Least Commitment Planning for the Object Scouting Problem

Max Merlin[1*], Ziyi Yang[1], George Konidaris[1†], David Paulius[1†]

***Abstract*— State uncertainty is a primary obstacle to effective long-horizon robot task planning. State uncertainty can be decomposed into spatial uncertainty—resolved using SLAM—and uncertainty about the objects in the environment, formalized as the *object scouting problem* and modeled using the *Locally Observable Markov Decision Process* (LOMDP). We introduce a new planning framework specifically designed for object scouting with LOMDPs called the *Scouting Partial-Order Planner* (SPOP), which exploits the characteristics of partial order and regression planning to plan *around* knowledge gaps the robot may have about the existence, location, and state of relevant objects in its environment. Our results highlight the benefits of partial-order planning, demonstrating its suitability for object scouting due to its ability to identify absent but task-relevant objects, and show that it outperforms comparable planners in plan length, computation time, and execution time.**

I. Introduction

Any robot attempting to complete a task in the real world must successfully navigate through space and interact with the objects in it. One major complicating factor is the *state uncertainty* pervading this problem: the robot cannot simply know the state of its environment, but must instead estimate that state using its sensors. These sensors are typically limited by range and line of sight, thus narrowing effective perception to what is *near the robot* and *unobstructed from view*. Further, individual sensor readings are themselves unreliable, generally requiring uncertainty to be modeled across the entire state, even parts that can be currently measured. Modeling the uncertainty in large environments is intractable, so roboticists often turn to modeling assumptions and abstractions to isolate elements of uncertainty that can be resolved using specialized methods. For example, uncertainty in robot navigation is typically formalized as a SLAM (*Simultaneous Localization And Mapping*) problem [1], [2]. Solutions to SLAM allow a robot to accurately estimate its position and resolve spatial uncertainty despite the noise from individual sensor measurements, enabling efficient navigation within an envelope of known space.

When it comes to manipulation, uncertainty primarily has to do with objects, and several works have focused on different subproblems stemming from that uncertainty. Object search [3], [4], [5] focuses on navigation to resolve the uncertainty about an object's location. Active perception [6] approaches assume that an object is within sensor range and attempt to resolve the low-level uncertainty about object classification and poses by repeatedly generating informative sensor poses. However, mobile manipulation requires further *task-relevant* reasoning and interactive perception [3]: many objects may be present in the environment, but a robot should focus on locating and observing those that would progress it towards its goal. In the **object scouting problem**, a robot must find, localize, and state estimate the objects *relevant to* solving its task.

The *Locally Observable Markov Decision Process* [7] (LOMDP) is a task-level formulation of object scouting. Objects in a LOMDP 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 and repeatedly sensed using closed-loop active perception, allowing us to model them as fully observed. In addition, manipulating an object requires that it 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. This is analogous to the way that SLAM constructs an envelope of known space within which path planning is effective. When a task cannot be solved with known objects, the robot must decide which type of object to find (using *object search*) and then observe (using *active perception*) next. Merlin et al. [7] showed that the simplest LOMDP planner, which randomly selects the object to search for next and replans when new objects are observed, can solve much larger tasks than a POMDP planner. That naive approach, while effective, leaves plenty of room for improvement.

We introduce SPOP (**Scouting Partial-Order Planner**), a planner that exploits the synergy between least-commitment planning [8] and the LOMDP model to generate partial plans that become progressively more complete as the robot observes more objects. In particular, we exploit the freedom offered by its action ordering system to create an 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 task-specific information about which missing objects would help complete the plan. Once the relevant objects are found, a subplan with the new information fills the gap in the plan, obviating constant replanning.

Our key contributions are as follows:

- We propose SPOP, which supplies our object search method with which objects are task relevant.
- Through a series of experiments, we highlight increased performance in planning time when compared to our previous LOMDP planner [7].
- In addition, we demonstrate a significant reduction in plan length, such that the number of actions is more comparable to the fully observable baseline.

[1]Department of Computer Science, Brown University, RI, USA
[*]Corresponding Author (Email: max_merlin@brown.edu)
[†]Equal Advising

II. Background

State uncertainty is undesirable because it imposes enormous computational costs on task-level planning. Nevertheless, it is unavoidable in mobile manipulation tasks, which can be decomposed into uncertainty to do with *navigation*—predominantly concerning space and resolved by SLAM—and with *manipulation*—predominantly concerning object state [9]. Just as SLAM resolves uncertainty about a robot's pose, enabling it to navigate through its world using a learned map, the *object scouting* problem addresses uncertainty about object existence, location, and attributes, which is central to manipulation. In SLAM, the robot typically treats the map as if it were exact and plans accordingly, updating it as necessary if errors are encountered. So too in object scouting, a reasonable strategy is to construct closed-loop measurement routines that gain sufficient information about an object within sensor range to subsequently assume that its state is known for planning; if new objects or attributes become observed or state estimates change, the robot can simply replan.

