Memory-Efficient Graph Search in Planning and Model Checking

Peter Lamborn and **Eric A. Hansen**
Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
pcl16@msstate.edu

Abstract

Graph search plays a central role in both planning and model checking. We describe how to adapt an approach to memory-efficient graph search that has been used for domain-independent STRIPS planning in order to create a more memory-efficient domain-independent model checker. We discuss some changes to this approach that are required by the differences between planning and model checking, and report preliminary results that indicate its effectiveness.

Introduction

Model checking is a fully-automated formal technique for verifying that a property is satisfied, or modeled, by a transition system (Clarke, Grumberg, & Peled 2000). The transition system is often a protocol, such as a protocol for wireless communication or an embedded hardware device. A violation of the property is an error. For example, in a wireless protocol, an error could be starvation of a device. Model checking can find errors that are difficult to detect by testing and simulation, and is especially effective in finding errors with long traces or errors that occur after unusual sequences of events. Model checking can also verify that no errors exist, which is impossible using testing and simulation.

There are several important connections between research in model checking and research in automated planning and heuristic search. First of all, there is a large body of work on planning via model checking (Giunchiglia & Traverso 1999). In this work, planning domains are formalized as transition systems, goals are expressed as temporal formulas, and planning is done by using model checking to determine whether the temporal logic formula is true in the transition system. Another important connection between automated planning and explicit-state model checking is that both rely on state-space exploration using graph search. For both, scalability is limited by the state explosion problem – the size of the state space grows exponentially in the number of variables in its description. Finally, in both planning and model checking, there is an emphasis on domain-independent tools; there is a focus on developing domain-independent model checkers just as there is interest in developing domain-independent planners.

This short paper describes preliminary work on using an approach to memory-efficient graph-search originally developed for domain-independent planning in order to improve the scalability of a domain-independent model checker. The approach we adapt is called breadth-first heuristic search with layered duplicate detection (Zhou & Hansen 2006). Layered duplicate detection turns out to be a very effective approach to reducing the memory requirements of model checking because it can be easily implemented in a domain-independent way and works well in searching directed graphs; model checking always involves directed graphs. In our adaptation, we use breadth-first search without a heuristic to perform a complete search of a graph, in order to verify that a property always holds. We report preliminary empirical results that illustrate that this approach can significantly improve the range of models that can be verified.

Background

Graph search and model checking

In model checking, a transition system (e.g., for a protocol) is represented by a directed graph in which the nodes correspond to states of the system and the edges correspond to state transitions. For example, a state may indicate two packets in flight, a valid transition could be the reception of a packet, and the new state would contain just one packet in flight. The graph has an initial state, and the set of all possible paths in the graph represents the set of all possible behaviors of the protocol. Given this representation, a property can be verified by a complete search of the graph to make sure that no error states can be reached. If an error is found, a trace of the error (i.e., a path from the initial state to the error state) is returned and used to debug the protocol.

The scalability of model checking is limited by the size of the graph that must be searched. In explicit-state model checking, every generated state is stored in a hash table that is used for duplicate detection, which is the process of determining whether or not a newly-generated state is a duplicate of a previously-generated state. Since a complete search of the graph is needed to verify a system, the memory needed to store all generated states is the bottleneck of model checking. As a result, some approaches to model checking abandon verification and merely attempt to detect errors in a swift manner. Heuristic search algorithms such as A* have been used to efficiently find paths to error states, which are treated as goal states (Edelkamp, Lluch-Lafuente, & Leue 2001). Other approaches, such as randomized search, have also been used (Jones *et al.* 2003).

Frontier search and layered duplicate detection

Frontier search is a memory-efficient approach to graph search that only stores the Open list, and saves memory by not storing the Closed list (Korf *et al.* 2005). Instead of the traceback method of solution recovery, it uses a divide-and-conquer method that involves finding an intermediate node along an optimal path and using it to divide the search problem into two subproblems – the problem of finding an optimal path from the start node to the intermediate node, and the problem of finding an optimal path from the intermediate node to the goal node. The subproblems are solved recursively by the same search algorithm until all nodes along an optimal solution path for the original problem are identified.

