Model-based Programming for Cooperative Vehicles: Generative Activity Planner

(16.412J Cognitive Robotics)

Seung H. Chung

(joint project with Robert Effinger, Thomas Léauté, and Steven Lovell)

1 Introduction

A typical embedded program interacts with a plant through the sensor observations and commands as illustrated in Figure 1(a). A programmer for such embedded program must predetermine all possible observations and map them to the appropriate commands. This mapping between observations and commands, however, may be complex and not at all intuitive. Furthermore, as the system becomes more capable and more complex, this mapping will surely become more arduous.

A *model-based embedded program* [13] eliminates this difficulty through the use of *model-based executive*. Unlike the conventional embedded program aforementioned, a model-based embedded program interacts directly with the plant state as illustrated in Figure 1(b). Thus, a programmer can design an embedded program intuitively in terms of the desired evolution of plant state rather than sequence of commands. Since the plant state can be inferred directly, the desired evolution of the plant state can also be conditioned on the plant state rather than on sensor observations.

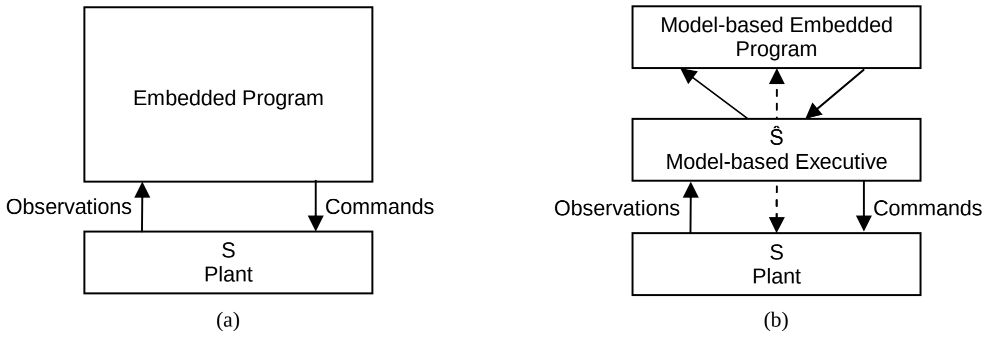

Figure 1. Model of interaction with the physical plant for traditional embedded languages (a) and model-based programming (b).

A model-based executive enables direct inference and direct control of the plant state. Williams et al. introduced a model-based executive called *Titan* with such capability [13]. Titan, however, is most well suited for a plant of hardware components. In this project we extend the model-based executive architecture to a plant involving a set of cooperative vehicles. In a sense, the vehicles are the components of a plant and the objective is to command the set of vehicles in a cooperative manner to achieve some mission objective.

1.1 Motivating Scenario

Consider a set of firefighting unmanned aerial vehicles (UAVs). A “seeker” UAV is equipped with an onboard surveillance camera with which the degree of fire containment can be analyzed. A large “water” UAV is equipped to pickup water from lakes and drop them at the desired location. The UAVs have finite range and must be refueled as necessary.

In a mountainous region, fire starts in two distant locations. The mission objective is to put out the fire autonomously using the UAVs. While the water UAV’s responsibility is to drop water over the fire, the seeker UAV’s responsibility is to take images of the fire before and after water is dropped, so that the progress of the mission can be analyzed. Figure 2 depicts the layout of the region. Initially, both UAVs are at the base. Within the region, there are two fuel stations and two lakes. Two regions are marked as no fly zones due to the high mountain peaks. The UAVs must fly around them. When the UAVs have completed their tasks, they all must return to the base.

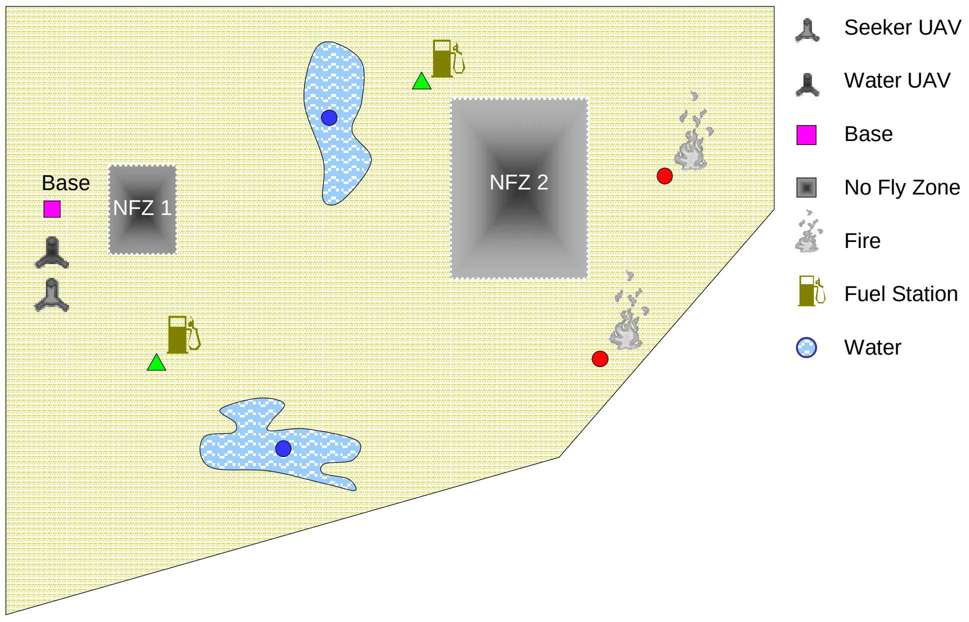

Figure 2. Map for the Firefighting UAV scenario.

This Firefighting UAV scenario will be used though out the remainder of this paper as an aid.

1.2 Model-based Executive for Cooperative Vehicles

The difference between the model-based executive for cooperative vehicles and Titan are the engines under the hood as illustrated in Figure 3. The proposed model-based executive for cooperative vehicles uses a Kirk planner [13] as the control sequencer, a roadmap path planner for estimation and prediction, and a combination of a generative activity planner and a kinodynamic path planner as the controller.

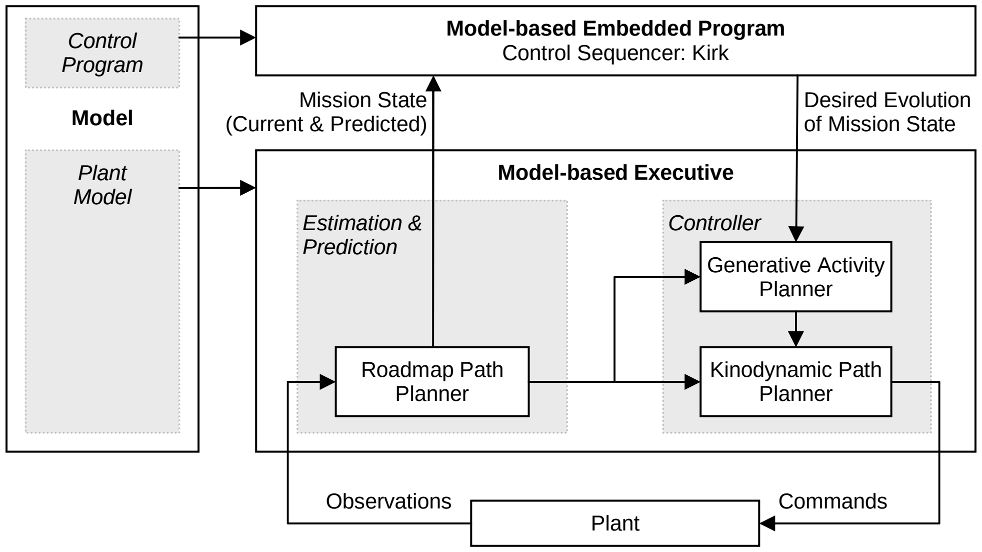

Figure 3. Architecture of a model-based executive for cooperative vehicles.

Control Sequencer

Kirk, as the control sequencer of the model-based executive, generates the mission plan in terms of the desired evolution of mission state. Kirk must choose the appropriate tactics and strategies given options and contingencies to cooperatively achieve the mission objective.

For example, in the Firefighting UAV scenario, the seeker UAV could be sent out first, and once its task is complete, the water UAV could then be sent out to drop water on fire. Finally the seeker UAV can go back to take images of the result. Another strategy may be to send the seeker UAV and water UAV simultaneously. In this case, however, the seeker UAV must be sure to take images of the fire before the water UAV drops water on them. If properly executed, this strategy should accomplish the mission within a shorter period of time. If truly urgent, one may even consider not sending out the seeker UAV, i.e. tradeoff time with uncertainty of the mission progress.

The possible tactics and strategies are the models for Kirk, and its task is to generate the mission plan with an appropriate strategy that can accomplish the mission objective. One of the important contributions of Kirk is that it quickly generates a plan with flexible time bounds. The flexible time bound makes the plan robust to execution time uncertainty. For the technical details of Kirk, refer to [13,3]. As a part of this project, [3] extends the original Kirk to use dynamic backtracking, enabling fast, anytime mission plan generation in best-first order.

Generative Activity Planner

The generative activity planner takes the mission plan from Kirk as an input to generate an actionable activity plan. While the mission plan describes the desired evolution of the mission state, an activity plan describes the sequence of actions, which when executed achieves the desired evolution of mission state, i.e. the mission plan. Furthermore, as the state of the mission and/or mission plan change, the generative activity planner replans as necessary.

