Ω^2: Optimal Hierarchical Planner for Object Search in Large Environments via Mobile Manipulation

Yoonyoung Cho[1,*], Donghoon Shin[2,*], and Beomjoon Kim[1]

***Abstract*— We propose a hierarchical planning algorithm that efficiently computes an optimal plan for finding a target object in large environments where a robot must simultaneously consider both navigation and manipulation. One key challenge that arises from large domains is the substantial increase in search space complexity that stems from considering mobile manipulation actions and the increase in number of objects. We offer a hierarchical planning solution that effectively handles such large problems by decomposing the problem into a set of low-level *intra-container* planning problems and a high-level *key place planning* problem that utilizes the low-level plans. To plan optimally, we propose a novel admissible heuristic function that, unlike previous methods, accounts for both navigation and manipulation costs. We propose two algorithms: one based on standard A* that returns the optimal solution, and the other based on Anytime Repairing A* (ARA*) which can trade-off computation time and solution quality, and prove they are optimal even when we use hierarchy. We show our method outperforms existing algorithms in simulated domains involving up to 6 times more number of objects than previously handled.**

I. Introduction

Consider a robot searching for a target object in a large home environment, densely populated with objects that occlude a target object from view as shown in Figure 1. The robot would have to manipulate occluding objects and navigate to different viewpoints to reveal novel volumes, while minimizing the operation time. Endowing robots with such capability would unlock vast opportunities not only in homes, but also warehouses, disaster sites, and military bases.

Our goal is to design a planner that can efficiently compute an optimal mobile manipulation plan for finding a target object in such large environments. This is a challenging problem involving a complex search space where the robot must reason over a large number of objects and plan motions in the joint configuration space of the robot base and manipulator. To guarantee optimality, the robot would also have to compute a plan that has the minimum operation time.

Given its importance, there have been several attempts to solve the object search problem, which can largely be divided into two classes: Active Visual Search (AVS) and Object Search through Manipulation (OSM). The goal of AVS is to discover the target object by planning the robot's viewpoints throughout the scene. While AVS has been applied to large, multi-room scale problems, they are confined to relatively uncluttered environments where the volumes in the environment can be readily observed by traveling to an appropriate viewpoint as they do not consider object manipulation [1]–[3]. The goal of OSM, on the other hand, is to discover the target object through sequential object manipulation [4]–[9]. These algorithms typically assume relatively small [7, 9] or no base movements [4, 6, 8], and are confined to environments with a smaller workspace than AVS such as a single cupboard or shelf. For these reasons, they only have been applied in domains with a relatively small number of objects than AVS.

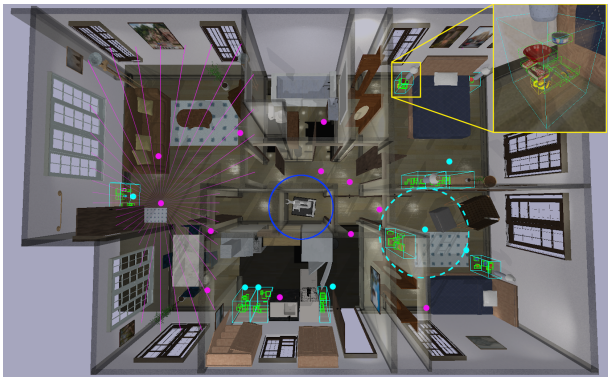


Fig. 1: A robot (blue circle) is searching for a target object (yellow cup, occluded in the zoomed-in container view, upper right) in a large cluttered environment, using a combination of navigation and manipulation actions to change its viewpoint, to get to a container and to clear occluding objects. In the figure, objects are marked with a green box; containers are marked with cyan, with the dashed circle indicating the robot workspace boundary enclosing the designated container; viewpoints are marked with magenta, with rays cast from the center that indicate observed regions.

For large, cluttered domains that we are contemplating, these approaches are not able to find an optimal solution either because they guarantee optimality in their original setup but there is no trivial adaptation to a mobile-manipulation setting, or simply because they cannot find the optimal solution in the first place. Moreover, because our domains have significantly more number of objects, these algorithms suffer from significantly larger branching factor and planning horizon, especially because they lack an informative heuristic function that considers both navigation and manipulation.

We propose an algorithm called Ω^2 Planner (Optimal Object search through Heuristic Hierarchical search using

[1]Intelligent Mobile Manipulation Lab, Korea Advanced Institute of Science and Technology, Department of Artifical Intelligence {yoonyoung.cho,beomjoon.kim}@kaist.ac.kr

[2]Dronebot Combat Development Center, Republic of Korea Army Training and Doctrine Command shinapses@gmail.com

