

Generative Planning for Hybrid Systems based on Flow Tubes

Hui X. Li and Brian C. Williams

MIT CSAIL MERS

32 Vassar St. 32-224, Cambridge, MA 02139

huili, williams@mit.edu

Abstract

When controlling an autonomous system, it is inefficient or sometimes impossible for the human operator to specify detailed commands. Instead, the field of AI autonomy has developed goal-directed systems, in which human operators specify a series of goals to be accomplished. Increasingly, the control of autonomous systems involves performing a mix of discrete and continuous actions. For example, a typical autonomous underwater vehicle (AUV) mission involves discrete actions, like get GPS and set sonar, and continuous actions, like descend and ascend, which involve continuous dynamics of the vehicle. Accordingly, we develop a hybrid planner that determines a series of discrete and continuous actions that achieve the mission goals.

In this paper, we describe a novel approach to solving the generative planning problem for hybrid systems, involving both continuous and discrete actions. The planner, Kongming¹, incorporates two innovations. First, it employs a compact representation of all hybrid plans, called a Hybrid Flow Graph, which combines the strengths of a Planning Graph for discrete actions and Flow Tubes for continuous actions. Second, it encodes the Hybrid Flow Graph as a mixed logic linear/nonlinear program, which it solves using an off-the-shelf solver. We empirically demonstrate that Kongming can efficiently plan for real-world scenarios that are based on science missions performed at the Monterey Bay Aquarium Research Institute (MBARI).

Introduction

It is often inefficient or impossible for the human operator to specify the detailed action sequence, needed to control an autonomous system. Especially for real-world missions, such as AUV seafloor mapping and multi-vehicle search-and-rescue tasks, it is crucial to automate the process, such that human operators only need to specify a set of goals that they want to accomplish, and have the planner itself produce a series of actions that achieve the mission goals, based on the model of the physical plant under control.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

¹Kongming is the courtesy name of the genius strategist of the Three Kingdoms period in ancient China.

Increasingly, the control of autonomous systems involves a mix of discrete and continuous actions. For example, a typical AUV mission is implemented by invoking a sequence of behaviors, which include discrete actions like get GPS and set sonar, and continuous actions that involve the continuous dynamics of the vehicle, such as descend and ascend. Hence, it is important for a generative planner to be able to reason about and plan with both discrete and continuous actions.

A large number of existing temporal planners (Smith and Weld 1999; Long and Fox 2002; Gerevini et al. 2003; Do and Kambhampati 2001; Frank and Jónsson 2003; Joslin 1996; Penberthy and Weld 1994; Vere 1983) are able to generate discrete action sequences to achieve goals, and handle time and other simple numeric constraints. Some of them have been used in real-world planning problems. For example, EUROPA (Frank and Jónsson 2003) has been deployed in space missions at NASA (Muscettola et al. 1998) and AUV missions at MBARI (McGann et al. 2007). However, the type of hybrid problems we are interested in requires more flexibility in representing the continuous actions. More specifically, we want to be able to represent the dynamics of the continuous actions as a set of ordinary differential equations (ODE). LPSAT planner (Wolfman and Weld 1999) solves resource planning problems where real-valued quantities are involved. TM-LPSAT (Shin and Davis 2005) extends it to handling problems with durative actions and linear continuous change. The difference from our work is that TM-LPSAT is limited to simple linear changes in terms of continuous effects and hence cannot handle complex robot dynamics or obstacle avoidance. In addition TM-LPSAT generates a feasible plan, while often times the capability of generating an optimal plan is desired.

In this paper, we introduce a novel approach to solving the generative planning problem for hybrid systems, where both continuous and discrete actions are involved. Currently actions are assumed to have fixed and equal durations for simplicity. The planner, Kongming, has the following key innovations. First, it employs a compact representation of all hybrid plans, called a Hybrid Flow Graph, which combines the strengths of a Planning Graph for discrete actions and Flow Tubes for continuous actions. Second, it encodes the Hybrid Flow Graph as a mixed logic linear/nonlinear program (ML(N)LP) that is then solved by an off-the-shelf solver. Kongming is implemented in Java, and uses the

ILOG CPLEX solver to solve the ML(N)LPs. We empirically evaluate Kongming on a set of real-world scenarios that are based on the AUV science missions, performed at MBARI. The results show that Kongming is capable of planning for complex hybrid problems efficiently.

Ocean Exploration using AUVs

Many real-world robot missions involve planning of a mix of discrete and continuous actions. At MBARI among other missions, marine scientists use AUVs to obtain high-resolution seafloor maps of the Monterey Canyon, one of the deepest underwater canyons along the continental United States. Currently the AUVs are controlled by pre-defined action sequences, specified by engineers. Having to design the action sequence for every new mission puts a significant cognitive burden on the human operator. A generative planner that can produce the sequences of both continuous and discrete actions automatically would save time, reduce human error and enable more complex missions.

