Least Commitment Planning for the Object Scouting Problem - Preliminary Results

Max Merlin[1*], David Paulius[1], George Konidaris[1]

***Abstract*—State uncertainty is one of the primary obstacles to effective long-horizon task planning in robotics. We tackle state uncertainty by decomposing it into spatial uncertainty—resolved by SLAM—and uncertainty about the objects in the environment, formalized as the *object scouting problem* and modelled using the *Locally Observable Markov Decision Process* (LOMDP). We introduce a new planning framework called *Scouting Partial Order Planner* (SPOP), specifically designed for object scouting with LOMDPs, which exploits the characteristics of partial order and regression planning to plan *around* gaps in knowledge the robot may have about the existence, location, and state of relevant objects in its environment. Our preliminary results demonstrate how the aspects of partial order planning make it uniquely suited to solving tasks in the object scouting context: SPOP significantly outperforms alternative planners in this setting.**

I. Introduction

Robots acting in the real world must navigate through space and interact with the objects around them. One complicating factor is the uncertainty that pervades this problem: the robot cannot simply know the state of the its environment, but must instead measure the world with its sensors to estimate that state. For navigation, this is typically resolved by SLAM (*Simultaneous Localization And Mapping*) problem [1, 2], which allows a robot to effectively estimate its position in the world and find a path to a desired reachable location. This removes the element of spatial uncertainty, enabling efficient navigation within an envelope of known space.

By contrast, there has been much less progress on resolving uncertainty for the object manipulation component of mobile manipulation. While this bears some resemblance to the general problem of object search [3, 4], mobile manipulation requires *task-relevant* reasoning: myriad objects may be present in the environment, but a robot should focus on locating and observing those that would progress it towards its goal. We call this problem setting the **object scouting problem**, where a robot must find, localize, and state estimate the objects required to solve its task.

Our recent work has modelled the task-level planning problem required to solve object scouting as a *Locally Observable Markov Decision Process* (LOMDP) [5]. Objects in LOMDPs are locally observable: no information is gained about objects outside of sensor range and line of sight, but the properties of those within can be effectively sensed until they are fully observed. Additionally, manipulating an object requires it to be observed. The result is a model in which the robot actively constructs an envelope of known objects, within which it can use an efficient Markov task-level planner. When a task cannot be solved using observed objects, the critical concern is to then decide which type of object to find and observe next. Merlin et al. [5] showed that even the simplest such planner, which repeatedly replans whenever a new object is observed and randomly selects the object to search for, could solve reasonably large tasks. That naive approach, while effective, leaves plenty of room for improvement.

We introduce a new LOMDP planner called the **Scouting Partial Order Planner** (**SPOP**) that exploits the synergy between least commitment planning [6] and the LOMDP model to generate partial plans that become progressively more complete as the robot gathers more information about its environment. In particular, we exploit the freedom offered by its action ordering system to create the initial partial plan based only on the currently observed objects, which will include gaps that must be resolved to obtain a complete plan. The gaps in these partial plans provide valuable information about which missing objects would help complete the plan. Once the relevant objects are found, a subplan utilizing the new information fills the gap in the plan, obviating the need to constantly replan from scratch. We demonstrate that this form of planning outscales all previous solution methods for the object scouting problem.

II. Background

A. Object Scouting

When a robot is planning to solve a goal-directed task, state uncertainty is undesirable because it imposes enormous computational costs. Nevertheless, state uncertainty is unavoudable in mobile manipulation tasks. This uncertainty can be decomposed into uncertainty to do with navigation—predominantly concerning space, resolved by SLAM—and to do with manipulation—predominantly concerning object state. Just as SLAM resolves uncertainty about the robot's pose, allowing it to effectively navigate through its world using a known map, the *object scouting* problem resolves uncertainty about object existence, location, and attributes, allowing the robot to plan object manipulation and interactions in the context of a "known" state.

When using SLAM, the robot does not typically consider uncertainty over the map: instead, it treat the map as if it was exact and plan with it, and if the robot encounters an error or new regions of map then we update our beliefs and adjust our plan. So too in object scouting, a reasonable strategy is to treat the measurement of objects in the robot's environment as exact and then plan accordingly. New objects or attributes may become observed, prompting an update or refresh of the plan, hence the suitability of LOMDPs. This is especially

[1]Brown University, RI, USA
[*]Corresponding Author (Email: max_merlin@brown.edu)

appropriate when the robot can repeatedly observe an object of interest to drive its estimate of that object's state towards certainty.

