# Deliberative Perception

Venkatraman Narayanan

CMU-RI-TR-17-67

*Submitted in partial fulfillment of the*
*requirements for the degree of*
*Doctor of Philosophy in Robotics*

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

**Thesis Committee**
Maxim Likhachev, *Chair*
Martial Hebert
Siddhartha S. Srinivasa
Manuela M. Veloso
Dieter Fox, *University of Washington*

August 2017

# Abstract

A recurrent and elementary robot perception task is to identify and localize objects of interest in the physical world. In many real-world situations such as in automated warehouses and assembly lines, this task entails localizing specific object instances with known 3D models. Most modern-day methods for the 3D multi-object localization task employ scene-to-model feature matching or regression/classification by learners trained on synthetic or real scenes. While these methods are typically fast in producing a result, they are often brittle, sensitive to occlusions, and depend on the right choice of features and/or training data.

This thesis introduces and advocates a deliberative approach, where the multi-object localization task is framed as an optimization over the space of hypothesized scenes. We demonstrate that deliberative reasoning — such as understanding inter-object occlusions — is essential to robust perception, and that discriminative techniques can effectively guide such reasoning. The contributions of this thesis broadly fall under three parts:

The first part, PErception via SeaRCH (PERCH) and its extension C-PERCH, formulates Deliberative Perception as an optimization over hypothesized scenes, and develops an efficient tree search algorithm for the same.

The second part focuses on accelerating global search through statistical learners, in the form of search heuristics (Discriminatively-guided Deliberative Perception), and by modulating the search-space (RANSAC-Trees).

The final part introduces general-purpose graph search algorithms that bridge statistical learning and search. Of these, the first is an anytime algorithm for leveraging edge validity priors to accelerate graph search, and the second, Improved Multi-Heuristic A*, permits the use of multiple, inadmissible heuristics that might arise from learning.

Experimental validation on multiple robots and real-world datasets, one of which we introduce, indicates that we can leverage the complementary strengths of fast learning-based methods and deliberative classical search to handle both "hard" (severely occluded) and "easy" portions of a scene by automatically sliding the amount of deliberation required.

# Acknowledgements

I cannot sufficiently express my gratitude for my advisor, Max, whose constant support, patience, and guidance has resulted in an enjoyable journey. I consider myself fortunate to have been your student, and for having been able to work on a number of interesting problems over the years. Our association has presented me with practical lessons in research philosophy, problem-solving, and integrity, that will stay for a long time to come.

I am thankful to my thesis committee members for their time and critique of this work. I am grateful to Sidd for the many valuable discussions and advice, to Martial, for the encouragement, and to Manuela, for all the insightful and big-picture questions. I thank Dieter for hosting me in his lab and for the collaboration from which I have learnt much. I must also thank Drew, for serving on my qualifier committee and for some of the most enriching classes at CMU.

SBPL has been my home away from home, and I am fortunate to have had wonderful lab-mates over time: Andrew, Ben, Brad, Ellis, Ishani, Fahad, Jon, John, Kalin, Kalyan, Karthik, Mike, Sameer, Sandip, Shivam, Siddharth, Sung, and Victor. I am particularly indebted to Mike for teaching me much of what I know today, and for being the best mentor a fledgling graduate student could ask for. I sincerely hope that your time is rewarded.

Thanks are due to my fellow robo-grads, Abhijeeth, Arun, Humphrey, and Karthik for their time in proof-reading manuscripts and providing feedback on practice talks at different stages of this work. I am also grateful to Varun, for being a permanent source of wisdom, and Sanjiban, for providing timely feedback and partnering in visa ordeals. I must thank the administrative staff at the RI, Peggy, Suzanne, and Rachel for always being there to help and check on, despite their incredible amounts of work.

Life in Pittsburgh has been delightful thanks to a great set of friends. Madhu, Salini, Shivram, Sreekanth, and Swati: I will cherish forever the music sessions, travels, games, potlucks, and philosophical discussions. Abhijeeth and Sidzoo, I will miss our crossword "breaks". Going back in time, I am thankful to my friends from undergrad, Brain, Padhu, Prabhu, Praha, Deepak, Guru, RT, and Vijay, for their company, and for leading me into robotics.

Finally, none of this would have been possible without the support of my family. I thank my parents, Narayanan and Gomathi, for their unconditional trust and love, my sister Jayanthi for constantly reminding me of life outside the lab, and my extended family for providing a stimulating environment to grow in.

*For Mangala Paati and the Sankarans*

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

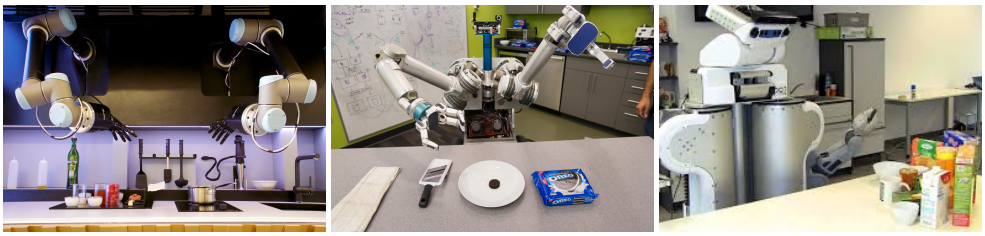
## 1.1 Motivation

(a) (b) (c)

FIGURE 1.1: Examples of object instance detection tasks in warehouse automation and flexible manufacturing settings. (a) Amazon Kiva warehouse. (b) Fetch robot in an automated warehouse. (c) Baxter robot in a flexible manufacturing setup.

Robots that interact with or monitor objects in the physical world invariably need to *perceive* the identity and whereabouts of those objects. Robotic assistants and industrial manipulators can grasp and manipulate an object of interest with a high degree of precision only when they know the object's pose (i.e, location and orientation), while autonomous cars require the poses of nearby actors—the heading of a pedestrian down the street, the precise location of a bus in the adjacent lane, and so forth—to safely and smartly interact with them. To deliver the promise of robotics and facilitate the exodus of robots from controlled laboratory environments to the real world, one would need to endow these systems with robust *perception* in the face of imperfect and high-variance sensor data.

Particularly important in settings such as automated warehouses or low-cost manufacturing is that of object *instance* detection, where a specific object needs to be localized. For example, the system is required to identify the whereabouts of a particular coffee mug, as opposed to *any* coffee mug. The specific instance that needs to be localized is often associated with a 3D model of that object, such as a CAD design, either known through manual design of the model or by performing a 3D scan on the physical instance. The instance detection problem is arguably simpler than the general perception problem (e.g, identifying all possible coffee mugs), and yet difficult to solve robustly.



(a) (b) (c)

FIGURE 1.2: Examples of object instance detection tasks in home automation. (a) Automated kitchen from Moley robotics. (b) HERB and (c) PR2 robots perceiving objects of interest.

For example, in an automated warehouse setting (Fig. 1.1), the inventory of objects is known ahead of time, and perception systems have a complete database of exact object instances they expect to encounter. Furthermore, the semi-structured nature of such warehouses might provide information on what objects are present in each shelf/bin. Despite the simplification, instance detection remains challenging: can the robot reliably localize an instance which is obscured by other ones, or an instance which has just been introduced to the warehouse? The same problems arise in low-volume and flexible automated manufacturing, where one often has knowledge of what objects or parts are expected to be seen, but precise localization and association is not available.

The task of object instance recognition and localization appears in home automation as well (Fig. 1.2). Robots that operate in known environments—i.e, environments which they have been familiarized to, often interact with the same object instances in different scenarios. As an example, a household robot operating in the kitchen might need to identify and localize a particular cutlery item to grasp and use it every time such a need arises.

This thesis focuses on developing robust and efficient approaches for a variety of object instance detection tasks, studying their theoretical properties, and evaluating them on real-world examples.

## 1.2 Conventional Approaches

Traditional methods for object instance detection have relied on matching hand-coded feature descriptors between the observed scene and 3D models, with recent data-driven methods permitting automated learning of those feature descriptors. While these methods, broadly classified as *discriminative*, provide attractive test-time speeds, they remain brittle in practice despite numerous variants that have shown promise. As an example, consider the scene in Fig. 1.3, where we need to identify and localize the Elmers' glue bottle, which is almost completely occluded by the shelf. Methods that employ feature correspondence matching fare poorly as key feature descriptors could be lost due to occlusion by the shelf, whereas learning-based methods could suffer as

(a)

(b)


(c)

FIGURE 1.3: A typical warehouse automation task where the robot needs to identify, grasp, and move a target item from the shelf to a tote. Here, the system has access to a list of objects in the bin, their 3D models, as well as the model of the shelf. (a) Carnegie Mellon University's robotic system in the 2016 Amazon Picking Challenge, employing techniques developed in this thesis. (b) An image of a single bin in the shelf from which we are required to identify and localize the Elmer's glue bottle, marked by a red bounding box. (c) An image of the Elmer's glue bottle that needs to be localized.

they might have not seen a similar training instance where only such a small portion of the object is visible.

Multi-object instance localization, the problem of identifying and localizing multiple objects in a scene, is particularly challenging for discriminative methods. The training data required to capture subtleties arising from inter-object occlusions scales exponentially with the number of objects that can be simultaneously present in a scene. Modern-day systems typically ignore the combinatorial aspect of this problem, and resort to post-hoc fine-tuning instead (Aldoma et al., 2012c).

### 1.2.1 Case in Point: The Amazon Picking Challenge

The persisting challenges of multi-object instance detection are perhaps best captured in a survey of the recently concluded 2015 Amazon Picking Challenge (Correll et al., 2016), where participants ranked perception as the most difficult aspect of the overall system, despite having *a priori* knowledge of exact object instances. Our own participation in the 2016 and 2017 editions of this challenge (picture of our system shown in Fig. 1.3) and the lessons learned second the survey results. While the most popular perception techniques used in this challenge were heavily reliant on learning-based methods, the performance of these systems was shown to degrade quite rapidly in the presence of occlusions (Zeng et al., 2017).

Let us now go back to the example scene in Fig. 1.3. Knowing that the objects are in a shelf bin, and that there are five specific objects allows us to jointly reason about or *deliberate* on the occlusion caused by the shelf and the positions of the other objects to infer the exact pose of the Elmer's glue bottle. Such a "deliberative reasoning" approach not only appears to address the problems faced by discriminative matching, but is also appealing from a computational standpoint; computers are adept at searching through millions of hypotheses, which in this case are possible explanations of the scene that result in the observed image. This deliberative reasoning forms the crux of our work.

## 1.3 Proposed Approach

This thesis proposes and advocates a deliberative approach to instance detection with model-based search as its central tenet. We argue and demonstrate that explicit reasoning over the physical structure of the problem (e.g., modeling inter-object occlusions) is essential for robust multi-object instance detection, study the tractability of such an approach, and provide principled tools for an efficient implementation of deliberative perception.

Consider an analogy with classical AI. Successes such as computer chess, and recently Go, can be attributed to the ability of computers to search through millions of hypothetical scenarios (i.e, *deliberate*), while using "discriminative" heuristics to focus the search on likely ones. A very recent example is that of Alpha Go (Silver et al., 2016), which guides Monte Carlo Tree Search with convolutional neural networks to achieve impressive results. State-of-the-art methods in machine perception however, rely solely on discriminative systems that match observed sensor data to training instances. Drawing parallels between the computer chess and machine perception problems, we argue that while powerful discriminative methods and heuristics are useful in obtaining a good "guess", they must be employed by a larger system that can systematically search through the "generative" state space.

In the context of multi-object pose estimation, an approach that can simultaneously perform global reasoning and be guided by discriminative patterns needs to employ two seemingly disparate AI tools: classical search techniques for combinatorial reasoning, and statistical learning for discriminative matching. Consequently, the thesis draws from both camps and takes a neoclassical approach to develop algorithms that integrate learning-based techniques seamlessly into search.

## 1.4 Thesis Overview

Chapter 2 reviews the related work in the field of object instance recognition, and presents a primer on graph search. The following chapters then present the core contributions of this thesis, and can be grouped under three parts:

1. Part I introduces and formalizes Deliberative Perception. Chapter 3 sets up the problem of multi-object instance localization as that of model-based search over the space of possible scenes that can be observed given the 6 DoF sensor pose and 3D models of objects. The presented algorithm, PERception via SearCH (PERCH) (Narayanan and Likhachev, 2016a), exploits structure in the optimization objective to cast the problem as one of tree search. In addition, this chapter introduces a notion of *completeness* for the multi-object localization problem. Chapter 4 presents an extension of PERCH to scenes containing extraneous clutter for which 3D models are not available (Narayanan and Likhachev, 2017a). Further, it discusses several algorithmic and implementation optimizations to improve the computational efficiency of PERCH.

2. Part II deals with the integration of learning-based approaches and global search. Discriminatively-guided Deliberative Perception, abbreviated D2P (Narayanan and Likhachev, 2016b) is introduced in Chapter 5 for leveraging arbitrary and multiple discriminative methods as heuristics in guiding search. Chapter 6 addresses the curse of dimensionality that arises in 6 DoF object pose estimation. We explore the use of discriminative learners in directly constructing a sampling-based search tree that we name RANSAC-Tree. This complements Chapter 5, where learners are used exclusively as heuristics in guiding a predefined implicit search tree.

3. Part III presents the core graph search algorithms that connect learning and search. Chapter 7 delves into graph search problems where edge evaluations are time-consuming, as is the case in this thesis. A novel anytime algorithm formulation (Narayanan and Likhachev, 2017b) is presented, with a definition for anytime algorithm "optimality". Finally, Chapter 8 presents the Improved Multi-Heuristic A* (Narayanan, Aine, and Likhachev, 2015) algorithm for leveraging multiple, arbitrary functions as heuristics to guide search while preserving formal guarantees on solution quality. The algorithms presented in both these chapters are applicable to arbitrary graphs, and we take some digressions here

to demonstrate their generality through evaluation on robot motion planning domains.

The algorithms introduced in these chapters provide attractive theoretical guarantees on completeness and solution quality. The implication of solution quality guarantees for practitioners is that it delineates cost-function design from algorithmic efficiency improvements, thereby providing a clear handle on the accuracy-efficiency tradeoff.

Chapter 9 concludes with a summary of contributions, discussion, and avenues for future work.

**Note for Readers:**

Although the primary contribution of the thesis is in the area of 3D perception, associated contributions and the related work that it builds on span multiple fields including 3D perception, classical AI and search, machine learning, and motion planning. The thesis is structured such that the first two parts are accessible to the reader with a more extensive background in perception than heuristic search, whereas readers coming from a classical AI or planning background would find the final part self-contained.

# Chapter 2

# Background

We first review the literature on object instance detection from 3D sensor data, and then provide a refresher on heuristic search—the tool of choice in this thesis.

## 2.1 Object Instance Detection

### 2.1.1 Local and Global 3D Feature Descriptors

Model-based object recognition and localization in the present 3D era falls broadly under two approaches: *local* and *global* recognition systems. The former class of algorithms operate in a two step procedure: i) compute and find correspondences between a set of local shape-preserving 3D feature descriptors on the model and the observed scene and ii) estimate the rigid transform between a set of geometrically consistent correspondences. A final, optional and often used step is to perform a fine-grained local optimization to align the model to the scene and obtain the pose. Examples of local 3D feature descriptors range from Spin Images (Johnson and Hebert, 1999) to Fast Point Feature Histograms (FPFH) (Rusu, Blodow, and Beetz, 2009), whereas final alignment procedures include Iterative Closest Point (ICP) (Chen and Medioni, 1991) and Bingham Procrustrean Alignment (BPA) (Glover and Popovic, 2013). The survey paper by Aldoma et al. (Aldoma et al., 2012a) provides a comprehensive overview of other local approaches.

The second, *global* recognition systems employ a single-shot process for identifying object type and pose jointly. Global feature descriptors encode the notion of an object and capture shape and viewpoint information jointly in the descriptor. These approaches employ a training phase to build a library of global descriptors corresponding to different observed instances (e.g., each object viewed from different viewpoints) and attempt to match the descriptor computed at observation time to the closest one in the library. Additionally, unlike the local methods, global ones require points in the observed scene to be segmented into different clusters, so that descriptors can be computed on each object cluster separately. Some of the global recognition systems include Viewpoint Feature Histogram (VFH) (Rusu et al., 2010), Clustered Viewpoint Feature

Histogram (CVFH) (Aldoma et al., 2011), OUR-CVFH (Aldoma et al., 2012b), Ensemble of Shape Functions (ESF) (Wohlkinger and Vincze, 2011), and Global Radius-based Surface Descriptors (GRSD) (Marton et al., 2011). Other approaches to estimating object pose include local voting schemes (Drost et al., 2010) or template matching (Hinterstoisser et al., 2013) to first detect objects, and then using global descriptor matching or ICP for pose refinement.