For example, the mission plan may require water UAV to drop water on a fire. Depending on the current state, the UAV may have to first fly over to the lake, pick up water, fly over to the fire, then drop the water.

The generative activity planner determines the sequence of actions necessary to achieve the desired mission state while concurrently achieving other subsequent evolution of the mission states.

Kinodynamic Path Planner

The kindoynamic path planner takes the activity plan from the generative activity planner, and for each motion activity, it generates a trajectory through which the desired destination can be reached while assuring that its motion is bound by the kinodynamics of the vehicle.

In general, kindoynamic path planning is a time consuming process, thus the trajectory is computed for some finite horizon. This approach is referred to as receding horizon kinodynamic path planning. In addition to the quick computation capability of receding horizon kinodynamic path planning, it also opens up to the ability to adapt to uncertainty through continuously replanning. Bellingham et al. took this approach to path planning for aerial vehicles using mixed integer linear programming [1]. As a part of this project, Léauté builds up on this approach so that a trajectory can be designed for sequence of motions with flexible time bounds [9].

Roadmap Path Planner

The roadmap path planner estimates the distance between two locations. It provides the distance estimates to the control sequencer, the activity planner, and the kinodynamic path planner.

For this project, D*lite was chosen as the roadmap path planner for its quick replanning capability [8]. Its computed distance is used to compute the lower time bound estimate on motion activities for the control sequencer and the generative activity planner. In the generative activity planner, D*lite's estimate of distance between locations is used to also compute the fuel consumption. Finally, D*lite's use to generate a cost-to-go map, which is used as a heuristic in kinodynamic path planning. Lovell discusses the details on the implementation and the integration into the model-based executive [10].

1.3 Objective

While the control sequencer, kinodynamic path planner, and the roadmap path planner are discussed in [3,9,10], this paper focuses on the generative activity planner. Recall that the generative activity planner generates an actionable activity plan with flexible time bounds given the mission plan from the control sequencer (see Figure 4). For this project, a planner called Linear Planning Graph (LPG) [5] is adapted to serve this purpose.

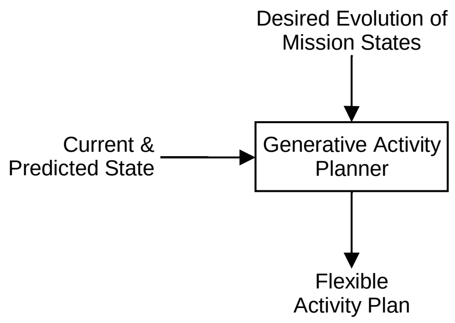
Desired Evolution of Mission States
Current & Predicted State
Generative Activity Planner
Flexible Activity Plan

Figure 4. Generative Activity Planner architecture.

In the third IPC, LPG was among the fastest general purpose planners in the time and metric resource domains. LPG's fast planning capability mainly arises from the combination of three techniques. One, it uses a modified plan-graph, temporal action graph, that can represent time within the plan-graph like structure. LPG seems to benefit from the compact encoding of the plan space, similar to the way many planners benefited from the compact plan-graph structure and it's use of mutex relations. Two, it uses relaxed plan-graph as a heuristic that guides the search. Finally, it uses a randomized local search, similar to WalkSAT. As randomized local search has shown to be very effective in solving SAT problems, randomized local search seems to be very effective in planning as well

The contribution of this paper is the use of an existing temporal generative planner, LPG, to refine a complex sequence of goals, i.e. a mission plan, into an actionable activity plan. In the following sections, we will describe Planning Domain Definition Language (PDDL) 2.1 used by LPG and the PDDL2.1 domain description of the Firefighting UAV scenario, followed by the use of LPG as a generative activity planner. The paper will conclude with the discussion on the implementations and remarks regarding to this approach.

2 Generative Activity Planner

In this section we briefly describe the PDDL2.1 in general and the PDDL2.1 domain description designed for the Firefighting UAV scenario.

2.1 PDDL2.1

Within the planning community, PDDL has become the standard language for describing planning domains and problems. While PDDL was initially inspired by STRIPS formulation of planning problems, it has been through constant revision and update for the enhancement of the expressivity of the language. The last major PDDL version was PDDL2.1 [4], which was used in the third International Planning Competition (IPC). Various features available in PDDL2.1 are distinguished by the level of expressivity, where:

Level 1: STRIPS
Level 2: adds the numeric extensions (i.e. arithmetic preconditions and assignments in effects) to Level 1.
Level 3: adds discretized durative actions (i.e. fixed time duration on actions) to Level 2.
Level 4: adds continuous durative actions (i.e. flexible time bounds and continuous effects) to Level 3.
Level 5: adds spontaneous events and physical processes to Level 5.

While PDDL2.1 Level 4 includes the expressivity of continuous duration that is essential in describing the flexible time bounds of the mission plan, since LPG is limited to a subset of PDDL2.1 Level 3, the generative activity planner is also limited to Level 3. For the details of PDDL2.1, refer to [4]. In general PDDL2.1 is very expressive. The STRIPS-like syntax, however, makes description of complex domain difficult and sometimes non-intuitive. Also, since flexible time bound cannot be expressed using PDDL2.1 Level 3, the lower time bound is used as the fixed duration of an action.

2.2 PDDL2.1 Firefighting UAV Domain in PDDL2.1

The main components of a PDDL2.1 domain description are types, predicates, functions, and action definitions. Types list the types of objects that are in the world; predicates are used to describe the state of a world, functions are used to describe numeric values of the world; and action describe how the state of the world evolves over time.

Types

There are only two types of objects in the Firefighting UAV domain: vehicles and locations.

Predicates

The following are the predicates used in the Firefighting UAV scenario:

```
(available ?v - vehicle)
(seeker-uav ?v - vehicle)
(water-uav ?v - vehicle)
(have-water ?v - vehicle)
(full-fuel ?v - vehicle)
(at ?v - vehicle ?l - location)
(have-image ?v - vehicle ?l - location)
(water ?l - location)
(fire ?l - location)
(base ?l - location)
(fuel-station ?l - location)
(dropped-water ?v - vehicle ?l - location)
(reachable ?l0 ?l1 - location)
```

Note that these describe the state of the world that can be inferred from the estimation module of the model-based executive.

Functions

The following are the functions used to represent numeric effects.

```
(fuel-level ?v - vehicle)
(fuel-capacity ?v - vehicle)
(fuel-consumption-rate ?v - vehicle)
(total-fuel-used)
(min-distance ?from ?to - location)
(speed ?v - vehicle)
```

Note that the “min-distance” function specifies the minimum travel distance between two locations. The minimum travel distance between two locations may change as new obstacles are detected. Thus the value of “min-distance” function is updated as necessary by querying the roadmap path planner.

Actions

The following are the actions created for the Firefighting UAV scenario (see Appendix A for the definitions of the actions):

```
move(?v - vehicle, ?from - location, ?to - location)
refuel(?v - vehicle, ?l - location)
take-image(?v – vehicle, ?l-location)
pickup-water(?v – vehicle, ?l-location)
drop-water(?v – vehicle, ?l-location)
```

3 LPG-based Generative Activity Planner

Recall that a generative activity planner generates an actionable activity plan with flexible time bounds given the mission plan from the control sequencer (see Figure 4). In this section the inputs, outputs, and the internals of the LPG-based generative activity planner are described.

3.1 Input: Mission Plan and Current State

Mission Plan

A mission plan generated by Kirk is in the form of a simple temporal network (STN) [2]. An example of a mission plan is illustrated in Figure 5. This mission corresponds to the case in which the seeker UAV and the water UAV both try to achieve their tasks simultaneously. The mission plan has been simplified by no requiring the seeker UAV to take the second image after water has been dropped on fire and the two UAVs to return to the base.

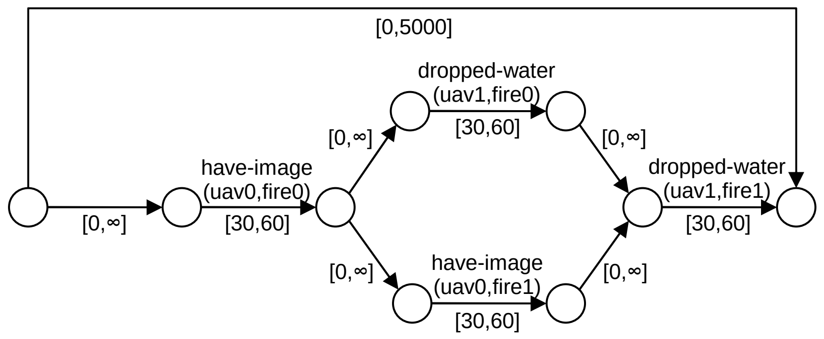

Figure 5. An example of a mission plan represented in a Simple Temporal Network (STN).

In an STN, each arc corresponds to an event and a node corresponds to either a start or an end of an event. Each arc has a label that specifies the time interval of the event using a pair of real numbers, where the first and the second specify the lower and the upper time bounds, respectively. A label that does not denote the time interval specifies a state the mission must be in during the specified time interval. A set of out arcs from a node represents a set of events that must occur in parallel, and two subsequent arcs represent sequential events.

In the mission plan illustrated in Figure 5, the seeker UAV, uav0, is first required to “have-image” of fire0 for 30 to 60 time units. Once the image has been acquired, the water UAV, uav1, can have water dropped, i.e. “dropped-water”, on the first fire, fire0, for 30 to 60 time units. In the mean time, the seeker UAV is to acquire the image of the second fire, fire1. After the water is dropped on the first fire and the image of the second fire is acquired, the water UAV can drop water on the second fire. Notice the arcs with no mission state constraint. These arcs can be interpreted as a period of time that is required to prepare for the achievement of the mission state specified by the subsequent events. Finally, the arc with the time bound of [0,5000] can be interpreted as a period over which the whole mission objective must be achieved.