Object scouting fuses two families of methods for object sensing: *object search* and interactive *object perception*. While object search focuses primarily on locating objects [4] (which may require some degree of manipulation to observe and localize objects [10]), object scouting must also reason about the states of said objects to determine whether they can be used to satisfy a task. Specifically, the robot must decide which task-relevant objects it must find, and it must resolve both their pose and their state. For example, consider a robot that must fill a coffee machine with water using a cup. Object search would focus solely on locating *any* cup that exists in its environment, regardless of its state. Object scouting must locate a cup that contains water; if the robot can only locate an empty cup, it should subsequently plan to make it full.

A. Locally Observable Markov Decision Processes

Locally Observable Markov Decision Processes (LOMDP) are a task-level model of object scouting, which assumes the existence of accurate closed-loop object state estimation routines that can resolve uncertainty over a given object's state. The state of a LOMDP is factored into objects that may be fully observed or unobserved. This means that the set of observed objects inherently compose an incomplete MDP model of the world state, which expands and updates as the agent explores and discovers new objects. The LOMDP also introduces a locality function that describes the set of states that would allow the agent to observe a given object. The locality function assists in transitioning an object from unobserved to observed, as passing through an object's locality guarantees its being observed.[1]

Merlin et al. [7] introduced the basic framework for planning only with what is known (i.e., observed) to the robot while searching for new, unknown objects to reinitialize planning with additional information. Since the robot may not know the exact objects in a domain, it uses *Skolem objects*: hypothetical objects of each object class that allow the robot to reason about where it may find new objects *if* they existed. When searching for a new object, the LOMDP provides the agent with locales from which it can attempt to observe each Skolem object. Each locale is a target region for the object search routine that will terminate after 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. LOMDPs are well suited for other problem settings in which local observability can be exploited, such as navigation in partially-mapped environments [11], [12], [5] and guiding information-seeking actions [13], [14].

[1] At the task level, an object remains known after observation. However, at a lower level, achieving observability may involve repeated views, next-best-view selection, modeling and updating state uncertainty, etc.

III. Planning for Object Scouting

Object scouting using LOMDPs can solve tasks in partially observable environments while only using Markov planners. This is similar to exploring unknown environments via SLAM, where a robot navigates within an explored region while occasionally finding frontiers that expand its map. A key requirement for effectively navigating partially known environments is the ability to select frontiers likely to reveal parts of the map that help the robot achieve its goal [15].

Similarly, LOMDPs allow an agent to represent its task plan as a fully observable environment with "frontiers" that expand the set of known objects. An effective planner should identify a relevant object to search for that would expand the known state, thus helping it complete the task. For example, a planner tasked with making a sandwich should know to look for bread and not milk. Because planning occurs within a known envelope that expands over time, solvers must *interleave* planning and execution; planning in the expanded space occurs only after executing the actions that caused the expansion. We use PDDL to represent the portion of the state that is fully observed. Therefore, a planner of this type—as laid out in [7]—comprises an *inner planning loop* and an *outer planning loop*. The inner loop plans over a PDDL file (representing only the known objects) to generate a plan. The outer loop executes that plan and generates an updated PDDL file that represents the state of all known objects, repeating the inner planning step on the new model until the task has been achieved.

A. Least-Commitment Planning

Solving the object scouting problem requires a robot to plan with potentially insufficient information. This could be done by generating incomplete plans, leaving gaps in the plan that will be filled once new relevant information is gathered. This aligns strongly with the idea of *least-commitment planning* [8], in which a solver logically decomposes a goal in a nonlinear fashion to find a plan and resolve action ordering as planning progresses. A least-commitment planner outputs a plan represented as a tuple $(\mathcal{A}, \mathcal{O}, \mathcal{L})$, where $\mathcal{A}$ 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 ordering constraint $\mathcal{O}$ is $(a_1 < a_2, a_1 < a_3, a_2 < a_3)$, meaning that action a_1 must come before a_2 and a_3, and action a_2 must come before a_3. An example link is $(a_1 \xrightarrow{p_1} a_3)$, denoting that action a_1 fulfills predicate p_1 for a_3. Plans may have multiple ways to resolve the ordering constraints. For example, when making 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.

This method initializes the search from a goal given as a set of predicates, which are added to the initial agenda of the plan. One of those predicates is dequeued from the agenda, and the planner attempts to connect it to an action that will make that predicate true. First, it attempts to link the predicate to an action already in the set of planned actions $\mathcal{A}$, including linking to the start state. If no valid actions already in the plan can fulfill a predicate, it then searches all possible actions and adds one that will resolve the predicate at random. The ordering $\mathcal{O}$ and links $\mathcal{L}$ are updated to reflect that this action fulfills a predicate and the ordering constraints associated with it. If the newly added action has any preconditions, those are 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 actions are causally consistent with each other.