Let's consider an example. Fig. 1 shows part of a seafloor mapping mission of an AUV in 2D. The actions and their preconditions, effects and dynamics are listed in Fig. 2. We use first-order velocity limited dynamics in the example for simplicity, but we are not limited to such dynamical systems. There's an obstacle on the way. The objective may be the shortest path, minimum battery use, or best mapping image resolution, which translates to traveling close to the seafloor as long as possible.

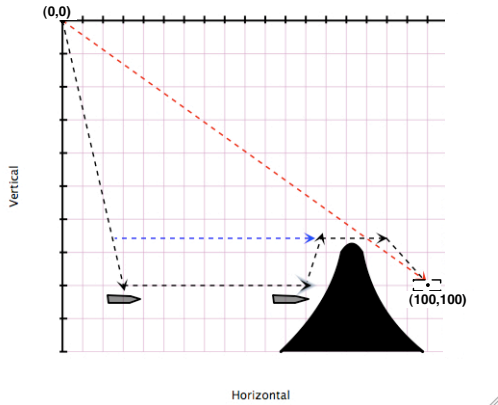


Figure 1: Part of an AUV seafloor mapping mission

Problem Statement

The inputs to the system are:

- A set of real-valued state variables, $s \in \mathbb{R}^n$, e.g. the x, y position of the physical plant, and a set of real-valued input variables, $u \in \mathbb{R}^n$, e.g. the x, y velocity;
- An initial condition I and a goal condition G , each denoted by a conjunction of (in)equalities over the state variables and propositions, e.g. the initial condition for the example shown in Fig. 1 is $\{x = 0, y =$

(:action glide	(:action getGPS
:duration (d)	:duration (d)
:precondition (\neg rudder)	:precondition ($(\text{and}(\neg\text{GPS})(y = 0))$
:effect ()	:effect (GPS)
:dynamics ($-10 \leq vx \leq 10, vy = 0$)	:dynamics ()
(:action ascend	(:action startRudder
:duration (d)	:duration (d)
:precondition ($(\text{and}(\text{rudder})(y \geq 3))$:precondition (\neg rudder)
:effect ()	:effect (rudder)
:dynamics ($4 \leq vx \leq 8, -5 \leq vy \leq -2$)	:dynamics ()
(:action descend	(:action stopRudder
:duration (d)	:duration (d)
:precondition ($(\text{and}(\text{rudder})(y \leq 200))$:precondition (rudder)
:effect ()	:effect (\neg rudder)
:dynamics ($4 \leq vx \leq 8, 3 \leq vy \leq 6$)	:dynamics ()

Figure 2: Available actions of the seafloor mapping example.

$0, \neg\text{GPS}, \neg\text{rudder}\}$, and the goal condition is $\{95 \leq x \leq 105, 98 \leq y \leq 102, \text{GPS}, \text{rudder}\}$;

- A set of hybrid actions, each of which has *preconditions*, *effects*, *dynamics* and a duration. For simplicity we currently assume all actions have fixed and equal durations. Future work will extend Kongming to handle varying durations with flexible temporal bounds.
 - Preconditions can be continuous or discrete. A continuous precondition is a conjunction of (in)equalities over state variables, and a discrete precondition is a conjunction of propositions.
 - Effects are discrete facts. An effect is a conjunction of propositions.
 - Dynamics are represented by bounds on the control input $|u| \leq u_{\text{limit}}$, and the state equation $\dot{s} = As + Bu$. They are mapped into flow tubes, as described in next section. The continuous effect of the action is then the goal region of the flow tube.
- A hybrid action is continuous when there are dynamics, e.g. action *descend* in Fig. 2. A hybrid action is discrete when no dynamics are involved, e.g. action *getGPS* in Fig. 2.
- Two types of obstacles, inside obstacles and outside obstacles:
 - An inside obstacle defines the boundaries of the world the robots maneuver in, e.g. the map boundary in Fig. 1. It defines the feasible region as inside the obstacle and convex;
 - An outside obstacle is an obstacle our robots avoid, e.g. the seamount in Fig. 1. It defines the feasible region as outside the obstacle and non-convex.
- An objective function f to be optimized.

The outputs are an optimal solution to $s \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$, and the corresponding action sequence.

In our framework, a *valid hybrid plan* is defined as follows. First, it follows the definition in Graphplan. A valid

plan is a set of actions and times in which each is to be carried out; actions can be concurrent as long as they do not interfere with each other; a valid plan may perform an action if all its preconditions are true; and finally a valid plan makes the goal condition of the problem true at the final time step. Second, a valid plan is a state trajectory that goes from a point in the initial region of the problem to a point in the goal region of the problem, without going outside any of the connected flow tubes, while avoiding obstacles.