Current State

The current state of the world is inferred from the state estimation module of the model-based executive. For example, the state estimation module may determine that the seeker UAV has crashed and is no longer available. The roadmap path planner may determine that the minimum travel distance from the base and to the nearest fire has increased to 2 km because of the new obstacle detected in the path.

3.2 Output: Flexible Activity Plan

Recall that the objective of a generative activity planner is to design an actionable activity plan that achieves the desired evolution of mission state specified in the mission plan. Figure 6 illustrates a partial activity plan that achieves one of the desired mission state, namely “dropped-water(uav1,fire0)”. Given that the water UAV is initially at the base with no water, the generative activity planner must figure out that the water UAV must first go to a lake, pick up water, go to the fire, and then drop the water. The act of dropping water achieves the desired “dropped-water(uav1,fire0)” state. Such activity plan must be generated for the whole mission plan.

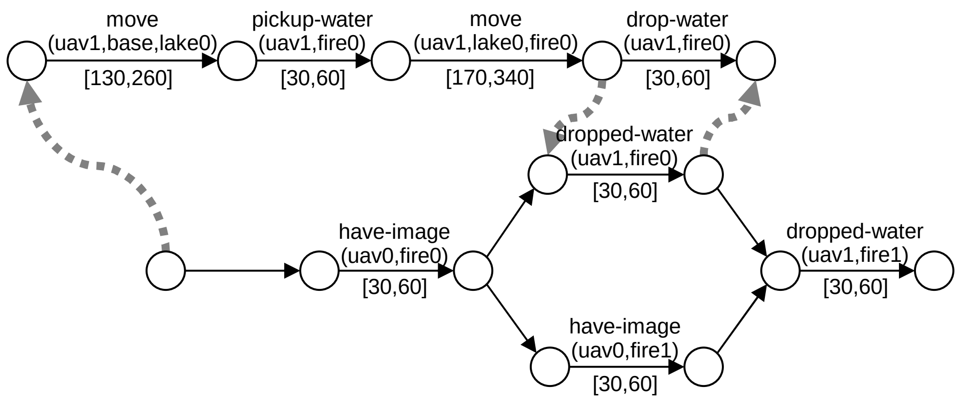

Figure 6. An example of a partial activity plan for a given mission plan.

3.3 LPG-based Generative Activity Planner Architecture

Figure 7 illustrates the architecture of the LPG-based generative activity planner and depicts the process of generating a plan. First, the current state and the mission plan must be translated into a PDDL2.1 problem that LPG can understand. More specifically, the problem description generator incorporates the current state information to generate a PDDL problem description. The domain description generator incorporates the mission plan to generate a PDDL domain description. The reason for such particular translation schema will be discussed in the subsequent sections.

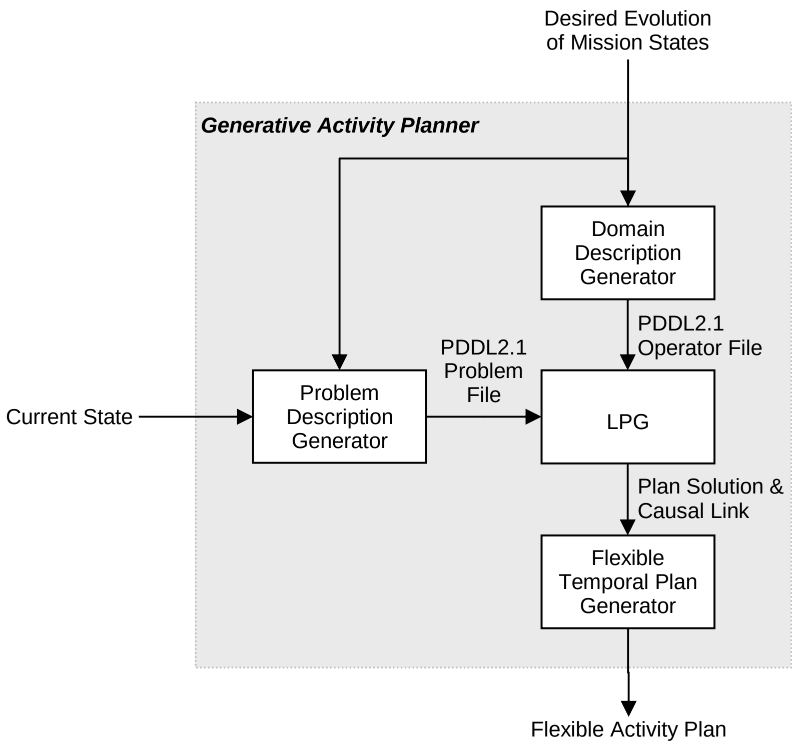

Figure 7. LPG-based Generative Activity Planner architecture.

One of the limitations of LPG is its inability to reason about actions with flexible time bounds. As a result, LPG can only generate a plan with a fixed time bound, rather than a flexible time bound. In order to maintain the execution robustness of a flexible temporal plan, we modify the plan generated by LPG into a flexible temporal plan. This is handled by the flexible temporal plan generator.

4 Domain Description Generator

The key to the domain description generator is translating the mission plan in STN into a PDDL2.1 domain description. Intuitively, the properties encoded by the STN must be translated into a PDDL2.1 domain description. That is, for each arc *A* in STN, create a PDDL action *action-A* that can only be executed for the duration specified by *A* if and only if the actions corresponding to the predecessor arcs have been executed and the state asserted by *A* is true during the execution of *action-A*.

4.1 Domain Description Generator Pseudo Code

The following is the pseudo code used to generate the PDDL2.1 domain description.

```
1. For each out arc $A$ from the root node, create a (started-<$UID(A)$>) predicate, where $UID(A)$ is the UID of the out arc $A$.
2. Traversing through the STN in the depth-first order
     a. For each out arc $A$, create a PDDL action $Act(A)$, where:
          i. The duration of the operator is the lower time bound of the arc.
          ii. For each predecessor arc $Pre(A)$, add a predicate that corresponds to the execution completion of the action of the predecessor arc as the start condition, i.e. (at start (<$UID(Pre(A))$>-<$UID(A_i)$>)). If no predecessor exists, add the start predicate (at start (started-<$UID(A)$>)).
          iii. As the condition during the time of execution, add the predicate that corresponds to the state asserted by $A$, e.g. (over all (take-image(uav0,fire0))).
          iv. For each predecessor arc $Pre(A)$, add a predicate that corresponds to the execution completion of the action of the predecessor arc as the delete effect at the start of the execution, i.e.
              (at start (not (<$UID(Pre(A))$>-<$UID(A)$>))).
          v. For each successor arc $Suc(A)$, add a predicate that corresponds to the execution completion of $Act(A)$ as the add effect at the end of the execution, i.e. (at end (<$UID(A)$>-ended-<$UID(Suc(A))$>))). If no successor exists, add
              (at end (<$UID(A)$>-ended-goal))) to the effect.
3. When the terminal node is reached, create a goal action $Act_{goal}$ using the same rule as before, but with the addition of an add effect that signifies the achievement of the mission plan, i.e.
   (at end (goal-achieved)).
4. Define all used predicates in the domain.
```

In the pseudo code line 2.a.i, only the lower time bound is used since PDDL2.1 Level 3 recognized by LPG cannot handle time interval as the duration, i.e. inequality on the time bound. This will cause the planner to always generate the activity plan with the shortest duration. At the same time, one may expect the planner may fail to generate an activity plan due to the over restriction on the duration length. Though this is true by the PDDL encoding, due to the use of no-op within LPG, an arbitrarily long time can be inserted in between each subsequent event. As such, the restriction of the duration to the lower time bound does not prevent LPG from finding a solution. On the other hand, this mapping to PDDL2.1 cannot guarantee that each subsequent event will start right after anther, for the same reason.

The pseudo code Line 2.a.ii guarantees that the action is not executed unless the predecessor actions have been executed. Line 2.a.iii guarantees that the desired mission state is achieved. Line 2.a.iv guarantees that no action is executed more than once, i.e. modification is made to the mission plan. Line 2.a.v enables the subsequent action to be executed. Furthermore, by adding one add effect for the precondition of each successor, it guarantees that the successor actions are not mutexed. This is necessary when used with a planner such as LPG that uses plan-graph-like algorithm. The action generated by Line 3 is used to determine when the mission plan has been complete.

4.2 Firefighting UAV Domain Example

Figure 8 illustrates the depth-first labeling of the arcs of the STN from the Firefighting UAV scenario. In the following examples, we will refer to each by the label specified in Figure 8.

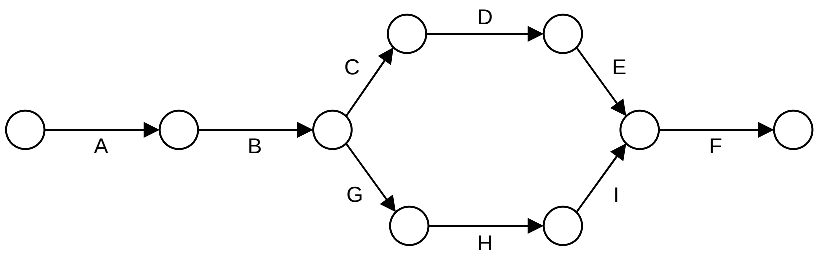

Figure 8. Depth-first STN arc labeling.