Computation at each node is more expensive than a linear forward or reverse planner due to its larger search space [16]. However, partial-order planners are well suited to the object scouting problem, as they naturally handle information gathering processes critical to resolving uncertainty [17] through their task decomposition. Furthermore, even with a partial plan, we would know which preconditions need to be fulfilled and, in turn, the objects required to fulfill them, which can still inform future planning processes [18], [19].

B. Scouting Partial-Order Planner (SPOP)

SPOP is a novel planning algorithm extending the POP (*Partial-Order Planner*) algorithm by Weld [8]. As a plan-space planner [17], POP plans from the goal, changing the branching factor of planning from the number of actions possible in a given state to the number of actions that can fulfill a predicate. In addition, the nonlinear nature of POP leaves gaps in plans that are resolved as the algorithm progresses. These properties are highly desirable for the object scouting problem. First, the nonlinearity of POP allows the robot to find and resolve objects multiple times in a given plan. Second, the object scouting problem is innately high level: it exists in the context of a robot with high-level manipulation skills and a model of its environment. When planning at this level, many predicates are only resolved by a single action. For example, when making a peanut butter and jelly sandwich (Figure 1), there may exist a single skill that will fulfill the predicate `(sandwich-made)`, one that will fulfill `(jelly-spread)`, one for `(PB-spread)`, etc. Least-commitment planning is uniquely suited to efficiently handle such instances.

In detail, our algorithm is first given a goal state defined as a set of predicates, which are added to a queue or *agenda* of predicate and action pairs. As in the POP algorithm, a plan is represented as a tuple $(\mathcal{A}, \mathcal{O}, \mathcal{L})$ (see Section III-A). At each step of the algorithm, a tuple $(pred, A_{need})$ is removed from the agenda, where A_{need} is an action that requires that $pred$ is true (line 2). During each iteration, the algorithm identifies the actions that can satisfy $pred$ and splits them into three sets (lines 3–5): `possible_old`, the list of actions already in the plan $\mathcal{A}$ that contain $pred$ as an effect; `possible_new`, the list of all fully instantiated actions that have $pred$ as an effect (This excludes any actions which take in any skolem object as a parameter); and `possible_skolem`, which considers Skolem actions that contain $pred$ as an effect.

Algorithm 1: Scouting POP

```
1  Input: plan (A, O, L), agenda agenda;
2  pred, A_need = pop(agenda);
3  possible_old ← get_satisfying_acts(A);
4  possible_new ← get_satisfying_acts(all_actions);
5  possible_skolem ← get_satisfying_acts(skolem_actions);
6  while not_empty(agenda) do
7      if not_empty(possible_old) then
8          A_add = pop(possible_old);
9          O ← (A_add, A_need);
10         L ← (A_add, pred, A_need);
11     else if not_empty(possible_new) then
12         A_add = pop(possible_new);
13         A ← A_add;
14         O ← (A_add, A_need);
15         L ← (A_add, pred, A_need);
16         for precond in get_preconditions(A_add) do
17             agenda ← (precond, A_add);
18     else if not_empty(possible_skolem) then
19         A_skolem = pop(possible_skolem);
20         A_resolve = make_resolve_action(pred);
21         A ← A_resolve;
22         O ← A_resolve, A_need);
23         L ← (A_resolve, pred, A_need);
24         for precond in get_preconditions(A_skolem) do
25             if Is_Skolem(precond) then
26                 A_find = make_find_action(precond);
27                 A ← A_find;
28                 O ← (A_find, A_resolve);
29     else
30         return None;
31     plan_valid = check_consistency(ordering, links);
32     if plan_valid then
33         return SPOP((A, O, L), agenda);
34     else
35         continue;
36 return (A, O, L), agenda;
```

If there are any actions in `possible_old`, SPOP removes an action from the list and attempts to add it to the plan by updating ordering and link constraints ($\mathcal{O}$ and $\mathcal{L}$, respectively) with the new action A_{add} (lines 7–10). After

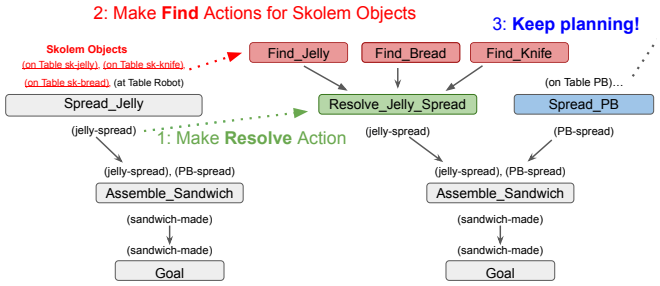

Fig. 1. Generating `resolve` and `find` actions from a Skolem action.

