the heuristic (Edelkamp 2002). An extension to this form of abstraction is to have *multiple* abstract heuristic functions and then admissibly fusing the individual $h()$ values into an admissible best bound (Katz and Domshlak 2010).

Note that the value of our heuristic function (and variants) is not limited to our own heuristic search system. It could also be used with some sampling methods, including our MCTS-based approach, or with find-and-revise based methods.

We are also considering whether to experiment with AO* *branch-and-bound* search (Marinescu and Dechter 2009; Otten 2013) as an alternative to AO* search. The guarantee to provide the optimal solution first, provided by A* search, and its AO* variant, frequently comes at the space cost of having to actively maintain a large portion of the search space, and the time cost of constantly switching attention from one portion of the search space to another. In practice A* search is often too inefficient to be practical, and AO* is likely to share this problem. A branch-and-bound search variant, quickly finding a (generally suboptimal) solution, and then using that solution to prune substantial parts of the search space, may provide a better technique for solving these problems. Branch and bound search might also combine well with an unsystematic approach such as our MCTS solver: an initial result from MCTS would drive extensive pruning, and then could either be validated as optimal, or improved by systematic search.

Acknowledgments Thanks to Jordan Thayer and Dan Bryce for many helpful discussions about search and the use of decision diagrams. Thanks to Dave Parker for assistance navigating the internals of PRISM. This research was sponsored by the Air Force Office of Scientific Research and AFRL via PO 284227 under CMU prime contract FA9550-12-1-0146.The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Bagchi, A., and Mahanti, A. 1983. Admissible heuristic search in AND/OR graphs. *Theoretical Computer Science* 24(2):207–219.

Bahar, R.; Frohm, E.; Gaona, C.; Hachtel, G.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic decision diagrams and their applications. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD-93)*, 188–191.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-dynamic Programming*. Belmont, MA: Athena Scientific.

Boldt, M. W.; Goldman, R. P.; and Musliner, D. J. 2015a. Offline Monte Carlo tree search for statistical model checking of Markov decision processes. Submitted.

Boldt, M. W.; Goldman, R. P.; and Musliner, D. J. 2015b. Online Monte Carlo tree search for sampling large Markov decision processes. Submitted.

Bonet, B., and Geffner, H. 2003. Faster heuristic search algorithms for planning with uncertainty and full feedback. In Gottlob, G., and Walsh, T., eds., *Proceedings of the International Joint Conference on Artificial Intelligence*, 1233–1238. Morgan Kaufmann.

Brázdil, T.; Chatterjee, K.; Chmelik, M.; Forejt, V.; Kretínský, J.; Kwiatkowska, M. Z.; Parker, D.; and Ujma, M. 2014. Verification of markov decision processes using learning algorithms. In Cassez, F., and Raskin, J., eds., *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, 98–114. Springer.

Edelkamp, S., and Schrödl, S. 2012. *Heuristic Search - Theory and Applications*. Academic Press.

Edelkamp, S.; Schuppan, V.; Bosnacki, D.; Wijs, A.; Fehnker, A.; and Aljazzar, H. 2008. Survey on directed model checking. In *MoChArt*.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, 274–283. AAAI.

Hansson, H., and Jonsson, B. 1994. A logic for reasoning about time and reliability. *Formal aspects of computing* 6(5):512–535.

Henriques, D.; Martins, J.; Zuliani, P.; Platzer, A.; and Clarke, E. 2012. Statistical model checking for markov decision processes. *9th International Conference on Quantitative Evaluation of SysTems (QEST)*.

Katz, M., and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174(12-13):767–798.

Kolobov, A.; Mausam; Weld, D. S.; and Geffner, H. 2011. Heuristic search for generalized stochastic shortest path MDPs. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *ICAPS*. AAAI.

Kolobov, A.; Mausam; and Weld, D. S. 2012. A theory of goal-oriented MDPs with dead ends. In de Freitas, N., and Murphy, K. P., eds., *UAI*, 438–447. AUAI Press.

Kwiatkowska, M.; Norman, G.; Parker, D.; and Sproston, J. 2006. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design* 29:33–78.