Finally, a number of recent works have employed feature-descriptors, either manually specified or learnt from data, to build complete vision pipelines for RGB-D data. Kevin Lai's thesis work (Lai, 2014) was amongst the first to explore the use of RGB-D data for robotic vision, and focused on efficient learning-algorithms for joint category, instance and pose estimation from RGB-D images. The MOPED system (Collet, Martinez, and Srinivasa, 2011) introduced a novel Iterative Clustering-Estimation for scoring object hypothesis and extended its usage to RGB-D images Fouhey et al., 2012 by developing techniques for handling missing depth data.

Although both local and global feature-based approaches have enjoyed popularity owing to their speed and intuitive appeal, they suffer when used for identifying and localizing multiple objects in the scene. The limitation is perhaps best described by the following lines from the book by Stevens and Beveridge (Stevens and Beveridge, 2000b):

> "*Searching for individual objects in isolation precludes explicit reasoning about occlusion. Although the absence of a model feature can be detected (i.e., no corresponding data feature), the absence cannot be explained (why is there no corresponding data feature?). As the number of missing features increase, recognition performance degrades*".

Global verification (Aldoma et al., 2012c; Aldoma et al., 2013; Doumanoglou et al., 2016) and filtering (Figueiredo et al., 2013) approaches aim to address the occlusion problems faced by feature-based methods through a joint optimization procedure over candidate object poses. However, these are restricted by the fact that initial predictions for object poses are provided by a system that does not model occlusion. In this thesis, we aim to explicitly reason about the interactions between multiple objects in the observed data by hypothesizing or rendering scenes, thereby overcoming the limitations of methods that train on single-object instances.

### 2.1.2 Generative Approaches

The idea of using search to "explain" scenes was popular in the early years of 2D computer vision: Goad (Goad, 1987) promoted the idea of treating feature matching between the observed scene and 3D model as a constrained search while Lowe (Lowe, 1987) developed and implemented a viewpoint-constrained feature matching system.

Grimson (Grimson and Lozano-Perez, 1987) introduced the Interpretation Tree to systematically search for geometrically-consistent matches between scene and model features, while using various heuristics to speed up search. Our work is also based on a search system, but it differs from the aforementioned works in that the search is over the space of full hypothesized/rendered scenes and not feature correspondences. In fact, our proposed approach can operate without any feature descriptors at all.

Several approaches (Hsiao and Hebert, 2014; Xiang and Savarese, 2013; Meger et al., 2011) have been proposed to incorporate explicit occlusion reasoning in vision systems. A common motif to these methods is the notion of an "occlusion prior" that is statistically learnt from data. These priors are integrated with an object detector to improve performance in cluttered scenes. Broadly speaking, these approaches are *top-down* in that they use occlusion reasoning as a "soft"-prior on top of existing object detectors. On the other hand, the use of exact 3D models in our proposed approach enables a completely *bottom-up* model-based search. Yet another related generative approach is the scene parsing system described by Hager and Wegbreit (Hager and Wegbreit, 2011). While the authors formulate the scene explanation problem as an optimization-based one similar to our approach, the focus of their work is on capturing scene dynamics and evolution of the scene over time as objects get removed and added. Further, their work assumes simple initialization of object poses based on geometric primitives for the objects. On the other hand, this thesis concentrates on the multi-object pose estimation problem for static scenes with arbitary objects, and develops efficient algorithms for the same.

The philosophy of the Render, Match and Refine (RMR) approach proposed by Stevens and Beveridge (Stevens and Beveridge, 2000a) is congruent with ours. RMR explicitly models interaction between objects by rendering the scene and uses occlusion data to inform measurement of similarity between the rendered and observed scenes. It then uses a global optimization procedure to iteratively improve the rendered scene to match the observed one. Our proposed approach, although similar in philosophy, is different in several ways: First, the optimization objective we use to compare the rendered and observed scene is based purely on 3D sensor data, as opposed to the 2D edge-feature and per-pixel depth differences used in RMR that make it vulnerable to the pitfalls of feature-based methods. Second, our optimization objective can be decomposed over the objects in the scene, thereby overcoming an intractable exhaustive search over the joint object poses. Finally, our model-based search framework lends itself to powerful heuristic search tools that can leverage discriminative methods for guidance.

### 2.1.3 Learning-based Approaches

Convolutional Neural Networks (CNNs) have revolutionized the field of object detection in RGB images through excellent representation learning capabilities (Krizhevsky, Sutskever, and Hinton, 2012). Consequently, recent works in RGB-D object detection

have also focused on using deep learning (Schwarz, Schulz, and Behnke, 2015; Eitel et al., 2015; Wu et al., 2015) for automatic representation learning. In a related work, Krull et al. (2015) train a CNN to compare rendered and observed depth images. Despite the promise shown, deep learning methods by themselves are ill-suited for multi-object instance detection problems since the required training data is combinatorial in the number of objects. Our proposed approach however, can use discriminative learners such as CNNs as heuristics in guiding model-based search (Chapter 5), thereby shifting responsibility from discrimination to deliberation.

## 2.2 Heuristic Search

Heuristic Search is the core of several planning algorithms, in robotics and beyond. Problems that can be cast as finding cost-minimizing paths in graphs or trees lend themselves to a rich toolbox of heuristic search algorithms. For example, in the field of robot motion planning where the task is to find collision-free paths (often with some cost-minimization objective), the problem can be cast as graph search. Such an approach has been used with success in a variety of robot motion planning problems: planning for autonomous navigation (Likhachev and Ferguson, 2008), navigation in dynamic environments (Phillips and Likhachev, 2011; Narayanan, Phillips, and Likhachev, 2012), navigation under topological constraints (Vernaza, Narayanan, and Likhachev, 2012; Narayanan et al., 2013), manipulation planning for robotic arms (Cohen, Chitta, and Likhachev, 2010; Cohen, Phillips, and Likhachev, 2014) and navigation for aerial vehicles (MacAllister et al., 2013; Butzke et al., 2016). Since this thesis makes use of, and contributes to the field of heuristic search, we provide a brief primer on the topic.

### 2.2.1 A* Search

A* (Hart, Nilsson, and Raphael, 1968) is a widely used graph search algorithm for finding shortest paths in graphs. It takes as input a graph $G(S, E)$ with states $s \in S$, an edge-cost function $c : S \times S \rightarrow \mathbb{R}^+$ that maps every edge to a strictly positive cost , a start state $s_{start}$ and a goal $s_{goal}$. The output is a path (sequence of edges) from $s_{start}$ to $s_{goal}$ that has the smallest possible cost accumulated over the edges. A*'s efficiency arises from a *heuristic* function $h : S \rightarrow \mathbb{R}$ that estimates the cost-to-goal for any given state $s$ in the graph. A* is guaranteed to find the optimal (cost-minimum) path only if the heuristic is *admissible*: i.e, it does not overestimate the cost-to-goal for any state $s$.

#### Algorithm

The algorithm (Alg. 1) assigns three values to each state in the graph: $g(s)$, the cost of the best known path from $s_{start}$ to $s$; $h(s)$, the heuristic estimate for state $s$; and

**Algorithm 1** A* Search

```
1: procedure EXPANDSTATE(s)
2:     Remove s from OPEN
3:     for all s' ∈ SUCCESSORS(s) do
4:         if s' was not seen before then
5:             g(s') ← ∞
6:         if g(s') > g(s) + c(s, s') then
7:             g(s') ← g(s) + c(s, s')
8:             Insert/Update s' in OPEN with priority f(s') = g(s') + h(s')
9: procedure A*(s_start, s_goal, c, h)
10:    OPEN ← ∅
11:    g(s_start) ← 0
12:    f(s_start) ← h(s_start)
13:    Insert s_start in OPEN with priority f(s_start)
14:    while s_goal not expanded do
15:        if OPEN.EMPTY() then return null
16:        s ← OPEN.TOP()                    ▷ State with smallest f(s) in OPEN
17:        EXPANDSTATE(s)
18:    return RECONSTRUCTPATH(s_goal)
```

$f(s) = g(s) + h(s)$, an estimate of the best path cost from $s_{start}$ to $s_{goal}$ through state $s$. Initially, $g(s)$ is set to $0$ for $s_{start}$, and $\infty$ for all other states (this is often done *lazily*—as and when states are "discovered" for the first time). During its operation, A* repeatedly *expands* a state $s$ with the smallest $f(s)$ value until $s_{goal}$ is about to be expanded. There are two steps in expanding a state $s$. First, we look at the successor states for $s$ and update their $g$-values if they can be reached via $s$ through a better-cost path than the currently known path. Second, we mark $s$ as expanded. These steps are typically implemented by maintaining an OPEN list of not-yet-expanded states sorted by $f(s)$ values. Once the search terminates, we can reconstruct the solution path by greedily following the best predecessor (defined by lowest $g$-value) from $s_{goal}$ recursively. A* is guaranteed to not expand a state more than once if the heuristic is *consistent*—i.e, it satisfies the triangle inequality $h(s) \leq h(s') + c(s, s')$, $\forall s, s' \in S$ and $h(s_{goal}) = 0$. Note that consistency implies admissibility, but not vice versa. Finally, A* is *complete*: it finds a solution if one exists, or reports failure in finite time if there is no path from start to goal.

### 2.2.2 Variants of A*

A fundamental limitation of optimal heuristic search techniques such as A* (Hart, Nilsson, and Raphael, 1968) is their prohibitively large computation time in high-dimensional state spaces—i.e, large graphs that result from high-dimensional state representations. Consequently, much of the recent research in applying heuristic search for high-dimensional problems has focused on achieving practical computation times by settling for bounded suboptimality. These work by inflating the heuristic, as in

Weighted A* (Pohl, 1970; Likhachev, Gordon, and Thrun, 2004), or using effort estimates (an estimate of how many expansions are needed to terminate) as in Bounded Suboptimal Search (Thayer and Ruml, 2011; Hatem and Ruml, 2014). Weighted A* works identically to A* search, except that the heuristic is inflated by a factor $w \geq 1$ to make the search more goal-directed. It can be shown that using a consistent heuristic yields solutions that are within $w$ of the optimal solution (Pohl, 1970), and that no state in the graph needs to be expanded more than once (Likhachev, Gordon, and Thrun, 2004).

The notion of *anytime* search is another pertinent concept to this thesis. An anytime algorithm is one that finds an initial solution quickly although the found solution might be suboptimal, and continues to improve the quality of the solution given more time. For instance, Anytime A* (Hansen and Zhou, 2007) and Anytime Repairing A* (Likhachev, Gordon, and Thrun, 2004) find initial solution paths quickly by using a Weighted A* search with large $w$, and continue searching (using different schemes) for improving the solution quality over time.

In this thesis, we make two core contributions to the heuristic search literature. The first one is a technique that can simultaneously utilize multiple inadmissible heuristics to efficiently search high-dimensional state spaces. This is motivated by the observation that the performance of A*-like algorithms is dictated strongly by the quality of the heuristic used—i.e, how correlated is the heuristic with the true "cost-to-go". Details of these techniques, their theoretical properties and applicability to arbitrary domains are presented in Chapter 8. The second one is an anytime search algorithm for graphs where evaluating the existence of an edge is time-consuming. This algorithm introduces a novel criterion for the design of an anytime algorithm, and derives an "optimal" anytime algorithm for edge evaluation.

# Part I

# Foundations

# Chapter 3

# PERCH: Perception via Search for Multi-Object Instance Recognition

This chapter introduces the problem tackled by this thesis and the model-based search framework that will be employed in the remaining chapters. We begin with a formal description of the task and the assumptions made.

## 3.1 Setup

The problem statement is as follows: given 3D models of $N$ unique rigid objects, a point cloud ($I$) (equivalently, a depth-image) of a scene containing $K \geq N$ objects (possibly containing replicates of the $N$ unique objects), and the 6 DoF pose of the sensor, we are required to find the 3 DoF pose $(x, y, \theta)$ of each of the $K$ objects in the scene. While rigid objects are fully characterized by their 6 DoF poses, we will assume for the time being that they are constrained in their other degrees of freedom: roll, pitch and translation along the $z$-axis. This assumption is reasonable in settings where objects rest on a supporting plane, such as in tabletop manipulation or warehouse bin picking. Further, if we treat distinct stable configurations of an object (typically much fewer in number than all possible 6 DoF poses) in the absence of other objects as unique models, the 3 DoF assumption is not as restrictive. Nevertheless, we will discuss approaches to handle full 6 DoF objects in Chapter 6.

We make the following assumptions:

- The number ($K$) and type of objects in the scene are known ahead of time (but not the correspondences themselves).
- The objects in the scene vary only in position $(x, y)$ and yaw $(\theta)$—3 DoF, with respect to their 3D models.

This chapter is based on material from Venkatraman Narayanan and Maxim Likhachev (2016a). "PERCH: Perception via Search for Multi-Object Recognition and Localization". In: *ICRA*. IEEE.

TABLE 3.1: Symbols and Notation.

| | |
|---|---|
| $\boldsymbol{I}$ | The input point cloud |
| $K$ | The number of objects in the scene |
| $N$ | The number of unique objects in the scene ($\leq K$) |
| $O$ | An object state specifying a unique ID and 3 DoF pose |
| $\boldsymbol{R_j}$ | Point cloud corresponding to the rendering of a scene with $j$ objects $O_{1:j}$ |
| $\boldsymbol{\Delta R_j}$ | Point cloud containing points of $R_j$ that belong exclusively to object $O_j$: $\boldsymbol{\Delta R_j} = \boldsymbol{R_j} - \boldsymbol{R_{j-1}}$ |
| $V(O_j)$ | The set of points in the volume occupied by object $O_j$. |
| $V_j$ | The union of volumes occupied by objects $O_{1:j}$ |

- The input point cloud/depth image can be preprocessed (supporting plane, background filtered etc.) such that the points in it belong to the objects of interest only.
- We have access to the intrinsic parameters of the sensor, so that we can render scenes using the available 3D models.

We specifically note that we *do not* make any assumptions about the ability to "cluster" points into different object groups as is done by most global 3D object recognition methods such as the Viewpoint Feature Histogram (VFH) (Rusu et al., 2010).

## 3.2 Notation

Throughout this thesis, we will use the following notation:

- $O$: An object state characterized by $(\text{ID}, x, y, \theta)$, the unique object ID, position and yaw.
- $\boldsymbol{I}$: The input/observed point cloud from the depth sensor.
- $\boldsymbol{R_j}$: A point cloud generated by rendering a scene containing objects $O_1, O_2, \ldots, O_j$.
- $\boldsymbol{p}$: A point $(x, y, z)$ in any point cloud.
- $\boldsymbol{\Delta R_j} = \boldsymbol{R_j} - \boldsymbol{R_{j-1}}$, the point cloud containing points in $\boldsymbol{R_j}$ but not in $\boldsymbol{R_{j-1}}$. In other words, the set of points belonging to object $O_j$ that would be visible given the presence of objects $O_1, O_2, \ldots, O_{j-1}$.
- $V(O_j)$: The set of all points in the volume occupied by object $O_j$. When it is not possible to compute this in closed form, this can be replaced by an admissible/conservative approximation, for example, the volume of an inscribed cylinder.
- $V_j = \cup_{i=1}^{j} V(O_i)$, the union of volumes occupied by objects $O_1, O_2, \ldots, O_j$.

We use upper-case bold-faced letters to denote point clouds (set of points in $\mathbb{R}^3$), and lower-case bold-faced letters to denote a point in $\mathbb{R}^3$. Table 3.1 summarizes the notation used.

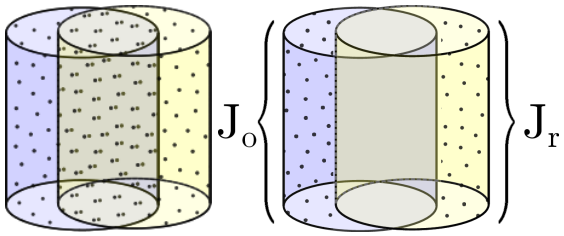



FIGURE 3.1: Illustration showing the computation of the explanation cost. The figure on the left shows the superposition of the observed point cloud (in blue) and the rendered point cloud (in yellow) of a cylindrical object. Object boundaries and volumes are shown merely for illustration. The total explanation cost (see figure on the right) is the number of unexplained points in the observed point cloud ($J_o$) and the number of unexplained points in the rendered point cloud ($J_r$).

## 3.3 Optimization Formulation

We formulate the problem of identifying and obtaining the 3 DoF poses of objects $O_1, O_2, \ldots, O_K$ as that of finding the minimizer of the following *explanation cost*:

$$J(O_{1:K}) = \underbrace{\sum_{\boldsymbol{p} \in \boldsymbol{I}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{R_K})}_{J_{observed}(O_{1:K}) \text{ or } J_o} + \underbrace{\sum_{\boldsymbol{p} \in \boldsymbol{R_K}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I})}_{J_{rendered}(O_{1:K}) \text{ or } J_r} \tag{3.1}$$

in which $\textsc{Outlier}(\boldsymbol{p}|\boldsymbol{P})$ for a point cloud $\boldsymbol{P}$ and point $\boldsymbol{p}$ is defined as follows:

$$\textsc{Outlier}(\boldsymbol{p}|\boldsymbol{P}) = \begin{cases} 1 & \text{if } \min_{\boldsymbol{p'} \in \boldsymbol{P}} \|\boldsymbol{p'} - \boldsymbol{p}\|_2 > \delta \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

for some sensor noise threshold $\delta$. We will use the notation $J_o$ and $J_r$ to refer to the observed and rendered explanation costs respectively.

