Toward Real Time Autonomous Robotics Through Integration of Hierarchical Goal Network Planning and Low Level Control Libraries

Nicholas Fung

November 29, 2017

Abstract

Automated planning has become an increasingly influential area of research in the realm of artificial intelligence. Task based planning algorithms provide a number of advantages including the ease of human readability when creating mission length plans. However, task based planning algorithms are rarely implement on real world robotic systems because they require additional domain specific knowledge to define tasks and are generally not as flexible as other planning techniques.

This paper documents work to integrate a hierarchical goal network planning algorithm with low level path planning. The system utilizes the Goal Decomposition with Landmarks (GoDeL) planner for plan generation at an abstract level and the Searched Based Planning Library (SBPL) for low level control. The system is used to direct a robot through an office setting within a simulation environment. We then discuss incorporating an "in the now" approach to the GoDeL algorithm to make the system more robust to a dynamic environment. The resulting algorithm is more suited for use in real time applications such as autonomous robotics.

1 Introduction

Automated planning is a powerful tool in the field of artificial intelligence that generates a sequence of actions in order to accomplish a specific goal. Planning techniques are often used when creating intelligent systems and are of particular interest for the implementation of autonomous robotic platforms that can utilize set action patterns in a specific order in order to accomplish complex tasks. Within the field of autonomous planning, task planning has a number of advantages including the ability to form long, human readable plans. However, they have largely remained in the area of theoretical because of the challenges encountered when integrating into a complex, autonomous system. Ghallab, et al. [2] [3] show that tight coupling between planning and acting is necessary in order to bring such planning algorithms into practical application.

We utilize the Goal Decomposition with Landmarks (GoDeL) algorithm, created by Shivashankar, et al. [9] to perform mission planning through goal decomposition. We interface this symbolic mission planning algorithm with a navigation system that utilizes the Search-Based Planning Library (SBPL) [6] to interface with the motor controllers of an autonomous robotic platform. The resulting system is able to create symbolic mission plans that are readable by a human operator, but is also able to control the robotic platform without occupying the user. We look into the shortcomings of utilizing this system in a dynamic environment and develop an approach to mitigate these risks. An "in the now" approach to planning and acting is integrated into the GoDeL algorithm. In this system partial plans are executed before a complete plan is created. The goal of this approach is to make efficient use of time in a dynamic domain that will likely require alterations to the initial plan. The algorithm will also be more resilient to any disruptions in execution of the plan.

2 Background

2.1 The Planning Problem

We use formalisation of the planning problem from Ghallab, et al. [1]. A planning domain D consists of a finite sets of states S, a finite set of operators O, and a state transition function γ. A state $s \in S$ quantifies the current configuration of the system. An operator is defined as a triple $o = (head(o), precond(o), effects(o))$. $head(o)$ contains the name and argument list of o. $precond(o)$ are the ground atoms that the state must satisfy for the operator to be executed. $effects(o)$ are the ground atoms that become true through the execution of the operator. A ground instance of an operator is known as an action. The state transition function γ defines the change in the system that occurs upon execution of an operator.

A planning problem P consists of the triple (D, s_0, gn) where D is the planning domain, s_0 is the initial state of the system, and gn is a goal network consisting of a pair $(T, \prec)$. T is a finite nonempty set of nodes, each containing a goal g_t in disjunctive normal form, and $\prec$ is an ordering over T.

A solution plan π under goal network gn is a sequence of actions $\langle a_1, ..., a_n \rangle$ such that each action a_i is executable on s_{i-1} (where s_{i-1} is the state attained after execution of action a_{i-1}) and each subgoal in gn is satisfied within the sequence of states subject to $\prec$. A planning algorithm is *sound* if it returns a plan π; it is a valid plan that solves the intended planning problem. A planning algorithm is *complete* if it returns a valid plan π for the intended planning problem if such a plan exists.