Kwiatkowska, M.; Norman, G.; and Parker, D. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., and Qadeer, S., eds., *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, 585–591. Springer.

Kwiatkowska, M.; Norman, G.; and Sproston, J. 2002. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In Hermanns, H., and Segala, R., eds., *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, 169–187. Springer.

Marinescu, R., and Dechter, R. 2009. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence* 173(16–17):1457–1491.

Otten, L. 2013. *Extending the Reach of AND/OR Search for Optimization in Graphical Models.* Ph.D. Dissertation, UCLA.

Parker, D. 2011. Probabilistic model checking. Lectures notes available through PRISM website: `http://www.prismmodelchecker.org/lectures/pmc/`.

Puterman, M. L. 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons, Inc.

Sokolova, A., and De Vink, E. P. 2004. Probabilistic automata: system types, parallel composition and comparison. In *Validation of Stochastic Systems*. Springer. 1–43.

Somenzi, F. 2001. Efficient manipulation of decision diagrams. *STTT* 3(2):171–181.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2000. APRICODD: Approximate policy construction using decision diagrams. In *NIPS-2000*, 1089–1095.

A AO* algorithm

backpropagate(s) subroutine

1: $o \leftarrow \text{solved}(s)$; ▷ Old values of f and solved
2: $f \leftarrow f(s)$;
3: **if** AND(s) **then**
4: $f(s) \leftarrow \sum_{c \in \text{successors}(x)} f(c) \cdot p(c|s)$;
5: **for all** $c \in \text{successors}(s)$ **do**
6: $\text{markedChildren}(s) \leftarrow \text{markedChildren}(s) \cup c$;
7: $\text{solved}(s) \leftarrow \bigwedge_{c \in \text{successors}(s)} \text{solved}(s)$;
8: **end for**
9: **else** ▷ s is an OR node
10: $b \leftarrow \text{argmax}_{c \in \text{successors}(s)} f(c)$; ▷ b is best child
11: $\text{solved}(s) \leftarrow \text{solved}(b)$;
12: $\text{markedChildren}(s) = \{b\}$;
13: **end if**
14: **return** $\neg o \wedge \text{solved}(s) \vee f \neq f(s)$; ▷ parent needs update

AO^* Search

```
▷ Data structures:
PriorityQueue ntt ← newPQ();                                   ▷ the openlist (nonterminal tip nodes)
Set dup ← ∅;                                                   ▷ duplicates table: persistent
Set of states markedChildren(s);                               ▷ children of s in best partial solution
▷ the following are repeatedly cleared and repopulated
Priority Queue Z
Dequeue π                                                      ▷ best partial solution
OrState s ← new OrState(model initial state);
boolean done ← ⊥;
repeat
    done ← ⊤;
    ntt ← ∅;
    clear(π);
    push(π, s);                                                ▷ Initial state is always in the solution
    ▷ Refill the NTT (non-terminal leaf states) queue with the NTT states from the partial solution, π:
    Set v ← ∅;                                                 ▷ Visited set
    while π ≠ ∅ do
        SearchState c ← pop(π);
        if c ∈ v or solved(c) then
            continue;
        else
            v ← v ∪ c;
        end if
        if isNTT(c) then
            ntt ← ntt ∪ c
        else
            π ← π ∪ markedChildren(c);
        end if
    end while
    ▷ Expand the next search state:
    SearchState u ← pop(ntt);
    successors(u) ← expand(h, dup);
    for all SearchState v ∈ successors(u) do
        done ← ⊥;
        if solved(v) then
            continue;
        else if isGoal(v) then
            h(v) ← 1;
            solved(v) ← ⊤;
        else if isTimeout(v) then
            h(v) ← 0:
            solved(v) ← ⊤;
        end if
    end for
    ▷ Propagate updates from expanded state
    clear(Z);                                                  ▷ Z is the set of states that need to be updated.
    Z ← Z ∪ u;
    while Z ≠ ∅ do
        SearchState x ← pop(Z);                                ▷ select from Z an element with no descendant in Z
        boolean updateParent ← backpropagate(x);
        if updateParent then
            Z ← Z ∪ ancestors(x);
        end if
    end while
until solved(s) ∨ done;
return s;
```