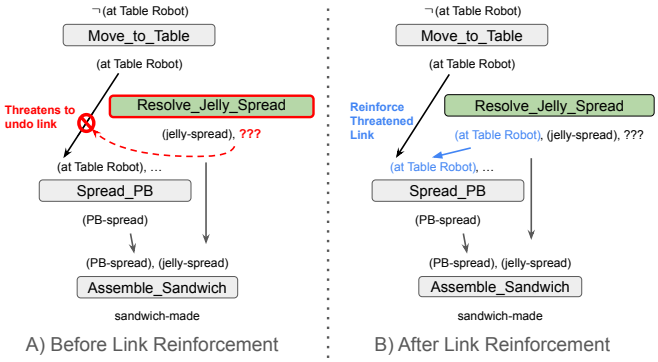

Fig. 2. When executing a `resolve` action, other causally linked predicates may be threatened. To prevent this, any threatened links are added to the goal of the `resolve` subplan, so that they are true after `resolve` is completed.

attempting to add any action, the algorithm then verifies that the ordering and link constraints are met using the same process as in [8] (line 31). If the logical consistency check is passed, SPOP recurses with the updated plan $(\mathcal{A}, \mathcal{O}, \mathcal{L})$, moving to resolve the next item $(pred, A_{need})$ in the agenda. If there are no actions already in the plan that can resolve $pred$ or if all existing actions fail, it then attempts to add a new action from `possible_new` to the plan. $(\mathcal{A}, \mathcal{O}, \mathcal{L})$ are again updated. However, when adding a new action, we must then add all of the preconditions of that action to the agenda (lines 11–17).

As this planner resolves its agenda of predicates, available actions may be limited due to a lack of domain knowledge. For example, in Figure 1, we try to add the action `Spread_Jelly` to the plan to resolve the `(jelly-spread)` predicate, but `Spread_Jelly` has three parameters that refer to Skolem objects: objects that have not been found but could exist. We call this a *Skolem action* (A_{skolem}). If a valid Skolem action exists in `possible_skolem`, the planner does *not* add it to the plan since the action is based on hypothetical objects. Instead, it inserts a `resolve` action ($A_{resolve}$) linked to $pred$, denoting that there is a possible future subplan to resolve once more information is gathered (lines 20–23). It then checks each parameter of the Skolem action to identify what objects need to be found. For each parameter that refers to a Skolem object, the planner creates a `find` action and updates $\mathcal{A}$ and $\mathcal{O}$ (lines 26–28). In Figure 1, `Spread_Jelly` relies on three Skolem objects: `sk-jelly`, `sk-sknife`, and `sk-bread`. A `find` action for each Skolem object is added to the plan.

C. Plan Execution

The SPOP algorithm 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 are performed as normal, but `find` and `resolve` actions trigger specialized subroutines. The `find` action behaves no differently than described in prior work [7]; 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 lacks the knowledge to do so. When `resolve` 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 would not know all of its effects, only that one of those effects must be the predicate that caused its creation. When planning within the `resolve` block, some actions may have effects that threaten an existing causal link in the macro-plan. Since the agent has already executed portions of the plan, it cannot run the link protection process with those links, since that requires the ability to rearrange the actions that have already been taken. Instead, we use a process called *link reinforcement* (illustrated in Figure 2). Before executing a `resolve` subplan, SPOP checks to see if any causal links are both supplied by an action before the `resolve` action and required by an action after it. Any predicates that meet these criteria are added to the goal of the resolve subplan so that they are true when it terminates.

IV. Experiments

This section experimentally compares our planning framework with the existing LOMDP planner [7]. This baseline (hereafter **SFD** for *Scouting Fast-Downward*) iteratively generates PDDL as more information about the scene is collected and plans with Fast-Downward [20]. We highlight markedly better performance across several metrics in simulated domains, and showcase task executions using plans obtained from each method in AI2THOR [21]. We report performance using average planning time (in seconds) as well as average plan length in terms of the number of actions.

A. RoboHome Domain

We first test our planner in a simulated household domain called RoboHome (inspired by environments such as AI2THOR [21]). This domain was inspired by the peanut butter and jelly (PBJ) domain from Merlin et al. [7], in which there are several locales (i.e., a table, a counter, and some number of cupboards), task-relevant objects (i.e., knife, bread, jelly, and peanut butter), and some number of objects irrelevant to the task. Objects may be found at each locale; most locales, like cupboards, need to be opened before the agent can observe what is inside. The robot is equipped