* Yoonyoung Cho and Donghoon Shin are co-first authors

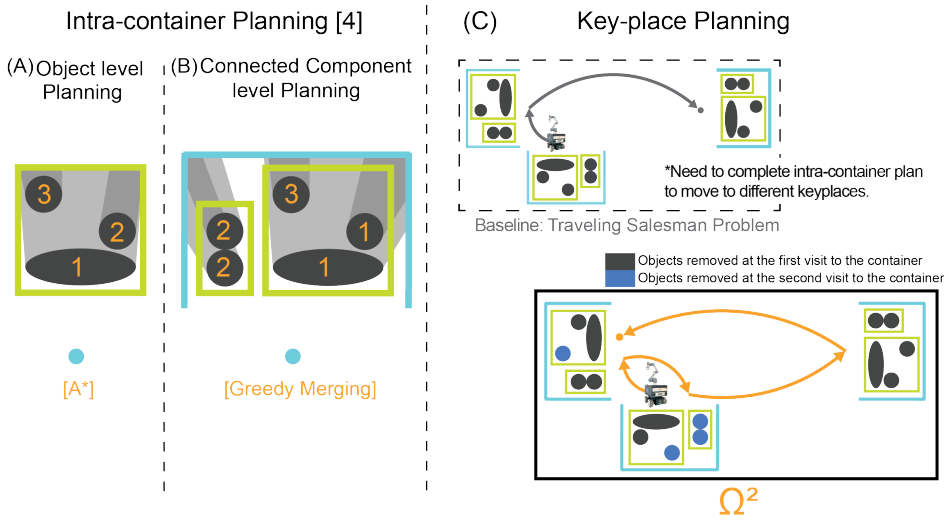


Fig. 2: An overview of our framework. Black ellipses denote objects, light and dark grey shade denotes individually and jointly occluded volume, green rectangles denote the connected components, cyan rectangles denote containers, and cyan filled circle denotes container key-place. **A** and **B** show the intra-container planning that only use manipulation actions in [4], and **C** shows key-place planning that only uses navigation actions. Naively using a traveling salesman approach that must complete intra-container plan before moving its base leads to a suboptimal plan. In contrast, Ω^2 that can move between key-places during intra-container plan leads to an optimal plan.

Mobile Manipulation Planner), which efficiently computes an optimal plan in such complex search space by using a hierarchy and novel heuristic function. Our main intuition is to utilize the hierarchy in the way objects are typically arranged in large cluttered environments. In most of these environments, objects are arranged into different regions such as cupboards, shelves, or table-tops that we call *containers*. Objects in different containers do not have a joint occluding volume, or affect each other's reachability, allowing us to treat each container as an independent OSM problem.

To exploit this, we first discretize the space of robot base poses given the locations and shapes of objects except those of the target object so that each base pose is associated with a container, or reveals a volume of the environment. Unlike existing OSM methods which uniformly discretizes the space [7], we discretize the space using an extension of a sensor placement algorithm [10] into *key-places*. Specifically, it allocates *viewpoint key-places* to the poses where the robot can observe a part of the environment, such that views at all key-places cover the entire environment. Additionally, it allocates *container key-places* such that at each key-place, all objects in that container are within the robot's reach.

Using the key-places, Ω^2 performs a hierarchical search. At a lower-level, we use an OSM algorithm [4] to compute the intra-container plan for each container. At a higher level, we perform *key-place level* planning, where we plan a sequence of key-places and the corresponding navigations. If we visit a container key-place, we follow the intra-container plan at that container. This hierarchical decomposition greatly reduces the search complexity by breaking the large problem into smaller independent problems, and reducing the branching factor from the number of objects to the number of key-places.

To perform the key-place level planning, we can naively adapt [4] by greedily merging different intra-container plans at different key-places. However, while the greedy approach can guarantee optimality in OSM problems [4], it loses optimality in mobile-manipulation setup as the navigation cost is not considered. Alternatively, we can treat this as a traveling salesman problem (TSP) and compute the plan that visits all the key-places with the least navigation cost. However, this also gives a suboptimal plan because we cannot compare the cost between removing an object at the current container versus that from another container, even though there may be more valuable actions at other containers.

We propose an admissible heuristic function that considers both the navigation and manipulation cost-to-go, by decomposing cost contributions from these two action types and estimating the lower bound for each part by solving a relaxed sub-problem. During planning, we allow interleaving low-level intra-container plans based on this heuristic function instead of committing to an intra-container plan. Figure 2C compares our approach with the TSP approach.

Based on this hierarchical decomposition and heuristic function, we offer two search algorithms. The first is based on A*, and the second is an anytime algorithm based on Anytime Repairing A* (ARA*) [11, 12], and we prove that both algorithms guarantee an optimal solution.

We evaluate the performance of our algorithm by comparing against state-of-the-art OSM algorithms [4, 7] that are adapted to mobile manipulation settings in four different simulated environments from iGibson [13]. Compared to previous works that only addressed up to 12 objects [4]

limited to a single container seen from at most 5 viewpoints [7], we test in 4 different environments that contain up to 10 containers, 18 viewpoints, and 75 objects. Our largest domain involved branching factor (BF) of 93 while previous works had at most 17 [4, 7], and we show that our algorithm consistently outperforms baselines in all four domains.

To summarize, our key contributions are:

- an object search algorithm for large-scale domains through mobile manipulation;
- an efficient search through hierarchical decomposition of mobile manipulation sub-problems;
- an optimal search through novel admissible and consistent heuristic function;
- practical trade-off between compute time and solution quality through an anytime extension.

II. Related work

A. Active Visual Search

The objective of AVS is to find the sequence of robot viewpoints that maximizes the probability of detecting the target object. One branch of AVS focuses on modelling accurate probabilities of target object location using prior semantic information such as object-room co-occurrence [1, 2, 14], object-object co-occurrence [2, 14, 15], and spatial relations between objects [16, 17]. Our framework can readily incorporate these prior probabilities.

Other branch of AVS focuses on planning under uncertainty. Aydemir et al. [1] model the problem as a Markov Decision Process for planning the sequence of next best views while exploiting spatial relations between objects. Wang et al. [3] model the problem as a Partially Observable Markov Decision Process (POMDP), and search object with a partial map of an environment. They model the unexplored part as belief state and use a graph structure to reduce the search space. Wandzel et al. [18] also model the problem as a POMDP, and search multiple target objects using an efficient forward search algorithm called POMCP [19]. They form potential target objects locations as belief state, while updating it with not only visual observations but also language commands. We do not consider uncertainty, and leave that as future work. As we will empirically show, even in a deterministic setting, efficiently computing an optimal plan in large multi-room environments with several tens of objects is already challenging for state-of-the-art algorithms.