Since frontier search only stores nodes on the frontier, it needs to prevent already-explored nodes that are no longer in memory from being regenerated. In undirected graphs, Korf et al. (2005) use a technique called *used-operator bits* that is very effective. But directed graphs present more of a challenge. In the approach proposed by Korf et al., each time a node is expanded and its successors generated, all predecessors of these successors are also generated, even if no path has yet been found to these nodes. These *dummy nodes* are stored in the Open list with an g-cost of infinity until a path to them is found, at which point they acquire the g-cost of that path. Although this guarantees that no node is generated more than once, the dummy nodes would not be generated by a standard graph-search algorithm and there is no bound on the number of dummy nodes that can be generated. In some cases, the additional overhead for generating dummy nodes can make search performance worse.

Zhou and Hansen (2006) propose an alternative approach to duplicate detection that can be used in breadth-first search. In this approach, called layered duplicate detection, the Closed list is stored in layers, one for each g-cost, and earlier layers are deleted to recover memory. In undirected graphs, they point out that keeping just one previous layer in memory is sufficient to detect all duplicates. In directed graphs, they propose keeping one or more previous layers, where the number needed to prevent all duplicates depends on the structure of the graph. They also point out that even if no previous layers are kept in memory, the number of times a node can be regenerated is bounded by the depth of the search. In depth-first search, by contrast, the number of node regenerations can be exponential in the depth of the search.

Layered duplicate detection is easy to implement in a domain-independent way, in contrast to the approach to duplicate detection used in Korf et al.'s implementation of frontier search. Therefore, we use frontier breadth-first search with delayed duplicate detection to reduce the memory requirements of our domain-independent model checker.

Algorithm and preliminary results

We briefly describe our approach to memory-efficient model checking and present some preliminary experimental results.

Algorithm

The starting point for our implementation is the Murϕ model checker, a domain-independent tool that takes a description of a model as input and uses breadth-first search to verify that the model is correct (Dill 1996). If an error is found, it returns an error trace.

Since the graphs that are searched in model checking are directed graphs, we modify the breadth-first search algorithm to use layered duplicate detection. First of all, this requires indexing the hash table of stored states by layers, so that individual layers can be deleted as the search progresses. The breadth-first search proceeds as usual until it begins to run out of memory. At that point, it recovers memory by deleting layers of generated states from memory. The layers it deletes are those that are furthest from the frontier (i.e., the shallowest layers), since they are less likely to be useful in duplicate detection. Any closed nodes are eligible for deletion, if more memory is needed. Thus, as long as the Open list can fit in memory, the search continues. (If the Open list does not fit in memory, we could use beam search to continue to search for an error, but we do not consider this possibility in this paper.)

If an error is found, the error trace is recovered by using the traceback method to follow pointers backwards from the error state through as many layers as still reside in memory. If some layers are missing, the shallowest state in the error trace is treated as a goal state, and another search is conducted to find a path from the initial state to this state, in order to finish recovering the error trace. This is a modification of the divide-and-conquer technique for solution recovery.

Theoretical properties

Because the search is breadth-first, if an error is found, the error trace is guaranteed to be a shortest path to the error state. If no error is found, the search terminates when the Open list is empty and there are no more states of the graph to explore. If the search terminates in this way without finding an error, the model is verified.

However, when layers of the search graph are deleted to save memory, the search is no longer guaranteed to terminate with an empty Open list, even if the graph is finite. If some layers of the search graph are removed from memory, it is possible for duplicate nodes to be generated during the search. Zhou and Hansen (2006) give some conditions under which no duplicates will be generated, but these do not hold in general. In the worst case, the number of times a node can be regenerated is bounded by the depth of the search (d) and the number of of layers stored (l), and no state can be duplicated more than d/l times. In practice, duplicates are generated even less frequently than this. But the possibility of regenerating previously explored nodes means that there is no guarantee the search will terminate with an empty Open list, since the search can repeatedly regenerate and re-explore the same parts of the search graph.