2.2 Hierarchical Planning

A hierarchical task network (HTN) is a structure used to create plans to accomplish specific goals. The graph structure is created by breaking down *tasks* into subtasks using *actions* and *methods* that are defined within the domain. A sample HTN in the Blocks World domain can be seen in Figure 1. A *task* defines an advancement of the system state to attain a specific subsequent state and is of the form *task(literal1,literal2,...)* for any number of literals. For example, the task *pick-up(apple,kitchen_table,right_hand)* and represents the act of picking up an apple from the kitchen table with your right hand. A *primitive task* is a task that can be accomplished by a single operation. An *action* is an operator that accomplishes a *primitive task*. A *complex task* is a task that requires more than one operation. A *method* is similar in application to an action, but is applied to a complex task. The method breaks down the complex task into multiple subtasks, and requires the subsequent application of multiple actions or methods. Complex tasks can continually be broken down through the application of methods until only primitive tasks remain. These primitive tasks are then accomplished through application of a sequence of actions, defining a *plan* that satisfies the goal of the planning problem. Because each task and each literal can be named according to the user defined domain documentation, the plan can easily be interpreted by a human reader. Similarly, a system state is equally readable, making it easy for a human to issue goals to the planner.

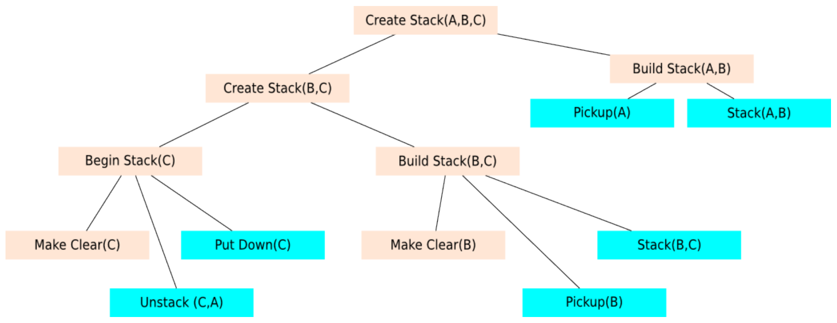

Figure 1: HTN of a Sample Problem in the Blocks World Domain

Under an HTN planning problem, we make a distinction between actions and methods. An action $a = (name(a), precond(a), effects(a))$ is a ground primitive operator that can accomplish a primitive task. An action a is applicable to a task t if $name(a) = t$ and the current state $s \models precond(a)$. Execution of a produces the resulting state $\gamma(s, a) = (s - effects^{-}(a)) \cup effects^{+}(a)$. A method m imparts knowledge to the planner in order to plan for a series of actions and is defined as $m = (name(m), task(m), subtasks(m), constr(m))$. The $name(m)$ is of the form $n(x_1, ..., x_k)$ where n is a unique method symbol and $(x_1, ..., x_k)$ are all of the variable symbols that occur in m. The $task(m)$ is a nonprimitive task to which the method can be applied. The $subtasks(m)$ are a set of tasks that are constrained by $constr(m)$ that, when accomplished, will satisfy $task(m)$.

Hierarchical goal networks (HGNs) [10] differ from HTNs through the use of subgoals rather than subtasks. While HTN planners break down tasks into subtasks and actions, an HGN breaks down goals into subgoals. This structure requires different definitions of actions and methods. For HTN planners, actions are applied to complete primitive tasks and methods are applied to decompose complex tasks into subtasks. In HGN planners, actions achieve a specific goal and methods are applied to decompose a specific goal into multiple subgoals. The planner iteratively plans on each goal in the graph structure. Each subgoal is assigned an action to achieve this subgoal or is decomposed into more subgoals. When each subgoal can be accomplished through the execution of a single action, this sequence of actions is a valid plan to attain the overall goal.

An HGN method m can be applied as other operators according to $precond(m)$. Each method contains a sequence of subgoals $sub(m) = \langle g_1, ..., g_k \rangle$ and a post-condition $post(m) = g_k$ rather than an $effects$ function as would be contained in an action. Although a method shares similar structure to an operator, it is not directly executed upon the system, but instead is used in the planning process to produce subgoals. The planner creates plans to achieve each subgoal through application of actions.

One of the difficulties in applying HTN planners to complex domains found in real world scenarios is that the domain must be explicitly defined. Tasks, actions, and methods must be defined with preconditions and expected outcomes so that the planner can determine which operations can be applied in a specfic state and what the outcome of the operation will produce. This work may require a significant amount of a priori work by the user. HGN planners can minimize this work through the use of more loosely defined methods that are more similarly defined to operators with pre and post conditions. This allows for less effort by the user in constructing HGN methods as opposed to HTN methods that require additional construction of tasks. In addition, this planning structure affords the planner more freedom to apply low level operators to attain subgoals rather than being tied to specific subtasks. Shivashankar, et al. show that their Goal Decomposition Planner (GDP) [10] is both sound and complete.