The following is a PDDL2.1 action generated for arc A:

```
(:durative-action action-A
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (started-A))))
                :effect (and (at start (not (started-A)))
                            (at end (A-ended-B))))
```

Since arc A’s lower time bound is zero, the duration of the `action-A` is also zero. The condition for `action-A` is `(at start (started-A))` since no in arcs exist for the start node of A. The effect `(at start (not (started-A))` of `action-A` guarantees that `action-A` never starts again. The effect `(at end (A-ended-B))` of enables `action-B` to start when `action-A` ends.

The following is a PDDL2.1 action generated for arc B:

```
(:durative-action action-B
                :parameters ()
                :duration (= ?duration 30)
                :condition (and (at start (A-ended-B))
                              (over all (have-image uav0 fire0)))
                :effect (and (at start (not (A-ended-B)))
                            (at end (B-ended-C))
                            (at end (B-ended-G))))
```

Since arc B requires an image of fire0, `(over all (have-image uav0 fire0))` is added to the condition. Also, since there are two out arcs from the end node of B, the effect has two add effects that signify the completion of `action-B`, i.e. `(at end (B-ended-C))` and `(at end (B-ended-G))` so that each of actions corresponding to arcs C and G can start without being mutually exclusive.

The following is a PDDL2.1 action generated for arc F:

```
 (:durative-action action-F
                :parameters ()
                :duration (= ?duration 30)
                :condition (and (at start (E-ended-F))
                             (at start (I-ended-F))
                             (over all (water-dropped uav1 fire1)))
                :effect (and (at start (not (E-ended-F)))
                           (at start (not (I-ended-F)))
                           (at end (F-ended))))
```

Since the start node for arc F has two in arc, `action-F` has two conditions that require the predecessor actions of E and I to have completed.

The following is a PDDL2.1 action generated to signify the achievement of the goal:

```
(:durative-action action-GOAL
              :parameters ()
              :duration (= ?duration 0)
              :condition (at start (F-ended-1))
              :effect (and (at start (not (F-ended-1)))
                      (at end (goal-achieved))))
```

See Appendix A for the complete PDDL2.1 domain file for the Firefighting UAV scenario.

5 Problem Description Generator

Problem description generator is quite straightforward. Given the current state, it translates the information into PDDL2.1 using the appropriate predicate or function. For example, if the vehicle uav0 is available for the mission the fact `(available uav0)` would be added to the `:init` section, i.e. initial fact, section of the PDDL2.1 problem file. Similarly, if the roadmap path planner determines that the minimum travel distance from the base to closest fire, fire0, is 2 km, the fact `(= (min-distance base fire0) 2000)` would be added to the `:init` section of the problem file. In such a way, the current state of the mission is mapped to the initial facts.

In addition, for each out arc *A* of the root node, add the fact `(start-<`UID(*A*)`>)` to the `:init` section, where UID(*A*) is the UID of the arc *A*. This will enable the execution of the action corresponding to *A*. As the goal, `(goal-achieved)` is added to `:goal` section of the problem file. Finally, the desired cost function that defines the quality of the plan is added to the `:metric` section of the problem file. For the Firefighting UAV scenario, `minimize (+ (* 100 total-time) (* 1 total-fuel-used))` was used as the cost function. See Appendix B for an automatically generated full problem file.

6 Flexible Temporal Plan Generator

When LPG generates a plan, the time bounds on the actionable activities are fixed to the lower time bound as illustrated in Figure 9. To transform the plan into a flexible temporal plan, we simply add the upper time bound of the activities (note that the upper time bound must be predefined for each activity), thus transforming it into a plan illustrated in Figure 6. Since the temporal bounds on the activities are flexible, i.e. represented by an interval, the temporal consistency of the resulting flexible temporal plan must be checked.

To check the temporal consistency, we must know the constraints between the activities and a temporal consistency checker. Theoretically, the constraints between the activities can be extracted from the temporal action graph of LPG. Then, we can use the incremental temporal reasoning algorithm [11] in Kirk to check the consistency of the flexible activity plan. If the flexible activity plan is temporally inconsistent, a new plan must be generated.

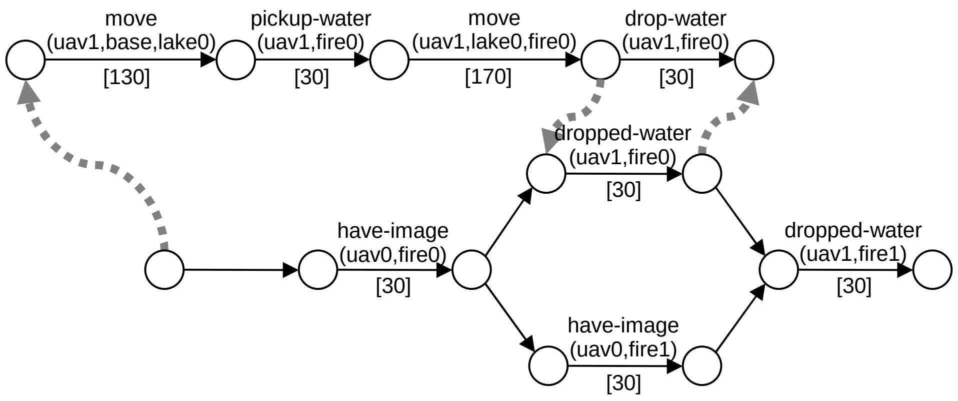

Figure 9. A partial activity plan generated by LPG for the Firefighting UAV scenario.

7 Conclusions

The Domain Description Generator and the Problem Description Generator was in C++. While the original proposal included the implementation of the Flexible Temporal Plan Generator, due to the time constraint, it was de-scoped. In order to implement the Flexible Temporal Plan Generator, the LPG source code has to be modified. Unfortunately, fixing the provided LPG source code to even compile was in itself a time consuming process and a such no time was available to thoroughly understand the LPG code and modify it.

The performance of LPG within the context of the generative activity planner was disappointing. For what seems to be relatively a simple problem (see Appendix A and B), LPG was slow. That is, LPG took anywhere from 0.06-600 seconds when a solution was found, but sometime, LPG couldn't solve the problem at all. Over all, however, the problem is solved within tens of seconds. I have also noticed that the performance of LPG is reduced considerably if the vehicle is required to make multiple trips to refuel. While other mission plan to PDDL encoding options have been considered, the poor performance seems to result form the inherent difficulty of the problem. In general, LPG seems to perform worse as the problem becomes more constrained.

Also, as mentioned before, the use of no-op within LPG prevents us from guaranteeing that the resulting activity plan adheres to the specified mission plan. This is one of the leading reasons for the development of a generative planner called Spock [6].

While this project has introduced a method for mapping a complex mission plan into a PDDL2.1 problem, the result seems to suggest that the use of LPG within this framework is not desirable. A new approach that combines Spock generative planner the LPG-like temporal planning may be feasible, but further analysis is necessary. It would also be very beneficial to fully analyze the reason for LPG's poor performance.

8 References

1. J. Bellingham, A. Richards, and J. How. "Receding Horizon Control of Autonomous Aerial Vehicles." In Proceedings of *the IEEE American Control Conference*, May 2002.

2. R. Dechter, I. Meiri, and J. Pearl. "Temporal constraint networks." *Artificial Intelligence*, 49:61-95, May 1991.

3. R. Effinger. "MIT 16.412J Course Project." May 2004.

4. M. Fox and D. Long. “PDDL2.1: An extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research: Special Issue on the 3rd International Planning Competition*, 2003.

5. A. Gerevini and I. Serina. “LPG: A planner based on local search for planning graphs.” In Proceedings of *the Sixth International Conference on AI Planning and Scheduling (AIPS’02)*, 2002.

6. J. Kennell. "Generative Temporal Planning with Complex Processes." *M. Eng. Thesis*, Massachusetts Institute of Technology, October, 2003.

7. P. Kim, B. C. Williams, and M. Abramson. “Executing reactive, model-based programs through graph-based temporal planning.” In Proceedings of *the IJCAI-2001*, 2001.

8. S. Koenig and M. Likhachev. “Improved Fast Replanning for Robot Navigation in Unknown Terrain.” In Proceedings of *the International Conference on Robotics and Automation*, 2002.

9. T. Léauté. “MIT 16.412J Course Project.” May 2004.

10. S. Lovell. “MIT 16.412J Course Project.” May 2004.

11. I. Shu. “Enabling Fast Flexible Planning through Incremental Temporal Reasoning.” *M. Eng. Thesis*, Massachusetts Institute of Technology, September, 2003.

12. B. C. Williams, M. Ingham, S. H. Chung, and P. H. Elliott. “Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers," invited paper in Proceedings of *the IEEE: Special Issue on Modeling and Design of Embedded Software*, vol. 9, no. 1, pp. 212-237, January 2003.