All of these algorithms consider planning viewpoints and navigation motions, but not manipulation. As a result, none of them can search objects in a cluttered environment with object occlusions.

B. Object Search through Manipulation

Unlike AVS, OSM uses manipulation to find a target object in cluttered environments. Wong et al. [20] introduced an algorithm to address OSM in large environments, but was limited to a greedy algorithm that only finds sub-optimal solutions. Our hierarchical problem decomposition is inspired by Dogar et al. [4], where they define a hierarchy based on a notion of *connected component*, which is a set of objects that have intersecting occluding volumes or influence each other's reachability. Because different connected components are independent, we can break down the problem first into planning for each connected component, and then greedily merge these connected-component-level plans. We take a step further, and add one more level of hierarchy to handle key place planning. Moreover, because greedy merging is sub-optimal when we consider navigation, we introduce a novel heuristic function that guarantees optimality when paired with A^* search. See Figure 2 for details.

Lin et al. [7] extends [4] to combine manipulation and navigation. But because connected components are ill-defined with multiple viewpoints, they lose the benefit of hierarchical planning, limiting their scalability to larger scenes.

Li et al. [8] and Xiao et al. [9] formulate object search in cluttered environment as a POMDP with sampling-based online method that can address perception and manipulation error [8, 9, 19, 21]. Li et al. addresses OSM as choosing the target object among partially visible objects through manipulation. Xiao et al. combines manipulation planning and continuous viewpoint planning with base movements while considering perception and manipulation error. Unlike these algorithms, we do not model perception error or occlusion. Instead, we focus on addressing the challenge of optimal planning in domains with large branching factors.

III. Problem Formulation

The robot seeks to find the target object o^* of a known shape in a multi-room environment filled with movable objects, where only the robot is allowed to move objects. We are given a map $\mathcal{M}$ that indicates the locations and shapes of immovable room structures and non-target movable objects in the scene. Like previous works [4, 5, 9], we assume perfect perception and deterministic transition model, that all manipulation motions are pick, and do not consider object placement; rather, an object is removed from the scene upon picking as opposed to being re-positioned.

The robot state is described as $s^{rob} = \{\{x, y, \psi\}, \vec{\theta}\} \in SE(2) \times \mathbb{R}^{|\vec{\theta}|}$, where (x, y, ψ) denotes the position and orientation of the robot's base and $\vec{\theta}$ denotes the arm joint configurations. The state space consists of s^{obj}, the locations of remaining objects in the scene where each location is expressed in $SE(2)$, s^{loc}, the discrete set of viewpoint key-places that have not been visited, and s^{rob}. The initial state s^0 is denoted as $(s_0^{obj}, s_0^{loc}, s_0^{rob})$. The goal state is $(\emptyset, \emptyset, s^{rob})$ – that is, all non-target objects are removed and all viewpoint key-places are visited.

The action space is represented as a set of viewpoint key-places and objects. Note that the action space is hybrid: we have a discrete choice over viewpoint key-places and objects, and continuous choice over motions. We pre-compute the motions for each object and a pair of key-places, and perform discrete graph search on the viewpoint key-places and objects. We denote a continuous motion as τ which is either a navigation or pick motion depending on the circumstance.

Our goal is to compute an optimal motion sequence that discovers o^*. Because our goal is the same as [4], we use

the same optimality criteria as [4], called *minimum expected execution time*, defined as

$$\min_{\tau_{1:H}} \mathbb{E}[T[\tau_{1:H}]] = \min_{\tau_{1:H}} \sum_{t=1}^{H} P(\tau_t) \cdot T(\tau_{1:t}) \ , \qquad (1)$$

where $\tau_{1:H}$ is a sequence of motion plans of length H, $P(\tau_t)$ denotes the probability of finding o^* after using the motion at time step t, and $T(\tau_{1:t})$ denotes the time taken to execute the motion plans up to time step t.

Similar to [4], we assume $P(\tau)$ is proportional to the size of the volume revealed by taking τ, denoted $V_{\text{rev}}(\tau)$, weighted by a given prior coefficient $w(\tau)$. $w(\tau)$ reflects that each container has different probability of containing the target object. For instance, screwdrivers are more likely to be in a toolbox compared to a refrigerator.

So minimizing Eqn. 1 is equivalent to the following:

$$\min_{\tau_{1:H}} \mathbb{E}[T[\tau_{1:H}]] = \min_{\tau_{1:H}} \sum_{t=1}^{H} w(\tau_t) \cdot V_{\text{rev}}(\tau_t) \cdot T(\tau_{1:t}) \ . \qquad (2)$$

For OSM problems, where the robot only has a single container, time for moving an object stays the same across different time steps. For us, because our τ_t may involve navigation that varies across the source and target key-place that the robot will travel to, this is no longer true. So instead, we use the following equivalent objective function,

$$\min_{\tau_{1:H}} \sum_{t=1}^{H} V_{\text{unseen}}(\tau_t) \cdot T(\tau_t) \ , \qquad (3)$$

where $V_{\text{unseen}}(\tau_t) = (V_{\text{total}} - \sum_{t'=1}^{t-1} w(\tau_{t'}) \cdot V_{\text{rev}}(\tau_{t'}))$ is the total volume that have not been revealed up to time t. Note that eqn. 2 and 3 compute the same quantities, as illustrated in Figure 3A.