The explanation cost essentially counts the number of points in the observed scene that are not explained by the rendered scene and the number of points in the rendered scene that cannot be explained by the observed scene. While it looks simplistic, the cost function forces the rendering of a scene that as closely explains the observed scene as possible, from both a 'filled' (occupied) and 'empty' (negative space) perspective. Figure 3.1 illustrates the computation of the 'explanation cost'. Another interpretation for the explanation cost is to treat it as an approximation of the difference between the union volume and intersection volume of the objects in the observed and rendered scenes.

In the ideal scenario where there is no noise in the observed scene and where we have access to a perfect renderer, we could do an exhaustive search over the joint object

poses to obtain a solution with zero cost. However, this naive approach is a recipe for computational disaster: even when we have only 3 objects in the scene and discretize our positions to 100 grid locations and 10 different orientations, we would have to synthesize/render $10^9$ scenes to find the global optimum. This immediately calls for a better optimization scheme, which we derive next.

## 3.4 Monotone Scene Generation Tree

### 3.4.1 Construction

The crux of our algorithm exploits the insight that the explanation cost function can be decomposed over the set of objects in the scene. To see this, we first note that the rendered scene containing $K$ objects, $R_K$ can be incrementally constructed:

$$\boldsymbol{R_K} = \cup_{i=1}^{K} \boldsymbol{\Delta R_i} \qquad \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i}$$

where $\Delta \boldsymbol{R_i} = \boldsymbol{R_i} - \boldsymbol{R_{i-1}}$ and $\boldsymbol{R_0}$ is assumed to be an empty point cloud. The constraint $\boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i}$ translates to saying that the addition of a new object to the scene does not 'occlude' the existing scene, thereby guaranteeing that every point in $\boldsymbol{R_{i-1}}$ exists in $\boldsymbol{R_i}$ as well. In other words, the number of points in the rendered point cloud can only increase with the addition of a new object. The above constraint implicitly assumes that the scene does not contain objects which can simultaneously occlude an object and also be occluded by another object, such as horseshoe-shaped objects[1]. Using the above, we can write the rendered explanation cost as follows:

$$\begin{aligned} J_r &= \sum_{\boldsymbol{p} \in \boldsymbol{R_K}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I}) \\ &= \sum_{i=1}^{K} \sum_{\boldsymbol{p} \in \boldsymbol{\Delta R_i}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I}) \qquad \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i} \end{aligned}$$

We then similarly decompose the observed explanation cost:

$$\begin{aligned} J_o &= \sum_{\boldsymbol{p} \in \boldsymbol{I}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{R_K}) \\ &= \sum_{\boldsymbol{p} \in \boldsymbol{I}} \prod_{i=1}^{K} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{\Delta R_i}) \qquad \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i} \\ &= \sum_{i=1}^{K} \sum_{\boldsymbol{p} \in \{\boldsymbol{I} \cap V(O_i)\}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{\Delta R_i}) \\ &\quad + \sum_{\boldsymbol{p} \in \{\boldsymbol{I} - V_K\}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{R_K}) \qquad \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i} \end{aligned}$$

[1] In theory, we could still handle such objects by decomposing them into multiple surfaces that satisfy the non-occlusion constraint. We omit the details for simplicity of explanation.

With the above decompositions, we can re-write the overall optimization objective as:

$$J(O_{1:K}) = \sum_{i=1}^{K} \Delta J^i \qquad \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i} \tag{3.3}$$
$$= \sum_{i=1}^{K} \Delta J_r^i + \Delta J_o^i \qquad \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i}$$

where

$$\Delta J_r^i = \sum_{\boldsymbol{p} \in \boldsymbol{\Delta R_i}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I})$$
$$\Delta J_o^i = \sum_{\boldsymbol{p} \in \{\boldsymbol{I} \cap V(O_i)\}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{\Delta R_i}) + \text{residual}(i)$$
$$\text{residual}(i) = \begin{cases} \sum_{\boldsymbol{p} \in \{\boldsymbol{I} - V_K\}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{R_K}) & \text{if } i = K \\ 0 & \text{otherwise} \end{cases}$$

Equation 3.3 defines a pairwise-constrained optimization problem, the constraint being that the assignment of the $i^{\text{th}}$ object does not occlude the scene generated by the assignment of the previous objects 1 through $i-1$. A natural way to solve this problem is to construct a tree that satisifies the required constraint, and assigns the object poses in a sequential order. This is precisely our approach and the resulting tree we construct is called the Monotone Scene Generation Tree (MSGT), with 'monotone' emphasizing that as we go down the tree, newly added objects cannot occlude the scene generated thus far (Fig. 3.2). We note that while a particular configuration of objects can be generated by choosing different assignment orders, only one is sufficient to retain as all those configurations have identical explanation costs. Thus, we obtain a tree structure instead of a Directed Acyclic Graph (DAG). Formally, any vertex/state in the MSGT is a partial assignment of object states: $s = \{O_{1:j}\}$, with $j \leq K$. For a MSGT state $s$ with an assignment of $j$ objects, the implicit successor generation function and the associated cost are defined as follows:

$$\textsc{Succ}(s) = \{s' | s' = s \cup O_{j+1} \wedge \boldsymbol{R_j} \subseteq \boldsymbol{R_{j+1}}\} \tag{3.4}$$
$$\textsc{c}(s, s') = \Delta J^{j+1} = \Delta J_r^{j+1} + \Delta J_o^{j+1} \tag{3.5}$$

The root node of the tree $s_{\text{root}}$ is an empty state containing no object assignments, while a goal state is any state $s$ that has an assignment for all objects. Given the MSGT construction, the multi-object localization problem reduces to that of finding the cheapest-cost path in the tree from the root state to any goal state.

#### A Note on the Cost Function

While we choose a relatively simple outlier-based cost function in this work, a larger class of sophisticated cost functions that accurately model various aspects of sensor

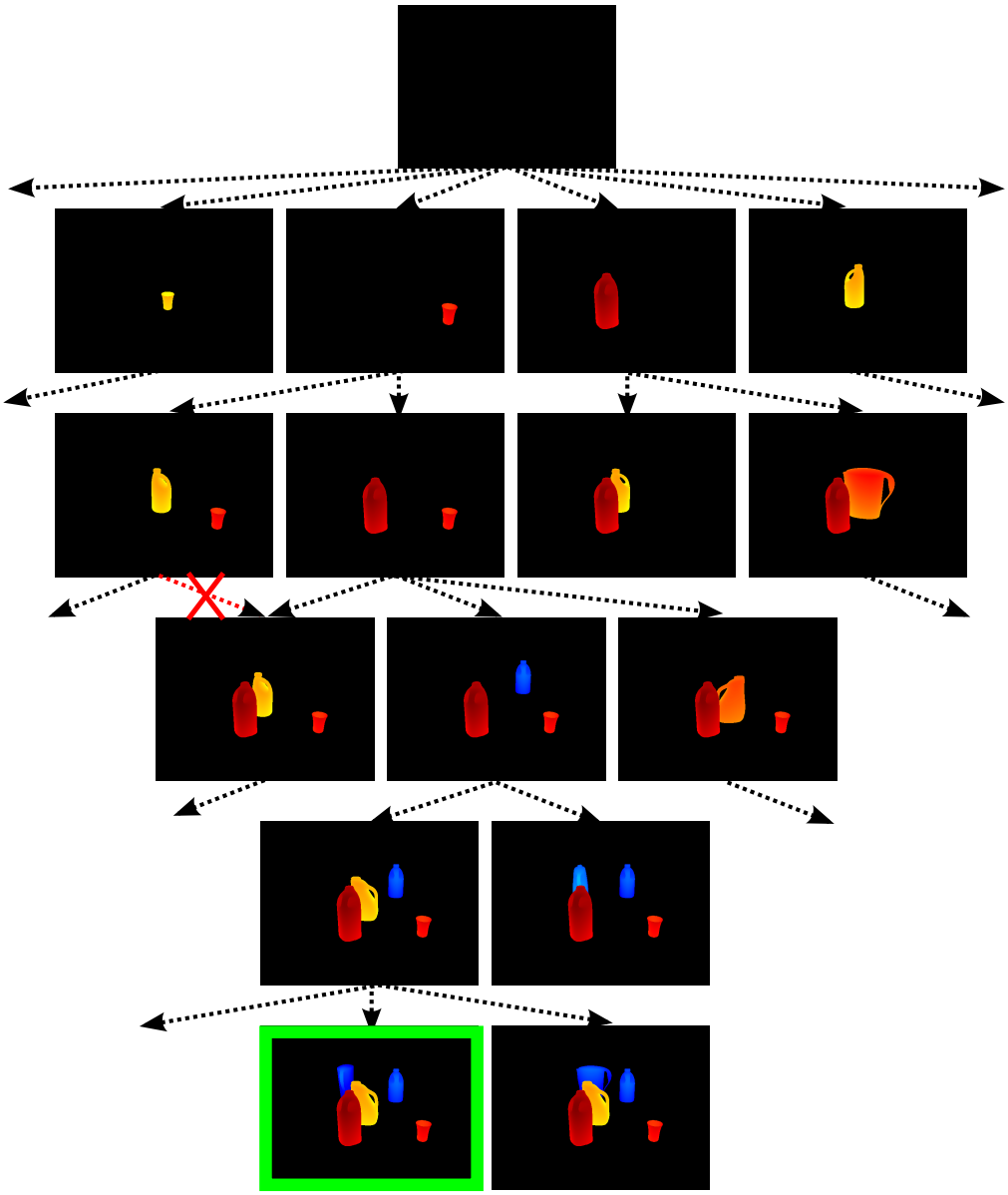


FIGURE 3.2: Portion of a Monotone Scene Generation Tree (MSGT): the root of the tree is the empty scene, and new objects are added progressively as we traverse down the tree. Notice how child states never introduce an object that occludes objects already in the parent state. A counter-example (marked by the red cross) is also shown. Any state on the $K^{\text{th}}$ level of the tree is a goal state, and the task is to find the one that has the lowest cost path from the root—marked by a green bounding box in this example.

noise can be used. The main constraint on the choice of cost function is that it is decomposable—a constraint easily satisfied by any "local" cost-function. An example of a non-decomposable cost function would be a color-difference cost, which depends on all objects in the scene since lighting and shadow effects are now inter-dependent. This prevents an object-wise decomposition of the cost function.

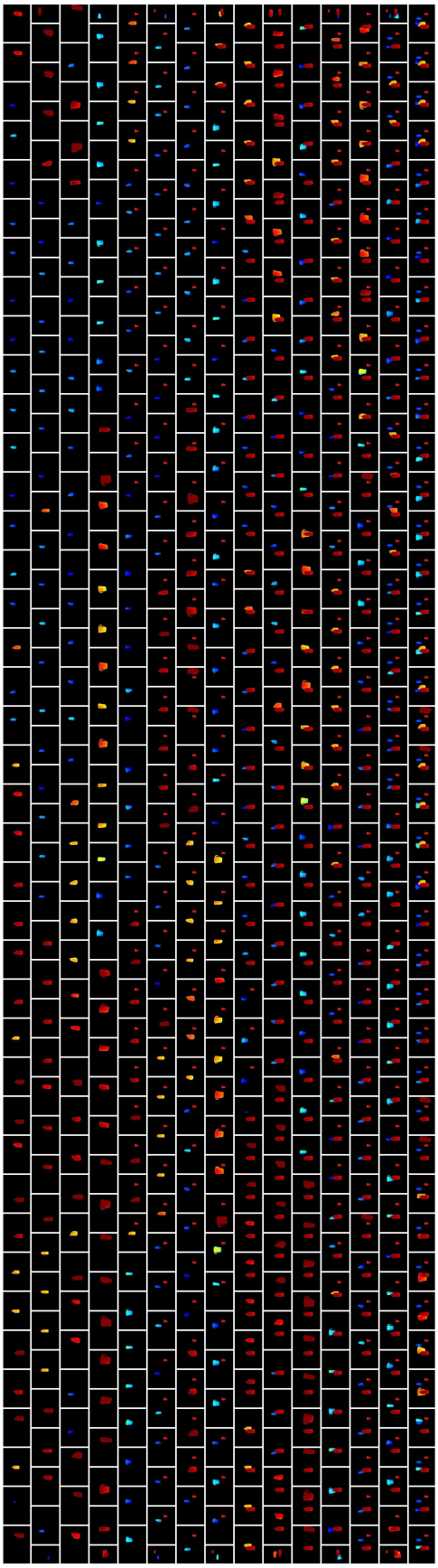


FIGURE 3.3: A subset of all the states 'generated' during the tree search for the scene in Fig. 3.2. This figure is best viewed with digital zoom.

### 3.4.2 Search

Although we have replaced exhaustive search with tree search, the problem still remains daunting owing to its branching factor. Assume that we have $d$ possible configurations $(x, y, \theta)$ for each object. Then, the worst-case branching factor for the MSGT is $d^K$ for all levels if we allow repetition of objects in the scene, or $d^{K-i}$ for level $i$ if there is no repetition. Figure 3.3 illustrates this by showing a subset of the states generated during the tree search corresponding to the scene in Fig. 3.2. While heuristic search techniques such as A* are often a good choice for such problems, they require an *admissible* heuristic that provides a conservative estimate of the remaining cost-to-go. Usual heuristic search methods are limited by the following: i) admissible heuristics are non-trivial to obtain for this problem, and ii) they cannot support multiple heuristics, each of which could be useful on their own. In this thesis, we develop novel graph-search techniques that can incorporate multiple inadmissible heuristics to guide the search, without compromising guarantees on solution quality. The next chapter presents the details of these techniques, and their properties.

While the choice of heuristic search technique is independent of the construction of the MSGT, Algorithm 2 shows an instantiation of PERCH with the Focal Multi-Heuristic A* (MHA*)[2] search algorithm (Narayanan, Aine, and Likhachev, 2015) as an example. At a high level, Focal-MHA* operates much like A* search. Like A*, it maintains a priority list of states ordered by an estimate of the path cost through that state and repeatedly 'expands' the most promising states until a goal is found. The difference from A* is in that Focal-MHA* interleaves this process with expansion of states chosen greedily by other heuristics. Finally, Focal-MHA* guarantees that the solution found will have a cost which is bounded by $w \cdot \text{OPT}$, where OPT is the optimal solution cost and $w(\geq 1)$ is a user-chosen suboptimality bound.

#### Heuristics

Focal-MHA* requires one admissible and multiple inadmissible heuristics. Constructing an informative admissible heuristic is non-trivial for this setting, and thus we set our admissible heuristic to the trivial heuristic that returns $0$ for all states. We next describe our inadmissible heuristics.

The large branching factor of the MSGT might result in the search 'expanding' or opening every node in a level before moving on to the next. To guide the search towards the goal, a natural heuristic would be a depth-first heuristic that encourages expansion of states further down in the tree. Consequently, our first inadmissible heuristic for Focal-MHA* is the depth heuristic that returns the number of assignments left to make:

$$h_{depth}(s) = K - |s|$$

[2] See next chapter.

**Algorithm 2** PERCH

**Inputs:**

The implicit MSGT construction (Eq. 3.4 and Eq. 3.5).
Suboptimality bound factor $w$ ($\geq 1$).
1 admissible heuristic $h$ and $n$ arbitrary, possibly inadmissible, heuristics $h_1, h_2, \ldots, h_n$.

**Output:**

An assignment of object poses $s_{\text{goal}}$ with $|s_{\text{goal}}| = K$ whose cost is within $w \cdot \text{OPT}$.

```
1:  procedure MAIN()
2:      s_root ← {}
3:      planner ← Focal-MHA*-Planner()
4:      planner.SETIMPLICITTREE(SUCC(s), c(s, s'))
5:      planner.SETSUBOPTIMALITYFACTOR(w)
6:      planner.SETSTARTSTATE(s_root)
7:      planner.SETHEURISTICS(h, h_1, ..., h_n)
8:      planner.SETGOALCONDITION(return true if |s| = K)
9:      {s_root, s_1, s_2, ..., s_goal} ← planner.COMPUTEPATH()
10:     return s_goal
```

As a reminder, states with *smaller* heuristic values are expanded ahead of those with larger values. Next, it would be useful to encourage the search to expand states that have maximum overlap with the observed point cloud $I$ so far, rather than states with little overlap with the observed scene. Our second heuristic is therefore the overlap heuristic that counts the number of points in $I$ that *do not* fall within the volume of assigned objects:

$$h_{overlap}(s) = |I| - \sum_{\boldsymbol{p} \in \boldsymbol{I}} \text{OUTLIER}(\boldsymbol{p}|V_j)$$

where $j = |s|$ and $V_j$ is the union of the volumes occupied by the $j$ assigned objects. Another interpretation for this heuristic is the number of points in the observed scene that are outside the space carved by objects assigned thus far.