Appendix A. Firefighting UAV Domain File

The following is a PDDL2.1 domain automatically generated for the Firefighting UAV scenario:

```
(define (domain fireFightingUAV)
  (:requirements :strips
                 :typing
                 :negative-preconditions
                 :disjunctive-preconditions
                 :equality
                 :fluents
                 :durative-actions)
  (:types vehicle location)

  (:functions (fuel-level ?v - vehicle)
              (fuel-capacity ?v - vehicle)
              (fuel-consumption-rate ?v - vehicle)
              (total-fuel-used)
              (min-distance ?from ?to - location)
              (speed ?v - vehicle))

  (:predicates (available ?v - vehicle)
               (seeker-uav ?v - vehicle)
               (water-uav ?v - vehicle)
               (have-water ?v - vehicle)
               (full-fuel ?v - vehicle)
               (at ?v - vehicle ?l - location)
               (have-image ?v - vehicle ?l - location)
               (water-between ?l0 ?l1 - location)
               (water ?l - location)
               (fire ?l - location)
               (base ?l - location)
               (fuel-station ?l - location)
               (dropped-water ?v - vehicle ?l - location)
               (reachable ?l0 ?l1 - location))

  (:durative-action move
               :parameters (?v - vehicle ?from - location ?to - location)
               :duration (= ?duration (/ (min-distance ?from ?to) (speed ?v)))
               :condition (and (at start (available ?v))
                               (at start (at ?v ?from))
                               (at start (>= (fuel-level ?v)
                                            (* (/ (min-distance ?from ?to)
                                                  (speed ?v))
                                               (fuel-consumption-rate ?v))))
                               (over all (available ?v))
                               (over all (>= (fuel-level ?v) 0)))
               :effect (and (at start (not (at ?v ?from)))
                            (at start (not (full-fuel ?v)))
                            (at start (not (have-image ?v ?from)))
                            (at end (increase (total-fuel-used)
                                              (* ?duration
                                                 (fuel-consumption-rate ?v))))
                            (at end (decrease (fuel-level ?v)
                                              (* ?duration
                                                 (fuel-consumption-rate ?v))))
                            (at end (at ?v ?to))))

  (:durative-action refuel
               :parameters (?v – vehicle ?l - location)
               :duration (= ?duration 30)
               :condition (and (at start (available ?v))
                               (at start (not (full-fuel ?v)))
                               (at start (fuel-station ?l))
                               (at start (at ?v ?l))
                               (over all (at ?v ?l))
```

```
                         (over all (available ?v))
                         (at end (at ?v ?l)))
             :effect (and (at end (full-fuel ?v))
                       (at end (assign (fuel-level ?v) (fuel-capacity ?v)))))

(:durative-action take-image
             :parameters (?v – vehicle ?l - location)
             :duration (= ?duration 30)
             :condition (and (at start (available ?v))
                         (at start (seeker-uav ?v))
                         (at start (at ?v ?l))
                         (over all (at ?v ?l))
                         (over all (available ?v))
                         (at end (at ?v ?l)))
             :effect (and (at end (have-image ?v ?l))))

(:durative-action pickup-water
             :parameters (?v – vehicle ?l - location)
             :duration (= ?duration 30)
             :condition (and (at start (available ?v))
                         (at start (water ?l))
                         (at start (at ?v ?l))
                         (at start (water-uav ?v))
                         (at start (not (have-water ?v)))
                         (over all (at ?v ?l))
                         (over all (available ?v))
                         (at end (at ?v ?l)))
             :effect (and (at end (have-water ?v))))

(:durative-action drop-water
             :parameters (?v – vehicle ?l - location)
             :duration (= ?duration 30)
             :condition (and (at start (available ?v))
                         (at start (water-uav ?v))
                         (at start (have-water ?v))
                         (at start (at ?v ?l))
                         (over all (available ?v)))
             :effect (and (at end (not (have-water ?v)))
                       (at end (dropped-water ?v ?l))))

(:predicates (start-finished)
           (goal-0032D108-0-finished)
           (goal-0032D288-0-finished)
           (goal-0032D288-1-finished)
           (goal-0032E1B8-0-finished)
           (goal-0032E338-0-finished)
           (goal-0032E858-0-finished)
           (goal-0032E858-1-finished)
           (goal-0032F1D8-0-finished)
           (goal-0032F358-0-finished)
           (goal-0032F8A8-0-finished)
           (goal-0032F8A8-1-finished)
           (goal-00560298-0-finished)
           (goal-00560418-0-finished)
           (goal-00560948-0-finished)
           (final-goal-finished)
           (goal-00560AC8-0-finished)
           (goal-00560C48-0-finished)
           (goal-00561178-0-finished)
           (goal-005612F8-0-finished)
           (goal-005617F8-0-finished)
           (goal-0032FA28-0-finished)
           (goal-0032FBA8-0-finished)
           (goal-00560118-0-finished)
           (goal-00560118-1-finished)
           (goal-0032E9D8-0-finished)
           (goal-0032EB58-0-finished)
           (goal-0032F058-0-finished)
           (goal-0032F058-1-finished))
```

```
(:durative-action goal-action-0032D108
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (start-finished)))
               :effect (and (at start (not (start-finished)))
                          (at end (goal-0032D108-0-finished))))

(:durative-action goal-action-0032D288
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032D108-0-finished))
                             (at start (have-image seeker-uav0 fire0)))
               :effect (and (at start (not (goal-0032D108-0-finished)))
                          (at end (goal-0032D288-0-finished))
                          (at end (goal-0032D288-1-finished))))

(:durative-action goal-action-0032E1B8
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032D288-0-finished)))
               :effect (and (at start (not (goal-0032D288-0-finished)))
                          (at end (goal-0032E1B8-0-finished))))

(:durative-action goal-action-0032E338
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032E1B8-0-finished))
                             (at start (dropped-water water-uav0 fire0)))
               :effect (and (at start (not (goal-0032E1B8-0-finished)))
                          (at end (goal-0032E338-0-finished))))

(:durative-action goal-action-0032E858
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032E338-0-finished)))
               :effect (and (at start (not (goal-0032E338-0-finished)))
                          (at end (goal-0032E858-0-finished))
                          (at end (goal-0032E858-1-finished))))

(:durative-action goal-action-0032F1D8
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032E858-0-finished))
                             (at start (goal-0032F058-0-finished)))
               :effect (and (at start (not (goal-0032E858-0-finished)))
                          (at start (not (goal-0032F058-0-finished)))
                          (at end (goal-0032F1D8-0-finished))))

(:durative-action goal-action-0032F358
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032F1D8-0-finished))
                             (at start (dropped-water water-uav0 fire1)))
               :effect (and (at start (not (goal-0032F1D8-0-finished)))
                          (at end (goal-0032F358-0-finished))))

(:durative-action goal-action-0032F8A8
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032F358-0-finished)))
               :effect (and (at start (not (goal-0032F358-0-finished)))
                          (at end (goal-0032F8A8-0-finished))
                          (at end (goal-0032F8A8-1-finished))))

(:durative-action goal-action-00560298
               :parameters ()
               :duration (= ?duration 0)
               :condition (and (at start (goal-0032F8A8-0-finished))
                              (at start (goal-00560118-0-finished)))
                :effect (and (at start (not (goal-0032F8A8-0-finished)))
                            (at start (not (goal-00560118-0-finished)))
                            (at end (goal-00560298-0-finished))))

(:durative-action goal-action-00560418
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-00560298-0-finished))
                              (at start (at water-uav0 base0)))
                :effect (and (at start (not (goal-00560298-0-finished)))
                            (at end (goal-00560418-0-finished))))

(:durative-action goal-action-00560948
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-00560418-0-finished)))
                :effect (and (at start (not (goal-00560418-0-finished)))
                            (at end (goal-00560948-0-finished))))

(:durative-action goal-action-final-goal-operator
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-00560948-0-finished))
                              (at start (goal-005617F8-0-finished)))
                :effect (and (at start (not (goal-00560948-0-finished)))
                            (at start (not (goal-005617F8-0-finished)))
                            (at end (final-goal-finished))))

(:durative-action goal-action-00560AC8
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-0032F8A8-1-finished))
                              (at start (goal-00560118-1-finished)))
                :effect (and (at start (not (goal-0032F8A8-1-finished)))
                            (at start (not (goal-00560118-1-finished)))
                            (at end (goal-00560AC8-0-finished))))

(:durative-action goal-action-00560C48
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-00560AC8-0-finished))
                              (at start (have-image seeker-uav0 fire1)))
                :effect (and (at start (not (goal-00560AC8-0-finished)))
                            (at end (goal-00560C48-0-finished))))

(:durative-action goal-action-00561178
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-00560C48-0-finished)))
                :effect (and (at start (not (goal-00560C48-0-finished)))
                            (at end (goal-00561178-0-finished))))

(:durative-action goal-action-005612F8
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-00561178-0-finished))
                              (at start (at seeker-uav0 base0)))
                :effect (and (at start (not (goal-00561178-0-finished)))
                            (at end (goal-005612F8-0-finished))))

(:durative-action goal-action-005617F8
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-005612F8-0-finished)))
                :effect (and (at start (not (goal-005612F8-0-finished)))
                            (at end (goal-005617F8-0-finished))))

(:durative-action goal-action-0032FA28
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-0032E858-1-finished))
                               (at start (goal-0032F058-1-finished)))
                :effect (and (at start (not (goal-0032E858-1-finished)))
                            (at start (not (goal-0032F058-1-finished)))
                            (at end (goal-0032FA28-0-finished))))

(:durative-action goal-action-0032FBA8
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-0032FA28-0-finished))
                               (at start (have-image seeker-uav0 fire0)))
                :effect (and (at start (not (goal-0032FA28-0-finished)))
                            (at end (goal-0032FBA8-0-finished))))

(:durative-action goal-action-00560118
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-0032FBA8-0-finished)))
                :effect (and (at start (not (goal-0032FBA8-0-finished)))
                            (at end (goal-00560118-0-finished))
                            (at end (goal-00560118-1-finished))))

(:durative-action goal-action-0032E9D8
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-0032D288-1-finished)))
                :effect (and (at start (not (goal-0032D288-1-finished)))
                            (at end (goal-0032E9D8-0-finished))))

(:durative-action goal-action-0032EB58
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-0032E9D8-0-finished))
                               (at start (have-image seeker-uav0 fire1)))
                :effect (and (at start (not (goal-0032E9D8-0-finished)))
                            (at end (goal-0032EB58-0-finished))))

(:durative-action goal-action-0032F058
                :parameters ()
                :duration (= ?duration 0)
                :condition (and (at start (goal-0032EB58-0-finished)))
                :effect (and (at start (not (goal-0032EB58-0-finished)))
                            (at end (goal-0032F058-0-finished))
                            (at end (goal-0032F058-1-finished)))))
```