One of the primary difficulties in integrating an HTN or HGN based planner with a low level acting agent on a robotic platform is in translating the actions generated by the planner. Because tasks and how they are decomposed into action, methods, and subtasks are user defined specifically for a particular domain, they are typically complex concepts. For example, an action may be "pick up box A". While this instruction is easily understandable by a human, this requires numerous motor control instructions for a robot to execute. If instead actions are defined at the motor control level, the action definitions would too complex for a human author. This paper documents efforts to integrate an HGN planner with a navigation system on a robotic platform within a simulation environment. The HGN planner is used at a high level to form an overall mission plan. The navigation system uses a low level search based action planner in order to form the motor control signals.

3 Planning and Acting System

3.1 GoDeL Planner

The GoDeL planner, is an HGN planning algorithm that utilizes landmarks to break down goals into subgoals. GoDeL operates as an HGN planner as described in section 2.2 with additional rules to aid the planner under sparse domain information. As in generalized HGN planners, the algorithm assigns actions to achieve goals and breaks down goals into subgoals using methods when applicable. However, when a goal cannot be attained directly through an action or decomposed by a method, GoDeL will work to decompose the goal using landmarks. A landmark is a specific state that must be satisfied in any valid plan to attain a specific goal. For example, any plan to put an object down must at some point contain the state of holding the object. GoDeL uses the LAMA [8] algorithm to identify landmarks for a given subgoal. If the algorithm encounters a goal that does not have an a relevant action or method and cannot be decomposed into landmarks, the state space is searched through nondeterministic action chaining. Through this progression, GoDeL utilizes whatever information it has to limit the search space and run more efficiently. However, with limited access to useful methods, the algorithm will still search through the entire search space and will find a valid solution plan if it exists.

3.2 Search-Based Planning Library

The GoDeL mission planning system is paired with a navigation system through a waypoint instruction interface. The waypoint navigation system works to provide semi-autonomous capabilities to a robotic platform. The system takes in a goal orientation, known as a waypoint, consisting of a map coordinate and pose. A path to this goal orientation, consisting of a series of poses, is then generated. The robot attempts to follow the path as closely as possible in order to obtain the goal pose.

The path is obtained through the use of the SBPL. This library contains a generic set of domain independent graph search planners, meant to be applicable to a variety of systems. This waypoint navigation implementation makes use of the ARA* any time planner. The system takes in a goal location and orientation and uses the SBPL to generate a series of motor control instructions to the robot in order to follow the path and obtain the goal.

3.3 Test Bed Implementation

The integration of the planner and navigation system was implemented under the Robot Operating System (ROS) software architecture. ROS is an open source project that is widely utilized on robotic platforms to provide a flexible architecture to integrate code for autonomous platforms. We use the integrated planner and actor code to operate a robotic platform within a ROS simulated environment. The simulated robot, modeled after the iRobot Packbot platform,

Algorithm 1 GoDeL Algorithm

```
1:  **Procedure** $GoDeL(D, s, gn, M, \pi)$
2:  **if** $gn$ is empty **then**
3:      **return** $\pi$
4:  **end if**
5:  nondeterministically choose a goal formula $g$ in $gn$ without any predecessors
6:  **if** $s \models g$ **then**
7:      **return** $GoDeL(D, s, gn - \{g\}, M, \pi)$
8:  **end if**
9:  $\mathcal{U} \leftarrow$ {operator and method instances applicable to $s$ and relevant to $g$}
10: $\mathcal{U} \leftarrow \mathcal{U} -$ used_methods[$s$]
11: **while** $\mathcal{U}$ is not empty **do**
12:     nondeterministically remove a $u$ from $\mathcal{U}$
13:     **if** $u$ is an action **then**
14:         res1 $\leftarrow GoDeL(D, \gamma(s, u), gn, M, \pi \circ u)$
15:     **else**
16:         add $u$ to used_methods[$s$]
17:         set subgoals_inferred to false
18:         res1 $\leftarrow GoDeL(D, s, subgoals(u) \circ gn, M, \pi)$
19:     **end if**
20: **end while**
21: **if** subgoals_inferred $\neq$ true **then**
22:     subgoals_inferred $\leftarrow$ true
23:     lm $\leftarrow Infer - Subgoals(D, s, g)$
24:     **if** lm $\neq \emptyset$ **then**
25:         res2 $\leftarrow GoDeL(D, s,$lm$\circ gn, M, \pi)$
26:         **if** res2 $\neq$ failure **then**
27:             **return** res2
28:         **end if**
29:     **end if**
30: **end if**
31: $\mathcal{A} \leftarrow$ {operator instances applicable to $s$}
32: **if** $\mathcal{A} = \emptyset$ **then**
33:     **return** failure
34: **end if**
35: nondeterministically choose an $a \in \mathcal{A}$
36: **return** $GoDeL(D, \gamma(s, a), gn, M, \pi \circ a)$
```