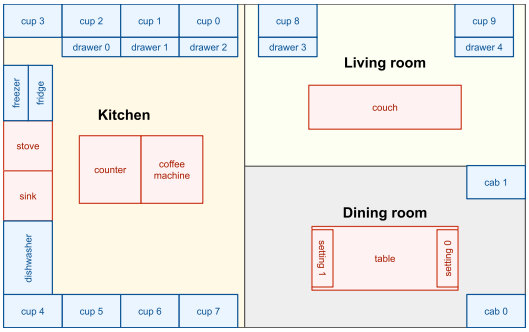

Fig. 3. Conceptual map of the RoboHome Domain (Section IV). Locations that can be opened or closed (e.g., cabinets or cupboards) are shown in blue, while other locations not exhibiting this property as shown in red (best viewed in color). The simulated robot must use open actions on openable locations to observe objects that may or may not be contained within them.

with skills for picking and placing objects, moving between locations, and performing various actions that change the state of each object, such as slicing cheese or spreading jelly on bread. In addition to making a PBJ sandwich, five more objectives have been added using new objects: 1) *grilled cheese sandwich*, 2) *omelet*, 3) *coffee*, 4) *setting a dining table*, and 5) *find my phone*. Each objective requires distinct object subsets, where each object must be moved to key locations to allow the robot to execute task-relevant actions. For instance, the robot must move items to the countertop for slicing objects with a knife: for toast, it must slice bread, while for a grilled cheese sandwich, it must slice cheese. We assume that a single instance of every object type is present, regardless of the task the robot has to perform.

The map of the (purely symbolic) RoboHome domain models is shown in Figure 3. At the start of each trial, objects are randomly distributed across locations. This is guided by a prior distribution, where each object can appear in 3 to 6 possible locations, one of which is most likely (65% of the time). The only notable exception is the cellphone, which has a uniform probability of appearing in any location. These prior distributions are used as heuristics for planner settings that use priors, indicated by **-PR** after the algorithm name.

Methods: In our experiments with the RoboHome domain, we compare **SPOP** against **SFD**, with and without priors. We also compare against a fully observed oracle. The five algorithms are: 1) **MDP:** a planner with a fully observable MDP; 2) **SFD:** our baseline planner solving the task under local observability as in [7]; 3) **SFD-PR:** Same as SFD, but with object locale priors `find` action; 4) **SPOP:** SPOP solving the task under local observability; and 5) **SPOP-PR:** same as SPOP, but with object locale priors. Both LOMDP planners have access to the same `find` action. When given priors, the `find` action has information on the most likely locations of a given object. Otherwise, it searches at random from among the 3-6 possible locations for that object.

Results: For every objective, both versions of SPOP consistently outperform the baseline methods in average computation time and average plan length (Figure 4). This highlights the importance of identifying which objects to

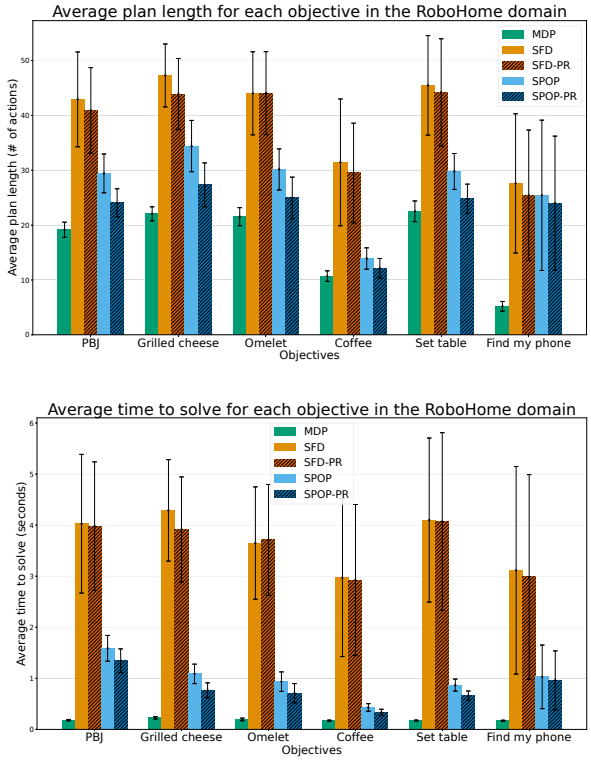

Fig. 4. Graphs showing average plan lengths and plan computation times for each domain objective using both SFD (Scouting Fast-Downward) and SPOP with and without priors over object location. SPOP-PR generally outperforms other locally observable planners, with the exception of the *find my phone* task, where priors are uniformly distributed.

locate for a given task. SFD often picks a task-irrelevant object to locate and searches unhelpful locations. In addition, searching for the wrong object drastically increases the plan length and, more importantly, the number of times the planner is run, leading to longer plan times. From the perspective of real robot execution, the additional actions due to incorrect `find` actions is undesirable, demanding both time and energy to unnecessarily navigate and explore its environment. In contrast, SPOP identifies useful objects to search for, resulting in fewer actions needed to solve a given task. This also results in fewer `find` actions, state updates, and planning calls, beating SFD in runtime despite the higher computational cost of the internal POP planning loop.