## 3.5 Completeness

Finally, we introduce a notion of completeness for the multi-object instance detection and pose estimation problem:

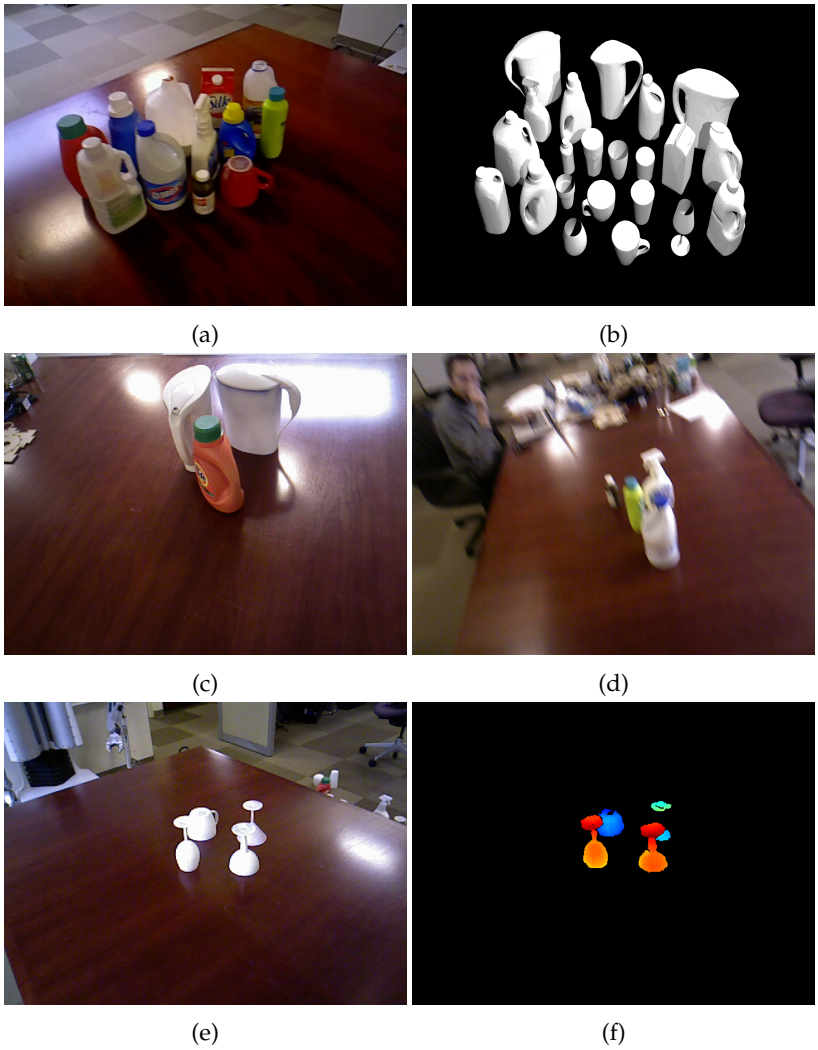
**Definition 1** (Completeness)**.** *An algorithm for multi-object pose estimation of $K$ objects is complete if it returns* any *feasible solution (i.e, a solution that contains guaranteedly collision-free pose estimates for all $K$ objects) when one exists, and correctly identifies in finite time that no solution exists otherwise.*

This definition mirrors the notion of completeness in motion planning (LaValle, 2006). We also say an algorithm is resolution-complete if it satisfies the above requirements in

a smaller solution space obtained after discretization. PERCH is resolution-complete for the chosen discretization. This follows from the completeness property of Focal-MHA*. We stress upon the notion of completeness since popular approaches to this problem proceed by obtaining individual hypotheses for each object and then performing a global refinement (Aldoma et al., 2012c), which leads to a restricted solution space and hence algorithm incompleteness.

## 3.6 Evaluation

### 3.6.1 Dataset



(a) (b) (c) (d) (e) (f)

FIGURE 3.4: (a) and (b) A subset of the objects in the RGB-D occlusion dataset. (c), (d) and (e) Representative test scenes from the dataset. (f) Depth image (mapped to jet color) corresponding to (e) obtained after preprocessing to remove background and table.

To evaluate the performance of PERCH for multi-object recognition and pose estimation in challenging scenarios where objects could be occluding each other, we pick the occlusion dataset described by Aldoma et al. (Aldoma et al., 2012a) that contains objects partially touching and occluding each other. The dataset contains 3D CAD models of 36 common household objects, and 23 RGB-D tabletop scenes with 82 object instances in total. With the exception of one scene, all scenes contain objects only varying in translation and yaw, with some objects flipped up-side down. Since PERCH is designed only for 3D pose estimation, we drop the one non-compatible scene from the dataset, and preprocess the 3D CAD models such that they vary only in translation and yaw with respect to the ground truth poses. Figure 3.4 shows some examples from the dataset.

### 3.6.2 Implementation Details

#### Compensating for Discretization

The most computationally expensive part of PERCH is that of generating successor states for a given state in the MSGT. This involves generating and rendering every state that contains one more object than the number in the present state, in every possible configuration. Several elements influence this branching factor: the number of objects in the scene, the chosen discretization for object poses, whether objects are rotationally symmetric (in which case only $(x, y)$ is of interest) etc. In our implementation, we limit the complexity by favoring coarse discretization and compensating with a local alignment technique such as ICP (Chen and Medioni, 1991). Specifically, every time we render a state with a new object, we take the non-occluded portion and perform an ICP alignment in the local vicinity of the observed point cloud. This allows us to obtain accurate pose estimates while retaining a coarse discretization. We do note that the underlying MSGT now becomes a function of the observed point cloud due to the ICP adjustment. Consequently, the algorithms are still resolution-complete, but in a solution space that itself depends on the ICP refinement instead of the chosen discretization. Finally, we plan to investigate multi-resolution approaches to further improve efficiency by reducing the branching factor.

#### Parallelization

The generation of successor states is an embarrassingly parallel process. We exploit this in our implementation by using multiple processes to generate successors in parallel. Theoretically, with sufficient number of cores, the time to expand a state would simply be the time to render a single scene.

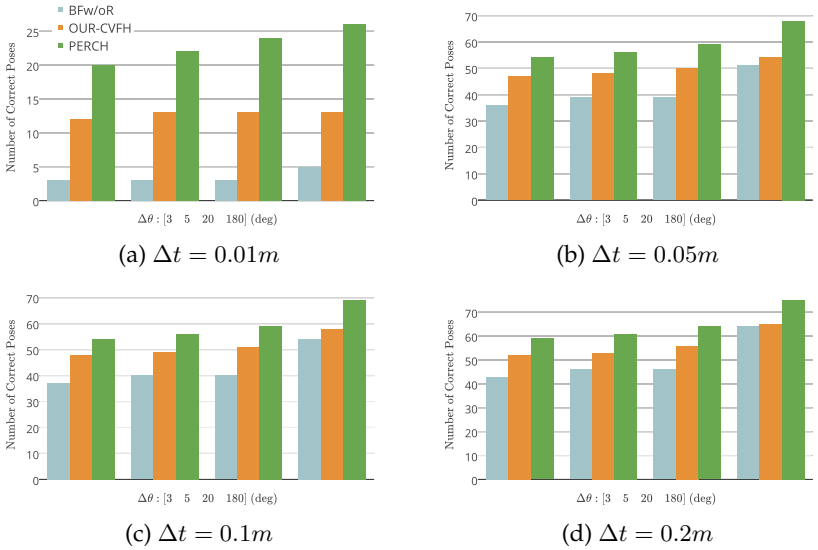
#### PERCH Setup

Since PERCH requires that points in the scene only belong to objects of interest, we first preprocess the scene to remove the tabletop and background. Then, based on the RANSAC-estimated table plane, we compute a transform that aligns the point cloud from the camera frame to a gravity aligned frame, to simplify construction of the MSGT. PERCH has two parameters to set: the sensor noise threshold $\delta$ for determining whether a point is 'explained' (Eq. 3.2), and the suboptimality factor $w$ for the Focal-MHA* algorithm. In our experiments, we set $\delta$ to $3$ mm to account for uncertainty in the depth measurement from the Kinect sensor, as well as inaccuracies in estimating the table height using RANSAC. For the suboptimality factor $w$, we use a value of $3$. While this results in solutions that can be suboptimal by a factor of up to $3$, it greatly speeds up the search, since computing the optimal solution typically takes much more time (Pohl, 1970). Finally, for defining the MSGT we pick a discretization resolution of $4$ cm for both $x$ and $y$ and $22.5$ degrees for yaw. The adaptive ICP alignment (Sec. 3.6.2) is constrained to find correspondences within $2$ cm, which is half the discretization resolution.

### 3.6.3 Baselines

#### OUR-CVFH

Our first baseline is the OUR-CVFH global descriptor (Aldoma et al., 2012b), a state-of-the-art global descriptor designed to be robust to occlusions. By clustering object surfaces into separate smooth regions and computing descriptors for each portion, OUR-CVFH can handle occlusions better than descriptors such VFH and FPFH. Furthermore, it has the added advantage of directly encoding the full pose of the object, with no ambiguity in camera roll. We build the training database by rendering 642 views of every 3D CAD model from viewpoints sampled around the object. Then, for computing the training descriptors we use moving least squares to upsample every training view to a common resolution followed by downsampling to the Kinect resolution of $3$ mm as suggested in the OUR-CVFH paper (Aldoma et al., 2012b). Since the number and type of models in the test scene is assumed to be known for PERCH, we use the following pipeline for fair comparison: for the $K$ largest clusters in the test scene we obtain the histogram distance to each of the models we know that are in the scene. Then, we solve a min-cost matching problem to assign a particular model (and associated pose) to each cluster and obtain a feasible solution. Finally, we constrain the full 6 DoF poses returned by OUR-CVFH to vary only in translation and yaw and perform a local ICP alignment for each object pose.




(a) $\Delta t = 0.01m$ (b) $\Delta t = 0.05m$

(c) $\Delta t = 0.1m$ (d) $\Delta t = 0.2m$

FIGURE 3.5: Number of objects whose poses were correctly classified by the baseline methods (BFw/oR, OUR-CVFH) and PERCH, for different definitions of 'correct pose'.

**Brute Force ICP**

The second baseline is an ICP-based optimization one, which we will refer to as Brute Force without Rendering (BFw/oR). Here, we slide the 3D model of every object in the scene over the observed point cloud (at the same discretization used for PERCH), and perform a local ICP-alignment at every step. The location $(x, y, \theta)$ that has the best ICP fitness score is chosen as the final pose for that model and made unavailable for other objects that have not yet been considered. Since the order in which the models are chosen for sliding can influence the solution, we try all permutations of the ordering ($K!$) and take the overall best solution based on the total ICP fitness score.

### 3.6.4 Results

To evaluate the accuracy of object localization, we use the following criterion: a predicted pose $(x, y, \theta)$ for an object is considered correct if $\|(x, y) - (x_{\text{true}}, y_{\text{true}})\|_2 < \Delta t$ and $\text{SHORTESTANGULARDIFFERENCE}(\theta, \theta_{\text{true}}) < \Delta\theta$. We then compute the number of correct poses produced by each method for different combinations of $\Delta t$ and $\Delta\theta$. Figure 3.5 compares the performance of PERCH with BFw/oR and OUR-CVFH. Immediately obvious is the significant performance of PERCH over the baseline methods for $\Delta t = 0.01m$. PERCH is able to correctly estimate the pose of over 20 objects with translation error under $1$ cm and rotation error under $5$ degrees. While the baseline methods have comparable recall for higher thresholds, they are unable to provide as many precise poses as PERCH does. Further, PERCH consistently dominates the baseline methods for all definition of 'correct pose'. Among all methods, BFw/oR performs

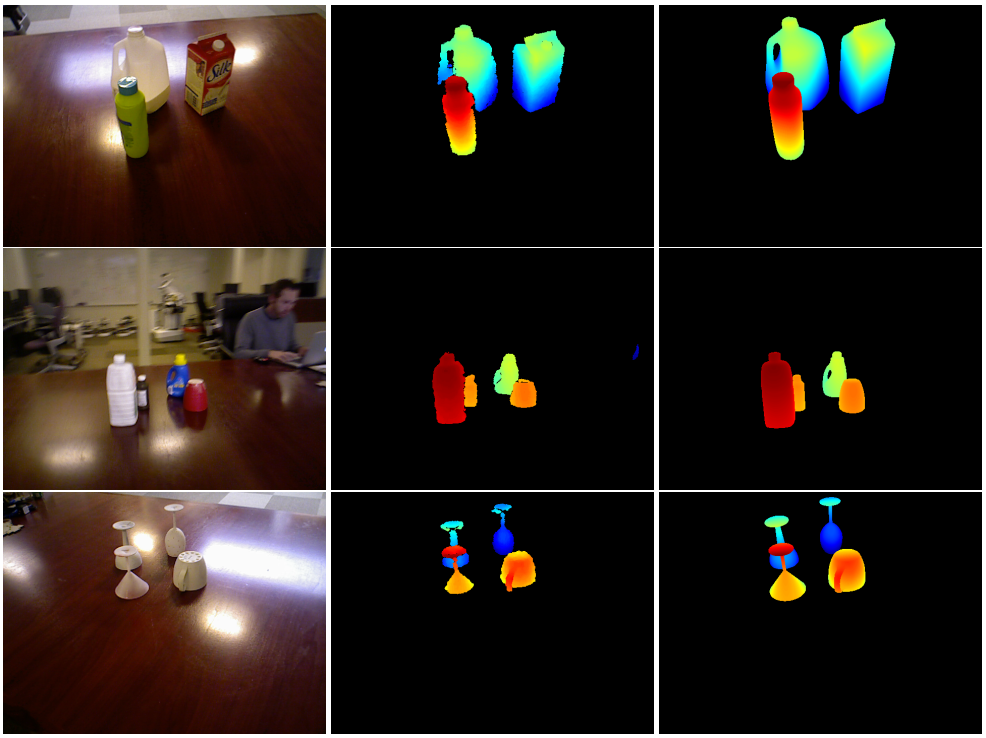


FIGURE 3.6: Examples showing the output of PERCH on the occlusion dataset. *Left*: RGB-D scenes in the dataset. *Middle*: Depth images of the corresponding input RGB-D scenes, *Right*: The depth image reconstructed by PERCH through rendering object poses.

the worst. This is mainly due to the fact that it uses the point cloud corresponding to the complete object model for ICP refinement, rather than the point cloud corresponding to the unoccluded portion of the object. Again, this showcases the necessity to explicitly reason about self-occlusions as well as inter-object occlusions.

The last column of the histogram in Fig. 3.5d (corresponding to $\Delta t = 0.2$, $\Delta\theta = 180$) is essentially a measure of recognition alone—PERCH can correctly identify 69 of the 80 object instances, where 'identified' is defined as obtaining a translation error under 10 cm. Figure 3.6 shows some qualitative examples of PERCH's performance on the occlusion dataset. Further examples and illustrations are provided in the supplementary video.

#### Computation Time and Scalability

Unlike global descriptor approaches such as OUR-CVFH which require an elaborate training phase to build a histogram library, PERCH does not require any precomputation. Consequently, the run time cost is high owing to the numerous scenes that need to be rendered. However, as mentioned earlier, the parallel nature of the problem and the easy availability of cluster computing makes this less daunting. For our experiments, we used the MPI framework to parallelize the implementation and ran


(a) (b)

FIGURE 3.7: (a) A scene showing multiple chess pieces occluding each other. (b) *Top:* The depth image from a Kinect sensor, colored by range. *Bottom:* The best-match depth image produced by our algorithm PERCH through searching over possible poses of the chess pieces.

the tests on a cluster of 2 Amazon AWS m4.10x machines, each having a 40-core virtual CPU. For each scene, we used a maximum time limit of 15 minutes and took the best solution obtained within that time. Overall, the mean planning time was 6.5 minutes, and the mean number of hypotheses rendered (i.e, states generated) was 15564.

Finally, to demonstrate that PERCH can be used for scenes containing several objects, we conducted a test on a chessboard scene (Fig. 3.7a). We captured a Kinect depth image of the scene containing 12 pieces, of which 6 are unique and 4 are rotationally symmetric. We ran PERCH with suboptimality bound factor $w = 15$ and sensor resolution $\delta = 7.5$ mm, and took the best solution found within a time limit of 20 minutes. The solution found (i.e., the depth image corresponding to the goal state) is shown in Fig. 3.7b.

# Chapter 4

# Extension to Unmodeled Clutter and Optimizations

The optimization formulation presented in Chapter 3 assumes that everything in the observed scene can be explained through known models. In other words, it does not account for unmodeled clutter, and assumes that the input scene is comprised solely of objects for which we have 3D models. Unfortunately, this assumption can fail in many real-world scenarios where we simply cannot procure models for every object in the scene.

