Intention Progression with Maintenance Goals

Di Wu
Zhejiang University of Technology
wudi@zjut.edu.cn

Yuan Yao
University of Nottingham Ningbo
n.a.alechina@uu.ul

Natasha Alechina
Utrecht University
n.a.alechina@uu.ul

Brian Logan
Utrecht University
b.s.logan@uu.nl

John Thangarajah
RMIT University
john.thangarajah@rmit.edu.au

In this paper, we propose SA$_M$, a Monte-Carlo Tree Search (MCTS)-based solver for intention progression problem with both achievement and maintenance goals. We evaluate the performance of our approach in a range of benchmark scenarios with increasing difficulty. The results suggest that SA$_M$ significantly improves the performance of agents with maintenance goals.

Intention Progreesion For BDI Agent

- Plan selection: the problem of choosing which plan to the goal
- Intention selection: the problem of deciding which intention to execute
- Inention progression problem(IPP): Intention selection + Plan selection

- goal-plan tree(gpt): a hierarchical tree structure to represent the relationship between goal, plans and action and all the way to achieve a top-level goal
- intention: The agent's current intentions are represented by a set of pairs, $I = \{(t_1, s_1), \ldots, (t_n, s_n)\}$, where each $t_i \in T$ is a goal-plan tree corresponding to a top-level goal of the agent, and $s_i \in S$ is the current step of t_i.

- IPP with gpts: the problem of choosing a current step $s_i \in S$ to progress and decide how to progress it (e.g., if s_i is a (sub)goal, the problem also involves choosing a plan to achieve it) at each deliberation cycle so as to maximise the number of goals achieved (or other utility).

Maintenance Goals

Maintenance goals are goals that specify a state of the environment an agent should maintain. We Consider two ways of implementing maintenance goals:

- Proactive Maintenance Goals: If the violation of a maintenance goal can be reliably predicted, given the agent's other intentions, the agent can adopt a recovery plan to prevent the maintenance goal from being violated.
- Reactive Maintenance Goals: If the violation of a maintenance goal cannot be reliably predicted, the agent must adopt a recovery plan reactively, i.e., after the violation occurs.

SA$_M$ Scheduler

Each iteration of the SA$_M$ algorithm consists of 4 phases: selection, expansion, simulation and back-propagation.

An iteration of the SA$_M$ algorithm

As with MCTS and SA, each iteration of SA$_M$ consists of four main phases: selection, expansion, simulation and back-propagation.

Selection: Starting from the root node, we recursively follow child nodes with highest UCT value until a leaf node n_e is reached.

Expansion: n_e is expanded by creating all its child nodes which correspond to all possible action executions, and adding them to the search tree.

Simulation: One of the newly created child nodes, n_s, is selected at random. The value of the state in n_s is estimated by performing β pseudorandom simulations.

Back-propagation: All the simulation value is then back-propagated from the simulated states to all states on the path to the initial state.

To support maintenance goals, we modified the expansion and simulation phases of SA: We add additional functionality to deal with maintenance goals.

```
function expand(n_e)
    N ← ∅
    G_m ← getMaintenanceGoals()
    if satisfied(G_m) and has no achievement goals then
        continue
    for each g ∈ G_m do
        if one recovery plan for g is already adopted then
            continue
        if (isRMG(g) and isViolated(g, n_e)) or isPMG(g)
    then
            Act ← getExecutableActions(g)
            for each a ∈ Act do
                n' ← genNode(n_e, a)
                N ← N ∪ {n'}
    n_e.setChildren(N)
    for each n_i ∈ N do
        n_i.setParent(n_e)
```

```
function simulate(n)
    N ← expand(n)
    if N ≠ ∅ then
        n ← random(N)
        simulate(n)
    return n
```

After α iterations, the step that leads to the best child n_b of the root node n_0 is returned.

Evaluation

We evaluate the performance of SA$_M$ based on a Mars rover example [?]. The environment is a grid consisting of 20 × 20 cells (see Figure 7).