Object scouting unifies two families of methods for object sensing: object search and interactive object perception. While object search primarily focuses on locating objects within a robot's environment [3], which may require some degree of manipulation to observe and localize objects in its space [7], object scouting must also reason about the states of said objects to determine whether they can be used to satisfy a task subgoal, considering two differences: 1) it must choose which objects to find based on the task; and 2) it must also resolve their state, not just their pose. For example, let us consider a robot needing to fill a coffee machine with water using a cup. On one hand, object search simply focuses on locating *any* cup that exists in its environment regardless of its state. On the other hand, object scouting must consider not only locating a cup but finding one such that it would satisfy the current subgoal. Specifically, the robot must find a cup that has enough water in it to then pour into a coffee maker; if it does not have enough water, then the robot must further determine how it can acquire a cup with enough water to proceed with coffee making. Locally Observable Markov Decision Processes (LOMDPs) were proposed as a task-level model of the object scouting decision process.

B. Locally Observable Markov Decision Processes

A LOMDP [5] is defined as the tuple:

$$(S, A, O, \Omega, L, R, T, \gamma),$$

extending the definition of Partially Observable Markov Decision Processes (POMDP) [8] by including additional structure. First, the state is factored into n objects: $S = S_r \times S_{o_1} \times ... \times S_{o_n}$. Next, because observations are fully accurate but spatially constrained, the observation space takes the form $\Omega = S_r \times \{S_{o_1} \cup \phi\} \times ... \times \{S_{o_n} \cup \phi\}$. Information gained about a given object is independent of every other object, and for a given object, the observation at a given time step is either its true state value or nothing (denoted as ϕ). Finally, the LOMDP introduces a locality function that describes the set of states that would allow the agent to observe a given object i: $L(i, v) = \{s \in S \mid O(i) = s_{o_i} \wedge (s_{o_i} = v)\}$. The locality function assists in transitioning an object from unobserved to observed, as passing through an object's locality guarantees its becoming observed.

Merlin et al. [5] provided the basic framework for planning using only what is known (i.e., observed) to the robot and seeking to search for new, unknown objects to reinitialize planning again with additional information. Since the robot may not know the exact objects in a domain, it uses the existence of *Skolem objects*, hypothetical objects of each object class that allow the robot to reason about where it might be able to find new objects *if* they existed. When searching for a new object, the LOMDP provides the agent with locales that it can plan to reach in order to observe each Skolem object (if it is at that locale). Each locale is essentially a goal for a subplan that will terminate with observing any objects present at the locale. As an alternative to the locale system, objects can be found using any off the shelf object search algorithm. Recent work has utilized LOMDPs, as they rely upon the property of local observability for their task settings [9, 10].

Aside from the LOMDP framework, similar formulations integrating structure to MDPs have been proposed by related work. Koller and Parr [11] introduced factored MDPs, in which the state is broken down into subsets of state variables (known as factors) that are independent to most other factors and operate under different transition dynamics. Diuk et al. [12] introduced object-oriented MDPs (OO-MDPs), which factors the state space based on objects. Ong et al. [13] proposed mixed observability MDPs (MOMDP), where the state is decomposed into fully observable and partially observable factors. The key distinction between MOMDPs and LOMDPs is that while MOMDPs assume that factors are immutable, where fully observable factors cannot become partially observable and vice-versa, LOMDPs are object oriented, and allow partially observable objects to become fully observable as more information is gathered about the robot or agent's environment.