IV. Method

A. Method Overview

We now describe Ω^2 that computes a plan that solves the problem given in Eqn. 3. The key idea is to break the problem into a set of lower-level intra-container planning problems that are efficiently solved by [4], and use higher-level key-place planning to use the intra-container plans. As we will describe below, this effectively reduces the branching factor from number of objects to number of key-places, and finds an optimal plan based on our admissible heuristic function.

The algorithm is described in Algorithm 1. It takes as inputs $\mathcal{M}$, O, the locations and shapes of non-target objects in the scene, o^*, the shape of the target object, and $h_{\Omega^2}(\cdot)$, an admissible heuristic function that takes account of both navigation and manipulation costs.

Our algorithm begins by discretizing the search space into a set of key-places K. We allocate *viewpoint* key-places to the poses where the robot can observe a part of the environment, and *container* key-places for each container so that the robot can remove objects to reveal the occluded volumes. We first compute the container key-places. To compute this, we assign a key-place for each container such that all of the objects in the container are within the robot's reach, and the volume inside the container is completely visible from that key-place. If the container is large and a single key-place is insufficient, we assign multiple key-places until the container is completely covered in terms of reachability and visibility.

Algorithm 1 $\Omega^2(\mathcal{M},\ O,\ o^*, h_{\Omega^2}(\cdot))$

1: $K \leftarrow$ ComputeKeyPlaces($\mathcal{M}$, o^*)
2: $s_0 = (\mathcal{M}.s_0^{obj}, K.s_0^{loc}, s_0^{rob})$
3: $\Pi \leftarrow$ ComputeMotionPlans(K, O)
4: $C(\cdot,\cdot) \leftarrow$ GetEdgeCost(Π)
5: $\Xi \leftarrow$ ComputeIntraContainerPlan($\mathcal{M}, \Pi$)
6: $\tau_{1:H} \leftarrow$ ComputeKeyPlacePlan($s_0, K, h_{\Omega^2}(\cdot), C(\cdot,\cdot), \Xi$)
7: **return** $\tau_{1:H}$

We then compute viewpoint key-places. Allocating viewpoint key-places is akin to the *Point-Guard Art Gallery Problem*: we seek a set of positions whose sensing area covers the scene volume. Based on this, we adapt the method in [10] to our problem. First, we employ the dual sampling proposed in [10] to the voxelized volumes of the scene to quickly sample key-places that cover the volumes. Then, we employ set-cover optimization of key-places to reduce redundant samples. During this process, we also consider container key-places as viewpoint key-places, and only the remainder of the unseen volume within the environment is accounted for during viewpoint key-place allocation. Examples of key-places are given in Figure 1.

Using the key-places and $\mathcal{M}$, we then compute the initial state. Afterwards, we pre-compute the set of motions, denoted Π, for navigating between all pairs of key-places and the motions for picking objects. This is then used to compute the edge cost of each action, $C(s_t, \tau_t) = V_{\text{unseen}}(\tau_t) \cdot T(\tau_t)$.

We then use the algorithm from [4] to compute the intra-container plans for each container, denoted Ξ. This lower-level planner uses a set of objects in that container as its action space, instead of all the objects in the scene, effectively reducing branching factor and planning horizon. Each container is now associated with an intra-container plan, and we have a set of viewpoint key-places. Using the set of key-places, denoted K, as the action space, we compute the optimal key-place level plan using either A* or ARA* [11] in line 6, which will yield Ω^2-A* and Ω^2-ARA* respectively. For a chosen action $k \in K$, the associated motion plan is a navigation motion from the current key-place to k if k is a viewpoint key-place. If k is a container key-place, then the associated motion is the navigation motion followed by a manipulation motion that corresponds to a step in the intra-container plan for that container.

Note that using our hierarchy, we reduced a large problem that involves all of the objects and viewpoint key-places into a set of intra-container planning problems and a problem of planning over key-places. We now describe our heuristic function that enables us to compute an optimal plan over the key-places, h_{Ω^2}.

B. Admissible Mobile-manipulation Heuristic Function

Since our plan has both navigation and manipulation motions, we construct our heuristic as a sum of two independent heuristics per motion type: h_{manip} for costs incurred during manipulation and h_{nav} for costs incurred during navigation. So $h_{\Omega^2}(s) = h_{\text{nav}}(s) + h_{\text{manip}}(s)$, where s is a state.

For h_{manip}, we estimate the cost-to-go by optimistically assuming that the navigation cost is zero, and treat the set of containers in the scene as a single container, as illustrated in Figure 3B, and apply the algorithm from [4] to obtain a plan. Our heuristic is the cost of this plan, defined as

$$h_{\text{manip}}(s) = \sum_{t=1}^{H} V_{\text{unseen}}(\tau_t^{(M)}) T(\tau_t^{(M)}) \ , \tag{4}$$

where $\tau_t^{(M)}$ is a manipulation motion. By optimality of [4], $h_{\text{manip}}(s)$ is consistent. Moreover, the plan is extremely quick to compute because [4] uses a greedy strategy when merging different parts of the plan.

When computing h_{nav}, we assume the manipulation cost is zero. In other words, when the robot visits a container key-place, it removes all objects instantly. However, unlike h_{manip} where you can simply compute manipulation motions, for h_{nav} we still need to solve a significant search problem to obtain the lower-bound of the navigation cost, which can take a significant amount of time when $|K|$ is large. Instead, we optimistically assume that the robot will move directly from the current key-place to other key-places, rather than visiting them sequentially (See Figure 3C). Further, we optimistically estimate $V_{\text{unseen}}^{(k)}$ by only considering the occluding volumes of objects at the destination key-place. So, h_{nav} is defined as

$$h_{\text{nav}}(s) = \sum_{k \in K} V_{\text{unseen}}^{(k)} \cdot T(\tau_{s,k}^{(N)}), \tag{5}$$

where $\tau_{s,k}^{(N)}$ is a navigation motion from the robot base pose in state s to the key-place k, and $V_{\text{unseen}}^{(k)}$ is the sum of all the volumes occluded by the objects in k if k is a container key-place, or the volume that will be revealed at k if it is a viewpoint key-place. If a volume is visible from multiple viewpoints, we optimistically assume that it will be revealed from the viewpoint nearest to the current position. As we assume minimum time to reveal each volume, $h_{\text{nav}}(s)$ is consistent by triangle inequality.