Appendix B. Firefighting UAV Problem File

The following is a PDDL2.1 problem automatically generated for the Firefighting UAV scenario:

```
(define (problem FFUAV-scenario)
  (:domain fireFightingUAV)
  (:objects base0 - location
          fire0 fire1 - location
          fuel-station0 fuel-station1 - location
          lake00 lake01 lake10 lake11  - location
          seeker-uav0 water-uav0 - vehicle)
  (:init (base base0)
         (fire fire0)
         (fire fire1)
         (fuel-station fuel-station0)
         (fuel-station fuel-station1)
         (water lake00)
         (water lake01)
         (water lake10)
         (water lake11)
        (water-between lake00 lake01)
        (water-between lake01 lake00)
        (water-between lake10 lake11)
        (water-between lake11 lake10)
        (seeker-uav seeker-uav0)
        (full-fuel seeker-uav0)
        (= (fuel-capacity seeker-uav0) 120)
        (= (fuel-consumption-rate seeker-uav0) 0.2)
        (= (speed seeker-uav0) 20)
        (water-uav water-uav0)
        (full-fuel water-uav0)
        (= (fuel-capacity water-uav0) 500)
        (= (fuel-consumption-rate water-uav0) 0.8)
        (= (speed water-uav0) 20)
        (available seeker-uav0)
        (at seeker-uav0 base0)
        (= (fuel-level seeker-uav0) 120)
        (available water-uav0)
        (at water-uav0 base0)
        (= (fuel-level water-uav0) 500)
         (= (total-fuel-used) 0)

        (reachable base0 fire0)
        (= (min-distance base0 fire0) 4971.66)
        (reachable fire0 base0)
        (= (min-distance fire0 base0) 4971.66)
        (reachable base0 fire1)
        (= (min-distance base0 fire1) 5399.79)
        (reachable fire1 base0)
        (= (min-distance fire1 base0) 5399.79)
        (reachable base0 fuel-station0)
        (= (min-distance base0 fuel-station0) 1551.6)
        (reachable fuel-station0 base0)
        (= (min-distance fuel-station0 base0) 1551.6)
        (reachable base0 fuel-station1)
        (= (min-distance base0 fuel-station1) 3394.47)
        (reachable fuel-station1 base0)
        (= (min-distance fuel-station1 base0) 3394.47)
        (reachable base0 lake00)
        (= (min-distance base0 lake00) 2511.8)
        (reachable lake00 base0)
        (= (min-distance lake00 base0) 2511.8)
        (reachable base0 lake01)
        (= (min-distance base0 lake01) 3076.53)
        (reachable lake01 base0)
        (= (min-distance lake01 base0) 3076.53)
```

```
(reachable base0 lake10)
(= (min-distance base0 lake10) 2563.43)
(reachable lake10 base0)
(= (min-distance lake10 base0) 2563.43)
(reachable base0 lake11)
(= (min-distance base0 lake11) 2694.4)
(reachable lake11 base0)
(= (min-distance lake11 base0) 2694.4)
(reachable fire0 fire1)
(= (min-distance fire0 fire1) 1593.05)
(reachable fire1 fire0)
(= (min-distance fire1 fire0) 1593.05)
(reachable fire0 fuel-station0)
(= (min-distance fire0 fuel-station0) 3936.58)
(reachable fuel-station0 fire0)
(= (min-distance fuel-station0 fire0) 3936.58)
(reachable fire0 fuel-station1)
(= (min-distance fire0 fuel-station1) 2753.35)
(reachable fuel-station1 fire0)
(= (min-distance fuel-station1 fire0) 2753.35)
(reachable fire0 lake00)
(= (min-distance fire0 lake00) 3336.75)
(reachable lake00 fire0)
(= (min-distance lake00 fire0) 3336.75)
(reachable fire0 lake01)
(= (min-distance fire0 lake01) 2561.27)
(reachable lake01 fire0)
(= (min-distance lake01 fire0) 2561.27)
(reachable fire0 lake10)
(= (min-distance fire0 lake10) 2756.56)
(reachable lake10 fire0)
(= (min-distance lake10 fire0) 2756.56)
(reachable fire0 lake11)
(= (min-distance fire0 lake11) 3329.6)
(reachable lake11 fire0)
(= (min-distance lake11 fire0) 3329.6)
(reachable fire1 fuel-station0)
(= (min-distance fire1 fuel-station0) 4766.03)
(reachable fuel-station0 fire1)
(= (min-distance fuel-station0 fire1) 4766.03)
(reachable fire1 fuel-station1)
(= (min-distance fire1 fuel-station1) 2290.57)
(reachable fuel-station1 fire1)
(= (min-distance fuel-station1 fire1) 2290.57)
(reachable fire1 lake00)
(= (min-distance fire1 lake00) 4428.62)
(reachable lake00 fire1)
(= (min-distance lake00 fire1) 4428.62)
(reachable fire1 lake01)
(= (min-distance fire1 lake01) 3757.54)
(reachable lake01 fire1)
(= (min-distance lake01 fire1) 3757.54)
(reachable fire1 lake10)
(= (min-distance fire1 lake10) 2852.21)
(reachable lake10 fire1)
(= (min-distance lake10 fire1) 2852.21)
(reachable fire1 lake11)
(= (min-distance fire1 lake11) 3063.81)
(reachable lake11 fire1)
(= (min-distance lake11 fire1) 3063.81)
(reachable fuel-station0 fuel-station1)
(= (min-distance fuel-station0 fuel-station1) 3293.27)
(reachable fuel-station1 fuel-station0)
(= (min-distance fuel-station1 fuel-station0) 3293.27)
(reachable fuel-station0 lake00)
(= (min-distance fuel-station0 lake00) 969.477)
(reachable lake00 fuel-station0)
(= (min-distance lake00 fuel-station0) 969.477)
(reachable fuel-station0 lake01)
```

```
      (= (min-distance fuel-station0 lake01) 1636.7)
      (reachable lake01 fuel-station0)
      (= (min-distance lake01 fuel-station0) 1636.7)
      (reachable fuel-station0 lake10)
      (= (min-distance fuel-station0 lake10) 2309.88)
      (reachable lake10 fuel-station0)
      (= (min-distance lake10 fuel-station0) 2309.88)
      (reachable fuel-station0 lake11)
      (= (min-distance fuel-station0 lake11) 2862.44)
      (reachable lake11 fuel-station0)
      (= (min-distance lake11 fuel-station0) 2862.44)
      (reachable fuel-station1 lake00)
      (= (min-distance fuel-station1 lake00) 3427.22)
      (reachable lake00 fuel-station1)
      (= (min-distance lake00 fuel-station1) 3427.22)
      (reachable fuel-station1 lake01)
      (= (min-distance fuel-station1 lake01) 3116.91)
      (reachable lake01 fuel-station1)
      (= (min-distance lake01 fuel-station1) 3116.91)
      (reachable fuel-station1 lake10)
      (= (min-distance fuel-station1 lake10) 983.536)
      (reachable lake10 fuel-station1)
      (= (min-distance lake10 fuel-station1) 983.536)
      (reachable fuel-station1 lake11)
      (= (min-distance fuel-station1 lake11) 788.406)
      (reachable lake11 fuel-station1)
      (= (min-distance lake11 fuel-station1) 788.406)
      (reachable lake00 lake01)
      (= (min-distance lake00 lake01) 801.877)
      (reachable lake01 lake00)
      (= (min-distance lake01 lake00) 801.877)
      (reachable lake00 lake10)
      (= (min-distance lake00 lake10) 2494.17)
      (reachable lake10 lake00)
      (= (min-distance lake10 lake00) 2494.17)
      (reachable lake00 lake11)
      (= (min-distance lake00 lake11) 3207.24)
      (reachable lake11 lake00)
      (= (min-distance lake11 lake00) 3207.24)
      (reachable lake01 lake10)
      (= (min-distance lake01 lake10) 2302.26)
      (reachable lake10 lake01)
      (= (min-distance lake10 lake01) 2302.26)
      (reachable lake01 lake11)
      (= (min-distance lake01 lake11) 3084.16)
      (reachable lake11 lake01)
      (= (min-distance lake11 lake01) 3084.16)
      (reachable lake10 lake11)
      (= (min-distance lake10 lake11) 792.577)
      (reachable lake11 lake10)
      (= (min-distance lake11 lake10) 792.577)
      (available seeker-uav0)
      (at seeker-uav0 base0)
      (= (fuel-level seeker-uav0) 150)
      (available water-uav0)
      (at water-uav0 base0)
      (= (fuel-level water-uav0) 500)
      (start-finished))
(:goal (final-goal-finished))
```

(:metric minimize (+ (* 100 total-time) (* 1 total-fuel-used))))