C. Least Commitment Planning

In the prior lomdp work, they replan from scratch whenever new information is found. However, it would be beneficial to be able to retain some of the plan derived from the known portion of the state for future planning. To that end, we adopt the principle of least commitment planning, which aims to generate a plan by logically decomposing the goal rather than exploring actions in a linear sequence. The output of a least-commitment planner is a plan represented as a tuple $(\mathcal{A}, \mathcal{O}, \mathcal{L})$. $\mathcal{A}$ corresponds to the actions in the plan, $\mathcal{O}$ the ordering constraints between each action a, and $\mathcal{L}$ the causal links between actions. An example $\mathcal{O}$ could be $(a_1 < a_2, a_1 < a_3, a_2 < a_3)$, meaning that action a_1 must come before a_2 and a_3, and action a_2 must come before a_3. One possible link would then be $(a_1 \xrightarrow{p_1} a_3)$, denoting that action a_1 is fulfilling a predicate p_1 for a_3. Plans defined this way may have multiple ways to resolve the ordering constraints. For example, to make coffee, it might not matter if water or coffee grounds are put in the coffee machine first, only that they are both in before the machine is turned on. Either of those plans is valid and consistent with the output of the planner. That is why this method is referred to as least commitment: it does not commit to a single order for its actions, only a set of ordering constraints.

Searching for a plan using this method begins with a goal given as a set of predicates, which are added to the initial agenda of the plan. One of those predicates is popped from the agenda, and the planner attempts to connect it to an action that will make that predicate become true. First it will attempt to link the predicate to an action that is already in the set of planned actions $\mathcal{A}$, including linking to the start state. If there are no valid actions already in the plan that can fulfil a predicate, it will then search all possible actions and add one at random that will resolve the predicate. The ordering $\mathcal{O}$ and

links $\mathcal{L}$ are updated to reflect that this action is fulfilling a predicate and the ordering constraints that come with that. If the newly added action has any preconditions, those must now be added to the agenda to be resolved later. Before recursing to resolve the next predicate in the agenda, a link protection process must be run on the new ordering and links to ensure that all of the actions are causally consistent with each other.

The computation at each node is more expensive than a linear forward or reverse planner due to the infinite nature of the plan-search space [14]. However, partial order planners are well-suited to the object scouting problem, as they naturally handle information gathering processes critical to resolving uncertainty [15] while also integrating with high-level task planning and reasoning that are resolved at run time [16].

III. Planning in the Object Scouting Problem

A. Scouting Partial Order Planner

We developed **SPOP** as an extension to the POP (Partial Order Planner) algorithm introduced by Weld [6] for two main reasons. First, regression planning changes the branching factor when planning from the number of actions possible in a given state to the number of actions that can fulfil a predicate. The object scouting problem is innately high-level: it exists in the context of a robot with high-level manipulation skills and modeling of its environment. When planning at high levels such as this, there are a significant number of predicates that are only resolved by a single action. Take for example making coffee using a coffee machine. There may be only one skill that will fulfil the predicate `machine_on`, one that will fulfil `water_in_machine`, one for `pot_in_machine`, etc. Least commitment planning is uniquely suited to resolve such instances quickly. Second is the nonlinear nature of least commitment planning, which allows the agent to leave gaps in the plan for finding and resolving objects multiple times in a given plan. On top of that, the partial plan that are generated can give us valuable information about what might need to happen in those gaps.

Whenever planning is triggered, a PDDL [17] file is generated from the known portion of the state and current goal. This file is passed into and solved by the **SPOP** algorithm, as shown in Algorithm 1. As with the POP method, a plan is represented as a tuple $(\mathcal{A}, \mathcal{O}, \mathcal{L})$. At each step a predicate is taken from the agenda and matched with an action that will fulfill that predicate, updating $\mathcal{A}, \mathcal{O}, \mathcal{L}$. However, these domains contain *Skolem objects* as introduced by the LOMDP; these are hypothetical objects of each type that represent the possible existence of that object that has not yet been observed. The action space is then populated—at least in part—with actions that rely on Skolem objects as their parameters. These *Skolem actions* are considered separately from fully instantiated actions. As **SPOP** resolves its agenda of predicates, some of those predicates may have no fully instantiated actions that can resolve them due to a lack of domain knowledge. In that case, the planner will check if any Skolem actions could resolve the predicate instead. If a valid Skolem action exists it will *not* add it to the plan because the action is still hypothetical. Instead it will insert a **resolve** action linked to the target predicate, denoting that there is a possible future subplan (or concept of a plan) that can achieve that predicate once more information is gathered.

