CSE 574: Class Project

Opportunistic Re-planning Agent

Anupam Khulbe

1. Introduction

This report describes an attempt at creating an opportunistic re-planning agent. The primary objective of the re-planning agent was to accommodate new goals coming in at execution time, while still being biased towards the plan at hand.

Preliminary setup of the project:

1. The agent uses a conservative approach in the sense that it only tries to fit a new goal in its existing set of schemes without giving up on it's current goals. So the agent can be looked at as being *opportunistic* in the sense that it sees opportunities where it can piggyback new goals on the existing plan.
2. The domain chosen was Logistics, consisting of an agent that can move around a set of locations, picking and dropping packages. The cost/utility model chosen was PSP.

2. Approach

A straight-forward and blatant approach could have been to invoke the planner to re-plan from scratch. Under such a scheme, whenever a new goal comes in, the planner would be invoked with the *initial* state being the current state of world, *goals* being the set of yet unachieved goals and the newly assigned goal.

There are a certain drawbacks of such a scheme:

1. Planning for the full set of goal would be costlier in terms of time and computing resources. This is a real issue when the actions have durations and goals come with deadlines.
2. It doesn't allow the agent to take advantage of the existing plan at hand.
3. Also, even if concurrent planning and execution is allowed, an agent which starts re-planning from scratch from the current state has to halt execution until it has the plan to follow.

It would be preferable to have use the planner only for making localized changes to the existing plan while trying to accommodate new goal(s) into it. The approach taken was to divide the plan into “phases”, where a phase is nothing but a section of the plan marked with some events (explained ahead). The idea is to accommodate a goal into that section of the plan from where the goal can be achieve with minimum digression from the current plan and then re-planing the digression and reentering the plan, while trying to keep the rest of the plan as intact as possible.

2.1 Taking advantage of the existing plan

It was observed that in order to do opportunistic re-planning for a new goal, it was very much desirable to break the goal achievement into appropriate phases, which could then be accommodated into the existing plan at appropriate points by re-planning around those points.

Advantage of breaking the goal achievement into phases:

1. Better biasing with the plan
   Each phase can be completed wherever it is nearest to the existing plan trajectory.

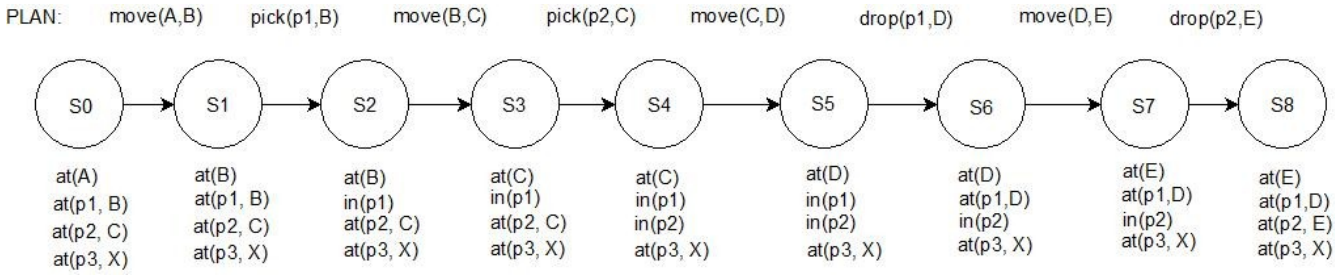
2. Easier to figure out if the new goal is worth chasing
   Re-planning for a new goal can be abandoned as soon as the plan cost for achieving any of the phases exceeds the goal utility.

3. Potentially better for the temporal case
   Phases can be reasoned about from temporal aspects individually. In particular, if parallel planning and execution is not allowed, each phase can be planned whenever there are slack time in the plan (with respect to current goal deadlines)

Motivating example: Logistics domain

Consider the agent working on goals of delivering two packages, p_1 and p_2.
Initial state: {at(A), at(p1, B)}
Goal: { g_1 = at(p_1, D), g_2 = at(p_2, E) }

Current plan and the state trajectory. Predicates (connected ?x ?y – location) not shown for succinctness.