Although providing priors over object locations reliably improves SPOP (SPOP-PR), it does not consistently improve SFD (SFD-PR). We hypothesize that the lack of priors over object locations does not affect the chances of SFD accidentally finding a useful object when searching for an incorrect one. The average plan length of SPOP-PR is only slightly above the fully observable oracle (MDP), further highlighting the benefits of identifying the right objects to seek. The notable exception is *find my phone*, which requires the agent to find and retrieve a cellphone that is equally likely to be at any location. This means that when SFD picks a task-irrelevant object to find, it is just as likely to find the cellphone by accident as when SPOP correctly tries to find the phone. This results in SPOP and SFD having similar plan lengths, though SPOP remains firmly ahead in runtime.

TABLE I

RESULTS OVER 100 TRIALS ON THE PBJ DOMAIN WITH VARYING NUMBER OF OBJECTS AND CUPBOARDS

# Extra Objects	Planner	10 Cupboards		30 Cupboards		50 Cupboards	
		Avg. Time (s)	*Avg. Plan Length*	*Avg. Time (s)*	*Avg. Plan Length*	*Avg. Time (s)*	*Avg. Plan Length*
0	MDP (Oracle)	0.11 ± 0.010	17.49 ± 1.330	0.12 ± 0.006	18.42 ± 0.890	0.16 ± 0.006	18.63 ± 0.787
	SFD	1.17 ± 0.167	28.07 ± 2.914	2.90 ± 0.551	58.38 ± 10.405	5.74 ± 0.975	91.38 ± 14.961
	SFD-PR	1.06 ± 0.253	25.00 ± 4.573	2.46 ± 0.861	48.68 ± 16.199	4.55 ± 2.072	70.98 ± 30.784
	SPOP	0.65 ± 0.138	30.02 ± 3.137	1.57 ± 0.213	63.11 ± 9.483	3.36 ± 0.575	93.72 ± 18.381
	SPOP-PR	$\mathbf{0.59 \pm 0.123}$	$\mathbf{22.30 \pm 3.611}$	$\mathbf{0.97 \pm 0.256}$	$\mathbf{27.66 \pm 12.420}$	$\mathbf{1.46 \pm 0.502}$	$\mathbf{28.03 \pm 16.794}$
5	MDP (Oracle)	0.12 ± 0.011	18.48 ± 1.878	0.14 ± 0.004	18.91 ± 1.055	0.19 ± 0.007	18.84 ± 0.813
	SFD	1.36 ± 0.278	28.43 ± 3.613	3.54 ± 0.645	61.12 ± 9.538	6.81 ± 1.271	92.83 ± 15.733
	SFD-PR	1.36 ± 0.348	27.04 ± 4.566	3.24 ± 0.931	54.12 ± 14.033	6.12 ± 2.021	80.67 ± 24.619
	SPOP	0.81 ± 0.141	30.39 ± 2.954	2.01 ± 0.246	63.29 ± 9.378	4.38 ± 0.721	93.74 ± 15.719
	SPOP-PR	$\mathbf{0.68 \pm 0.145}$	$\mathbf{23.10 \pm 3.842}$	$\mathbf{1.19 \pm 0.338}$	$\mathbf{29.52 \pm 13.072}$	$\mathbf{1.78 \pm 0.736}$	$\mathbf{29.36 \pm 17.667}$
10	MDP (Oracle)	0.12 ± 0.009	19.11 ± 2.044	0.15 ± 0.007	19.08 ± 1.098	0.21 ± 0.007	19.18 ± 0.757
	SFD	1.48 ± 0.289	29.42 ± 3.346	3.98 ± 0.818	60.29 ± 9.623	7.58 ± 1.776	91.04 ± 18.369
	SFD-PR	1.49 ± 0.327	28.10 ± 3.826	3.75 ± 0.978	55.42 ± 12.359	7.39 ± 2.143	85.21 ± 22.552
	SPOP	0.93 ± 0.197	31.12 ± 3.796	2.44 ± 0.363	62.28 ± 9.395	5.56 ± 1.117	93.93 ± 18.780
	SPOP-PR	$\mathbf{0.78 \pm 0.156}$	$\mathbf{24.10 \pm 3.961}$	$\mathbf{1.36 \pm 0.446}$	$\mathbf{29.36 \pm 12.879}$	$\mathbf{2.33 \pm 1.310}$	$\mathbf{34.67 \pm 23.973}$
20	MDP (Oracle)	0.14 ± 0.012	21.67 ± 2.555	0.19 ± 0.009	19.95 ± 1.592	0.27 ± 0.010	19.40 ± 1.092
	SFD	1.73 ± 0.323	32.19 ± 3.987	4.84 ± 1.091	60.69 ± 10.567	9.50 ± 2.069	92.76 ± 15.763
	SFD-PR	1.74 ± 0.399	31.07 ± 4.098	4.92 ± 1.0325	59.99 ± 9.957	9.52 ± 2.454	88.89 ± 18.601
	SPOP	1.35 ± 0.231	33.83 ± 3.715	3.28 ± 0.637	61.39 ± 10.147	8.09 ± 1.609	94.33 ± 16.840
	SPOP-PR	$\mathbf{1.00 \pm 0.211}$	$\mathbf{26.52 \pm 4.126}$	$\mathbf{1.59 \pm 0.500}$	$\mathbf{27.22 \pm 9.532}$	$\mathbf{2.94 \pm 1.664}$	$\mathbf{33.25 \pm 20.270}$
50	MDP (Oracle)	0.24 ± 0.019	26.69 ± 3.946	0.34 ± 0.011	22.07 ± 2.438	0.48 ± 0.028	20.67 ± 1.730
	SFD	2.52 ± 0.466	37.20 ± 4.930	8.04 ± 1.819	64.56 ± 9.511	15.67 ± 3.598	94.82 ± 15.658
	SFD-PR	2.49 ± 0.546	36.58 ± 4.951	7.42 ± 2.149	60.00 ± 12.262	16.04 ± 4.253	93.58 ± 17.614
	SPOP	2.87 ± 0.551	38.62 ± 4.555	7.24 ± 1.354	65.35 ± 9.848	16.18 ± 3.926	92.89 ± 18.568
	SPOP-PR	$\mathbf{2.00 \pm 0.508}$	$\mathbf{31.01 \pm 4.702}$	$\mathbf{3.05 \pm 1.503}$	$\mathbf{30.79 \pm 12.971}$	$\mathbf{4.96 \pm 3.358}$	$\mathbf{31.72 \pm 18.577}$
100	MDP (Oracle)	0.51 ± 0.028	35.91 ± 7.624	0.71 ± 0.027	24.67 ± 2.958	0.99 ± 0.043	22.67 ± 2.128
	SFD	4.70 ± 0.877	46.84 ± 8.005	14.94 ± 3.543	65.92 ± 9.727	29.72 ± 8.615	94.20 ± 18.168
	SFD-PR	$\mathbf{4.43 \pm 1.082}$	45.84 ± 8.319	15.17 ± 3.691	65.78 ± 10.593	30.34 ± 7.764	94.46 ± 16.698
	SPOP	8.50 ± 1.930	48.67 ± 7.699	16.30 ± 3.021	68.03 ± 9.643	33.85 ± 6.939	96.84 ± 15.948
	SPOP-PR	5.72 ± 1.944	$\mathbf{40.50 \pm 8.506}$	$\mathbf{6.43 \pm 3.365}$	$\mathbf{33.59 \pm 12.226}$	$\mathbf{10.35 \pm 8.331}$	$\mathbf{35.70 \pm 22.209}$