navigates its way through a known area in order to attain the specified mission goals.

The system architecture can be seen in figure 2. The ROS code base provides protocols to pass data between nodes. We utilize the existing communications and integrate the GoDeL code into a ROS node that we call the GoDeL Node. The GoDeL code produces a symbolic action plan that is transferred to the GoDeL to Navigation Node. This node translates the symbolic plan into navigations calls that can be interpreted by the Path Planning Agent. The Path Planning Agent is code that interfaces with the SBPL. This code was authored by the US Army Research Laboratory as part of its efforts to research autonomous behaviors to be implemented on Army robotic platforms. The Path Planning Agent then interfaces with the robotic platform to provide motor control and move the robot according to the plan provided by the Path Planner.

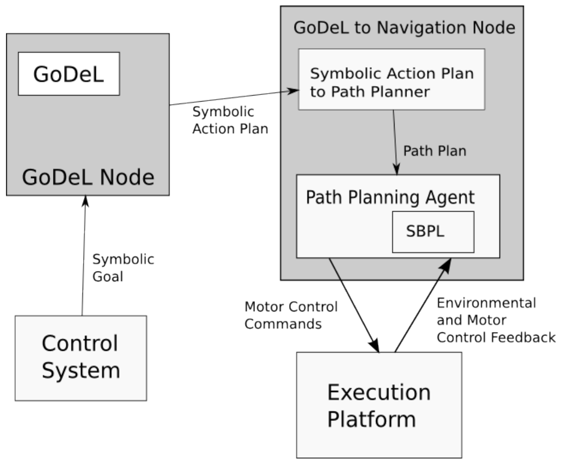

Figure 2: System Architecture: Integrating GoDeL with the SBPL

In the simulated scenario, a robot is tasked to move between areas of the environment. Practical applications could be to patrol an area of the building while continuously taking sensor measurements. Doors that may be open or closed are incorporated within the scenario to increase complexity in addition to movement actions. While the waypoint navigation planner can create paths between open rooms, closed doors may block a valid path from being created. In order to create a complete plan, the simulated robot must make use of an additional action to open doors. Although the system does not include motor control functions for the Packbot to open doors, the simulated robot was given this ability to increase the complexity of the problem. This could have a real world analogy of using a remote control to open an electronic door or signaling a person to open the door for the robot.

The test domain is specifically kept to a small number of actions because this system is designed as a proof of concept. The available actions are *move(robot,location1,location2)* and *open(robot,door)*. A method *traverse(robot,location1,location2)* is also included in the domain and is designed to address the movement of the robot across several rooms with closed doors. The method is applied to a location goal (e.g. *robot-at(robot,location)*) and breaks the goal down into subgoals of the robot at intermediate rooms and open doors between these rooms. GoDeL uses these actions and methods to create an action plan consisting of *move* and *open* commands. This queue of action commands is interpreted by C++ code contained in the Navigation Node. The behaviors of this node is tailored by the user for this specific domain. It translates *move* actions into waypoints to be sent to the waypoint navigation system. The node executes *open* actions by directly manipulating the environment, clearing the path between two rooms, simulating the act of opening a door.

In this manner, the system demonstrates additional capabilities that are added to the waypoint navigation planner. The HGN planner is able to create *move* and *open* commands to establish a valid plan. These commands are applied through the system into autonomous selection of waypoints and door opening commands. The HGN planner is also extended in its ability to carry out actions. While a command to *move(robot1,location1)* is abstract in the literal sense, the system is able to translate this into a waypoint. It can then use the waypoint navigation planner to provide motor control actions to the robot.