Suppose that when the agent is in state S_1, it is assigned a new goal to deliver package p_3 at D.
g_3 = at(p_3, D)

As the agent would be passing through the location D as part of its existing plan, it would be a favorable point to achieve at(p_3, D). But the achievement of this goal also depends on whether the agent has the package p_3, or can it pick it up (at a reasonably low cost, in PSP scenario).

A goal of delivering a package in the logistics domain can be broken down into 2 phases:
(i) picking up the package, and
(ii) dropping the package at the destination

2.2 Domain dimensionality analysis

One way of breaking goals into phases is to have HTN schema. But this requires domain engineering, and requires that the type of goals be known upfront (so that reduction schema can be written for them). What is rather needed was a domain-independent way of breaking the goal into appropriate phases which can then be inserted at into the plan.

Dimensionality views the *state* as a point in a n-dimensional space. The domain dimensions are defined by a set of orthogonal predicates (and not necessarily all the predicates). Actions can be seen as changing the agent's location (current state) along some (*ideally,* only one) dimensions, while keeping it intact in other dimensions.

Given a domain description, performing a dimensionality analysis to choose the right set of predicates making up the dimensions would provide the information needed for breaking goals into phases. Appropriate number of segments for a goal seems to be related to the dimensionality of the plan (and not the domain) for achieving it. Given a plan for achieving a goal, number of changes in the dimensions, along which the agent moves, gives the number of segments needed.

Illustration

Consider a travel domain with 2 predicates: at(?x - location) and connected(?x, ?y - location). Only action available is move(?x, ?y), which takes the agent from ?x to ?y, given at(?x) and connected(?x, ?y). Clearly, the action move(?x, ?y) transitions the agent only along the at(?x) dimension, without any changes to connectivity (defined by predicate connected) dimension. So although the domain is having two dimensions, any travel plan consisting of a sequence of move actions will transition the agent along only one dimension. A new goal in this domain will have to be achieved in a single phase.

On the other hand, lets consider the chosen logistics domain. The logistics domain can be visualized as consisting of 2 dimensions:

i. Agent's location – Action move changes the agent's location, without affecting the predicates describing the state of the package.
ii. State of the package – Actions pickup/drop change the package's location without affecting the predicates describing the agent's location.

A goal of delivering a package to a destination would consist of 2 phases:

i. First phase for picking up the package
ii. Second phase for dropping the package at the desired location

The point to consider here is that these two segments can be accommodated into the rest of the plan separately. So, the package can picked up when the agent is nearest to the pickup location and dropped when the agent is nearest to the desired location. Breaking the goal lets the agent take better advantage of the existing plan. The rest of the plan between the two phases moves independently, either because it is in the orthogonal dimension (only consisting of move actions) or any pickup/drop actions involved are not invoked on the package specified in the new goal.

2.3 Procedure for breaking the goal into phases

Breaking a goal into multiple phases in a domain-independent way would be highly desirable. But, for the purpose of this project, the approach used for doing in the chosen Logistics domain is as follows:

Procedure: *return_phase_actions*
Input: goal g (*of the form (at ?package ?location)*)
Output: **Ground** action tuple <phase1_action, phase2_action>

i. Create a plan for achieving the new goal g. (Alternately, a relaxed plan could be used)

ii. Let $\mathbf{a_k}$ be the last action in the plan, which would be drop(?package, ?location). It moves the agent along package dimension. The positive effect of this **ground** action would be the goal of the second phase, i.e. the end goal g itself

iii. Starting from the beginning of the plan, select the first action $\mathbf{a_i}$ which moves along the same dimension, which would be the ground action of type pick(?package, ?location).
NOTE that the ?package in this ground action has to be the same as the one in the action $\mathbf{a_k}$.
The positive effect of this **ground** action would be goal of the first phase i.e. picking up the package.