The Approach

Graphplan (Blum and Furst 1997) and Blackbox (Kautz and Selman 1997) have introduced two fundamental concepts to planning with discrete actions. Graphplan introduced the Planning Graph as a compact way of representing all possible plans, while pruning many invalid plans. Blackbox introduced the concept of searching the plan space by reformulating the Planning Graph as a SAT problem, and using the best off-the-shelf solvers to extract a plan. In this paper, we take an analogous approach, but for hybrid actions.

We first introduce an important concept to our approach, called *Flow Tubes*. Then we introduce the main innovation of the paper, a Hybrid Flow Graph, which is a compact representation of all valid hybrid plans. The second innovation of the paper is a constraint encoding of the Hybrid Flow Graph as an ML(N)LP, which we feed into a standard solver.

Flow Tube

When planning with purely discrete actions, the decision variables are discrete, and we only need to reason about a finite number of trajectories in plan space. However, when we include continuous actions and continuous variables in planning, we need a compact way of representing and reasoning about an infinite number of trajectories. Moreover, for real-world planning problems, we often need to take into account uncertainty in states and in actions. To address the above problems, we use flow tubes to represent the abstraction of the infinite number of trajectories.

Flow tubes have been used in the qualitative reasoning community to represent a set of trajectories with common characteristics that connect two regions. (Bradley and Zhao 1993) used flow tubes to characterize phase spaces during analysis of nonlinear dynamical systems. (Hofmann and Williams 2006) used flow tubes to represent sets of flexible state trajectories that can reach the goal region of foot placement.

We define a *flow tube* through the concept of *cross sections*. A *cross section* c is a function of an N dimensional initial region R_i , a duration d , and bounds on dynamics dB , $c = f(R_i, d, dB)$. Given an initial region, c is the set of states in N dimensions, such that they are all the states that can be reached from the initial region for a certain duration. A *flow tube* is a collection of all the trajectories that start from the initial region and reach the cross section. For example, Fig. 3 shows a simple flow tube of a first-order velocity limited system in 1D. The vertical line c_1 is the cross section for duration d_1 , and c_2 is the cross section for duration d_2 . In this case, $R_i = [xi_{min}, xi_{max}]$, $dB = [vx_{min}, vx_{max}]$, and $c = [xg_{min}(d), xg_{max}(d)]$.

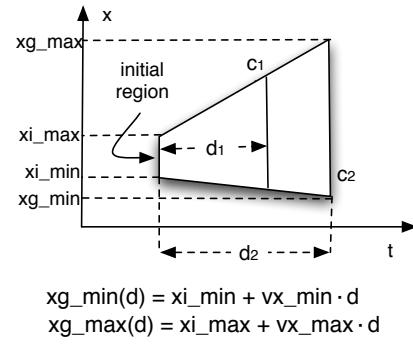


Figure 3: The flow tube of a 1D action for a first-order velocity limited dynamic system. $R_i = [xi_{min}, xi_{max}]$. $dB = [vx_{min}, vx_{max}]$. $c = [xg_{min}(d), xg_{max}(d)]$.

Flow tubes for the first-order velocity limited system are computed through a set of linear equations. For the 1D action in Fig. 3, $xg_{min}(d) = xi_{min} + vx_{min} * d$ and $xg_{max}(d) = xi_{max} + vx_{max} * d$. For higher dimensional, nonlinear dynamical systems, computing a flow tube is more complex. The example in Fig. 4 is a flow tube of a second-order acceleration limited system. Linear equations are no longer adequate. Therefore various approximation methods have been used to approximate the cross sections. (Hofmann and Williams 2006) use a polyhedral approximation, which approximates the tubes as slices of polyhedra for each time step. (Kurzanskiy 2006) uses an ellipsoidal calculus for approximation that has proven highly efficient.

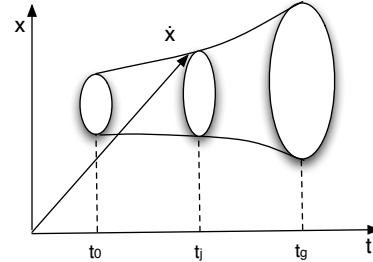


Figure 4: The flow tube of a 1D action for a second-order acceleration limited dynamic system.

Hybrid Flow Graph

Hybrid Flow Graphs are built upon Planning Graphs from Graphplan, augmented with flow tubes, to represent all valid plans and collections of feasible state trajectories. Before diving into the Hybrid Flow Graph, we need to explain two important processes. One is the mapping from a hybrid action to a flow tube. The other is the connection from one flow tube to another.