Algorithm 1: SPOP

```
(A, O, L), agenda
pred, A_need = pop(agenda);
possible_old ← get_satisfying_acts(A);
possible_new ← get_satisfying_acts(all_actions);
possible_skolem ← get_satisfying_acts(skolem_actions);
while not_empty(agenda) do
    if not_empty(possible_old) then
        A_add=pop(possible_old);
        O ← (A_add, A_need);
        L ← (A_add, pred, A_need);
    else if not_empty(possible_new) then
        A_add=pop(possible_new);
        A ← A_add;
        O ← (A_add, A_need);
        L ← (A_add, pred, A_need);
        for precond in get_preconditions(A_add) do
            agenda ← (precond, A_add);
    else if not_empty(possible_skolem) then
        A_skolem=pop(possible_skolem);
        A_resolve = make_resolve_action(pred);
        A ← A_resolve;
        O ← (A_resolve, A_need);
        L ← (A_resolve, pred, A_need);
        for precond in get_preconditions(A_skolem) do
            if Is_Skolem(precond) then
                A_find = make_find_action(precond);
                A ← A_find;
                O ← (A_find, A_resolve);
    else
        return None;
    plan_valid = check_consistency(ordering, links);
    if plan_valid then
        return SPOP((A, O, L), agenda);
    else
        continue;
return (A, O, L), agenda
```

The planner then checks each parameter of the Skolem action to see if it refers to a Skolem object or one that has already been found (at least one parameter must be Skolem since the action is Skolem). If a parameter refers to a Skolem object, **SPOP** it then adds a **find** action for that object type to the list of planned actions. It also updates the ordering constraints $\mathcal{O}$ to reflect that the **find** action must occur before the related **resolve** action. For example, when considering a robot making a sandwich, the action `spread_jelly` has four parameters of types `robot`, `jelly`, `bread`, and `knife` and is able to provide the target predicate `jelly_is_spread`. If the robot has not yet explored the kitchen, it does not have any found objects in the domain of the `jelly`, `bread`, and `knife` types. Those would be represented by skolem objects in the domain. However, there is already an object of type `robot` that could be slotted into the (Skolem) `spread_jelly` action. Therefore, find actions would only be generated for

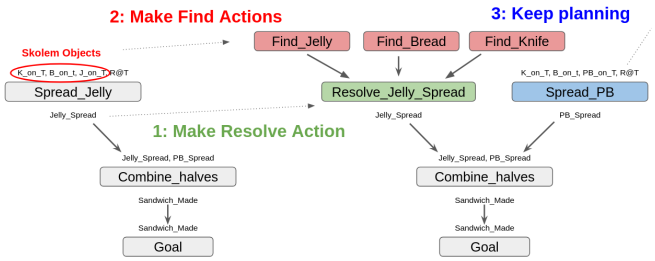

Fig. 1. Generating the resolve and find actions from a desired Skolem action.

`jelly`, `bread`, and `knife` type objects, but not a `robot` type object. This process, shown in Fig. 1, ensures that we will solve the subplan that achieves the target predicate (which triggered the creation of the **resolve** action) after it has found the necessary missing objects and generated actions pertaining to them. To continue the example, we will solve a subplan that achieves `jelly_is_spread` once we have found instances of `jelly`, `bread`, and `knife` objects. The **resolve** action is then linked in $\mathcal{L}$ as resolving the current predicate, and it continues recursing to link the predicates in rest of the agenda. Whenever a Skolem action is added to the plan, it indicates that the task cannot yet be solved until more objects are found and resolved. Therefore, when attempting to resolve each predicate from the agenda, we must sure that Skolem actions are only considered if no existing fully instantiated actions will suffice.

B. Plan Execution

SPOP returns a plan that may consist of any combination of fully instantiated actions, **find** actions, and **resolve** actions. When executing the plan, fully instantiated actions would be performed as normal, but **find** and **resolve** actions will enter their own specialized subroutines involving further planning and interaction with the world. The **find** action behaves no differently than described in prior work [5]; it identifies a locale that could be a potential source of the required object, plans to reach said locale, and then returns if the object is found or loops on to the next locale otherwise.