iv. Return <$\mathbf{a_i}$, $\mathbf{a_k}$>

3. Marking the plan trajectory with events (goal epochs)

Now that we have a way of breaking a new goal into phases, we also need a way to demarcate the existing plan with information about the achievement phases of current goals. This is the a way of keeping track of important events (**goal epochs**) along the plan trajectory, around which the agent need to re-plan. *This is the agent's way of remembering when are important things happening in the plan trajectory.*

Here is a procedure based on the technique (for Logistics domain) described earlier which marks a trajectory T with the 2 phases for a goal g. The trajectory T contains the state and action information for a plan P starting from a given initial state.

Procedure: *mark_goal_epochs*
Inputs: Plan Trajectory T, Goal g
Output: Trajectory T marked with epochs for goal g

1. phase2_state = achievement_point(T, g)
   // i.e. state in the plan trajectory from where g holds true
   // achievement_point(T, g) returns the state in T from where onwards goal g holds true.

2. phase2_action = precedingAction(phase2_state, T)
   // i.e. ground action in plan which results in phase2_state

3. dimension = actionDimension(phase2_action)
   // dimension along which this action moves. In logistics case, this would be either the
   // agent's location or the package dimension

4. phase1_action = actionInDimension(dimension, T, package(g))
   // the first action in T which also transitions in the dimension dimension
   // package(g) signifies the ?package involved in phase2_action of this goal.
   // phase1_action must also act on the same package (as explained in section 2.3)

5. phase1_state = resultingState(phase1_action, T)
   // the state resulting after the above ground action phase1_action in T

6. Mark phase1_state and phase2_state as the epochs for goal g
7. return T

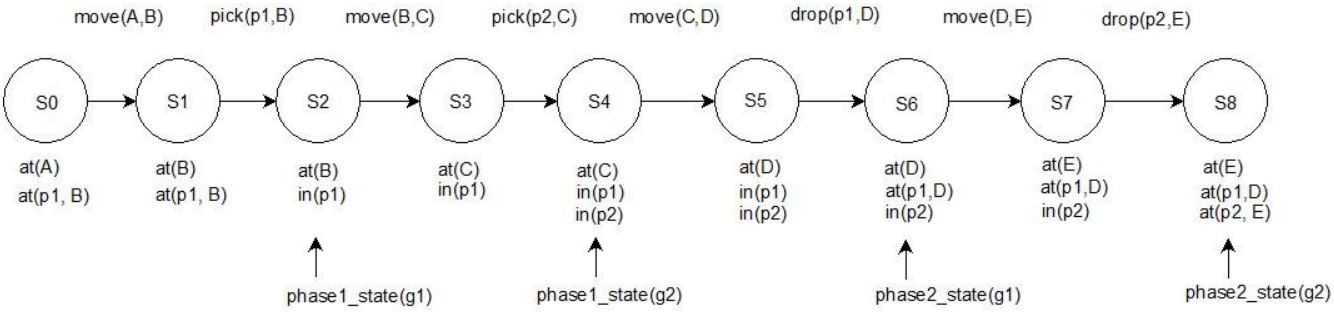
Marking the plan trajectory with all goal epochs
Given our way of identifying the goal epochs, we apply this technique to mark these events (epochs) for each goal on the original plan trajectory.