In this chapter, we first address the above limitation of PERCH, and then discuss several algorithmic and implementation improvements for computational efficiency. The main insight for the former is that we can augment the optimization objective that PERCH uses with a term that allows the algorithm to treat certain points in the input cloud as "clutter", thereby still being able to reason about occlusion explicitly. We present the details of this formulation, named C-PERCH (Clutter-PERCH) below.

## 4.1 C-PERCH

### 4.1.1 Notation

We adopt the same notation used in PERCH, and introduce some additional ones that are summarized in Table 4.1. As a reminder, upper-case bold-faced letters denote point clouds (set of points in $\mathbb{R}^3$), and lower-case bold-faced letters denote a point in $\mathbb{R}^3$.

### 4.1.2 Augmented Objective

The explanation cost used by PERCH is meaningful only when we want both the rendered and input point clouds to exactly match each other. In the presence of unmodeled clutter however, this fails on two counts: first, the rendered scene does not account

This chapter is based on material from Venkatraman Narayanan and Maxim Likhachev (2017a). "Deliberative Object Pose Estimation in Clutter". In: *ICRA*.

TABLE 4.1: Symbols and Notation for C-PERCH.

| | |
|---|---|
| $\boldsymbol{C}$ | Point cloud containing points in $\boldsymbol{I}$ which are considered as "clutter" by the algorithm |
| $\boldsymbol{R_j}\|\boldsymbol{C}$ | Ditto as $\boldsymbol{R_j}$, but considering points in $\boldsymbol{C}$ as occluders |
| $\boldsymbol{\Delta R_j}\|\boldsymbol{C}$ | $(\boldsymbol{R_j}\|\boldsymbol{C}) - (\boldsymbol{R_{j-1}}\|\boldsymbol{C})$ |
| $\boldsymbol{p} \prec \boldsymbol{P}$ | Point $\boldsymbol{p}$ occludes the point cloud $\boldsymbol{P}$ |
| $\boldsymbol{C_j}$ | Points in the clutter cloud $\boldsymbol{C}$ which occlude $\boldsymbol{R_j}$: $\boldsymbol{C_j} = \{\boldsymbol{p} \in \boldsymbol{C} : \boldsymbol{p} \prec \boldsymbol{R_j}\}$ |
| $\boldsymbol{\Delta C_j}$ | Points in the input cloud $\boldsymbol{I}$ which occlude $\boldsymbol{\Delta R_j}$: $\boldsymbol{\Delta C_j} = \{\boldsymbol{p} \in \boldsymbol{I} : \boldsymbol{p} \prec \boldsymbol{\Delta R_j}\}$ |

for occlusion by the clutter (the algorithm assumes that all occlusions occur between objects with known 3D models), and second, the cost function (specifically $J_{observed}$) would unnecessarily penalize points in the input cloud which are extraneous clutter that do not belong to the objects of interest. To overcome these limitations, we first propose a formulation that allows the algorithm to explicitly pick and treat some points as clutter, and secondly demonstrate that this does not add any significant complexity to the existing optimization solved by PERCH.

In our proposed extension C-PERCH (Clutter-PERCH), we jointly optimize over the object poses $O_{1:K}$ *and* a variably-sized clutter cloud $\boldsymbol{C} \subseteq \boldsymbol{I}$. The latter allows the algorithm to mark certain points in the input scene as clutter, so that it can use those as extraneous "occluders" when rendering a scene with known 3D models. However, complete freedom to mark points as clutter could be disastrous: the optimal solution might just be to treat the input entire cloud as clutter, claim that the desired object(s) to be localized are completely occluded by the clutter, and thus incur no cost at all (since the rendered point cloud would be an empty cloud). To strike a balance between optimizing the explanation cost and allowing the algorithm to treat certain points as clutter, we introduce an additional term to the cost function that penalizes the algorithm for marking too many points as clutter—in other words, we would like to minimize the explanation cost while not marking too many points treated as clutter (conversely maximize the number of points observed on the objects of interest). Of course, the amount to penalize depends on the scenario at hand as well. In extreme clutter, it would be okay if the algorithm marks several points as clutter, but in scenes with no clutter, we really don't want to mark any point as clutter. We model this using a multiplicative factor $\alpha$ on the clutter penalty, to represent our uncertainty about the true nature of clutter in the scene. The augmented cost function to minimize is:

$$
\begin{aligned}
J_\alpha(O_{1:K}, \boldsymbol{C}) &= J_o(O_{1:K}, \boldsymbol{C}) + J_r(O_{1:K}, \boldsymbol{C}) + \alpha|\boldsymbol{C}| \\
J_o(O_{1:K}, \boldsymbol{C}) &= \sum_{\boldsymbol{p} \in \boldsymbol{I} \cap \boldsymbol{V_K}} \textsc{Outlier}(\boldsymbol{p}|(\boldsymbol{R_K}|\boldsymbol{C})) \\
J_r(O_{1:K}, \boldsymbol{C}) &= \sum_{\boldsymbol{p} \in \boldsymbol{R_K}|\boldsymbol{C}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I})
\end{aligned}
\tag{4.1}
$$

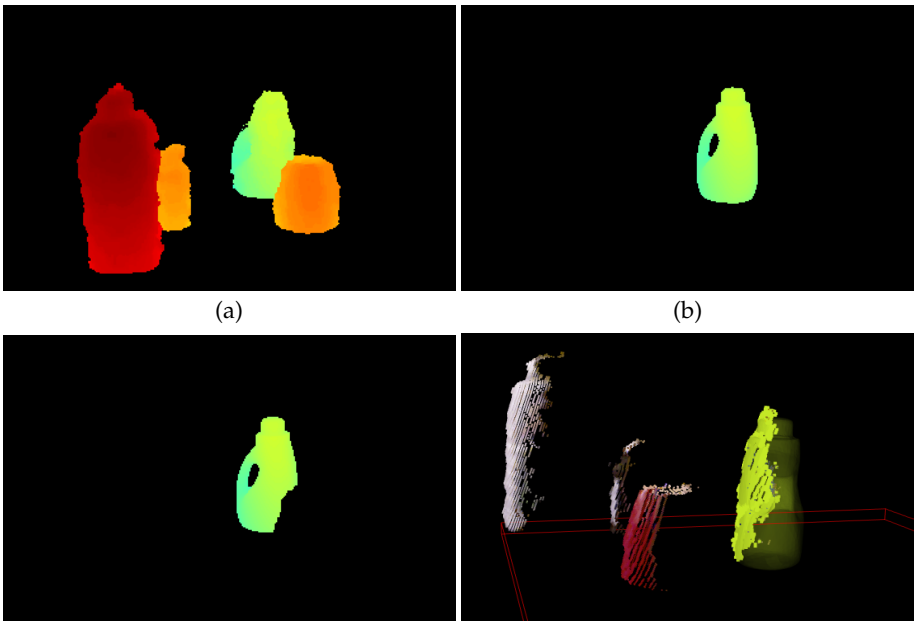
**Algorithm 3** C-PERCH: Generation of Successor States and Edge Costs

**procedure** GETSUCCESSORS($O_{1:j}$)
  $S \leftarrow \emptyset$
  $\Delta J \leftarrow \emptyset$
  // Iterate over objects not yet added to scene
  **for all** ID $\in$ {All Possible IDs} $\setminus$ IDs($O_{1:j}$) **do**
    // Iterate over all possible poses the new object can take
    **for all** $pose \in$ {All Discrete Poses} **do**
      $O_{j+1} = \{\text{ID}, pose\}$
      $s = O_{1:j} \cup O_{j+1}$
      **if** $s$ is not physically-plausible **then**
        **continue**
      $\boldsymbol{R_j}$ = render scene with $O_{1:j}$
      $\boldsymbol{R_{j+1}}$ = render scene with $O_{1:j+1}$
      $\boldsymbol{\Delta R_{j+1}} = \boldsymbol{R_{j+1}} - \boldsymbol{R_j}$
      **if** $\boldsymbol{\Delta R_{j+1}} \prec \boldsymbol{R_j}$ **then**
        // Prune if new object occludes existing scene
        **continue**
      $\boldsymbol{\Delta C_j}$ = Points in $\boldsymbol{I}$ which occlude $\boldsymbol{\Delta R_j}$
      Compute $\Delta J_o^{j+1}(O_{j+1}, \boldsymbol{\Delta R_{j+1}}, \boldsymbol{\Delta C_{j+1}})$ ▷ Eq. 4.3
      Compute $\Delta J_r^{j+1}(\boldsymbol{\Delta R_{j+1}}, \boldsymbol{\Delta C_{j+1}})$ ▷ Eq. 4.4
      Compute $\Delta J_{c,\alpha}^{j+1}(\boldsymbol{\Delta C_{j+1}})$ ▷ Eq. 4.5
      $\Delta J_\alpha^{j+1} = \Delta J_o^{j+1} + \Delta J_r^{j+1} + \Delta J_{c,\alpha}^{j+1}$
      $S \leftarrow S \cup \{s\}$
      $\Delta J \leftarrow \Delta J \cup \{\Delta J_\alpha^{j+1}\}$
  **return** $\langle S, \Delta J \rangle$

There are three changes from Eq. 3.1. First, both $J_o$ and $J_r$ use the cloud $\boldsymbol{R_K}|\boldsymbol{C}$ rather than simply $\boldsymbol{R_K}$ to explicitly acknowledge the occlusions caused by extraneous clutter. Second, the optimization objective has a penalty term for the number of points marked as clutter, weighed by $\alpha$, a term which models the amount of true clutter in the scene. Third, $J_o$ only penalizes points in the input cloud that fall within the volume of the modeled objects, rather than every point in $\boldsymbol{I}$. An illustration of the different point clouds used is presented in Fig. 4.1. Intuitively, the term $\alpha$ can be viewed as an occlusion prior or regularizer that balances the cost of explaining the rendered scene with the cost of labeling input points as belonging to extraneous clutter.

### 4.1.3 Tractability

With this formulation, it appears that we have made the problem intractable by introducing a variably-sized point cloud into the already combinatorial search space. However, it can be shown that $\boldsymbol{C}$ is only a dependent variable of $O_{1:K}$, and does not affect the decomposition used by PERCH under a reasonable constraint. Define $\boldsymbol{C}_{intrusive} = \{\boldsymbol{p} \in \boldsymbol{C} : \boldsymbol{p} \prec \boldsymbol{R_K}\}$, the set of clutter points which occlude the scene rendered by considering only the objects of interest, and $\boldsymbol{C}_{superfluous} = \boldsymbol{C} - \boldsymbol{C}_{intrusive}$. Quite clearly, we can replace $\boldsymbol{R_K}|\boldsymbol{C}$ with $\boldsymbol{R_K}|\boldsymbol{C}_{intrusive}$ in the optimization objective (since the superfluous clutter points do not affect the rendering of the objects).



(a) (b) (c) (d)

FIGURE 4.1: Illustrations for the notation used. (a) The input point cloud $\boldsymbol{I}$ (represented as a depth image and pseduo-colored). (b) Rendering $\boldsymbol{R_1}$ corresponding to a state with one object $O_1$ (c) Rendering $\boldsymbol{R_1}|\boldsymbol{\Delta C_1}$, considering points in $\boldsymbol{I}$ that occlude $\boldsymbol{R_1}$ (d) A profile view of the same scene, showing the volume $V(O_1)$, and the points in $\boldsymbol{I}$ contained in it.

Since $\boldsymbol{C} = \boldsymbol{C}_{intrusive} \cup \boldsymbol{C}_{superfluos}$ and all terms except the penalty depend only on $\boldsymbol{C}_{intrusive}$, the optimal solution would involve setting $\boldsymbol{C}_{superfluos} = \emptyset$. Hereon, we simply set $\boldsymbol{C} = \boldsymbol{C}_{intrusive}$ to factor this observation in to account.

If we now require the algorithm to definitely mark every occluding input point as clutter (for a given rendered scene), we can simply drop $\boldsymbol{C}$ from the argument list of $J_\alpha$, $J_o$ and $J_r$ because $\boldsymbol{C}$ is now purely a function of $\boldsymbol{R_K}$.

Note that $\boldsymbol{R_K}$ can be constructed in a monotone fashion by introducing objects in a non-occluding order, as shown in the previous chapter. Formally, $\boldsymbol{R_K} = \cup_{i=1}^{K} \boldsymbol{\Delta R_i}$, s.t, $\boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i}$. If we define $\boldsymbol{\Delta C_j} = \{\boldsymbol{p} \in \boldsymbol{I} : \boldsymbol{p} \prec \boldsymbol{\Delta R_j}\}$ and follow a decomposition procedure similar to the one adopted in PERCH, we obtain:

$$
\begin{aligned}
J_\alpha(O_{1:K}) &= \sum_{i=1}^{K} \Delta J_\alpha^i && \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i} \\
&= \sum_{i=1}^{K} \Delta J_o^i + \Delta J_r^i + \Delta J_{c,\alpha}^i && \text{s.t. } \boldsymbol{R_{i-1}} \subseteq \boldsymbol{R_i}
\end{aligned}
\tag{4.2}
$$

where

$$\Delta J_o^i = \sum_{\boldsymbol{p} \in \{\boldsymbol{I} \cap V(O_i)\}} \textsc{Outlier}(\boldsymbol{p} | (\boldsymbol{\Delta R_i} | \boldsymbol{\Delta C_i})) \tag{4.3}$$

$$\Delta J_r^i = \sum_{\boldsymbol{p} \in \boldsymbol{\Delta R_i}} \textsc{Outlier}(\boldsymbol{p} | \boldsymbol{I}) \tag{4.4}$$

$$\Delta J_{c,\alpha}^i = \alpha |\boldsymbol{\Delta C_i}| \tag{4.5}$$

The end result is that we can still maintain the decomposition of the objective function over individual objects despite introducing the clutter cloud in to the optimization process. Similar to PERCH, we solve the final optimization as a discrete tree-search under the constraint that objects are added in a non-occluding order to the Monotone Scene Generation Tree. The complete procedure followed to generate successor states for a parent state, and the corresponding edge costs is presented in Alg. 3.

## 4.2 Pose Uncertainty Estimates

We earlier mentioned how the factor $\alpha$ models the amount of clutter expected in the scene. Low values correspond to scenes with high anticipated clutter (where the penalty for marking points as clutter is low), and vice versa. Solving the optimization problem for different values of $\alpha$ yield potentially distinct solutions:

$$O_{1:K}^{\alpha} = \underset{O_{1:K}}{\operatorname{argmin}}\, J_{\alpha}(O_{1:K}) \tag{4.6}$$

Immediately, there is an opportunity to produce uncertainty estimates for the object poses based on the uncertainty in how much clutter there exists in the scene. If $p(\alpha)$ denotes the prior for $\alpha$ (which may be obtained by a priori analysis of the scene or assumed to be uniform), then the density estimate for the object poses (represented by the random variable $\Omega$) is given by

$$\begin{aligned} p(\Omega = O_{1:K}) &= \int_{\alpha} p(\Omega = O_{1:K}, \alpha) d\alpha \\ &= \int_{\alpha} p(\Omega = O_{1:K} | \alpha) p(\alpha) d\alpha \\ &= \int_{\alpha} \mathbb{1}\left(O_{1:K} = O_{1:K}^{\alpha}\right) p(\alpha) d\alpha \end{aligned} \tag{4.7}$$

Eq. 4.7 is hard to solve in closed form, but sampling from the distribution is trivial: we sample an $\alpha$ from $p(\alpha)$, and solve the optimization problem in Eq. 4.6 to get a solution,

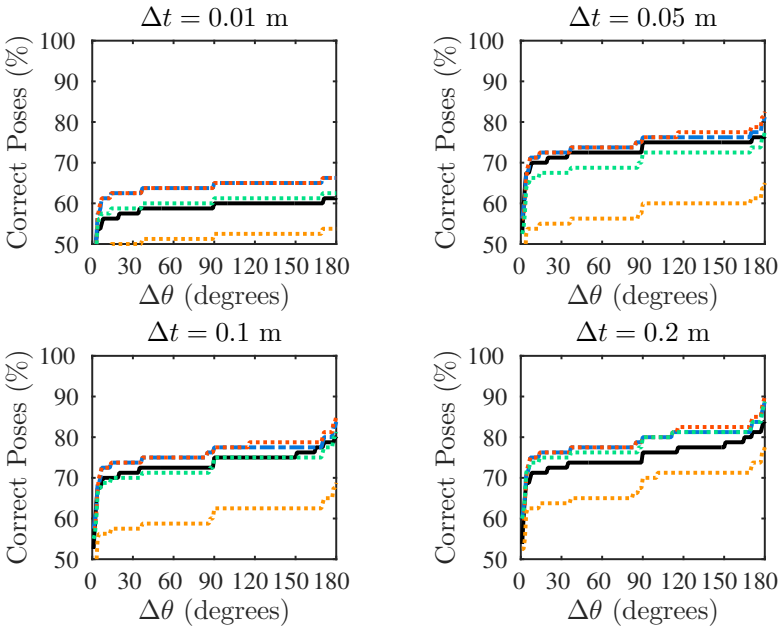



FIGURE 4.2: Comparison of PERCH (■) with C-PERCH for different values of $\alpha$: 1 (■) 0.5 (■) 0.25 (■) and 0 (■), and for different correctness measures. We omit $\alpha = 0.75$ since it yielded identical results to $\alpha = 1$.

which is a sample from $p(\Omega)$. Intuitively, if we find out that several values of $\alpha$ lead to the same solution for $O_{1:K}$, it implies that the scene clutter model has minimal effect on the object poses and we can therefore be confident about our pose estimate. On the other hand, if solutions are distinct for closely related values of $\alpha$, we would have greater uncertainty in our object poses due to a spread-out distribution.

