Towards Predictive Execution Monitoring in BDI Recipes

Extended Abstract

Mika Barkan
Bar Ilan University
Ramat Gan, Israel
barkanm1@biue.ac.il

Gal A. Kaminka
Bar Ilan University
Ramat Gan, Israel
galk@cs.biu.ac.il

KEYWORDS

Execution Monitoring; BDI

ACM Reference Format:
Mika Barkan and Gal A. Kaminka. 2019. Towards Predictive Execution Monitoring in BDI Recipes. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

Agents do not only generate and choose plans for execution, they also monitor the execution of plans and handle contingencies [1, 5, 6]. The capacity for *execution monitoring* allows agents to assess the execution of plans, determine the need for re-planning, identify opportunities, and re-evaluate goal selection.

In practice, many BDI plan execution systems focus only on the current plan step. They do not project ahead the current knowledge of the agent to determine implications on future steps. Thus a failure of a future plan-step, which may already be predictable with given the current knowledge of the agent, is not detected until the last possible moment.

This paper examines the task of *predictive* execution monitoring in BDI recipes. Such capability is similar in principle to BDI planning [4, 7, 8], in the sense that both tasks require prediction of future states, based on simulation of actions taken. However, monitoring of recipes does not require ordering decisions, this is already defined by the structure of the recipe, and would seem to therefore require lighter computation. Alas, this is not the case.

We discuss a base algorithm for predictive execution monitoring, intended for hierarchical recipes. We show that its complexity is super-exponential in the general case. We then discuss several methods for reducing the projected execution space. We evaluated these methods in various combinations in hundreds of experiments.

2 BACKGROUND AND RELATED WORK

Unlike plan-based monitoring, BDI recipes have only partial information about the effects of plan steps, and thus the number of possible changes that can occur grows super-exponentially. Moreover, BDI recipes are very often hierarchical. Thus execution monitoring of hierarchical plans is relevant. de Silva *et al.* has shown the close similarities between BDI systems and HTN planning [3]. In their work they compare the runtime of an HTN planner and BDI system in both static and dynamic environments, using the blocks world environment. Their work shows that the BDI system has better results both in static and dynamic environments. However, the problems are created such that there is no need for an HTN-style lookahead (prediction). This is done since BDI does not have the capabilities to do so. In contrast, such capability for prediction is exactly what we seek to investigate.

Sardina *et al.* [7] used HTN planner to add lookahead capabilities to BDI for planing purposes. As in [3] the HTN planner derives its knowledge from the plan library of the BDI agent and its beliefs. The HTN planner is invoked and does a full lookahead search. If a plan is found then the BDI agent will follow it until goal is reached or until a step in the plan is no longer possible. Detection of such a failure occurs late. To address this, the algorithms we present attempts to provide early detection of failures.

Belker *et al.* [2] used HTN planning to estimate the outcome of actions in navigation tasks. This in turn allows the agent to choose alternative actions (if available) that improve the projected outcome over the original chosen action, and results in a considerable performance improvement (42%). Encouraged by this, we seek to use predictions to improve the execution of BDI plans in general.

3 PREDICTIVE RECIPE MONITORING

Predictive execution monitoring begins with (i) a recipe, (ii) the current execution state in the recipe (that is, which behaviors are currently selected), and (iii) the current knowledgebase of the agent. It then projects ahead, given the current beliefs of the agent, whether any future behaviors can be shown to be *un-selectable*, given potential changes to the beliefs of the agent, by behaviors preceding this future behavior, in the execution.

3.1 BDI Beliefs, Recipes, and Plans

A BDI agent has a knowledgebase of beliefs, which are revised and modified during the operation of the agent.

The BDI recipe is an augmented connected directed graph where vertices represent *behaviors* (see below). Hierarchical *task-decomposition* edges, allow a higher-level behavior to be broken down into lower level behaviors, until reaching a primitive behavior. *Sequential* edges, constrain the execution order of behaviors, a sequential edge from b_1 to b_2 specifies that b_1 must be executed before executing b_2. Sequential edges may form circles, but hierarchical edges cannot.

Behaviors change the current beliefs of the agent (knowledgebase), and its state in the world (e.g., a command to move forward, changing its position in the world). Every behavior b is associated with the following: preconditions, a set of beliefs that need to be true (in the knowledgebase) in order for this behavior to be selectable by the agent; termination conditions, a set of beliefs that signal that execution of the behavior should terminate (typically, because of the achievement of the behavior goal, or its failure); and support, a set of beliefs whose value might be changed by the behavior.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

3.2 Predicting Execution Possibilities

A base line algorithm searches *the space of possible recipe executions*. Each discrete point in this space is a combination of a valid path through the recipe graph (along hierarchical and sequential edges), coupled with the knowledgebase which holds at the end of the path. With each search iteration, the algorithm considers extending the path structurally. Each such expansion can involve multiple possible knowledgebase revisions. Thus each search iteration results in multiple discrete points in the search space, to be considered.

The algorithm proceeds by iterating over a queue of execution traces to be considered. Each search node q contains a valid *possible execution path*. This path records a potential execution trace (behaviors and beliefs), beginning with the agent's beliefs and behaviors when the algorithm was called. The execution path contains a sequence of behaviors selected for execution by the BDI executive, in response to possible revisions to the knowledgebase, made by behaviors and the knowledgebase at the time of selection of each node. If q is a leaf then it is a possible termination of the execution, and the path leading to it ($q.path$) is added to the set of successful executions. Otherwise, if all edges of the recipe graph are accounted for in the set of successful paths, then this means that no future behavior can be proven to be unselectable at this point, and thus the search can terminate.