Because we underestimate the costs with optimistic assumptions, the heuristic function is admissible. So if we use A^* with h_{Ω^2} in the original action space defined by objects and viewpoint key-places, we would get an optimal plan in terms of the sequence of objects and viewpoint key-places. We will call this approach FLAT-Ω^2. Now we will show Ω^2-A* that uses hierarchy will yield the same optimal solution as FLAT-Ω^2. Denote FLAT-τ^* as the optimal plan found by FLAT-Ω^2, and Ω^2-τ^* as the one found by Ω^2-A*. We say containers are *independent* if a robot cannot move an object from one container to access another container, or reveal a volume in another container. We have the following theorem.

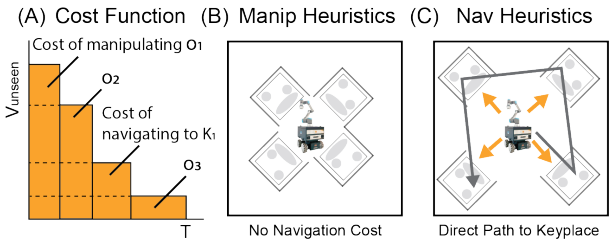


Fig. 3: **A**. Our cost function is a sum of all the edge costs for manipulating objects $O_{1,2,3}$ and navigating to a viewpoint key-place K_1. Both Eqns. 2 (sum of dotted horizontal bars) and 3 (sum of solid vertical bars) compute the area under the curve, the product of unseen volume and incurred time. **B**. Illustration of optimistic assumption for h_{manip}. We assume navigation costs are zero, so all four containers are accessible by the robot. **C**. Illustration of optimistic assumption for h_{nav}. We ignore all manipulation costs, and assume that the robot reaches each key-place with a direct path(orange arrow) from the current location, as opposed to a real path(grey arrow).

Theorem. *Assume that for any container, all objects are within the robot's reach only at the associated key-place, and that objects at different containers do not have a joint occluding volume. Then, Ω^2-A* finds an optimal sequence of objects and viewpoint key-places.*

Proof This would be true if (1) for each container, the sequence of objects in FLAT-τ^* is equivalent to the sequence of objects found by [4], and (2) the order of key-place visits in FLAT-τ^* and Ω^2-τ^* are the same. We show (1) first.

Suppose we have more than one key-places, at least two of which is a container key-place. Without loss of generality, consider a container key-place C. FLAT-τ^* and Ω^2-τ^* would be different if by visiting other key-places, the utility $\frac{V_{rev}(\tau)}{T(\tau)}$ of objects in C changes, where by [4] utility determines the sequence of objects within the container to minimize the sum of costs. But by our assumption, the occluding volumes of objects and their reachability in one container are independent of those in other containers, or revealing of a volume at a viewpoint key-place. Therefore, FLAT-τ^* and Ω^2-τ^* must have equal intra-container plan for C.

We now show that the sequence of key-places in Ω^2-τ^* is equivalent to that of FLAT-τ^*. While Ω^2 uses the set of key-places as its action space, for each action, it reduces to removing an object according to the intra-container plan found by [4]. We have established above that Ω^2-τ^* and FLAT-τ^* have the same intra-container plan for any container in the scene. And for Ω^2-A*, since it uses A* search with an admissible heuristic, it is guaranteed to find an optimal plan that merges these intra-container plans with viewpoint key-places. So, Ω^2-τ^* and FLAT-τ^* must have the same sequence of key-places. □

V. Results

A. Domain Description

To evaluate Ω^2, we perform experiments in realistic large-scale household environments with multiple rooms from

iGibson library [13] built on Pybullet [22] simulator. In each environment, we spawn at most 8 objects per each container, randomly selected from the YCBObjects dataset [23]. Each object is randomly set to a stable pose within the container, and sequentially inserted into the container until it is infeasible to insert it without colliding with existing objects. After spawning the objects, we compute the occluding volumes by raycasting from each key-place. Below is a summary of our environments:

- *Pomaria2*: BF=31 (24 objects, 3 containers and 7 viewpoints). We set $w(\tau) = 0.01$ for the viewpoint key-places. Out of 3 containers, we set $mean(w(\tau)) = 14.8$ for container 0, $mean(w(\tau)) = 9.6$ for container 1, and $w(\tau) = 1$ for container 2. This models a scenario where we have a prior belief that containers 0 and 1 contain the target object with higher probability than other containers or viewpoints.
- *Merom0*: BF=43 (32 objects, 4 containers and 11 viewpoints). We set $w(\tau) = 0.01$ for the viewpoint key-places, and $w(\tau) = 1$ for container key-places, since o^* is much more likely to be in a container.
- *Benevolence2*: BF=40 (21 objects, 3 containers and 19 viewpoints). Uses the same $w(\tau)$ as *Merom0* domain.
- *Merom1*: BF=93 (75 objects, 10 containers and 18 viewpoints). We set $w(\tau) = 0.01$ for the viewpoint key-places. Out of 10 containers, we set 3 containers with high $w(\tau)$. $mean(w(\tau)) = 28$ for container 0, $mean(w(\tau)) = 10$ for container 1, and $mean(w(\tau)) = 2.35$ for container 2. We applied $w(\tau) = 1$ for remaining containers.