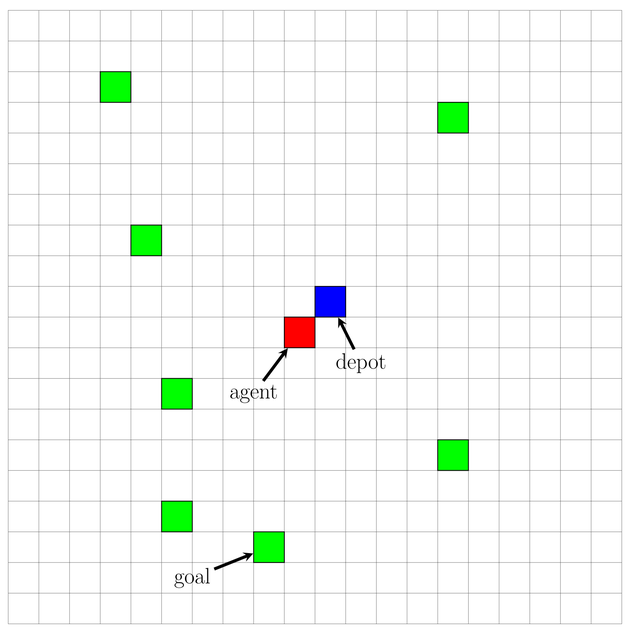

Figure 7: A Mars rover example

Each cell in the grid is a location represented as a pair of integer numbers (x, y), where x and y are the x-axis and y-axis values respectively. A Mars rover can move up, move down, move left and move right, and is asked to visit different locations in the grid. There are four actions the Mars rover agent can take to change its current locations, i.e., moveUp, moveDown, moveLeft and moveRight.

In all the experiments reported below, we assume the Mars rover starts in the depot cell, and all its goals (i.e., where to visit) are randomly generated. We measure the performance of the Mars rover agent based on two criteria: the number of goals achieved and the amount of battery consumed. To evaluate the performance of SA$_M$, the experiments were conducted in 4 cases: RMG, PMG, RMCTS and PMCTS. Both RMG and PMG are from [?].

The results are shown in Figure 8.

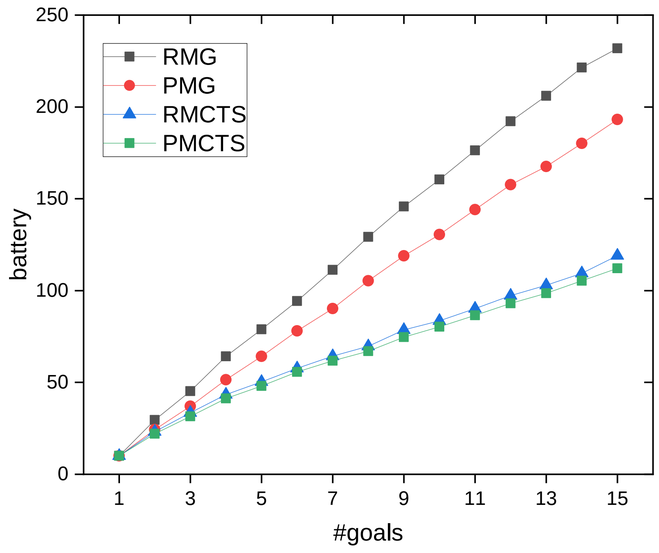

Figure 8: Battery consumptions with fixed capacity of 40

As can be seen, for all approaches the amount of battery consumption increases, as the number of goals increases. MCTS-based approaches (PMCTS, and RMCTS) outperform all other approaches, and approaches using proactive maintenance goals have better performance than those using reactive maintenance goals. As expected, RMG has the worst performance in all testing cases, i.e., it consumes more battery power than any other approaches. The PMG has better performance than RMG, as it can estimate the battery consumption before pursuing a goal. If the agent will trigger the maintenance goal half way achieving its next achievement goal, then it will choose to recharge first. Finally, both PMCTS and RMCTS have a clear advantage over PMG. Especially, when the given number of goals becomes larger, the differences between MCTS-based approaches and other approaches are more significant. The reason is that the MCTS-based approach can predict not only the possible violation of maintenance conditions during the execution but also possible synergies between different intentions (i.e., the agent can merge the same actions from different intentions to save time and resources).

Conclusion

The preliminary results suggested that even without giving any reliable prediction of future violation, SA$_M$ with reactive maintenance goal can still outperform both reactive and proactive approaches proposed in [?] in all cases.

The advantage of using SA$_M$ comes from not only the ability to avoid violation between achievement goals and maintenance goals, but also the nature that MCTS can interleave recovery plans and other intentions so as to avoid negative interaction and to exploit synergies if maintenance goals are implemented proactively.

Future work

- Extend SA$_M$ scheduler to work with nondeterministic actions in uncertain environment.
- Extend the current SA$_M$ to additional richer types of goals that are explicitly represented as Linear Temporal Logic formula.
- Investigate how to schedule maintenance goals that are only valid for a period of time.
- Investigate how to schedule multiple maintenance goals with different priorities or urgency.

References

[1] Brian Logan, John Thangarajah, and Neil Yorke-Smith. Progressing intention progression: A call for a goal-plan tree contest. 16th Conference on Autonomous Agents and MultiAgent Systems, pages 768–772. IFAAMAS, 2017.

[2] Y. Yao and B. Logan . Action-Level Intention Selection for BDI Agents. AAMAS-16, 1227-1236, 2016.