4 Simulation

Initial State:	Plan:
at(robot1,loc1)	1:move(robot1,loc1,loc2)
closed(door1)	2:open-door(robot1,door1,loc2,loc3)
	3:move(robot1,loc2,loc3)
Goal State:	4:move(robot1,loc3,loc4)
at(robot1,loc5)	5:move(robot1,loc4,loc5)

Table 1: Sample Plan From Simulation

We implemented the planning and acting system within a simulation environment to operate a robot within a known environment. The GoDeL Node is able to create valid plans consisting of *move* and *open* actions and utilize the *traverse* method to guide the action search. Table 1 depicts a sample run generated by the GoDeL Node. The planner utilized a *traverse* method during formulation to guide the search and generate the action plan. The system then translates the actions into instructions and utilizes the SBPL to derive motor control commands for each action. A screen shot of the simulation visualized in RVIZ can be seen in Figure 3. This figure demonstrates the robot moving along a path created by the navigation system to a waypoint selected by the HGN planner. This system successfully provides a proof of concept that justifies continued work in this direction.

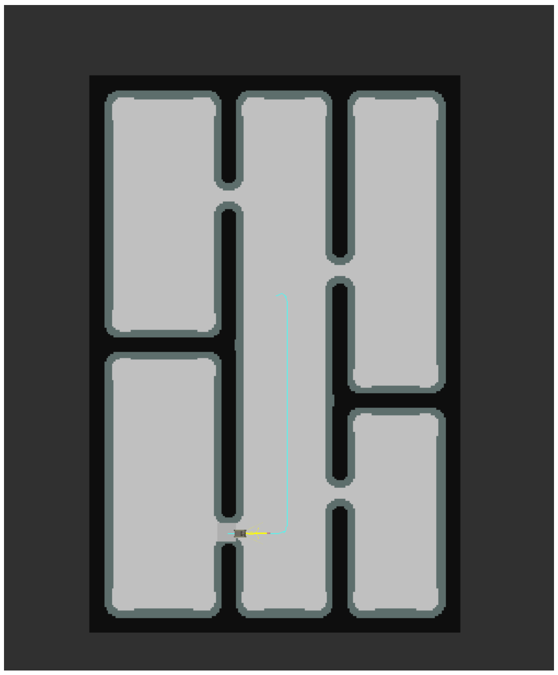

Figure 3: Screen Capture of Simulation in Progress

5 Planning in the Now

Planning in the now, as in the Hierarchical Planning in the Now (HPN) algorithm from Kaelbling [4], provides more flexible planning in a dynamic environment by avoiding commitments to actions in the long horizon. Instead, it uses a depth first approach to refine a hierarchical plan into ground actions only when they are needed by the system. The system is tightly coupled with an acting module so that actions are executed as they are generated. In the case described in section 3.3, this acting agent is the Path Planning agent that is tied to the SBPL. The primary advantage to this approach is to allow for the planner to adjust to unexpected outcomes from actions or dynamic events in the environment before a complete plan is generated. This can improve efficiency of a planning and acting system as computing power will not be consumed in generating portions of the plan that are likely to be discarded. Levihn, et al. [5] further the algorithm by taking advantage of reconsideration and foresight. Reconsideration is used to optimize the plan by initiating a local re-plan when a new, unexpected opportunity presents itself to increase efficiency. Foresight is used to exploit the structure of abstract methods to minimize interference of the immediate plan with future actions.

HPN is similar to other refinement algorithms such as GoDeL and Simple Hierarchical Ordered Planner 2 (SHOP2) [7] in its hierarchical structure. The

Algorithm 2 HPN Algorithm

```
1: Procedure $HPN(s_{now}, \gamma, \alpha, world)$:
2: $p = Plan(s_{now}, \gamma, \alpha)$
3: for each $(\omega_i, g_i)$ in $p$ do
4:     if $isPrimitive(\omega_i)$ then
5:         $world.execute(\omega_i, s_{now})$
6:     else
7:         $HPN(s_{now}, g_i, nextLevel(\alpha, \omega_i), world)$
8:     end if
9: end for
```

distinctive characteristic of HPN is that when the refinement is able to identify a primitive action as the next operator, the action is immediately executed. The algorithm then considers feedback from the acting agent in order to adjust its expected state and react to the actual state. Because subsequent steps of the plan are still at an abstract level, further refinements are created based off of the actual world state rather than the expected world state.