To compute navigation motions, we use A* search to plan a collision-free path between each pair of key-places. For the manipulation motions, we first assume all other objects in the containers have been removed, then we compute a feasible pick motion for each object by sampling a grasp pose within the sphere around the center of mass of the object. We only consider the pick motion to be feasible if: (1) the gripper can touch the object within some distance tolerance, (2) gripper pose is collision free, (3) we can compute a valid inverse kinematics solution for that pose based on UR5 kinematics model, and (4) we can compute a feasible motion plan from the robot to the sampled grasp via bidirectional RRT.

To estimate the execution time for each motion plan, we assume every motion has constant velocity, and the corresponding motion time was estimated by dividing path length by the velocity of the robot base (navigation time) set to 0.20m/s, and the joint speed of the end-effector (manipulation time) set to $30\,\mathrm{deg}\,/s$.

B. Baselines Description

We wish to validate two hypotheses. The first is that our algorithm can compute an optimal plan by searching through navigation and manipulation actions that inter-leaves intra-container plans at different key-places. To validate this hypothesis, we compare against:

- DOGAR-GREEDY: Extension of the algorithm suggested in [4] to a mobile-manipulation setup. After computing the intra-container plans, the algorithm extends the greedy merge strategy of [4] while additionally incorporating navigation time to interleave intra-container plans and viewpoint key-place visits.
- DOGAR-TSP: Computes the intra-container plans identically as above, but uses a TSP algorithm that plans the least-cost navigation path that visits all key-places, while committing to finishing an intra-container plan at each container key-place.

The second hypothesis is that, by exploiting the hierarchical decomposition into containers and key-places and using our novel heuristic function, we can plan more efficiently. To test this, we compare against:

- LIN: Extension of [7] that performs non-hierarchical planning. We converted LIN to be an anytime algorithm via ARA* [11].
- FLAT-Ω^2: Variant of Ω^2 that uses our heuristic function without hierarchically decomposing the problem. Uses ARA*.

Table I summarizes the characteristics of all algorithms.

To evaluate these algorithms, we compare the plan quality vs. planning time, and the time to find the best solution by each algorithm. For all runs, we used an Intel i9-10900K CPU at 3.7GHz, 32Gb memory, and 1 hour compute time, after which the planner was aborted. All anytime algorithms ran with ARA* [11] using the initial weight ϵ (heuristic inflation factor) of 4.0 decremented by 0.2 at each step.

C. Experimental result

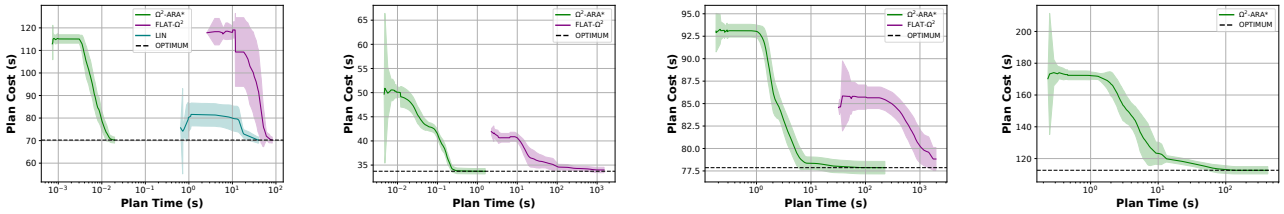
Figure 4 shows our results for anytime algorithms. Figure 4A shows that in *Pomaria2* domain with BF of 31, Ω^2-ARA* converges to an optimum in $10^{-2.57}$ seconds, while the best baseline LIN finds one in $10^{1.51}$ seconds, demonstrating a speed up by a factor of more than 10,000; FLAT-Ω^2 converges even slower, at around $10^{1.9}$ seconds. This shows that utilizing hierarchy is the key to efficient planning, which reduces BF from 31 to 10 by reducing the choice among 24 objects into choice among 3 containers.

Table II compares the results of non-anytime algorithms. It shows that Ω^2-A* finds the optimal solution in less than 0.005 seconds and while DOGAR-TSP and DOGAR-GREEDY algorithms finds solutions at comparable speeds, the costs of their best solutions are higher than the optimal cost by a factor of 2.89 and 1.67, respectively. This is because the highest utility object in one container misleads DOGAR-GREEDY to visit that container first, despite lower utility of objects in other containers. In contrast, Ω^2-A* and LIN do not suffer from this problem. Furthermore, as *Pomaria2* has high $w(\tau)$ in two containers, an optimal plan preferentially visits these containers before others. However, DOGAR-TSP chooses navigation actions to minimize the total navigation path, without accounting for manipulation costs. Due to this, DOGAR-TSP takes sub-optimal navigation sequences in *Pomaria2*, which results in a highly sub-optimal overall plan.