Procedure: *mark_all_goal_epochs*
Inputs: Plan trajectory T, Goals G
Output: Trajectory T marked with goal epochs

1. for each goal g belonging to G
   mark_goal_epochs(T, g)
2. return T

Illustration: *mark_all_goal_epochs*

Initial state: {at(A), at(p1, B)}
Goal: { g1 = at(p1, D), g2 = at(p2, E) }



4. Putting it all together

Given the method for breaking goals in phases and marking the plan with epochs, the re-planning approach can be explained as below:

Procedure: *Main*
Input: initial state I, Goals G

1. Invoke the planner to get the plan P for achieving the goal set G
2. Create plan trajectory T, given P and the starting state I
3. T = mark_all_goal_epochs(T, G)
4. Execution and re-planning phase
    1. If a new goal g' comes
        T = replan_for_new_goal(T, g')
        Go to step 4.
    2. Else
        execute next action in the current plan
        Go to step 4.

Procedure for re-planning for a new goal

Procedure: *replan_for_new_goal*
Input: Trajectory T, goal g
Output: Trajectory T'

1. T' = T

2. Break the goal into two phase actions
   <phase1_action, phase2_action> = *return_phase_actions*(g)

3. Plan $phase_1$
   a) deviate_state = state along T' from where $Effect^+$(phase1_action) is nearest (based on cost heuristic)
   b) reentry_state = first state after deviate_state along T' having a goal epoch (i.e. next important event)
   c) Setup new planning problem instance as:
      initial_state = deviate_state
      goals (hard) = reentry_state $\cup$ $Effect^+$(phase1_action) \ $Effect^-$(phase1_action)
   d) P_1 = plan(initial_state, goals)
   e) costIncrement(P_1) = cost(P_1) – originalCost(deviate_state, reentry_state, T)
      if costIncrement(P_1) >= utility(g) return T //i.e. goal is not worth chasing
   f) T' = Replace plan segment between deviate_state and reentry_state with P_1
   g) Propagate effects of P_1 on the trailing part of T'.
      Return T if propagation invalidates old plan.
      // This step updates the following states with the effects of new plan segment P_1.
      // E.g. Plan segment for digressing to pick package p3 and reentering back must add in(p_3)
      // and delete at(p_3, <pickup-location>) to all subsequent states. Less the effects propagated,
      // the better it is for making local changes in the plan.

4. Plan $phase_2$
   a) deviate_state = state along T' from where $Effect^+$(phase2_action) is nearest (based on cost heuristic)
   b) reentry_state = first state after deviate_state along T' having some goal epoch (i.e. next important event)
   c) Setup new planning problem instance as:
      initial_state = deviate_state
      goals (hard) = reentry_state $\cup$ $Effect^+$(phase2_action) \ $Effect^-$(phase2_action)
   d) P_2 = plan(initial_state, goals)
   e) costIncrement(P_2) = cost(P_2) – originalCost(deviate_state, reentry_state, T)
      if costIncrement(P_1) + costIncrement(P_2) >= utility(g) return T
   f) T' = Replace plan segment between deviate_state & reentry_state with P2
   g) Propagate effects on P2 on the trailing part of T'
      Return T if propagation invalidates old plan.

5. Return T'

note: originalCost(deviate_state, reentry_state, T) returns action costs for the plan segment between states deviate_state and reentry_state in the old plan trajectory T.

5. Outcome / conclusion

An implementation of this approach could not be completed in due time. For this reason, there is no empirical data to judge the performance/behavior of this re-planning approach.

Reasons for project not reaching its intended completion:

1. I took too much time to select and start working on a specific project topic.
2. For implementation, I chose to use an existing planner (Sapa PS). Although the objective was to build the agent on top of planner with minimal changes to the planner itself, it took substantial amount of time understanding and working with a new piece of code.

Nonetheless, as far as the idea itself it concerned, it can be refined and extended in multiple ways. I feel that it would be interesting to explore this idea further, especially on the lines of domain dimensionality analysis and finding ways to break the goal into right phases in a domain-independent way.

There are several interesting questions this approach opens. To name a few:

1. How to find the right set of predicates defining the domain dimensionality?
2. Instead of post-processing the plan to mark epochs, how beneficial/difficult would it be to have this information generated during the planning phase itself?
3. How to extend this for handling durative actions and goals with deadlines?