Flow Tube of An Action As described in Problem Statement, a hybrid action includes the following: a continuous precondition, characterized by a conjunction of

(in)equalities over the state variables, $\wedge_i f_i(s) \leq 0$; discrete preconditions and effects, both described as conjunctions of propositions; and dynamics of the physical plant, which consist of a state equation and the bounds on the control input, e.g. the velocity limits of a first-order velocity limited system. We only need to consider the continuous elements in a hybrid action in order to map it to a flow tube, i.e. the continuous precondition and the dynamics. In Fig. 5, a generic hybrid action is on the left, and its corresponding flow tube is on the right. Suppose the region s is the current region of feasible states in state space, and the region that corresponds to the continuous precondition has a non-empty intersection with the region s . We denote the non-empty intersection R_i , which is the initial region of the flow tube. The cross section c is calculated as a function of R_i , duration d and the dynamics bounds, and the function type depends on the state equation.

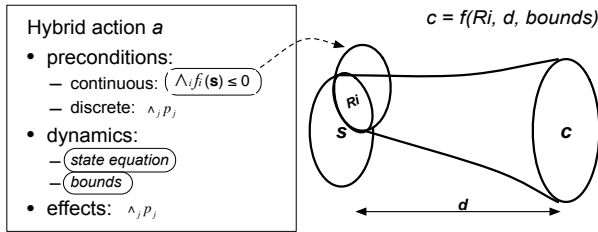


Figure 5: Mapping from a hybrid action to a flow tube.

Connecting Flow Tubes The condition under which action a_2 's flow tube is connected to action a_1 's flow tube is the following: connect a_2 to a_1 if a_1 's goal region R_{g1} intersects with a_2 's continuous precondition CP_2 , i.e. $R_{g1} \cap CP_2 \neq \emptyset$. In this case, we name R_{g1} the *resolved precondition* of action a_2 . A *resolved precondition* of action A is a continuous fact that has a non-empty intersection with A 's precondition. Fig. 6 shows an example for a first-order velocity limited system. We connect a_2 's flow tube to a_1 's flow tube, because a_2 's continuous precondition, represented by the vertical line, has a non-empty intersection with R_{g1} . The non-empty intersection, R_{i2} , is the initial region of a_2 's flow tube. This connection condition guarantees that all valid plans are included in the graph, so it is complete. However, the condition is not sound, meaning that not all plans in the graph are valid, which is in the same spirit as Graphplan. The step in Kongming, where it encodes the Hybrid Flow Graph as an ML(N)LP and solves it, makes sure that the output plan is valid and optimal.

A Hybrid Flow Graph is similar to a Planning Graph in that it is also a directed, leveled graph that alternates between *fact levels* and *action levels*. It represents all valid plans while pruning many invalid plans through mutual exclusion. The structure of a Hybrid Flow Graph is different from that of a Planning Graph in that, first, a fact level of the Hybrid Flow Graph includes two types of fact nodes, discrete and continuous; and second, an action level of the Hybrid Flow Graph contains hybrid actions, which can be

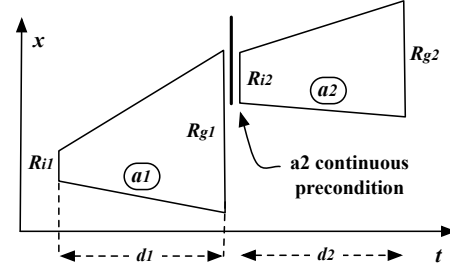


Figure 6: Connecting one flow tube to another.

either continuous or discrete, based on whether continuous dynamics are involved.

Fact Level A *fact level* includes two types of fact nodes: *continuous fact nodes* and *discrete fact nodes*. A *continuous fact node* is defined by a conjunction of (in)equalities over the state variables, $\wedge_i f_i(s) \leq 0$. A *discrete fact node* is defined by a proposition. In a Planning Graph a fact can be either a precondition or an effect; in a Hybrid Flow Graph, a fact can be a discrete precondition, an effect, a goal region of a flow tube, or a resolved precondition.

Action Level An *action level* contains hybrid action nodes, and each node can be either continuous or discrete. When continuous dynamics are not involved, a hybrid action is discrete. When there are dynamics involved, a hybrid action is continuous and is denoted by a flow tube, connecting an initial region and a goal region over a duration. From sub-section Connecting Flow Tubes, we can see that because there may be multiple continuous facts in a fact level that can be resolved preconditions of an action, there may be multiple flow tubes of the same action in one action level. In that case, the multiple flow tubes share the same dynamic bounds, but may connect different initial and goal regions. For example, in Fig. 7 in Action Level 1, there are two action nodes from the action *glide*, based on the different resolved preconditions in Fact Level 1. A no-op action node can be continuous or discrete depending on the type of the fact it carries forward. So for example, if the fact is continuous, then the no-op action node is a flow tube whose initial region and goal region are equal.