Appendix C. PDDL2.1 Domain and Problem Generator Code

The following is a C++ code that was used to generate the PDDL2.1 domain and problem files from an STN:

```
#include <cassert>
#include <cstdio>
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <string>
#include <cmath>
#include "../Kirk_v2/src/Parsers/XMLTag.h"
#include "../Kirk_v2/src/TPN/TPN.h"
#include "../Kirk_v2/src/STN/Primitive.h"
#include "../Kirk_v2/src/STN/Node.h"
#include "../Kirk_v2/src/STN/Arc.h"
using namespace std;

class Coordinate
{
public:
  Coordinate(double x, double y, double z)
  {
    xCoordinate = x;
    yCoordinate = y;
    zCoordinate = z;
  };
  double get_x() const {return xCoordinate;};
  double get_y() const {return yCoordinate;};
  double get_z() const {return zCoordinate;};
private:
  double xCoordinate, yCoordinate, zCoordinate;
};

bool get_reachability(const Coordinate &from,
                      const Coordinate &to,
                      double *distance = NULL)
{
  if (distance != NULL)
  {
    *distance = sqrt(pow(to.get_x() - from.get_x(),2) +
                     pow(to.get_y() - from.get_y(),2) +
                     pow(to.get_z() - from.get_z(),2));
  }
  return true;
}

string get_pddl_vehicle_initial_state()
{
  string vehicleInitialState =
    string("       (available seeker-uav0)") +
    string("\n       (at seeker-uav0 base0)") +
    string("\n       (= (fuel-level seeker-uav0) 150)") +
    string("\n       (available water-uav0)") +
    string("\n       (at water-uav0 base0)") +
    string("\n       (= (fuel-level water-uav0) 500)");
  return vehicleInitialState;
};

string get_pddl_location_info(map<string,Coordinate> &locations)
{
  string locationInfo;

  map<string,Coordinate>::const_iterator itrtrI;
  map<string,Coordinate>::const_iterator itrtrJ;

  for (itrtrI = locations.begin(); itrtrI != locations.end()--; itrtrI++)
  {
    for ((itrtrJ = itrtrI)++; itrtrJ != locations.end(); itrtrJ++)
    {
      double distance;
      bool isReachable = get_reachability(itrtrI->second, itrtrJ->second, &distance);
      if (isReachable)
      {
        ostringstream stringStream;
        stringStream << distance;
        locationInfo += string("\n       (reachable ") +
          itrtrI->first + string(" ") + itrtrJ->first + string(")");
        locationInfo += string("\n       (= (min-distance ") +
          itrtrI->first + string(" ") + itrtrJ->first + string(") ") +
          stringStream.str() + string(")");
        locationInfo += string("\n       (reachable ") +
          itrtrJ->first + string(" ") + itrtrI->first + string(")");
        locationInfo += string("\n       (= (min-distance ") +
          itrtrJ->first + string(" ") + itrtrI->first + string(") ") +
          stringStream.str() + string(")");
      }
    }
  }

  return locationInfo;
};

void generate_pddl_problem_file(string &problemFileName,
                                map<string,Coordinate> &locations)
{
  string predefinedProblemFileName = "pfileFFUAV.pre";

  ifstream predefinedProblemDescriptionStream(predefinedProblemFileName.c_str());
  ofstream problemDescriptionStream(problemFileName.c_str());

  problemDescriptionStream << predefinedProblemDescriptionStream.rdbuf() <<
    endl << get_pddl_location_info(locations) <<
    endl << get_pddl_vehicle_initial_state() <<
    endl << "       (start-finished))" <<
    endl << "  (:goal (final-goal-finished))" <<
    endl << "  (:metric minimize (+ (* 100 total-time) (* 1 total-fuel-used))))";

  predefinedProblemDescriptionStream.close();
  problemDescriptionStream.close();
}


void generate_pddl_domain_from_tpn(map<const Arc *,Primitive *> &primitives,
                                   set<const Node *> &visitedNodes,
                                   const Node &firstNode,
                                   string &goalPredicates,
                                   string &goalOperators)
{
  visitedNodes.insert(&firstNode);
  string goalPredicatePrefix = "goal-";
  string goalPredicateSuffix = "-finished";
  string goalActionPrefix = "goal-action-";

  // If there are no out arcs, it's the last node in the tpn.
  if (firstNode.getOutArcs().size() == 0)
  {
    string header = "\n\n  (:durative-action " + goalActionPrefix;
    string parameters = "\n                 :parameters ()";
    string duration = "\n                 :duration (= ?duration ";
    string condition = "\n                 :condition (and";
    string effect = "\n                 :effect (and";
  // If there are no in arcs, it's the first node in the tpn.
  if (firstNode.getInArcs().size() == 0)
  {
    goalPredicates += " (start" + goalPredicateSuffix + ")";
    condition += " (at start (start" + goalPredicateSuffix + "))";
    effect += " (at start (not (start" + goalPredicateSuffix + ")))";
  }
  // Otherwise, an operator's conditions and effect depends on the in arcs.
  else
  {
    // Generate the necessary conditions and effects. The achievement of the in
    // arcs is the start condition for the start of the out arcs. The effect of an
    // out arc starting is that the in arcs are not "just" achieved.
    for (unsigned int i = 0; i < firstNode.getInArcs().size(); i++)
    {
      // Just trying to make the output pretty.
      if (i == 0)
      {
        condition += " ";
        effect += " ";
      }
      else
      {
        condition += "\n                              ";
        effect += "\n                            ";
      }

      // Use the memory location of the arc as the name of the arc.
      stringstream arcNameStream;
      arcNameStream << firstNode.getInArcs()[i] << "-0";

      // Write condition associated with the in arc.
      condition += "(at start (" + goalPredicatePrefix + arcNameStream.str() +
        goalPredicateSuffix + "))";

      // Write the effect associated with the in arc.
      effect += "(at start (not (" + goalPredicatePrefix +
        arcNameStream.str() + goalPredicateSuffix + ")))";
    }
  }
  goalPredicates += "\n             (final-goal" + goalPredicateSuffix + ")";
  goalOperators += header + "final-goal-operator" + parameters + duration + "0)" +
    condition + ")" + effect +
    "\n                           (at end (final-goal" + goalPredicateSuffix +
    "))))";
}
// Otherwise, there are more operators to generate.
else
{
  // Generate an operator for each out arc.
  for (unsigned int i = 0; i < firstNode.getOutArcs().size(); i++)
  {
    string header = "\n\n  (:durative-action " + goalActionPrefix;
    string parameters = "\n                 :parameters ()";
    string duration = "\n                 :duration (= ?duration ";
    string condition = "\n                 :condition (and";
    string effect = "\n                 :effect (and";

    // If there are no in arcs, it's the first node in the tpn.
    if (firstNode.getInArcs().size() == 0)
    {
      goalPredicates += " (start" + goalPredicateSuffix + ")";
      condition += " (at start (start" + goalPredicateSuffix + "))";
      effect += " (at start (not (start" + goalPredicateSuffix + ")))";
    }
    // Otherwise, an operator's conditions and effect depends on the in arcs.
    else
    {
      // Generate the necessary conditions and effects. The achievement of the in
      // arcs is the start condition for the start of the out arcs. The effect of an
      // out arc starting is that the in arcs are not "just" achieved.
      for (unsigned int j = 0; j < firstNode.getInArcs().size(); j++)
      {
        // Just trying to make the output pretty.
        if (j == 0)
        {
          condition += " ";
          effect += " ";
        }
        else
        {
          condition += "\n                              ";
          effect += "\n                            ";
        }

        // Use the memory location of the arc as the name of the arc.
        stringstream arcNameStream;
        arcNameStream << firstNode.getInArcs()[j] << "-" << i;

        // Write condition associated with the in arc.
        condition += "(at start (" + goalPredicatePrefix +
          arcNameStream.str() + goalPredicateSuffix + "))";

        // Write the effect associated with the in arc.
        effect += "(at start (not (" + goalPredicatePrefix +
          arcNameStream.str() + goalPredicateSuffix + ")))";
      }
    }

    // If the arc has a primitive activity, the activity is a condition that
    // must be true during the execution of the operator associated to the
    // out arc.
    string goalCondition = "";
    map<const Arc *,Primitive *>::const_iterator itrtr =
      primitives.find(firstNode.getOutArcs()[i]);
    if (itrtr != primitives.end())
    {
      goalCondition += "\n                              (at start (" +
        itrtr->second->getName();
      const vector<string> &arguments = itrtr->second->getArguments();
      for (unsigned int j = 0; j < arguments.size(); j++)
      {
        goalCondition += " " + arguments[j];
      }
      goalCondition += "))";
    }

    // Use the memory location of the arc as the name of the arc.
    stringstream arcNameStream;
    arcNameStream << firstNode.getOutArcs()[i];

    stringstream lowerBound;
    lowerBound << firstNode.getOutArcs()[i]->getLowerBound();

    goalOperators += header + arcNameStream.str() + parameters + duration +
      lowerBound.str() + ")" + condition + goalCondition + ")" + effect;

    unsigned int numOfChildren =
      firstNode.getOutArcs()[i]->getEndNode()->getOutArcs().size();