The **resolve** action is unique to our methodology. **SPOP** generates **resolve** actions when it knows that a certain predicate must be made true, but it lacks the domain knowledge to do so. When the **resolve** action is triggered, it generates a new subplan with the current state as its starting state. The goal of the **resolve** subplan must include the target predicate that originally triggered the creation of the **resolve** action. However, when creating the **resolve** action, the planner cannot know what all of its effects would be, only that one of those effects has to be the predicate that caused its creation. There is a danger that, when planning within the resolve block, there is an effect of an action that threatens an existing causal link in the macro plan. Since we have already executed portions of our plan, we cannot run the link protection process as is standard, since that requires the ability to rearrange the actions that have already been taken. Instead, we run a process called **link reinforcement** (illustrated as Fig. 2). Before executing a

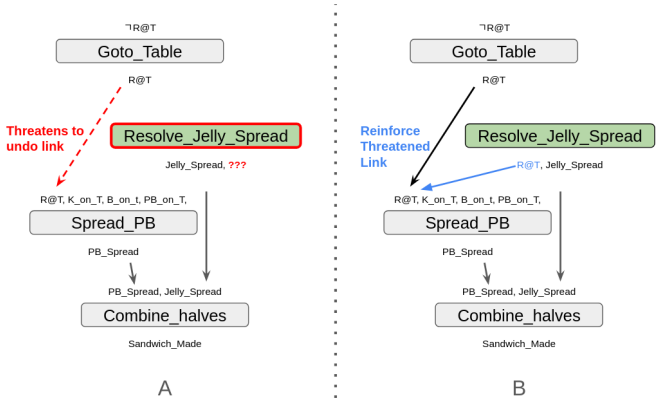

Fig. 2. When executing a resolve action, other causally linked predicates may be threatened. To prevent this, any threatened links are added to the goal of the resolve subplan, reinforcing that they will remain true for the action that needed them.

resolve subplan, we check to see if any causal links pass over the resolve action, i.e., if any predicates are supplied by an action before the resolve and needed by an action after the resolve. Any predicates that meet this criteria are added to the goal of the resolve subplan, reinforcing that they will be true once the resolve subplan terminates.

IV. Results

Since the main baseline for comparison is the Fast-Downward [18] planning loop implemented in Merlin et al. [5], we test our planner on the same domain: the PBJ domain. In this domain, the robot is in a kitchen with some number of cupboards. Each cupboard contains some number of objects organized as a stack, so reaching a given object requires removing all objects in front of it. The robot must retrieve bread, a knife, jelly, and peanut butter, get them onto a table, and then spread each ingredient and put together the two halves of the sandwich. The domain can scale up both the number of cupboards and the number of objects (in addition to the four necessary objects). Since this is done in simulation and some planning is done online actions are considered to be executed instantaneously, so the time recorded reflects only the planning time.

We show in Fig 3 the time comparisons for solving the PBJ domain using SPOP and Fast Downward (FD)[1]. In the small 1 cupboard domain FD outperforms SPOP by a small margin due to the increased complexity of the planning algorithm itself. With larger domains FD quickly fails to scale while SPOP continues to solve tasks within a handful of seconds. SPOP was able to scale up to much larger domains than we tested FD on as shown by I. The longest domain to complete on average was actually 10 cupboards and 100 objects. This is because fewer cupboards leads to more objects in a given cupboards, and therefore lengthier pick and place plans to retrieve a given object.

[1]The Fast Downward implementation used includes oracle heuristics for which objects are likely to advance the plan, as put forward in Merlin et al. [5]

	0 objects		5 objects		10 objects		20 objects		50 objects		100 objects	
	Avg Time (s)	STD Time(s)	Avg Time (s)	STD Time(s)	Avg Time (s)	STD Time(s)	Avg Time (s)	STD Time(s)	Avg Time (s)	STD Time(s)	Avg Time (s)	STD Time(s)
10 cpbrds	0.921	0.011	1.034	0.013	1.216	0.023	1.608	0.038	4.195	0.257	20.781	3.871
20 cpbrds	1.901	0.023	2.030	0.029	2.275	0.037	2.517	0.053	4.160	0.096	8.379	0.224
30 cpbrds	3.239	0.051	3.365	0.047	3.555	0.061	4.180	0.069	5.755	0.092	10.284	0.270