Exclusion Relations Similar to a Planning Graph, in a Hybrid Flow Graph, two action nodes, either continuous or discrete, at a given action level are *mutually exclusive* if no valid plan could contain both. Likewise, two facts at a given fact level are mutually exclusive if no valid plan could have both satisfied. Different from Graphplan are the exclusion rules that involve continuous facts or actions, i.e. Competing Needs II, Obstacle Avoidance and Causal Conflict. We propagate the exclusion relationships through the graph using the following rules. We use *mutex* as a short term for mutual exclusion in the rest of the paper.

- **Interference:** If an effect of action A negates an effect or a precondition of action B , then A and B are mutex.
- **Competing Needs I:** If a discrete precondition of A and

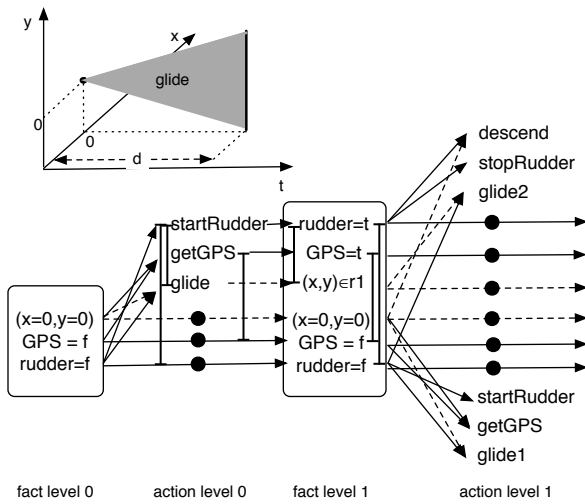


Figure 7: A Hybrid Flow Graph for the AUV seafloor mapping example. The dotted edges connect resolved preconditions to actions and actions to their goal regions. The solid edges connect preconditions to actions and actions to their effects. The big dots represent no-op actions. The vertical lines connect mutex actions or facts. The flow tube of action *glide* in action level 0 is shown on top.

a discrete precondition of B negate one another or are mutex, then A and B are mutex.

- **Competing Needs II:** If a resolved precondition of action A and a resolved precondition of action B are mutex or have an empty intersection, then A and B are mutex.
- **Obstacle Avoidance:** If the initial region or goal region of a continuous action node A or a continuous precondition of a discrete action node A has an empty intersection with the inside obstacle or is contained by an outside obstacle, then A is excluded.
- **Causal Conflict:** If each of fact A 's causal actions is mutex with each of fact B 's causal actions, then A and B are mutex, where a causal action of fact A is an action such that either A is its effect or A is its goal region.

The Competing Needs II rule indicates, whether two continuous actions are mutex does not depend on the continuous preconditions of the actions, but rather their resolved preconditions. Fig. 8 and Fig. 9 shows an example. We have two actions, listed on the right of Fig. 8. *getGPS*'s continuous precondition corresponds to the x axis, and *descend*'s continuous precondition corresponds to the region above $y = 50$. The two regions overlap, but we need to look at the resolved preconditions in order to decide about mutex. Suppose the fact level is as in Fig. 9, and the two continuous facts correspond to the two rectangular regions in Fig. 8. From the definition of resolved preconditions, $(x, y) \in ([0, 5], [-3, 3])$ can be the resolved precondition of *getGPS*, and $(x, y) \in ([10, 20], [10, 20])$ can be the resolved precondition of *descend*. Because they have an empty intersection, the two action instances are mutex.

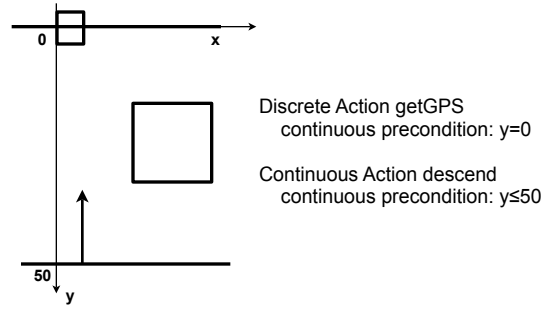


Figure 8: Competing Needs II example: part I

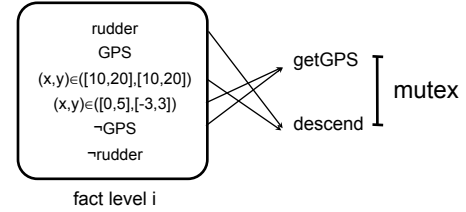


Figure 9: Competing Needs II example: part II

Expanding the Hybrid Flow Graph Similar to Graphplan, we expand the graph by starting from Fact Level 0, where the initial condition is contained; then keep adding an action level, followed by a fact level, until the goal condition is reached. As shown in the pseudo code ExpandGraph(), for each fact level we go through all the possible actions (line 4, 5) to check for the insertion conditions. The pseudo code of routine CheckInsertion() gives the insertion conditions for both actions and facts. The conditions are similar to Graphplan in checking for discrete preconditions, and inserting discrete effects. The conditions are different from Graphplan in checking for continuous preconditions, obstacles, and computing flow tube goal regions.