The *Plan* procedure takes in the current world state s_{now}, a goal γ, an abstraction level α, and the world model. For each action ω_i in the plan, the algorithm checks whether it is a primitive action or an abstract action. In the case of a primitive action, it is executed by the agent. Otherwise, HPN is recursively called to continue refining the plan.

The drawback of this approach is that, in general, greedy approaches to planning will not result in completeness. That is, the system cannot guarantee execution to complete all subgoals. In particular, non-serializable subgoals can cause inefficiencies or even failure. The Sussman anomaly [12] presents the case within the Blocks World domain wherein three blocks (A, B, C) are initially placed with $on\_table(A)$, $on\_table(B)$, and $on(C, A)$. When planning to achieve $on(A, B) \wedge on(B, C)$, a greedy algorithm will apply $unstack(C)$ followed by $move(A, B)$ which will achieve $on(A, B)$ but can then no longer achieve $on(B, C)$ without moving A. Alternatively, it could apply $move(B, C)$ to achieve the subgoal $on(B, C)$ but could then no longer proceed without invalidating this subgoal. HPN handles this case, albeit without optimal execution. It first resolves the $on(A, B)$ subgoal as above, by executing $(unstack(C)$ followed by $move(A, B)$. But the algorithm is then able to resolve $on(B, C)$ by moving A on to the table, B on to C, and finally A on to B. Specifically, HPN uses a depth first search and has the ability to form plan of any finite length necessary to satisfy subgoals at each step. Without look ahead, it can create inefficient plans by satisfying subgoals that must then be undone. However, as long as the state can be backtracked, HPN can search through actions and still complete the plan.

HPN is specifically meant to operate in domains where all actions are reversible. In the above example, we see that the system is able to solve the Susssman anomaly because it can reverse the achievement of $on(A, B)$. If, on the other hand, the domain includes irreversible actions, the system could enter a failure state even if a valid plan existed in the initial state. Consider in the above example if the end goal requires that the blocks be glued into position. When A is moved on to B as the first action, A is permanently glued on to B. If this stack cannot then be placed on to C, the system has entered a failure state. Because HPN executes actions as they are established through the refinement process, it can enter these failure states. Conversely, algorithms that form a complete plan before execution will avoid failure states because it is looking ahead and formulating to reach the final goal.

6 GoDeL Planning in the Now

We have shown that the GoDeL algorithm can be integrated with a motion planning agent in order to produce actionable plans for an autonomous robot agent. We now look to adapt the GoDeL algorithm to include the adaptability of HPN execution to increase its suitability for operation in a dynamic environment. We call this approach GoDeL Planning in the Now (GPN). The algorithm is similar in operation to GoDeL with the characteristics of planning in the now. The primary characteristic integrated into the algorithm is that when an abstract action is refined into a ground action, it is immediately executed before continuing with further goal decomposition.

By executing actions as they are established, we take advantage of several characteristics. First, the system can more easily react to changes in the system state that are not modeled by the transition function. As an example of the environment changing, consider a car parked along the road that blocks a passageway that was previously clear for traversal. Alternatively, consider unexpected outcomes from executing an action such as attempting to pick up an object and dropping it down a flight of stairs. Because the planner is not creating a complete plan, it can adjust and construct the next portion of the plan based off of the actual state rather than the state expected from the transition function. That is, plan repair and replanning in response to a broken plan will be more localized. Additionally, the planner is able to take advantage of beneficial, unexpected outcomes. If an action produces a result that unexpectedly satisfies additional subgoals and a complete plan has already been formulated, the agent may end up performing redundant actions to try and achieve these subgoals that have already been satisfied. Additionally, the planner can work to make on the spot optimizations. As an example, the planner can take advantage of the fact that a totally ordered plan has not already been completely formulated in order to reorder subgoals into a more optimal ordering. Causal links within the HGN structure will limited scope because much of the future plan is still abstracted, allowing for more freedom to alter or reorder subgoals. Another advantage is that the HGN methods used in a GoDeL inspired decomposition are well suited to provide domain knowledge to the refinement search. This may become very important in addressing the problems that arise from planning in the now.