If the model contains an error, however, breadth-first search is guaranteed to terminate by finding a shortest path to this error, no matter how many duplicates may be generated. This points to a second way to detect termination

Model	Peak Nodes in Memory	Total Nodes Generated	Number of Layers
cache32622	14,141	111,335	8
newlist8	3,930,856	24,714,307	44
arbiter13	1,521,655	11,545,717	17
arbiter14	1,429,446	11,255,853	14
adash1313e	684,946	13,480,127	15
ns22110	87,366	173,866	8
dynpart4	2,619,984	16,202,474	30

Table 1: Memory savings for seven different models.

and verify a model. If the diameter of the search graph is known, or can be bounded, and no error is found by the search algorithm in searching up to this depth, then no errors can be present and the model is verified. (If the search algorithm generates any nodes past this depth, they must be duplicates.) This method of detecting termination is similar to that used in bounded model checking, which uses satisfiability testing to verify that a model does not have any errors up to depth k (Clarke *et al.* 2001). If the model is verified for a depth that is equal to or greater than the diameter of the graph, called the *completeness threshold*, verification is complete. Various methods for determining a completeness threshold have been explored in the literature on bounded model checking, and can be applied in our approach to model checking.

Preliminary empirical results

Table 1 shows the performance of our modified algorithm in verifying seven different models. They include a cache coherency protocol with six caches and two disks (cache32622), a protocol for maintaining a linked list in a parallel environment with eight parallel nodes (newlist8), a mutual exclusion protocol with thirteen or fourteen threads and an arbiter in charge of who gets access (arbiter13 and arbiter14), the DASH communications protocol for a network of three nodes (adash1313e), the Needham-Schroeder protocol for secure communication for four participants with two intruders (ns22110), and an algorithm for dynamically partitioning a search with four processing nodes (dynpart4).

In model checking, the node data structure is typically very large because the state description includes many variables that can have a large range of possible values. For the cache coherency model, for example, the size of a single node is 167 bytes. Therefore, even though the models in Table 1 are relatively small, none can be verified using 1GB of memory without using our layered duplicate detection technique. The difference between the peak number of nodes stored and the number of nodes generated during the search gives a sense of how this approach improves scalability by allowing larger models to be verified using the same amount of memory. Note that because we use blind breadth-first search, the Open list it typically much smaller than the Closed list, and deleting the Closed list from memory saves a substantial amount of memory and significantly improves scalability.

Future Work

Besides testing this approach on larger and more varied models, we hope to eventually extend it in several ways. When the Open list does not fit in memory, we plan to use disk to continue the search. While disk provides much more storage than internal memory, it is also limited, and our approach will allow complete search of graphs that do not fit on disk. For example, Korf and Schultze (2005) describe a complete breadth-first search of the Fifteen Puzzle that requires 1.4 terabytes of disk just to store the search frontier. Their frontier search algorithm is for undirected graphs and specialized for the Fifteen Puzzle, whereas our approach will allow domain-independent search in directed graphs.

We will also consider parallel search algorithms. Our approach to both external-memory and parallel graph search will be based on *structured duplicate detection*, which has been shown to be very effective in domain-independent planning (Zhou & Hansen 2004). Thus, we will continue to leverage the close connections between graph search in domain-independent planning and model checking.

References

Clarke, E. M.; Biere, A.; Raimi, R.; and Zhu, Y. 2001. Bounded model checking using satisfiability solving. *Formal Methods in System Design* 19(1):7–34.

Clarke, E. M.; Grumberg, O.; and Peled, D. 2000. *Model Checking*. MIT Press.

Dill, D. 1996. The Murϕ verification system. In R. Alur, and T. Henzinger., eds., *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, 390–393. Springer Verlag.

Edelkamp, S.; Lluch-Lafuente, A.; and Leue, S. 2001. Directed explicit model checking with HSF-SPIN. In Dwyer, M. B., ed., *Proc. of the 8th Int. SPIN Workshop*, volume 2057 of *Lecture Notes in Computer Science*, 57–79.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *Proceeding of the Fifth European Conference on Planning (ECP 1999)*, volume 1809 of *Lecture Notes in Computer Science*, 1–20. Springer.

Jones, M.; Mercer, E.; Bao, T.; Kumar, R.; and Lamborn, P. 2003. Benchmarking explicit state parallel model checkers. In *Proceedings of the 2nd International Workshop on Parallel and Distributed Methods in Verification*, 84–98.

Korf, R., and Schultze, P. 2005. Large-scale parallel breadth-first search. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, 1380–1385.

Korf, R.; Zhang, W.; Thayer, I.; and Hohwald, H. 2005. Frontier search. *Journal of the ACM* 52(5):715–748.

Zhou, R., and Hansen, E. 2004. Structured duplicate detection in external-memory graph search. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, 683–688.

Zhou, R., and Hansen, E. 2006. Breadth-first heuristic search. *Artificial Intelligence* 170:385–408.