Let's revisit the AUV example. Fig. 7 shows part of the expansion of the flow graph for the planning problem in the example. The initial condition is in Fact Level 0. We insert discrete action *startRudder* because its only precondition is in Fact Level 0. We insert discrete action *getGPS* because its discrete precondition is in Fact Level 0 and its continuous precondition $y = 0$ has a non-empty intersection with fact $(x = 0, y = 0)$. We insert continuous action *glide* because its only precondition, $\neg rudder$, is in Fact Level 0. The initial region of the *glide* flow tube is $(x = 0, y = 0)$ and the intersection of its goal region with the inside obstacle and the complementary set of the outside obstacle is denoted by $r1$. Hence $r1$ is inserted in Fact Level 1 along with other effects. *startRudder* and *glide* are mutually exclusive due to the Interference rule. The Interference rule also applies to the other two pairs of mutex in Action Level 0. The three pairs of mutually exclusive facts in Fact Level 1 are all due to the Causal Conflict rule.

As mentioned before, similar to Graphplan, the way we expand the graph does not guarantee every plan in the graph

is valid, but it guarantees all valid plans are included. The step, where we encode the graph as an ML(N)LP and solve it, makes sure that every solution plan is valid and optimal.

ExpandGraph - returns a Hybrid Flow Graph

1. start with the initial condition - Fact Level 0
2. *while* goal condition is not contained *do*
3. find mutex facts in current fact level
4. *for* each action
5. CheckInsertion()
6. *for* each fact in current fact level
7. form no-op action
8. find mutex actions in current action level

CheckInsertion - returns next action level, next fact level

1. *if* action a is discrete
2. check 1: a 's discrete preconditions are contained in fact level
3. check 2: each of a 's continuous preconditions has a non-empty intersection with a continuous fact in fact level
4. check 3: no two preconditions or resolved preconditions of a are mutex
5. *if* the 3 checks are all true
6. insert a and a 's effects
7. *if* action a is continuous
8. check 1: a is not labeled as excluded
9. check 2: a 's discrete preconditions are contained in fact level
10. check 3: each of a 's continuous preconditions has a non-empty intersection with a continuous fact in fact level; the intersection is the initial region of a 's flow tube
11. check 4: the goal region of a 's flow tube computed from the initial region has a non-empty intersection r with the inside obstacle and the complementary set of the outside obstacles
12. check 5: no two preconditions or resolved preconditions of a are mutex
13. *if* the 5 checks are all true
14. insert a and r

Constraint Encoding

Kongming encodes the Hybrid Flow Graph as an ML(N)LP, based on a series of encoding rules. The rules are analogous to those for Blackbox, which encodes a Planning Graph as a SAT problem. The main difference from Blackbox is the introduction of continuous variables and constraints.

Depending on the type of the dynamical system and the obstacle representation used in a problem, the constraint encoding could be linear or nonlinear. Note that Kongming is not limited to any specific type of continuous dynamics or any dimensionality. The condition on which the flow tubes are connected, the expansion of the Hybrid Flow Graph, the mutex rules, and the constraint encoding rules are all independent of the dynamics, dimensionality or the obstacle representation. Aspects that are influenced are the method used to compute the flow tubes, and the type of solver we choose to solve the ML(N)LPs. The encoding includes both real-valued and boolean variables. The real-valued variables are the state variables for each fact level and the input variables

of each continuous action node. The boolean variables represent all the fact and action nodes in the graph. The objective function for a plan is over the variables and is any such function handled by the solver.

- If fact F is true, at least one of F 's causal actions is true.
- If continuous fact F in Fact Level i is true, the state variables of the fact level satisfy F .
- If action A is true, all A 's discrete preconditions and resolved preconditions are true.
- If continuous action A in Action Level i is true, the state variables of Fact Level i satisfy A 's initial region, and the state variables of Fact Level i and $i + 1$ and the input variables of Action Level i satisfy A 's dynamic ODEs.
- Mutex facts or actions cannot both be true.
- Obstacles are avoided at all times.