## 4.3 Experiments

**Experiment Setup.** We evaluate C-PERCH on the occlusion dataset of (Aldoma et al., 2012a) used in the earlier chapters. To test the ability of C-PERCH to handle extraneous clutter in the scene, we setup evaluation such that C-PERCH and the baselines are required to identify and localize exactly one object in the scene, treating the others as clutter. The process is repeated for every object in each scene.

For both PERCH and C-PERCH, we use a discretization of $0.05$ m for translation and $22.5$ degrees for yaw. Note that in this dataset, objects vary only in yaw with respect to the 3D models. We also use a locally-constrained ICP refinement (with a maximum of $20$ iterations) for every rendered state in both PERCH and C-PERCH to compensate for discretization artifacts. As usual, we measure accuracy of an algorithm by counting the number of objects that fall within a given error bound: an estimated object pose $(x, y, \theta)$ is marked 'correct' if $\|(x, y) - (x_{\text{true}}, y_{\text{true}})\|_2 < \Delta t$ and

FIGURE 4.3: Example that demonstrates how C-PERCH can be used to obtain pose uncertainty estimates, including multimodal distributions. The object to be localized in this scene (*left*) is the milk carton (for which we have a 3D model) and the other objects are considered as extraneous clutter (no models available). C-PERCH yields two distinct solutions across multiple values of $\alpha$, which are overlaid on top of the input RGB image (*right*).

SHORTESTANGULARDIFFERENCE$(\theta, \theta_{\text{true}}) < \Delta\theta$. The latter check is ignored for rotationally symmetric objects.

**Accuracy Comparisons.** Figure 4.2 compares the performance of PERCH (baseline) with C-PERCH configured with different values of $\alpha$, the clutter model parameter. All experiments were run on an m4.10x Amazon AWS instance. The first takeaway is that C-PERCH consistently outperforms PERCH, for $\alpha \in \{1, 0.5, 0.25\}$. This supports our hypothesis that modeling clutter explicitly in the optimization formulation will lead to better performance. A second observation is that C-PERCH with $\alpha = 0.5$ performs marginally better than $\alpha = 1$, and significantly better than $\alpha = 0$. This indicates that there is no one "correct" way to pick $\alpha$ unless we have some prior information about the clutter conditions. In some sense, being cautious ($\alpha = 0.5$) yields the best performance across a variety of scenes.

**Timing.** The average time taken by PERCH for a scene was $19.17$ s and for C-PERCH (across all values of $\alpha$) was $17.98$ s. While both methods generated the same number of scenes on average ($792.31$), PERCH takes slightly longer than C-PERCH since it computes the observed cost ($J_o$) over all points in the input point cloud, as opposed to C-PERCH which only looks at input points within the volumes of the assigned objects.

#### Illustration of Uncertainty Estimation

Next, we provide an example that demonstrates how C-PERCH can be used to generate uncertainty estimates for an object pose. In Fig. 4.3, the object desired to be localized is the milk carton, which is partially occluded by the milk jug. Large flat portions on the milk carton as well as on the Odwalla jug (far back on the right) cause some ambiguity to the algorithm since it deals only with the depth image (RGB is not used). If we proceed to estimate the object pose uncertainty (Eq. 4.7), by running the

optimization for values of $\alpha \in [0, 1]$ in steps of $0.01$ (i.e., assuming an uniform prior for $\alpha$), we observe that only two distinct solutions turn up, corresponding to the ranges $[0, 0.21]$ and $(0.21, 1]$. The pose uncertainty distribution can thus be represented as a particle distribution with two particles of weight $\sim 0.2$ and $\sim 0.8$ respectively, with the former corresponding to the partially occluded configuration.

## 4.4 Discussion

In summary, we presented C-PERCH, an extension to Perception via Search (PERCH), that allows deliberative perception algorithms to operate in scenes with unmodeled extraneous clutter. This significantly extends their practical relevance to real-world scenarios where 3D models cannot be obtained for every object in the scene. In addition, we also showed how C-PERCH can produce pose uncertainty estimates by reasoning about the amount of clutter in the scene. This is useful to systems with active sensing, and when introspection capabilities are required.

C-PERCH is rife with opportunities for future extensions. In the work presented, the occlusion prior / regularizer $\alpha$ was a free parameter for the algorithm. One line of future work would be to automatically determine the value of $\alpha$ to use for a given scene, by statistical or geometric analysis of the scene. First, note that $\alpha$ does not need to be uniform across the entire scene; one could use different values at different points in the input scene, depending on our estimate of how likely each point in the scene is occluded by extraneous clutter. This estimate could be generated statistically, as in the occlusion-predictive sensor model of (Herbst et al., 2011), or through geometric modeling of the occluder as in (Hsiao and Hebert, 2014). In the former, the occlusion likelihood produced by the sensor model increases as the difference between the predicted ("rendered") and measured depth increases, so long as the predicted depth is larger than the measured one. The latter models extraneous occluders as 3D boxes of varying dimensions to produce point-wise occlusion priors. With either of these models, one could obtain an $\alpha$ for every input point that can then be used in the optimization objective.

Another direction concerns the efficiency of generating solutions for different values of $\alpha$. In Sec. 4.2, multiple solutions were generated by solving the optimization for different values of $\alpha$, each independently of the other. However, one could use incremental search techniques such as LPA* (Koenig, Likhachev, and Furcy, 2004) to re-use search effort between successive episodes as the value of $\alpha$ is gradually increased. Note that solving the problem for intermediate values of $\alpha$ corresponds to finding solutions between the two extremes: labeling all input points as extraneous clutter, and labeling none of the input points as extraneous clutter. In some sense, the intermediate solutions can be viewed as pareto-optimal solutions (Choudhury, Dellin, and Srinivasa, 2016) corresponding to a bi-objective cost: the original PERCH cost and the penalty

cost. Nevertheless, the order in which the $\alpha$ values need to be traced out remains an open question.

## 4.5 Search Optimizations

A naive implementation of PERCH can result in poor computational performance. In the following, we will discuss three important optimizations for improving run times.

One of the most computationally expensive components of PERCH is the rendering of successor scenes when expanding a state in the tree. This is aggravated by the need to render each scene twice (first to obtain the point cloud that is used for ICP adjustment, and the second post-ICP to get the point cloud on which the edge-cost is computed). We propose two optimizations to accelerate this process: the first is memoization of first-level states (states with single objects) to quickly produce depth images for multi-object states in deeper levels of the tree. The second one is lazy evaluation (i.e, postponing exact evaluation until necessary) of edge costs. This minimizes the number of renderings required when expanding (generating successors of) a state in the tree. The final improvement is related to the efficient implementation of cost computation using distance fields. These are next discussed in detail:

### 4.5.1 Depth Image Memoization

Upon expanding the root node of the tree, successor states corresponding to all poses of individual objects are rendered. During this expansion, we cache depth images $D(O_j)$ corresponding to individual object states $O_j$. This allows to reconstruct the depth image for a multi-object state comprising of objects $O_{1:k}$ simply by taking the element-wise minimum of depth images $D(O_1), D(O_2), \ldots, D(O_j)$. Consequently, this eliminates the need to render multi-object successors for the most part, upon expanding a state in the tree.

There is an subtle detail however when the tree construction is interleaved with ICP for the visible portion of a newly added object. It is not guaranteed that the element-wise minimum of first-level depth images would produce the same result as the non-memoized depth image. Therefore, an additional step of performing ICP on the non-occluded portion is still needed to determine the final depth image.

### 4.5.2 Lazy Search

Depth-image caching by itself however, is not sufficient to accelerate successor generation. This is because the point cloud corresponding to the newly rendered object goes

through ICP refinement and subsequent re-rendering before the edge-cost can be evaluated. A similar bottleneck exists in heuristic search-based motion planning, where expanding a graph state requires time-consuming collision checking of the edges. This was addressed by (Cohen, Phillips, and Likhachev, 2014) in their lazy weighted A* algorithm. The key idea is that if we have a mechanism to inexpensively compute *admissible* estimates of the edge cost, then weighted A* search can simply use these proxy costs while inserting states into the frontier and look up the true cost only when a lazily evaluated state is about to be expanded. By using these admissible estimates (i.e, the estimated edge cost is lesser than or equal to the true cost), lazy weighted A* retains theoretical properties of bounded suboptimality. We apply the same idea to Focal-MHA*, albeit with minor differences. Since Focal-MHA* interleaves admissible expansions with expansions from inadmissible heuristics, we require that the true edge cost be evaluated any time it is about to be expanded, irrespective of whether it was chosen by the admissible heuristic, or an inadmissible one. This ensures bounded suboptimality of the solution returned by Focal-MHA* (the proof follows a similar structure to that of lazy weighted A*, but is omitted here for simplicity).

Next, we describe how a lazy admissible estimate of the edge cost can be obtained without rendering the successor state. Let $s_j$ be a newly generated successor state of $s_{j-1}$ with $O_j$ as the last introduced object, and $\boldsymbol{\Delta R_j}$ be the point cloud corresponding to the visible portion of object $O_j$ given the other objects in $s_j$. The lazy cost of the edge to $s_j$ is computed as follows:

1. Obtain the depth image corresponding to $s_j$ by composing the cached depth image of its parent (which exists by induction) with the cached depth image of $O_j$.
2. The differential partial cloud $\boldsymbol{\Delta R_j}$ of the newly introduced object is subject to ICP refinement, resulting in $\boldsymbol{R'_j}$.
3. Points in $\boldsymbol{\Delta R'_j}$ that are self-occluded and occluded by other objects in $s_j$ are removed to obtain $\boldsymbol{\Delta \tilde{R}_j}$. Removal of self-occluding points is done by projecting all points in $\boldsymbol{R'_j}$ to the depth image and retaining only the minimum depth for each pixel when multiple points project to the same one. Points occluded by existing objects in $s_j$ are similarly removed by taking the element-wise minimum of the re-projected depth image with the cached depth image of the parent state. Finally, $\boldsymbol{\Delta \tilde{R}_j}$ is obtained by "unprojecting" the depth image pixels corresponding to $O_j$, following the above process.
4. The lazy edge cost is then computed as

$$\tilde{c}(s_{j-1}, s_j) = \sum_{\boldsymbol{p} \in \boldsymbol{\Delta \tilde{R}_j}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I}) \tag{4.8}$$

**Theorem 1.** *Lazy estimates of the edge cost obtained by the above procedure are guaranteed to be admissible estimates of the true edge cost.*

*Proof* (Sketch): Let $s_j$ be the considered successor state of $s_{j-1}$ and $\boldsymbol{\Delta R_{j,\text{true}}}$ the differential point cloud corresponding to object $O_j$. By construction, we have $\boldsymbol{\Delta \tilde{R}_j} \subseteq \boldsymbol{\Delta R_{j,\text{true}}}$. Intuitively, re-rendering an object after ICP refinement will only introduce new unseen portions of the object, while the existing parts continue to be visible or become self-occluded/occluded by other objects. The true cost $c(s_{j-1}, s_j)$ is given by

$$
\begin{aligned}
c(s_{j-1}, s_j) &= \Delta J_r^j + \Delta J_o^j && \text{(From Eq. 3.5)} \\
&\geq \Delta J_r^j && \because \Delta J_o^j \geq 0 \\
&= \sum_{\boldsymbol{p} \in \boldsymbol{\Delta R_{j,\text{true}}}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I}) \\
&\geq \sum_{\boldsymbol{p} \in \boldsymbol{\Delta \tilde{R}_j}} \textsc{Outlier}(\boldsymbol{p}|\boldsymbol{I}) && \because \boldsymbol{\Delta \tilde{R}_j} \subseteq \boldsymbol{\Delta R_{j,\text{true}}} \\
&= \tilde{c}(s_{j-1}, s_j) && \square
\end{aligned}
$$

### 4.5.3 Edge Cost Normalization

The computation of edge costs as defined in Eq. 3.5 (i.e, the total number of unexplained rendered points and unexplained observed points for an object) can result in dramatically different values for objects of different sizes and depths. Objects that are small or further from the camera tend to have fewer points and consequently smaller edge costs. It is well-established in the heuristic search literature that having diverse edge costs (as opposed to a uniform cost for all edges) without a concentrated distribution of solution costs often results in a more difficult problem with many more expansions needed to find the best solution (Wilt and Ruml, 2011; Fan, Müller, and Holte, 2017) than otherwise. Secondly, if the scene were comprised of both small and large objects, large pose errors for the small object would incur the same cost as small pose errors for the large objects. Consequently, this could have the untoward effect of producing solutions where large objects have better accuracies than the smaller ones.

To address both of the above pitfalls, we normalize edge costs by taking the ratio of the explanation cost to the number of points over which the explanation cost was computed. Formally, we set $\Delta J_r^i$ to $\Delta J_r^i / |\boldsymbol{\Delta R_i}|$ and $\Delta J_o^i$ to $\Delta J_o^i / |\boldsymbol{I} \cap V(O_i)|$. Consequently, all edge costs, no matter the size or depth of the corresponding added object, range from 0 (best possible with perfect explanation cost) to 2 (worst possible with all points being unexplained).

### 4.5.4 Precomputed Distance Fields

The final optimization is on the implementation side. A vanilla implementation for computing the explanation cost would need to make several nearest neighbor calls. This is not a problem for $J_{\text{rendered}}$ because the points in the observed scene remain constant across the search and we can construct a nearest neighbor data structure such a

k-d tree once before the start of search. However, computing $J_{\text{observed}}$ is more problematic; we would have to construct a k-d tree for the newly rendered points of every single successor. While the number of points in the newly rendered successor is typically much smaller than the number of observed points, there is still an efficiency loss arising from repeated k-d tree construction.

Using a precomputed distance field for every object eliminates the above problem. Distance fields are simply voxel grids where the value for each voxel is the distance to the closest point on the object. The first step is to generate a distance field for every object with a voxel resolution that is at most twice the explanation cost outlier threshold, and using the reference frame of the 3D model. Then, for computing $J_{\text{observed}}$ for a newly added object $O_i$, we simply transform all observed points in the enclosed volume of the object $V(O_i)$ to the model's reference frame, and look up their distance values. Finally, the distances can be used in determinig the explanation cost (Eq. 3.1).

# Part II

# Discrimination and Deliberation

# Chapter 5

# Discriminatively-guided Deliberative Perception

In Chapter 3, we formulated multi-object localization as model-based search over the joint object poses. This formulation was completely *generative*, relying solely on the object models and rendered hypotheses at test-time to find the most likely scene. While this approach was robust to occlusions and required no training, it could be prohibitively slow at test-time.

In practice, scenes encountered by a perception system typically contain a mix of some "easy" portions, where objects are sitting in isolation, and some "hard" portions, where there might be severe occlusions. The former case is an example where fast discriminative methods do tend to work well, since testing and training data match. Consequently, it would be beneficial to have a system that can automatically figure out the tradeoff between the amount of deliberation versus discrimination required to produce high-quality solutions while being computationally efficient. To this end, we introduce the Discriminatively-guided Deliberative Perception (D2P) paradigm, which extends PERCH by leveraging discriminatively-trained algorithms to guide deliberative reasoning. This is made possible through the Focal-MHA* algorithm, whose details will be covered in Chapter 8. D2P has the following desirable properties: a) it is a *single* search algorithm that looks for the 'best' rendering of the scene that matches the input, b) can be guided by *any* and *multiple* discriminative algorithms, and c) generates a solution that is provably bounded suboptimal with respect to the chosen cost function.

## 5.1 Discriminative Heuristic Generation

We first discuss an approach for generating multiple discriminative heuristics. At a high level, the idea is to obtain a set of hypotheses for every object in the scene and

This chapter is based on material from Venkatraman Narayanan and Maxim Likhachev (2016b). "Discriminatively-guided Deliberative Perception for Pose Estimation of Multiple 3D Object Instances". In: *Robotics: Science and Systems*.