    if (numOfChildren == 0)
    {
      numOfChildren = 1;
    }
    for (unsigned int j = 0; j < numOfChildren; j++)
    {
      stringstream childNumberStream;
      childNumberStream << j;
```

```
        goalPredicates += "\n             (" + goalPredicatePrefix +
          arcNameStream.str() + "-" + childNumberStream.str() +
          goalPredicateSuffix + ")";

        goalOperators +=
          "\n                            (at end (" + goalPredicatePrefix +
          arcNameStream.str() + "-" + childNumberStream.str() +
          goalPredicateSuffix + "))";
      }
      goalOperators += "))";

      // Generating the operator for the end node, iff the node was not visited.
      if (visitedNodes.find(firstNode.getOutArcs()[i]->getEndNode()) ==
        visitedNodes.end())
      {
        generate_pddl_domain_from_tpn(primitives, visitedNodes,
          *firstNode.getOutArcs()[i]->getEndNode(),goalPredicates,
          goalOperators);
      }
    }
  }
}

/*void generate_pddl_domain_from_tpn(map<const Arc *,Primitive *> &primitives,
                             set<const Node *> &visitedNodes,
                             const Node &firstNode,
                             string &goalPredicates,
                             string &goalOperators)
{
  visitedNodes.insert(&firstNode);
  string goalPredicatePrefix = "goal-";
  string goalStartSuffix = "-started";
  string goalCompletionSuffix = "-finished";
  string goalActionPrefix = "goal-action-";
  string header = "\n\n  (:durative-action " + goalActionPrefix;
  string parameters = "\n                :parameters ()";
  string duration = "\n                :duration (= ?duration ";
  string condition = "\n                :condition (and";
  string effect = "\n                :effect (and";

  // If there are no in arcs, it's the first node in the tpn.
  if (firstNode.getInArcs().size() == 0)
  {
    goalPredicates += " (start" + goalCompletionSuffix + ")";
    condition += " (at start (start" + goalCompletionSuffix + "))";
  }
  // Otherwise, an operator's conditions and effect depends on the in arcs.
  else
  {
    // Generate the necessary conditions. The achievement of the in arcs is the
    // start condition for the start of the out arcs.
    for (unsigned int i = 0; i < firstNode.getInArcs().size(); i++)
    {
      // Just trying to make the output pretty.
      if (i == 0)
      {
        condition += " ";
      }
      else
      {
        condition += "\n                              ";
      }

      // Use the memory location of the arc as the name of the arc.
      stringstream arcNameStream;
      arcNameStream << firstNode.getInArcs()[i];

      // Write condition associated with the in arc.
      condition += "(at start (" + goalPredicatePrefix + arcNameStream.str() +
        goalCompletionSuffix + "))";
    }
  }

  // If there are no out arcs, it's the last node in the tpn.
  if (firstNode.getOutArcs().size() == 0)
  {
    goalPredicates += "\n             (final-goal" + goalStartSuffix +
      ")";
    goalPredicates += "\n             (final-goal" + goalCompletionSuffix +
      ")";
    goalOperators += header + "final-goal-operator" + parameters + duration +
      "0)" + condition +
      "\n                            (at start (not (final-goal" +
      goalStartSuffix + "))))" + effect + " (at start (final-goal" +
      goalStartSuffix + "))" +
      "\n                           (at end (final-goal" +
      goalCompletionSuffix + "))))";
  }
  // Otherwise, there are more operators to generate.
  else
  {
    // Generate an operator for each out arc.
    for (unsigned int i = 0; i < firstNode.getOutArcs().size(); i++)
    {

      // If the arc has a primitive activity, the activity is a condition that
      // must be true during the execution of the operator associated to the
      // out arc.
      string goalCondition = "";
      map<const Arc *,Primitive *>::const_iterator itrtr =
        primitives.find(firstNode.getOutArcs()[i]);
      if (itrtr != primitives.end())
      {
        goalCondition += "\n                             (over all (" +
          itrtr->second->getName();
        const vector<string> &arguments = itrtr->second->getArguments();
        for (unsigned int i = 0; i < arguments.size(); i++)
        {
          goalCondition += " " + arguments[i];
        }
        goalCondition += "))";
      }

      // Use the memory location of the arc as the name of the arc.
      stringstream arcNameStream;
      arcNameStream << firstNode.getOutArcs()[i];

      stringstream lowerBound;
      lowerBound << firstNode.getOutArcs()[i]->getLowerBound();

      goalPredicates += "\n             (" + goalPredicatePrefix +
        arcNameStream.str() + goalStartSuffix + ")";
      goalPredicates += "\n             (" + goalPredicatePrefix +
        arcNameStream.str() + goalCompletionSuffix + ")";

      goalOperators += header + arcNameStream.str() + parameters + duration +
        lowerBound.str() + ")" + condition + goalCondition +
        "\n                             (at start (not (" +
        goalPredicatePrefix + arcNameStream.str() +
        goalStartSuffix + "))))" + effect + " (at start (" +
        goalPredicatePrefix + arcNameStream.str() + goalStartSuffix + "))" +
        "\n                          (at end (" + goalPredicatePrefix +
        arcNameStream.str() + goalCompletionSuffix + "))))";

      // Generating the operator for the end node, iff the node was not visited.
      if (visitedNodes.find(firstNode.getOutArcs()[i]->getEndNode()) ==
        visitedNodes.end())
      {
        generate_pddl_domain_from_tpn(primitives, visitedNodes,
          *firstNode.getOutArcs()[i]->getEndNode(),goalPredicates,
          goalOperators);
      }
    }
  }
}*/

void get_pddl_domain_from_tpn(TPN &tpn,
                              string &goalPredicates,
                              string &goalOperators)
{
  map<const Arc *,Primitive*> primitives;
  set<Primitive *>::const_iterator itrtr;
  for (itrtr = tpn.getPrimitives().begin(); itrtr != tpn.getPrimitives().end();
    itrtr++)
  {
    primitives.insert(pair<const Arc *,Primitive *>((*itrtr)->getArc(),*itrtr));
  }

  set<const Node *> visitedNodes;

  goalPredicates += "  (:predicates";
  generate_pddl_domain_from_tpn(primitives, visitedNodes, *tpn.getStart(),
    goalPredicates, goalOperators);
  goalPredicates += ")";
}

void generate_pddl_domain_file(TPN &tpn, string &domainFileName)
{
  string predefinedDomainFileName = "FFUAV.pddl.pre";

  ifstream predefinedDomainDescriptionStream(predefinedDomainFileName.c_str());
  ofstream domainDescriptionStream(domainFileName.c_str());

  string *goalPredicates = new string();
  string *goalOperators = new string();
  get_pddl_domain_from_tpn(tpn,*goalPredicates,*goalOperators);

  domainDescriptionStream << predefinedDomainDescriptionStream.rdbuf() <<
    endl << *goalPredicates << *goalOperators << ")";

  delete goalPredicates;
  delete goalOperators;

  predefinedDomainDescriptionStream.close();
  domainDescriptionStream.close();
}

void generate_tpn_ps_file(vector<TPN *> &tpns, string &psFileName)
{
  string dotFileName = "tpn.dot";
  ofstream dotFileStream(dotFileName.c_str());
  tpns[0]->outputDOT(dotFileStream,string("TPN"));
  dotFileStream.close();
  string command = "dot -Tps " + dotFileName + " -o " + psFileName;
  system(command.c_str());
}

vector<TPN *> readin_XML_TPN(string &xmlFileName)
{
  // Read in an XML file into a string.
  ifstream xmlFileStream(xmlFileName.c_str());
  stringstream xmlStringStream;
  xmlStringStream << xmlFileStream.rdbuf();
  xmlFileStream.close();
  string xmlString = xmlStringStream.str();

  // Collect XML tags from the string.
  vector<XMLTag *> xmlTags;
  unsigned int i = 0;
  while (i < xmlString.size())
  {
    // Ignore whitespaces
    if (xmlString[i] == ' ' || xmlString[i] == '\t' || xmlString[i] == '\n')
    {
      i++;
    }
    else
    {
      // Collect XML tag
      XMLTag *tag = new XMLTag(xmlString, i);
      xmlTags.push_back(tag);
    }
  }

  // Generate TPNs.
  vector<TPN *> tpns;
  for (unsigned int i = 0; i < xmlTags.size(); i++)
  {
    tpns.push_back(new TPN(*xmlTags[i]));
  }

  // Delete XML tags.
  for (unsigned int i = 0; i < xmlTags.size(); i++)
  {
    delete xmlTags[i];
  }

  return tpns;
}

int main(int argc, char *argv[])
{

  // Read in location information.
  string locationInfoFile = "locations.txt";
  ifstream locationInfoStream(locationInfoFile.c_str());
  map<string,Coordinate> locations;
  string locationName;
  double xCoordinate, yCoordinate, zCoordinate;
  while (!locationInfoStream.eof())
  {
     locationInfoStream >> locationName;
     locationInfoStream >> xCoordinate;
     locationInfoStream >> yCoordinate;
     locationInfoStream >> zCoordinate;
     locations.insert(pair<string,Coordinate>(locationName,
       Coordinate(xCoordinate,yCoordinate,zCoordinate)));
  }
  locationInfoStream.close();

  string xmlFileName = "tpn.xml";
  vector<TPN *> tpns = readin_XML_TPN(xmlFileName);

  string psFileName = "tpn.ps";
  generate_tpn_ps_file(tpns, psFileName);

  string domainFileName = "FFUAV.pddl";
  generate_pddl_domain_file(*tpns[0], domainFileName);

  string problemFileName = "pfileFFUAV";
  generate_pddl_problem_file(problemFileName, locations);

  for (unsigned int i = 0; i < tpns.size(); i++)
  {
    delete tpns[i];
  }
```

```
    cout << endl;
    return 0;
  };
```