The form of an ML(N)LP is shown in Fig. 10, where Φ is recursively defined. It is MLLP when $f(x)$ and $g(x)$ are linear; it is MLNLP when $f(x)$ and $g(x)$ are nonlinear.

$$\begin{aligned}
 &\text{minimize } f(X) \\
 &\text{s.t. } \Phi(X) \\
 &\Phi(X) := \Phi(X) \wedge \Phi(X) \mid \Phi(X) \vee \Phi(X) \mid \neg \Phi(X) \mid \\
 &\Phi(X) \Rightarrow \Phi(X) \mid \Phi(X) \Leftrightarrow \Phi(X) \mid \text{proposition} \mid g(X) \leq 0
 \end{aligned}$$

Figure 10: The form of an ML(N)LP

High-level Algorithm

Roughly speaking, the high-level algorithm of Kongming interleaves between expanding the Hybrid Flow Graph and encoding and solving the ML(N)LP. As shown in the pseudo code Kongming(), the algorithm starts from Fact Level 0 and keeps expanding the graph until the goal condition is reached (line 1). Then it encodes the graph as an ML(N)LP and solves it with a standard solver (line 2). If no solution to the ML(N)LP exists, Kongming keeps expanding the graph one level at a time, until a solution is found (line 3-5). Unlike Graphplan, Kongming is not guaranteed to terminate if no valid plan exists, because there is no level-off in the presence of an infinite number of possible continuous action nodes and continuous fact nodes. This is not a major issue for the range of hybrid problems we are interested in, because it is rare that no valid plan exists.

If the solver outputs a solution, it is the optimal plan for the current k -stage Hybrid Flow Graph. Hence, it is a local optimum, rather than the global optimum. When the objective function is unrelated to the number of levels in the graph, the following statement is true: if the goal condition is contained in the Fact Level k and s^* is an optimal solution of the corresponding ML(N)LP, then there exists an optimal solution for every expansion of the graph after level k and the optimal solution is at least as good as s^* . This is because each (fact or action) level is a superset of its previous level, and mutex in each (fact or action) level is a subset of its previous level. Therefore, there are more choices in

terms of solving the ML(N)LP for a graph with more levels. Hence, for a minimization problem, Kongming keeps expanding the graph one level by one level until the optimal value no longer improves (line 7-10), and that optimal value is the global optimum.

Kongming()

1. ExpandGraph()
2. encode as ML(N)LP and solve
3. *while* no solution *do*
4. expand graph by one level
5. encode as ML(N)LP and solve
6. incumbent \leftarrow +infinity
7. *while* current optimal value < incumbent *do*
8. incumbent \leftarrow current optimal value
9. expand graph by one level
10. current optimal value \leftarrow encode as ML(N)LP and solve

Experimental Results

Kongming is implemented in Java, and tested on a variety of AUV mission and Unmanned Air Vehicle (UAV) fire-fighting examples involving linear dynamics. We use the ILOG CPLEX solver to solve the MLLPs.

For example, one test problem we use is as follows. In a science mission we need an AUV to go to two algal bloom regions to perform a survey. The mission requires the AUV to go up and down (a yoyo motion) inside the two regions to take samples. Going up or down requires the up or down rudder to be set within the desired range of depth. Taking samples requires a gulper, which "gulps" samples, to be set and the AUV to be inside the algal bloom regions while at the desired depth. The initial condition is $\{(x, y, z) \in [0, 2]^3, \neg sampleA, \neg sampleB, \neg gulper, \neg upRudder, \neg downRudder\}$, and the goal condition is, $\{(x, y, z) \in ([80, 82], [0, 2], [0, 2]), sampleA, sampleB\}$. The continuous actions are glide, descend and ascend, and the discrete actions are setGulper, takeSample, setUpRudder, setDownRudder. The objective function is to minimize the average depth z over time, in order to keep on the surface as long as possible, so that it achieves maximum GPS coverage. Kongming found the optimal solution in 13.1 seconds, from a graph with 14 levels. The left side of Fig. 11 shows the trajectory, i.e. the solution to the state variables, and the right side shows the corresponding action sequence. Note that the output also includes the optimal solution to the control input variables, in this example, the x, y, z velocity for each continuous action. In Fig. 11 along the optimal trajectory, due to the fixed duration assumption and the velocity limits, we cannot reach from point 6 to 8 in one time step, so there has to be a point 7 in the middle. Since in this case the cost is only on the z -axis, point 7 is required to be on the surface but not necessarily in a straight line with 6 and 8 in an optimal trajectory. Same with point 12.

To test the scaling of Kongming, we tested it on 48 problems with different numbers of discrete and continuous actions. The average computation time used to expand the Hybrid Flow Graph and to encode and solve the MLLPs is recorded in Fig. 12 and Fig. 13.

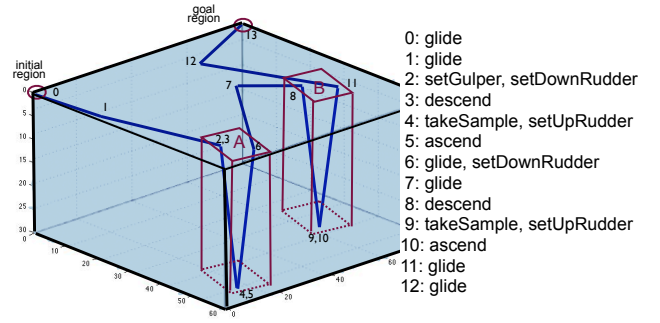


Figure 11: On the left: the optimal trajectory of the example problem. On the right: its corresponding action sequence. The numbers along the trajectory mark the fact level in the graph. The numbers next to the actions mark the action level in the graph.