B. PBJ Domain

Since RoboHome had a fixed size, we re-implemented the PBJ domain [7] such that SPOP and SFD (with and without priors) can be evaluated as we scale domains up in size (i.e., number of locations and objects). In this setting, a robot is in a kitchen with some number of cupboards, each containing some number of objects organized as a stack, where reaching a certain object requires removing all objects in front of it. The robot must locate and retrieve bread, a knife, jelly, and peanut butter, move them to a table, spread each ingredient onto two separate halves of bread, and then form the sandwich. In addition to the four task-relevant objects, we also include distraction objects irrelevant to the task. As with RoboHome, each object is given locale priors: objects could be found in any location, but each has a randomly assigned most likely location (60%), a less likely location (35%), and the remaining 5% of the distribution uniformly spread over the remaining locations.

Results: Table I shows that SPOP-PR performs best in almost all settings. The only exception is the domain with 10 cupboards and 100 objects, where both SFD and SFD-PR outperform SPOP and SPOP-PR by a small margin in terms of average computation time. This is because when there are many objects packed in the same cupboard, SFD is more likely to find the correct object despite choosing to search for an incorrect one. However, as the number of cupboards and objects increases, SFD quickly fails to scale while SPOP continues to solve tasks within a handful of seconds (Table I). As we involve more objects and locations, SFD-PR is more likely to choose to find an incorrect object, and it is less likely to find a correct object than SPOP-PR.

C. AI2THOR Domain

Finally, we use the AI2THOR [21] simulation environment to visually compare the better versions of the SPOP and SFD algorithms (i.e., SPOP-PR and SFD-PR). We designed the following tasks for AI2THOR: 1) *making toast*; 2) *making an omelet*; 3) *making coffee*; and 4) *making breakfast*, which is a combination of all three (3) tasks. Images from the perspective of the virtual agent are in Figure 5. We report two metrics in Table II: *average plan execution time* (in seconds) and *average plan length*. In general, we observed that SPOP-PR computes shorter plans, which resulted in

Fig. 5. A series of snapshots from AI2THOR [21], each taken after the agent has executed several actions to crack an egg. The agent first moves to the fridge and opens it, finding no egg. It then moves to the next locale, a cabinet, which does have the egg inside. Now that the egg is found, it brings the egg to the pan and cracks the egg.