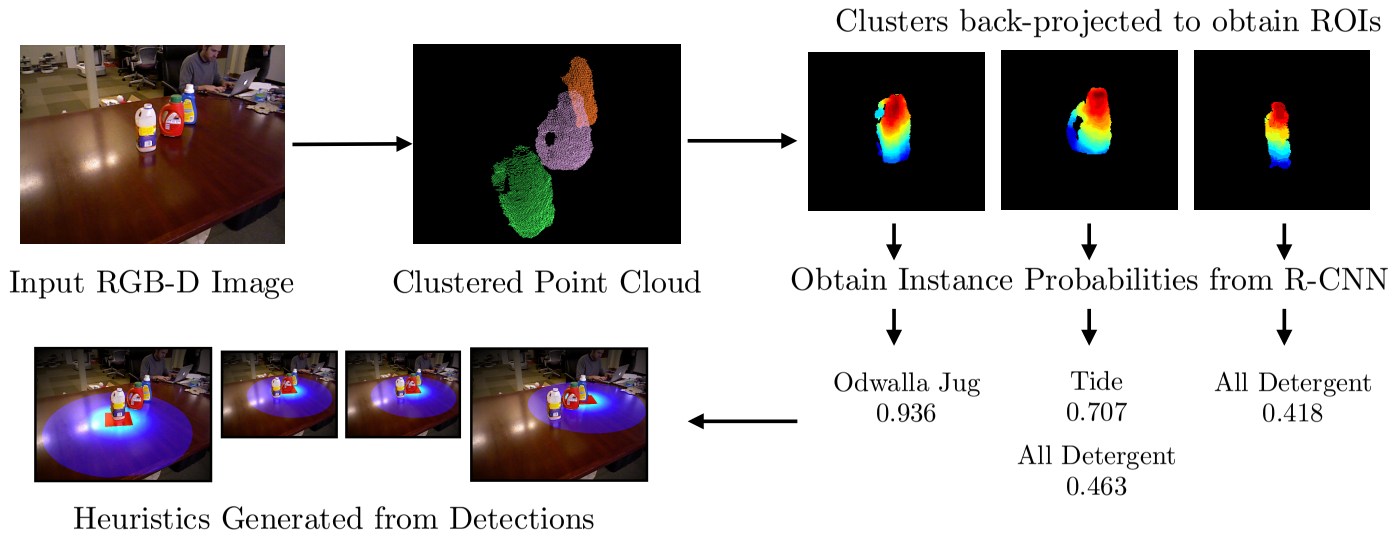



FIGURE 5.1: Discriminative heuristic generation pipeline: First, the point cloud corresponding to the input scene is clustered into $K$ components (for e.g., using PCL's Euclidean cluster extraction) and the points in each cluster are back-projected to obtain ROIs in the depth image. Then, the ROIs are fed to an R-CNN object detector trained on the complete object instance database, after appropriate scaling and colorization. Finally, every high-confidence class prediction for an ROI is converted to a heuristic for global search. In this example, we see that the R-CNN predicts two possible hypotheses for the center ROI, which results in two heuristics being created for that ROI.

treat each hypothesis as a separate heuristic in the Focal-Multi-Heuristic A* (MHA*) framework. This permits the global search to independently explore different routes down the tree by chaining different hypotheses. For instance, hypothesis 1 might help in selecting state $s_1$ from level 1 of the tree, while hypothesis 2 could then be used to evaluate all the states in level 2 that were generated as a consequence of expanding $s_1$. As a result, the search can quickly progress along the optimal route if the hypotheses turn out to be useful, while at all times retaining the ability to backtrack and explore alternative explanations.

While the proposed method is applicable to arbitrary learning algorithms that produce posterior distributions of individual object poses in the scene, we will describe our methodology in the context of object detectors that produce confidence scores for a given bounding box in the depth image, without additional information about 3 DoF pose. Evidently, this is motivated by the availability of successful object detectors from the 2D vision community (Krizhevsky, Sutskever, and Hinton, 2012).

Let $l$ denote the label associated with a unique object model, $B_i$ the set of ROIs (bounding boxes) in the depth image and $c(l|B_i)$ the confidence score for object instance $l$ being present in $B_i$. For every detection with $c(l|B_i) \geq c_{\text{thresh}}$, we generate a heuristic as follows:

$$\bar{p} = \textsc{ProjectToSupportPlane}(\textsc{Centroid}(\{p|p \in B_i\}))$$

$$h_{\text{bbox}}(s_j) = \begin{cases} \infty & \text{if id}(O_j) \neq l \\ 0 & \text{if } \|\bar{p} - T(O_j)\|_p \leq r_{\text{detector}} \\ \|\bar{p} - T(O_j)\|_p & \text{otherwise} \end{cases} \tag{5.1}$$

where $\|\cdot\|_p$ is the $p$-norm and $T(O_j)$ is object $O_j$'s center (assuming all models have been preprocessed such that the $z$-coordinate of their origins have been set to the height of the supporting surface), ignoring the orientation. Essentially, every heuristic acts as "do not care" when the last added object is different from the detection's label and equally prefers all states within the $r_{\text{detector}}$ $p$-ball of the detection's centroid if the labels match. Figure 5.1 illustrates the heuristic generation process. Note that we could have multiple hypotheses for the same ROI (e.g., when the bounding box covers multiple objects in the scene), and the onus falls on the search to resolve conflicts and produce a globally feasible solution.

We evaluate D2P on the real-world occlusion dataset of (Aldoma et al., 2012a), which was also used in Chapter 3.

## 5.2 D2P Implementation

**Parameters.** Table 5.1 lists all the parameters used in D2P. Unless otherwise specified

in particular experiments, we use the following values: $\delta = 0.003$ m, $dx = dy = 0.1$ m, $d\theta = 22.5°$, $w = 10$, max_icp_iter $= 20$, num_procs $= 40$. We perform local ICP adjustments for newly added objects by constraining ICP to match correspondences only if they are within a distance of $dx/2$. All experiments are performed on a single Amazon AWS m4.10x instance with 40 virtual cores, using MPI parallelization to compute edge costs for successor states in parallel.

TABLE 5.1: D2P Parameters.

| | |
|---|---|
| $\delta$ | Sensor noise resolution used in Eq. 3.2 |
| $dx, dy, d\theta$ | The discretizations for $(x, y, \theta)$ coordinates |
| $w$ | Suboptimality factor used by Focal-MHA* |
| max_icp_iter | Max. number of ICP iterations for refinement |
| $r_{\text{detector}}, c_{\text{thresh}}$ | Heuristic-generation parameters (Eq. 5.1) |
| num_procs | Number of processors used for parallelization |

### 5.2.1 R-CNN Heuristics

We generate heuristics for Focal-MHA* using an object detector as described in Sec. 5.1. We leverage a state-of-the-art implementation of region-based convolutional neural networks: Faster-RCNN (Ren et al., 2015). A common trend in the 2D object detection community is to use networks pre-trained on large training datasets such as Imagenet (Russakovsky et al., 2015) to initialize training on a custom dataset. We have two major concerns to address: the generation of training data for the 36 object models in our dataset, and a method to encode depth-images as 3-channel images—the input format used by available deep neural network implementations. We generate our training data by synthetically rendering every object in isolation from camera poses sampled uniformly on concentric cylinders around the object. We also create duplicates of the generated scenes by a) adding artificial noise—treating a randomly chosen $15\%$ of pixels in the rendered image as no-returns, and b) introducing occlusions in the form of a circle placed at a random valid image pixel. The radius of these circles is chosen to be one-third of the rendered object's bounding box. Finally, we obtain 108864 training images in total, with each annotated by a bounding box and label for the object present in it. For encoding depth-images as 3-channel images, we follow the method adopted by (Eitel et al., 2015) who apply a jet-coloring of the $[0 - 255]$ rescaled depth-image. While there is no theoretical justification for this process, the intuition is that jet-color maps encode discontinuities in depth as discontinuities in color, making them suitable for networks pre-trained on RGB images. We use the ZF network architecture (Ren et al., 2015) by modifying the final fully-connected (FC) layer train to span 36 object clases, and use the default 4-stage Faster-RCNN training settings. The training takes $\sim 30$ hours on an Amazon g2.2xlarge GPU-enabled instance. For generating heuristics for the search from object detections in the depth image, we use $c_{\text{thresh}} = 0.2$ (confidence scores are normalized to [0,1]), $r_{\text{detector}} = 0.1$ and $p = 1$ for the norm in Eq. 5.1. Note that we are able to use a high recall threshold because spurious

detections simply translate to uninformative heuristics for the search, without affecting the final solution quality. However, misleading heuristics could have a negative impact on the time taken to find a solution. In addition to the heuristics generated by the deep-learning procedure, we use one additional depth-first heuristic described in PERCH: $h_{depth}(s) = K - |s|$ , where $|s|$ is the number of objects in state $s$. This serves to prefer expanding states deeper in the tree that are closer to a potential goal state. Finally, the consistent heuristic used by Focal-MHA* is the trivial zero heuristic. A qualitative example of running D2P on a test instance is shown in Fig. 5.2.

### 5.2.2 Baseline Implementations

We compare the performance of D2P with PERCH, OUR-CVFH (Aldoma et al., 2012b) and a brute-force ICP (BF-ICP) baseline described in Chapter 3. We configure PERCH to use the same parameters as D2P where applicable, and include lazy edge-cost evaluation. We setup OUR-CVFH and BF-ICP identical to how it was used in the evaluation for PERCH.

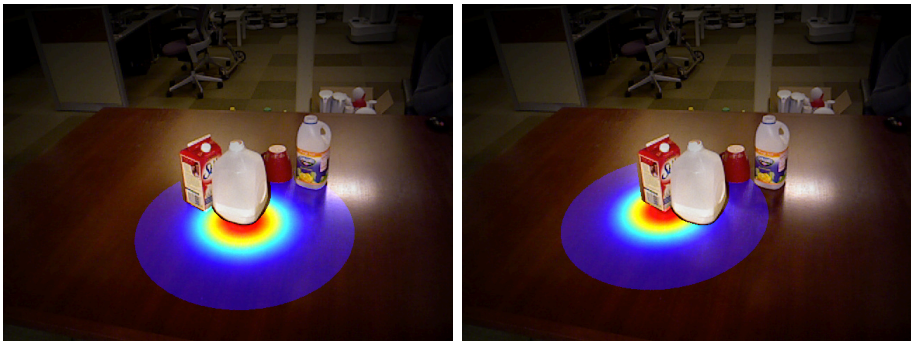
## 5.3 Results

### 5.3.1 Comparison with Baselines

As in Sec. 3.6.4, we measure accuracy of an algorithm by counting the number of objects that fall within a given error bound. Specifically, an estimated object pose $(x, y, \theta)$ is declared 'correct' if the translation error $\|(x, y) - (x_{\text{true}}, y_{\text{true}})\|_2 < \Delta t$ and rotation error $\text{SHORTESTANGULARDIFFERENCE}(\theta, \theta_{\text{true}}) < \Delta\theta$. The second portion is ignored for rotationally symmetric objects. Figure 5.3 compares the performance of D2P with PERCH configured with identical parameters (including lazy edge evaluation), but for the discriminative heuristics. We set an upper limit of 5 minutes for each scene and take the best solution discovered thus far if time runs out. While all experiments are done on m4.10x AWS instances, the objector detector outputs for D2P alone are precomputed for all scenes on an Amazon AWS g2.2xlarge GPU instance (which takes $\sim 0.2$ seconds per scene). The first four plots show the cumulative number of correct poses as $\Delta\theta$ is increased, for a fixed value of $\Delta t$. Two trends are evident: a) D2P dominates PERCH consistently, and b) lower suboptimality factors ($w$) produce more correct poses than higher ones. The latter is expected from the behavior of Focal-MHA*; however it comes at the price of a longer time to find a solution.

Figure 5.4 compares the run times of D2P and PERCH for every scene in the dataset, for a common suboptimality bound of $w = 10$. As we expect to see, D2P has a speedup over PERCH for majority of the scenes. An interesting observation is that there are also few cases where D2P is slower than PERCH. We find this to occur either in scenarios where the heuristics are misleading (e.g., false positives from the RCNN detector), or

(a) (b)



(c) (d)

FIGURE 5.2: (a) The input RGB-D scene. (b) The depth image reconstructed by our algorithm superimposed on the input. (c) & (d) High confidence detections from a region-based convolutional neural network for the milk jug and carton. This example shows how D2P can use the hypotheses generated by a discriminative learner in a global search for the best explanation of the scene.

when there are too many heuristics (due to very high recall) resulting in significant overhead for Focal-MHA*. We believe both these problems can be alleviated to some extent by following a technique similar to the one of (Phillips et al., 2015), where the "progress" made by each heuristic is monitored to intelligently schedule computation to each heuristic, rather than following the naive round-robin scheme used by Focal-MHA*.

Figure 5.5 depicts an identical comparison to OUR-CVFH and BF-ICP. We follow the same methodology as for the comparison with PERCH, and give D2P a maximum time limit of 5 minutes to find a solution. OUR-CVFH being a descriptor-matching method requires no time limit, whereas BF-ICP is provided sufficient time to exhaust all possible orderings of the objects. The results show that D2P consistently dominates the baselines, while showing most gain for strict error measurement criteria. Although BF-ICP performs an exhaustive search over all possible orderings of the object, the lack of any intermediate rendering to account for self-occlusions and occlusions by other objects inhibits its performance. The mean computation time per scene for BF-ICP, OUR-CVFH and D2P ($w = 5$) were 104.35, 5.02, and 139.74 seconds respectively. BF-ICP required no training, while OUR-CVFH needed $\sim$ 14 hours to render the objects from different viewpoints and build the descriptor database (Sec. 5.2.2). The training

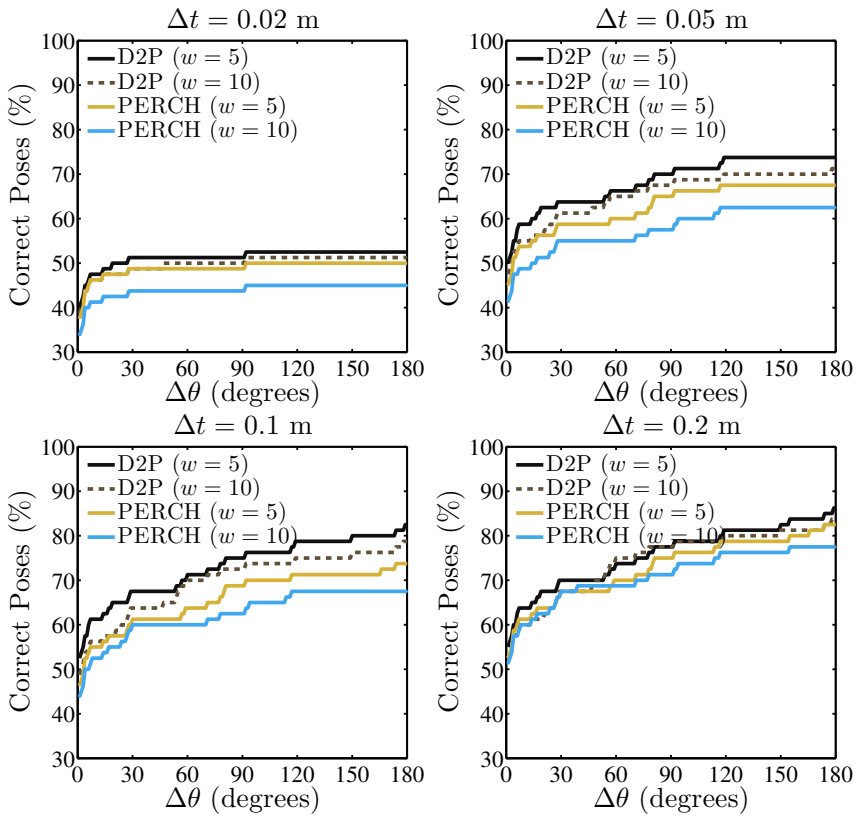



FIGURE 5.3: The first four plots show the percentage of correct poses produced by D2P and PERCH for suboptimality bounds of 5 and 10, where correctness is defined as having translation error within $\Delta t$ and rotation error within $\Delta\theta$. The discriminative heuristics used by D2P help produce a larger number of correct poses within the given time limit of 5 minutes, for identical suboptimality factors.