Figure 4B shows that in *Merom0* domain with BF of 43, Ω^2-ARA* finds the optimal solution within $10^{0.07}$ seconds

Algorithm	Planning?	Mobile Manipulation?	Use Hierarchy?	Nav-aware Heuristic?	Anytime?
DOGAR-GREEDY	X	O	O	O*	X
DOGAR-TSP	O	X	O	-	X
LIN	O	O	X	X	O*
FLAT-Ω^2	O	O	X	O	O
Ω^2-A*	O	O	O	O	X
Ω^2-ARA*	O	O	O	O	O

TABLE I: Summary of algorithms and their properties; *Planning* indicates whether the search algorithm performs planning; *Mobile Manipulation* indicates whether the search algorithm simultaneously considers navigation and manipulation costs while branching; *Use Hierarchy* column indicates whether the algorithm exploits the hierarchy of the scene; *Nav-aware Heuristic* indicates whether the algorithm incorporates navigation costs in the heuristic functions; *Anytime* indicates whether the algorithm is an anytime algorithm. (*: modified from original algorithm; -: not applicable)



(A) *Pomaria2* (BF=31) (B) *Merom0* (BF=43) (C) *Benevolence2* (BF=40) (D) *Merom1* (BF=93)

Fig. 4: Planning time (log-scale) vs. plan cost as defined in the Eqn 1. For each plot, we show the 95% confidence intervals over 10 different random seeds. The dotted line indicates the optimal solution. LIN only found a solution for *Pomaria2*.

Algorithm	*Pomaria2*		*Merom0*		*Benevolence2*		*Merom1*	
	$V^{\hat{\pi}^*}/V^{\pi^*}$	$T(\hat{\pi}^*)(s)$	$V^{\hat{\pi}^*}/V^{\pi^*}$	$T(\hat{\pi}^*)(s)$	$V^{\hat{\pi}^*}/V^{\pi^*}$	$T(\hat{\pi}^*)(s)$	$V^{\hat{\pi}^*}/V^{\pi^*}$	$T(\hat{\pi}^*)(s)$
Ω^2-A*	1.00 ± 0.02	0.00 ± 0.00	1.00 ± 0.02	0.83 ± 0.09	1.00 ± 0.01	100.30 ± 18.26	1.00 ± 0.02	214.77 ± 27.68
DOGAR-GREEDY	1.67 ± 0.06	0.00 ± 0.00	1.18 ± 0.02	0.00 ± 0.00	1.20 ± 0.01	0.00 ± 0.00	1.75 ± 0.05	0.01 ± 0.00
DOGAR-TSP	2.89 ± 0.06	0.01 ± 0.01	1.35 ± 0.04	0.01 ± 0.00	1.49 ± 0.05	0.02 ± 0.00	2.93 ± 0.02	0.03 ± 0.00

TABLE II: Comparison of the performance of the non-anytime algorithms. $V^{\hat{\pi}^*}$ denotes the cost of the best solution found by each algorithm, V^{π^*} denotes the cost of the optimal solution, and $T(\hat{\pi}^*)$ denotes the time (in seconds) to find the best solution within the time and memory limit. $V^{\hat{\pi}^*}/V^{\pi^*}$ indicates the ratio of the cost of solution found by each algorithm to the optimal cost. Each value was rounded to two decimal points. For each field, 95% confidence intervals over 10 different random seeds are additionally reported.

whereas FLAT-Ω^2 only finds a near-optimal solution within the compute budget, in $10^{2.71}$ seconds, demonstrating a speed up by a factor more than 400. This again demonstrates the efficacy of hierarchical planning, which reduces BF from 43 to 15 by reducing the choice among 32 objects into a choice among 4 containers. As *Merom0* has larger BF than *Pomaria2*, the LIN baseline failed to compute the solution within the compute budget, which also demonstrates the efficacy of the hierarchy.

Table II shows that Ω^2-A* finds the optimal solution at 0.83 seconds. DOGAR-TSP and DOGAR-GREEDY algorithms find solutions faster respectively at 0.01 seconds and less than 0.005 seconds, but again, their costs are each higher than the optimal cost by a factor of 1.35 and 1.18. This is due to the large number of navigation- and manipulation-actions, which renders DOGAR-TSP suboptimal as it does not simultaneously plan through mobile manipulation, and renders DOGAR-GREEDY suboptimal as it fully commits to a intra-container plan without being allowed to navigate to other containers. Note that compared to *Pomaria2*, the sub-optimality gap of DOGAR-GREEDY is smaller. This is because we have uniform $w(\tau) = 1.0$ for all containers in this domain, which means the myopic actions of DOGAR-GREEDY comparatively does not have as large of an effect.

Figure 4C shows that in *Benevolence2* domain with BF of 40, Ω^2-ARA* finds the optimal solution within $10^{2.02}$ seconds while FLAT-Ω^2 finds a near-optimal solution within the compute budget in $10^{3.07}$ seconds, demonstrating a speed up by a factor more than 11. LIN failed to compute the solution within the compute budget. Even though *Benevolence2* has less BF than *Merom0*, Ω^2-ARA* takes about 100 times longer time to find the optimal solution while FLAT-Ω^2 takes two times longer. This is because hierarchical planning in Ω^2 is more effective in *Merom0*, in which there are fewer viewpoint key-places and larger number of objects, which reduces to container key-place choices, than in *Benevolence2*.

Table II shows that Ω^2-A* finds the optimal solution in 100.30 seconds. While DOGAR-TSP and DOGAR-GREEDY algorithms find a solution faster, the cost of their solutions are worse than the optimal cost by a factor of 1.49 and 1.20 repsectively. Like the *Merom0* domain, this sub-optimality is because DOGAR-GREEDY is limited to single-step decisions

rather than planning ahead to optimize the overall trajectory; this leads DOGAR-GREEDY algorithm to spurious navigation actions between containers.

Figure 4D shows that in *Merom1* domain with the largest BF of 93, where Ω^2-ARA* finds the optimal solution in $10^{2.48}$ seconds. On the other hand, neither of the LIN and FLAT-Ω^2 baselines found a single sub-optimal solution within the budget. This shows the effectiveness of the hierarchical planning in large scenes, which reduces BF from 93 to 28 by reducing the choice among 75 objects into choice among 10 containers.