# discrete actions	2	4	8	16
Expand Graph	348	519	1825	3035
Encode & Solve MLLP	672	749	3915	4936

Figure 12: Computation time in milliseconds for different number of discrete actions. Meanwhile, continuous actions and continuous initial/goal conditions are kept the same.

Fig. 12 shows that scaling in terms of discrete actions is efficient. This is expected with respect to the time used to expand the graph, because it should follow the complexity of Graphplan, which is polynomial in the number of discrete actions. Comparing Fig. 13 to Fig. 12, we notice that scaling in terms of continuous actions in Kongming is more difficult than discrete actions. This is because while there can only be one instance of a discrete action in any action level and there are a finite number of discrete facts in a plan, there can be multiple flow tubes of a continuous action in any action level and there are potentially an infinite number of continuous facts. The good news is that in real-world problems, there are only a small number of continuous actions involved, because different continuous actions correspond to different dynamics of the robot. For example, an AUV engages 3 continuous actions: glide, descend and ascend. The scaling issue will be addressed further in our future work.

# continuous actions	2	4	8	16
Expand Graph	350	1702	12376	>10 ⁷
Encode & Solve MLLP	859	1663	3184	>10 ⁵

Figure 13: Computation time in milliseconds for different number of continuous actions. Meanwhile, discrete actions and discrete initial/goal conditions are kept the same.

Conclusion & Future Work

This paper provides a novel solution to the generative planning problem for hybrid systems, where both continuous and discrete actions are involved. The planner, Kongming, has the following key innovations. First, a compact representation of all hybrid plans, called a Hybrid Flow Graph, which combines the strengths of a Planning Graph for discrete actions and Flow Tubes for continuous actions. Second, a constraint encoding of the Hybrid Flow Graph in terms of an ML(N)LP that is then solved by an off-the-shelf solver.

Currently we are working on removing the assumption that actions have fixed and equal duration. It is not difficult to extend it to varying but fixed duration, in which case different actions have different duration, but for each action the duration is fixed. We can apply existing temporal planning methods, like LPGP or TGP, and keep most of our algorithm unchanged. Extending it to flexible duration is more challenging, because the time space is infinite and the infinity is propagated into the state space. Future work will also include extending the input from a goal condition to a set of goals with temporal constraints among them. As for real-world demonstration, we are collaborating with the AUV Lab at MIT Sea Grant, who has several different types of AUVs for sea deployment. Because the AUVs are controlled by low-level controllers that take as input a sequence of behaviors (i.e. actions) and way points, which are what Kongming outputs, there is a natural interface between our planner and the controllers.

Acknowledgments

This research is funded by The Boeing Company under grant MIT-BA-GTA-1.

References

- Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence*.
- Bradley, E., and Zhao, F. 1993. Phase-space control system design. *Control Systems* 13(2).
- Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of ECP*.
- Frank, J., and Jónsson, A. 2003. Constraint-based attributes and interval planning. *Journal of Constraints*.
- Gerevini, A.; Serina, I.; Saetti, A.; and Spinoni, S. 2003. Local search for temporal planning in lpg. In *Proceedings of ICAPS*.
- Hofmann, A., and Williams, B. 2006. Robust execution of temporally flexible plans for bipedal walking devices. In *Proceedings of ICAPS*.
- Joslin, D. 1996. Passive and active decision postponement in plan generation. *Ph.D. thesis, Carnegie Mellon University Computer Science Department*.
- Kautz, H., and Selman, B. 1997. Unifying sat-based and graph-based planning. In *Proceedings of ICAPS*.
- Kurzanskiy, A. 2006. Ellipsoidal toolbox: <http://www.eecs.berkeley.edu/~akurzhan/ellipsoids/>.
- Long, D., and Fox, M. 2002. Fast temporal planning in a graphplan framework. In *Proceedings of ICAPS*.
- McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2007. T-rex: A model-based architecture for auv control. In *Proceedings of ICAPS Workshop*.
- Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1-2):5-48.
- Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *Proceedings of AAAI*.
- Shin, J., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artificial Intelligence* 166.
- Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proceedings of IJCAI*.
- Vere, S. 1983. Planning in time: Windows and durations for activities and goals. *Pattern Matching and Machine Intelligence* 5.
- Wolfman, S., and Weld, D. 1999. The lpsat engine and its application to resource planning. In *Proceedings of IJCAI*.