The expansion of the search is as follows. First, the algorithm asks for the set of all possible expansions of the current search node q, by structural and belief revisions. This set is then pruned if possible, to reduce the number of such expansions (this key step is discussed below). Then, the new nodes are put on the queue and marked as visited, so they do not get expanded again.

The process bears some similarity to a BFS search through a graph, however we note that unlike BFS, the search does not stop when we found a single path to a target behavior, but continues examining other paths, to other behaviors.

3.3 Complexity

Even the simplest of recipes consisting of a single path of sequential links, has an exponential number of possible execution paths, due to the combinatorial explosion in the combination of termination conditions. Let us look at a simple graph with k nodes. Each expand called for an edge produces at the most t expanded nodes, the number of termination condition, since each termination condition can create a new and different knowledgebase. This expand is called for each combination of a node and a knowledgebase created before. This means that for the first edge there will be only t such paths created. The next edge will create for each such path another t expanded nodes, and so on. This gives us t^{k-1} search paths.

A directed acyclic graph (DAG) has combinatorial number of paths. For each path of length m we will have this complexity, this mean that even when we look at a graph plan that only has sequential edges and no cycles, we get an super-exponential worst case run time: a combinatorial number of paths, each generating a combinatorial number of execution paths to be considered. When we add hierarchical edges (which allow more complex paths), and when we allow cycles in our sequential edges, the runtime is exacerbated even further.

4 PRUNING POSSIBLE EXECUTIONS

We explore three different pruning methods, which cut the search space of possible executions.

Successful Visited. When a path p from successful paths already contains a tuple where the current vertex (from the expanded) with the same knowledge-base has been shown to be successful, then there is no use checking from the current node forward (the tuple from the successful path shows us that from this point on, the issue is resolved). Thus the only new information in the expanded tuple is in the prefix path, leading to the current node, which may be new. If so, we save it.

Cycle Detection. A cycle in a search graph is when we reach the same vertex again. In our case, since each search node also includes the path, and there are cycles in the recipe graph simply comparing the search node is not enough. Thus, to make sure the algorithm only goes in cycles through the graph until there is no new information to gain from the cycle. Cycle detection prunes a search node if a pair of the new knewledgebase and the node appear anywhere in the nodes path, excluding the last part we just added the pair too.

Merging paths. We observe that the role of the path p in each search node is to maintain information about which edges we can keep in our new plan. However, if a path leads to the same behavior with the same knowledgebase and same type of expand, then the checks from there on will be the same. So a new search node duplicating this check need not be added to the queue.

5 PRELIMINARY EXPERIMENTS

We empirically evaluated two independent issues. First, the impact of the graph structure and the impact of the knowledge state space size (as reflected by the number of termination conditions used in behaviors), on the actual runtime of the execution algorithm. Second, we seek to evaluate the efficacy of the different pruning methods we introduced.

Our preliminary results confirmed our complexity analysis, where we saw that the number of termination conditions for a behavior, increases the number of possible exploration options. The complexity of the problem is not only dependent on the number of behaviors in the recipe. Furthermore the experiments showed that breadth has more influence on the time complexity than depth.

In addition we saw that cycle detection does worse than the rest. Successful visited with cycle detection does slightly better than cycle detection alone, but not by much. On the other hand Merge Paths and all its combinations, finish faster and thus also finish more recipe graphs in the time given.

6 CONCLUSION

In this paper we presented the problem of predictive execution monitoring of BDI recipes; the ability to project forward the current knowledge of the agent to prevent future failures that can already be predicted. We then analyzed the complexity of the problem and showed that even on simple acyclic flat recipe graphs it is a super-exponential problem. We presented an algorithm that goes over the branches of the BDI recipe graph to try and find the branches that are predicted to fail. We then presented pruning methods to make the algorithm more efficient and reduce the running time. We proved that this pruning method are complete.

REFERENCES

[1] J. A. Ambros-Ingerson and S. Steel. 1988. Intergrating Planning, Execution and Monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)* (St. Paul, MN). AAAI Press, Minneapolis/St. Paul, MN.

[2] Thorsten Belker, Martin Hammel, and Joachim Hertzberg. 2003. Learning to optimize mobile robot navigation based on HTN plans. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, Vol. 3. IEEE, 4136–4141.

[3] Lavindra de Silva and Lin Padgham. 2004. A comparison of BDI based real-time reasoning and HTN based planning. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 1167–1173.

[4] Lavindra de Silva, Sebastian Sardina, and Lin Padgham. 2009. First principles planning in BDI systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 1105–1112. http://dl.acm.org/citation.cfm?id=1558167

[5] Charles Earl and R. James Firby. 1997. Combined Execution and Monitoring for Control of Autonomous Agents. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, W. Lewis Johnson (Ed.). ACM Press, Marina del Rey, CA, 88–95.

[6] Martha E. Pollack. 1990. Plans As Complex Mental Attitudes. In *Intentions in Communication*. MIT Press, 77–103.

[7] Sebastian Sardina, Lavindra de Silva, and Lin Padgham. 2006. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems* (2006) *(AAMAS '06)*. ACM, 1001–1008. https://doi.org/10.1145/1160633.1160813

[8] Andrzej Walczak, Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. 2006. Augmenting BDI agents with deliberative planning techniques. In *International Workshop on Programming Multi-Agent Systems*. Springer, 113–127. https://link.springer.com/chapter/10.1007/978-3-540-71956-4_7