An important shortcoming that is inherent in the planning in the now approach is that, as discussed in section 5, the planner cannot satisfy completeness. That is, the planner can lead the agent into a failure state from which it cannot recover, even if a valid plan could have been formulated from the initial state. In particular, systems that contain actions that are irreversible can lead to these failure states. As an example, if the mission requires that a robot lock the door to a room but the robot is incapable of unlocking the door, the mission could fail if the door is locked before the robot has completely searched it.

7 Future Work

7.1 Enforce Serializability in GPN

As discussed previously, planning in the now lacks completeness. In future work, we look to develop the algorithm and address this shortcoming by limited compromising of the 'in the now' technique. That is, we allow for some limited look ahead searching before execution of actions in order to avoid entering failure states. This poses a large hurdle, as a thorough look ahead search minimizes the advantages of planning in the now detailed above. To avoid failure states without constructing a complete plan, the system needs to look sufficiently ahead so that it can avoid irreversible actions that result in failure. In this case, the subgoals were not in a serialized ordering. In future work, we must endeavor to maintain serializable ordering of subgoals as they are refined into a ground state. We can alter the depth first search and refinement so that it looks far enough ahead in the planning process to ensure a serialized ordering of goals. How far ahead we need to search can be a difficult question to answer. Using user defined domain knowledge (HGN methods), we can attempt to establish enough detail in order to avoid execution of actions that result in failure states. That is, if the user defined HGN methods contain enough knowledge to enforce serializability in its subgoals, the algorithm will avoid failure states.

7.2 Integrate GPN with Simulation

We also look to integrate the GPN algorithm discussed in section 6 into the simulation environment with the SBPL navigation library discussed in section 3.3. With this simulation environment, we can further investigate the effectiveness of the algorithm when implemented on a robotic platform. Planning in the now is designed to be suited for use within a dynamic environment. This advantage would be verified through experimentation in the simulated environment.

Additionally, the simulation environment would provide the ability to experiment with a dynamically shifting domain in a controlled environment. A domain that can change states unexpectedly increases the complexity of the planning problem. For example, a person could close a door or a hallway could become blocked. The simulation can provide the opportunity to test the algorithm and provide experience in constructing applicable domain knowledge.

Finally, integration of the GPN algorithm with the SBPL navigation library can lead to further capabilities through integration with other robot behavior libraries. As an example, the MoveIt! library [11] provides a number of robot behaviors include complex manipulator control. Using MoveIt! capabilities, the GPN planner could be use to provide additional actions to our test scenario including the ability to pick up objects and move them to other rooms.

7.3 Dynamic Domain and Goals

As an additional area of research, a complete planning system for a dynamic system can include dynamic domain redefinitions. This capability is especially important for operation of the system with incomplete knowledge of the domain. For example, a robotic system working to explore an unknown building may find an unexpected door. A robust system would be able to add this door to the domain and invoke a replanning procedure. This may result in a more efficient plan to accomplish the goal state.

Similarly, the system could have cause to dynamically alter its goals. In the context of a search and rescue mission after a natural disaster, a robot could encounter an injured person. The system would have need to dynamically add a goal in order to rescue this person or call for aid with a high priority. A robust planning system would be able to add this goal and invoke a replanning procedure that would look to achieve this high priority goal as quickly as possible before continuing with the original plan.

References

[1] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[2] Malik Ghallab, Dana Nau, and Paolo Traverso. The actors view of automated planning and acting: A position paper. *Artificial Intelligence*, 208(Supplement C):1 – 17, 2014.

[3] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[4] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477. IEEE, 2011.

[5] Martin Levihn, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Mike Stilman. Foresight and reconsideration in hierarchical planning and execution. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 224–231. IEEE, 2013.

[6] Maxim Likhachev. Search-based planning library. [Online] Available: http://wiki.ros.org/sbpl.

[7] Dana S Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *J. Artif. Intell. Res.(JAIR)*, 20:379–404, 2003.

[8] Silvia Richter and Matthias Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

[9] Vikas Shivashankar, Ron Alford, Ugur Kuter, and Dana Nau. The godel planning system: a more perfect union of domain-independent and hierarchical planning. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2380–2386. AAAI Press, 2013.

[10] Vikas Shivashankar, Ugur Kuter, Dana Nau, and Ron Alford. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 981–988. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[11] Ioan A. Suscan and Sachin Chitta. Moveit! [Online] Available: http://moveit.ros.org.

[12] Gerald Jay Sussman. *A computer model of skill acquisition*, volume 1. American Elsevier Publishing Company New York, 1975.