Table II shows that Ω^2-A* finds the optimal solution at 214.77 seconds. Even though DOGAR-TSP and DOGAR-GREEDY algorithms find a solution a lot faster at 0.03 and 0.01 seconds, the cost of their solutions are worse than the optimal cost respectively by a factor of 2.93 and 1.75. Our setup for *Merom1* is similar to *Pomaria2*, but *Merom1* has 3 times as many objects and key-places as *Pomaria_2*, resulting in a much larger BF. Due to this, more sub-optimal choices are available to both DOGAR-TSP and DOGAR-GREEDY algorithms; specifically for DOGAR-GREEDY, the sub-optimal choices also accumulates with a longer planning horizon, resulting in a more sub-optimal plan.

VI. Conclusion

In this paper, we proposed Ω^2, an optimal and efficient object search planner that simultaneously reason through mobile manipulation in a large-scale multi-roomed cluttered environment. To reduce the branching factor, we employ hierarchical planning by separating intra-container planning and interleaving intra-container plans through key place planning. In addition, we propose a novel admissible and consistent heuristic function that accounts for navigation during key place planning for focused heuristic search via A*. Lastly, we offer an anytime extension of our algorithm via ARA* [11], so that the user can trade off the planning time with plan cost based on the application.

We validated that Ω^2 finds an optimal solution within tractable planning time in environments with as many as 75 objects and 7 rooms, where other state-of-the-art baselines either find highly suboptimal solutions, find an optimal solution with orders of magnitude longer time, or fail to find a feasible solution within compute budget. In the future, we would like to address object search under uncertainties arising from perception error or partial occlusion.

ACKNOWLEDGMENT

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)) and by Republic of Korea Army Training and Doctrine Command and Ulji Army Lab at KAIST.

References

[1] A. Aydemir, K. Sjöö, J. Folkesson, A. Pronobis, and P. Jensfelt, "Search in the real world: Active visual object search based on spatial relations," in *2011 IEEE International Conference on Robotics and Automation*, 2011.

[2] A. C. Hernandez, E. Derner, C. Gomez, R. Barber, and R. Babuška, "Efficient object search through probability-based viewpoint selection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[3] C. Wang, J. Cheng, J. Wang, X. Li, and M. Q.-H. Meng, "Efficient object search with belief road map using mobile robot," *IEEE Robotics and Automation Letters*, 2018.

[4] M. R. Dogar, M. C. Koval, A. Tallavajhula, and S. S. Srinivasa, "Object search by manipulation," in *2013 IEEE International Conference on Robotics and Automation*, 2013.

[5] X. Nie, L. L. Wong, and L. P. Kaelbling, "Searching for physical objects in partially known environments," in *2016 IEEE International Conference on Robotics and Automation*, 2016.

[6] A. Kurenkov, J. Taglic, R. Kulkarni, M. Dominguez-Kuhne, A. Garg, R. Martín-Martín, and S. Savarese, "Visuomotor mechanical search: Learning to retrieve target objects in clutter," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[7] Y.-C. Lin, S.-T. Wei, S.-A. Yang, and L.-C. Fu, "Planning on searching occluded target object with a mobile robot manipulator," in *2015 IEEE International Conference on Robotics and Automation*.

[8] J. K. Li, D. Hsu, and W. S. Lee, "Act to see and see to act: POMDP planning for objects search in clutter," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.

[9] Y. Xiao, S. Katt, A. t. Pas, S. Chen, and C. Amato, "Online planning for target object search in clutter under partial observability," in *2019 IEEE International Conference on Robotics and Automation*, 2019.

[10] H. González-Banos, "A randomized art-gallery algorithm for sensor placement," in *Proceedings of the 17th annual symposium on Computational geometry*, 2001.

[11] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, 2003.

[12] E. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, 2007.

[13] B. Shen, F. Xia, C. Li, R. Martín-Martín, L. Fan, G. Wang, C. Pérez-D'Arpino, S. Buch, S. Srivastava, L. Tchapmi, M. Tchapmi, K. Vainio, J. Wong, L. Fei-Fei, and S. Savarese, "iGibson 1.0: A simulation environment for interactive tasks in large realistic scenes," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

[14] Z. Zeng, A. Röfer, and O. C. Jenkins, "Semantic linking maps for active visual object search," in *2020 IEEE International Conference on Robotics and Automation*, 2020.

[15] T. Kollar and N. Roy, "Utilizing object-object and object-scene context when planning to find things," in *2009 IEEE International Conference on Robotics and Automation*, 2009.

[16] L. Kunze, K. K. Doreswamy, and N. Hawes, "Using qualitative spatial relations for indirect object search," in *2014 IEEE International Conference on Robotics and Automation*, 2014.

[17] P. Loncomilla, J. Ruiz-del Solar, and M. Saavedra, "A bayesian based methodology for indirect object search," *Journal of Intelligent & Robotic Systems*, 2018.

[18] A. Wandzel, Y. Oh, M. Fishman, N. Kumar, L. L. Wong, and S. Tellex, "Multi-object search using object-oriented POMDPs," in *2019 IEEE International Conference on Robotics and Automation*, 2019.

[19] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Advances in Neural Information Processing Systems*, 2010.

[20] L. L. Wong, L. P. Kaelbling, and T. Lozano-Pérez, "Manipulation-based active search for occluded objects," in *2013 IEEE International Conference on Robotics and Automation*, 2013.

[21] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," in *Advances in Neural Information Processing Systems*, 2013.

[22] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, roboticsand machine learning." `http://pybullet.org`, 2016–2018.

[23] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics*, 2015.