TABLE II

RESULTS OVER 50 TRIALS ON THE AI2THOR [21] DOMAIN

Objective	Planner	Avg. Time (s)	Avg. Plan Length
Coffee	SPOP-PR	$\mathbf{40.66 \pm 13.121}$	$\mathbf{11.98 \pm 2.43}$
	SFD-PR	66.974 ± 36.469	19.30 ± 9.22
Toast	SPOP-PR	$\mathbf{68.750 \pm 11.822}$	$\mathbf{16.70 \pm 1.30}$
	SFD-PR	106.013 ± 35.837	26.12 ± 8.17
Omelet	SPOP-PR	$\mathbf{104.040 \pm 17.788}$	$\mathbf{27.88 \pm 1.24}$
	SFD-PR	105.953 ± 28.577	30.98 ± 6.62
Breakfast	SPOP-PR	176.4096 ± 18.9404	$\mathbf{41.86 \pm 2.08}$
	SFD-PR	$\mathbf{175.349 \pm 33.088}$	49.56 ± 6.24

shorter average execution times. However, we observed that the only exception was in the breakfast task; this is because the agent is increasingly likely to explore all locales, as more objects are needed to complete all three objectives in one go.

V. CONCLUSIONS

Our work proposes a new planning algorithm, the Scouting Partial-Order Planner (SPOP), designed to solve the object scouting problem as first introduced by Merlin et al. [7]. We demonstrate that this algorithm, which builds upon least-commitment planning, is 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 information gaps, as well as providing useful subgoal heuristics for what relevant information must be found in order to fill those gaps. Our experiments show that SPOP is quick at finding low-cost solutions across several domains and objectives, and SPOP scales significantly better with larger domains than the previous methods as a result of exploiting the nature of partial-order planning.

ACKNOWLEDGEMENTS

This work was supported by the Office of Naval Research (ONR) under REPRISM MURI N000142412603 and ONR grant N00014-22-1-2592. Partial funding was also provided by the Robotics and AI Institute.

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] J. Bohg, K. Hausman, B. Sankaran, O. Brock, D. Kragic, S. Schaal, and G. S. Sukhatme, "Interactive Perception: Leveraging Action in Perception and Perception in Action," *IEEE Transactions on Robotics*, vol. 33, no. 6, pp. 1273–1291, 2017.

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

[5] A. Khanal and G. J. Stein, "Learning Augmented, Multi-Robot Long-Horizon Navigation in Partially Mapped Environments," in *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 167–10 173.

[6] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, "Revisiting active perception," *Autonomous Robots*, vol. 42, no. 2, p. 177–196, Feb. 2018.

[7] 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 (ICRA)*, 2024, pp. 1362–1368.

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

[9] E. Rosen, S. James, S. Orozco, V. Gupta, M. Merlin, S. Tellex, and G. Konidaris, "Synthesizing Navigation Abstractions for Planning with Portable Manipulation Skills," in *Proceedings of The 7th Conference on Robot Learning*, 2023, pp. 2278–2287.

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

[11] G. Stein, "Generating High-Quality Explanations for Navigation in Partially-Revealed Environments," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 17 493–17 506.

[12] A. Paudel and G. J. Stein, "Data-Efficient Policy Selection for Navigation in Partial Maps via Subgoal-Based Abstraction," in *Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2023, pp. 11 281–11 288.

[13] 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 *Proceedings of the 2021 IEEE International Conference on Robotics and Automation*, 2021, pp. 4830–4836.

[14] D. A. Shell and J. M. O'Kane, "Decision diagrams as plans: Answering observation-grounded queries," in *Proceedings of the 2023 IEEE International Conference on Robotics and Automation*, 2023, pp. 1659–1665.

[15] G. J. Stein, C. Bradley, and N. Roy, "Learning over Subgoals for Efficient Navigation of Structured, Unknown Environments," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 87. PMLR, 29–31 Oct 2018, pp. 213–222.

[16] A. Barrett and D. S. Weld, "Partial-order planning: Evaluating possible efficiency gains," *Artificial Intelligence*, vol. 67, no. 1, pp. 71–112, 1994.

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

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

[19] D. Paulius, A. Agostini, and D. Lee, "Long-Horizon Planning and Execution with Functional Object-Oriented Networks," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 4513–4520, 2023.

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

[21] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: An Interactive 3D Environment for Visual AI," *arXiv*, 2017.