TABLE I
SOLVING LARGER PBJ DOMAINS WITH SPOP, AVERAGES TAKEN OVER 50 TRIALS

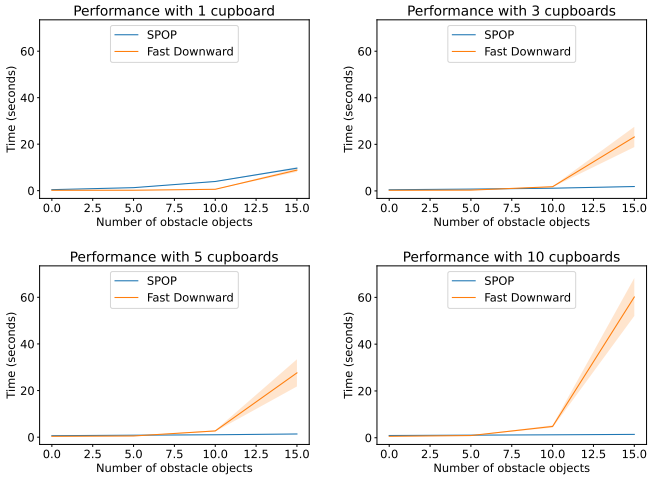

Fig. 3. The times it took to solve each domain size for both Fast Downward and SPOP

V. CONCLUSIONS

Our work introduces the concept of the object scouting problem, which can be well modeled as a LOMDP [5]. Since previous work in the LOMDP setting utilized only the most generic of planning approaches, we also introduce the **SPOP** algorithm, which we show to be particularly well suited to this problem formulation. **SPOP** synergizes uniquely with the object scouting problem by enabling the creation of partial plans that delay planning over gaps in information, as well as providing useful subgoal heuristics for what must be planned in those gaps once the relevant information is observed. Our preliminary results show that our **SPOP** algorithm scales at an order of magnitude faster than the previous methods as a result of exploiting the nature of partial order planning.

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2005.

[2] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1686–1705, 2023.

[3] K. Zheng, "Generalized Object Search," Ph.D. dissertation, Brown University, February 2023.

[4] A. Khanal and G. J. Stein, "Learning augmented, multi-robot long-horizon navigation in partially mapped environments," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 167–10 173.

[5] M. Merlin, S. Parr, N. Parikh, S. Orozco, V. Gupta, E. Rosen, and G. Konidaris, "Robot task planning under local observability," in *Proceedings of the 2024 IEEE International Conference on Robotics and Automation*, 2024, pp. 1362–1368.

[6] D. S. Weld, "An Introduction to Least Commitment Planning," *AI Magazine*, vol. 15, no. 4, pp. 27–27, 1994.

[7] M. R. Dogar, M. C. Koval, A. Tallavajhula, and S. S. Srinivasa, "Object search by manipulation," *Autonomous Robots*, vol. 36, pp. 153–167, 2014.

[8] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[9] C. Bradley, A. Pacheck, G. J. Stein, S. Castro, H. Kress-Gazit, and N. Roy, "Learning and Planning for Temporally Extended Tasks in Unknown Environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4830–4836.

[10] G. Stein, "Generating High-Quality Explanations for Navigation in Partially-Revealed Environments," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 17 493–17 506.

[11] D. Koller and R. Parr, "Policy Iteration for Factored MDPs," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, p. 326–334.

[12] C. Diuk, A. Cohen, and M. L. Littman, "An Object-Oriented Representation for Efficient Reinforcement Learning," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 240–247.

[13] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee, "Planning under Uncertainty for Robotic Tasks with Mixed Observability," *The International Journal of Robotics Research (IJRR)*, vol. 29, no. 8, pp. 1053–1068, 2010.

[14] A. Barrett and D. S. Weld, "Partial-order planning: Eval-

uating possible efficiency gains," *Artificial Intelligence*, vol. 67, no. 1, pp. 71–112, 1994.

[15] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.

[16] D. Paulius, "Object-Level Planning and Abstraction," in *CoRL 2022 Workshop on Learning, Perception, and Abstraction for Long-Horizon Planning*, 2022.

[17] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL – The Planning Domain Definition Language," CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Tech. Rep., 1998.

[18] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.