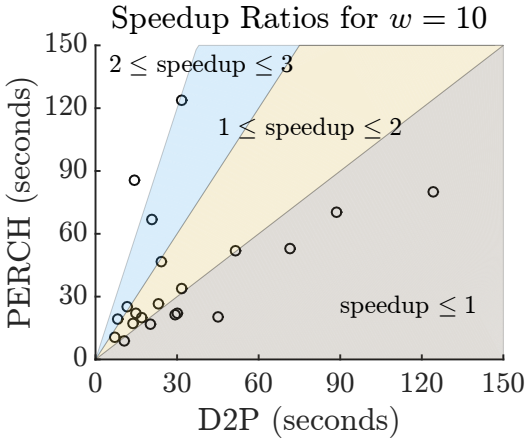



FIGURE 5.4: Each data point in the scatter plot shows the time taken by D2P and PERCH (both run with $w = 10$) for every scene, with different shaded regions representing distinct speedup intervals.

time for D2P depends on the discriminative learner used, which in our particular implementation is the R-CNN. As noted in Sec. 5.2, the training time for the ZF R-CNN

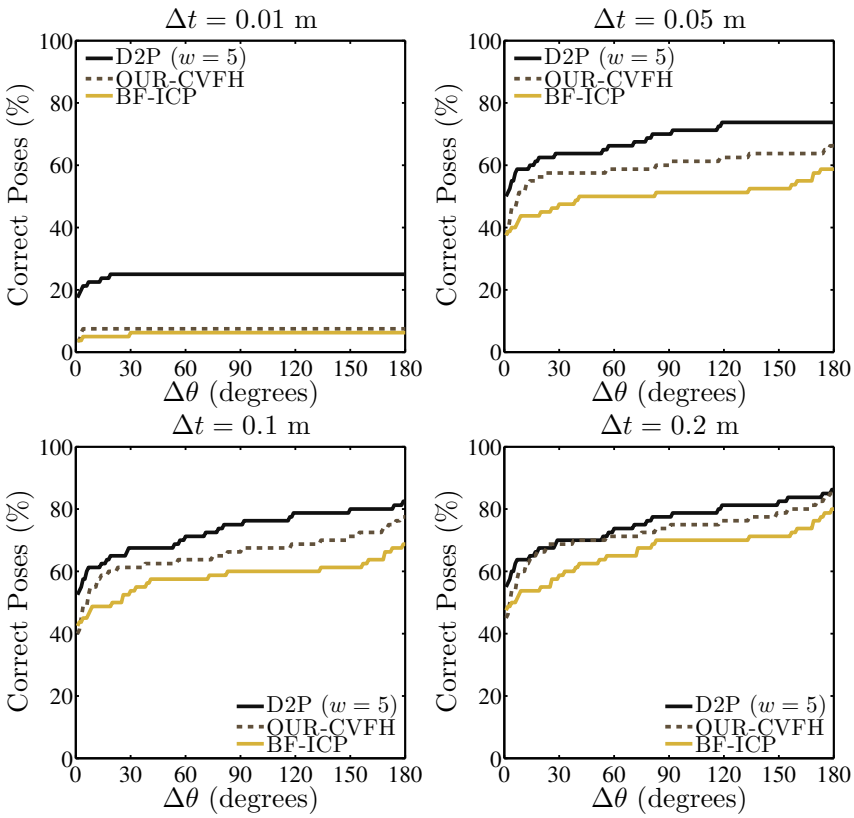



FIGURE 5.5: Comparison of D2P with OUR-CVFH and BF-ICP for different correctness criteria. D2P outperforms the baselines consistently, with the margin being larger for stricter correctness conditions.

was $\sim$ 30 hours.

### 5.3.2 Utility of Lazy Edge Evaluations

We next study how useful lazy edge cost evaluations are, with regard to the branching factor of the tree and the amount of parallelization available. Figure 5.6a plots the mean speedup of lazy D2P over non-lazy D2P for a varying number of processors available, setting $w = 10$. We observe that lazy evaluation is most useful when parallelization is limited and vice versa. If $t$ were the time required to compute the true cost of an edge, $t_{\text{lazy}}(<< t)$ the time to compute the lazy edge cost, $E$ the number of expansions required to find a solution for both lazy and non-lazy variants, $N$ the number of processors used to compute edge costs in parallel, and $b$ the branching factor for every tree state, then non-lazy D2P would take $t(b/N)E$ time to return a solution, whereas lazy D2P would take $tE + t_{\text{lazy}}(b/N)E$ to return a solution. Clearly, the benefits of lazy D2P are pronounced when the effective branching factor ($b/N$) is large. While we vary effective branching as a function of $N$ in Fig. 5.6a, a similar trend would show if we vary $b$ instead, e.g., using a finer discretization or more objects in the scene.

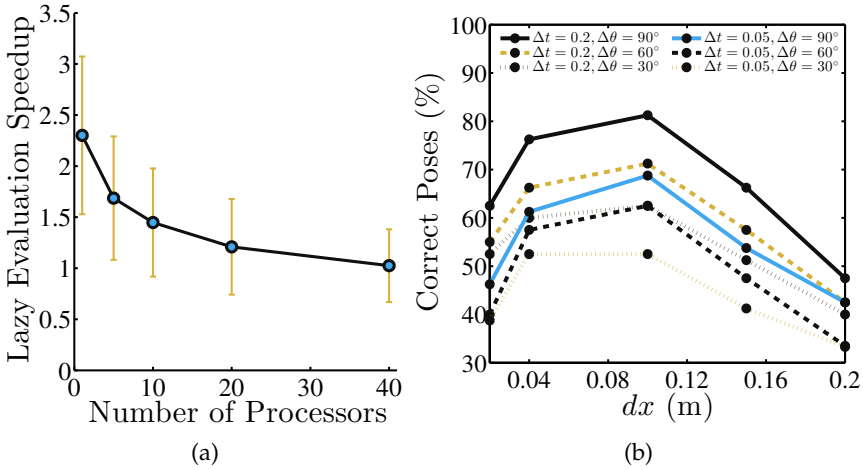



FIGURE 5.6: (a) Speedup obtained by lazy evaluation of edge costs as a function of parallelization. The values are mean speedups across every input scene, with the error bars representing one standard deviation. (b) Performance as a function of the translation-discretization, with each trace corresponding to a specific correctness criterion.

### 5.3.3 Discretization vs. ICP Tradeoff

A key implementation detail is that of local ICP refinement for every newly added object to a successor state. In our implementation, we restrict ICP refinement to only use correspondences that are within $dx/2$ when iteratively estimating the 3 DoF transformation, to keep ICP 'local' to the grid cell at which an object is placed by the search. This immediately introduces the following tradeoff: for coarse discretizations, ICP would have a larger basin of attraction to get to the best fit; the price being that the initial object locations generated by the search might altogether miss small objects. While using a finer resolution might help, it comes at the cost of a larger branching factor and restricted locality for ICP. This tradeoff is captured in Fig. 5.6b, which suggests a sweet-spot somewhere in the middle. More generally, our future work here entails using adaptive-resolution search as well as smarter local refinement techniques to combat the aforementioned problems. This experiment uses a maximum time limit of 10 minutes, to accommodate large branching factors resulting from finer discretizations.

### 5.3.4 Synthetic Example

We conclude the chapter with a synthetic example that reiterates the complementary strengths of discriminative guidance and systematic search. Figure 5.7 shows two scenes containing the same set of 5 objects in different configurations. In the first scene, the objects are mostly isolated and non-occluded. Search by itself takes a long time to obtain a solution (over 13 minutes for $dx = 0.1$ and 8-core parallelization) since it has no informative heuristic. However, by using the guidance from the discriminative R-CNN which correctly identifies all objects, the time to obtain a solution is reduced

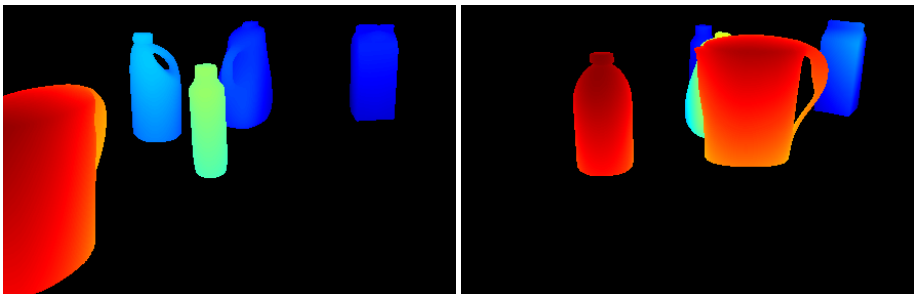


FIGURE 5.7: Synthetic example demonstrating the complementary strengths of discriminative and deliberative methods.

to just under 7 minutes. The second scene is more complex and features severe occlusions. Here, D2P manages to reconstruct the complete scene although the R-CNN correctly identifies only the 3 non-occluded objects.

**Source Code and Reproducibility.** Our implementation of D2P, and the complete experimental setup is available as open-source code at `http://www.github.com/venkatrn/perception`.

# Chapter 6

# RANSAC-Trees for 6 DoF Pose

The efficiency of global reasoning is dictated by two factors: heuristics that guide search, and the size of the search tree itself. While the previous chapter discussed the role of discriminative learners as heuristics, the current one considers the complementary role of statistical learners in constructing the search tree. The primary motivation here is to enable Deliberative Perception for multi-object 6 DoF pose estimation. While a simple discretization of the state space in combination with local alignment (ICP) sufficed for 3 DoF pose estimation, the curse of dimensionality thwarts a similar approach for the 6 DoF case. It becomes essential to consider "smarter" generation of object poses so that the search tree remains small and yet meaningful – i.e, there exists a solution in the tree which is close to the truly optimal one. Towards this objective, we introduce the "RANSAC-Tree" algorithm in this chapter.

## 6.1 Sampling-based Search and Sample Consensus

In tree search problems that deal with large state spaces, sampling-based strategies have been very effective. For instance, in the game of Go, Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006; Silver et al., 2016) has had great success in balancing exploration and exploitation despite the large branching factor. While a similar approach might seem applicable in the context of PERCH, there is a key difference to take into account: game trees are typically deep and care only about the "cost-to-go", whereas the PERCH tree is typically shallow, broad, and considers the "cost-to-come". The role of sampling in game trees is often to approximate the cost-to-go value for a search node, as opposed to limiting the branching factor.

Yet another domain in which sampling-based techniques have been popular is that of robot motion planning, where continuous action and state spaces lead to large search graphs. Over the years, a number of methods have been developed for sampling-based construction of compact search trees and graphs that lead to both feasible (Lavalle,

This chapter presents joint work with Tanner Schmidt, Yu Xiang, and Dieter Fox at the University of Washington, Seattle.

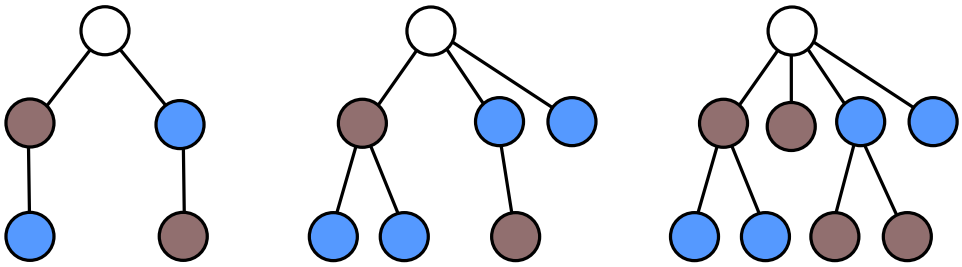


FIGURE 6.1: A toy example showing the incremental growth of a RANSAC-Tree for a scene with two objects (colored brown and blue respectively), batch size of one, and three episodes. The initial tree has one candidate pose for each object. The second episode adds a new candidate pose for the blue object, the final one introduces a new pose for the brown object. Notice how the additional candidates at each episode are added as successors in both levels of the tree.

Kuffner, and Jr., 2000; Kavraki et al., 1996), as well as close-to-optimal motion plans (Karaman and Frazzoli, 2010).

Sampling methods have a rich history in the computer vision field. Traditional pose estimation methods built on top of feature-matching between scene and model points often use a sample consensus technique, such as Random Sample Consensus (RANSAC) (Fischler and Bolles, 1981; Papazov and Burschka, 2010) to reject outlier matches and determine a set of matches that leads to the best "consensus". Of course, the effectiveness of these sample consensus methods relies heavily on the quality of the initial feature matches, which in turn is dependent on the discriminative technique employed.

Inspired by the successes of sampling-based search for large state spaces and sample consensus in single-object pose estimation, we propose the RANSAC-Tree algorithm. The algorithm is based on three observations: i) sampling is essential to control the branching factor of the search-tree, ii) RANSAC is a proven sampling-based technique to generate "good" pose candidates given a set of matches between scene and model points, and iii) discriminative learners can match scene points to model points accurately for the "most" part. Later, we discuss how the vague terms "good" and "most" can be translated to more formal notions of solution quality and likelihood.

Finally, we introduce the LOV (Learning about Objects from Video) RGB-D videos dataset comprising of 6-DoF pose annotated multi-objects scenes, with the objects chosen from the YCB dataset (Calli et al., 2015). The LOV dataset not only permits the study of multi-object pose estimation in complex scenes, but also allows us to leverage contemporary data-hungry deep learning methods in the context of scene-to-model point matching.

## 6.2 Algorithm

The Monotone Scene Generation Tree (MSGT) and PERCH from Chapter. 3 provide the framework for the RANSAC-Tree algorithm. However, unlike PERCH which is a single-shot algorithm (i.e, returns a single final solution), RANSAC-Tree is designed to be an anytime, incremental search which continuously provides better quality solution over time, while re-using search efforts over time. The intuition is that search can be conducted on an increasingly dense tree, where the so-called "densification" arises from using a growing set of individual object poses over time. Algorithm 4 presents the details of the approach and Fig. 6.1 illustrates the construction of a RANSAC-Tree.

As usual, we assume that the number of objects in the scene $K$ is known, and that the objective is to localize all of them (find their 6 DoF poses) in a depth image / point cloud of the static scene. The inputs to the algorithm are similar to that of PERCH (Alg. 2), except for an additional parameter, the batch count $l$. While the approach in previous chapters has been to start off with a fixed set of poses for each object (obtained by discretizing the 3 DoF configuration), we will use $l$ poses for each object to begin with, and consequently increment the number of available poses for an object by $l$ for successive episodes of the search using RANSAC. The object whose candidate pose set is expanded is chosen in a round-robin fashion between episodes (line 4).

To re-use search efforts between episodes rather than planning from scratch, RANSAC-Tree updates the latest search tree (line 26) by inserting edges corresponding to newly available object pose candidates. This is done by iterating through all states that were expanded by the planner during all previous episodes, and re-generating their successors. Consequently, new edges might be added in all $K$ levels of the tree, thereby "fattening" the tree. The newly generated successors are also inserted into the planner's current OPEN list, to make them available for expansion in the next episode. The termination condition for the algorithm could be one of several events: a time limit is exceeded, a maximum allowed number of episodes have been completed, or no more unique object candidate poses can be produced by RANSAC.

## 6.3 Theoretical Analysis

The questions of interest when dealing with sampling-based algorithms are a) is the approach *complete* and/or optimal given enough time (asymptotically)? and b) what can we say about the solutions that are obtained in the interim?

For the purposes of analysis, we will assume that there is a discriminative method that produces a set of $m_i$ matches between scene and model points for each object $i$ in the scene. Note that a RANSAC procedure operating on $m_i$ matches can produce at most $M_i = \binom{m_i}{3}$ distinct object poses.

**Algorithm 4** RANSAC-Tree Search

**Inputs:**
The implicit MSGT construction: SUCC(s) (Eq. 3.4) and edge cost $c(s,' s)$ (Eq. 3.5).
Suboptimality bound factor $w$ ($\geq 1$), and decrement $\Delta w$
Batch count $l$
1 admissible heuristic $h$ and $n$ arbitrary, possibly inadmissible, heuristics $h_1, h_2, \ldots, h_n$.

```
1: procedure UPDATECANDIDATEPOSES()
2:     poses ← RANSAC(object_idx, l)
3:     O_object_idx ← O_object_idx ∪ poses
4:     object_idx ← object_idx + 1 mod K
5: procedure FATTENTREE()
6:     for all s ∈ planner.EXPANDEDSTATES do
7:         for all s' ∈ SUCC(s) do
8:             planner.ADDEDGETOTREE(s, s', c(s, s'))
9: procedure MAIN()
10:    for all i ∈ 1...K do
11:        O_i ← RANSAC(i, l)
12:    object_idx ← 0
13:    s_root ← {}
14:    planner ← Focal-MHA*-Planner()
15:    planner.SETIMPLICITTREE(SUCC(s), c(s, s'))
16:    planner.SETSUBOPTIMALITYFACTOR(w)
17:    planner.SETSTARTSTATE(s_root)
18:    planner.SETHEURISTICS(h, h_1, ..., h_n)
19:    planner.SETGOALCONDITION(return true if |s| = K)
20:    while time limit not exceeded do
21:        success ← planner.COMPUTEPATH()
22:        if success then
23:            {s_root, s_1, s_2, ..., s_goal} = planner.SOLUTIONPATH()
24:            publish s_goal                          ▷ Incumbent solution
25:        UPDATECANDIDATEPOSES()          ▷ Increase available object poses
26:        FATTENTREE()          ▷ Update tree with newly available successors
27:        w ← max(1.0, w − Δw)
28:        planner.SETSUBOPTIMALITYFACTOR(w)
```

### 6.3.1 Asymptotic Properties

Let $T_{\text{universal}}$ denote the PERCH tree constructed using the maximum possible $p_i$ candidate poses for each object. Extending the notion of completeness from Chapter. 3, we define an algorithm as probabilistically complete with respect to $T_{\text{universal}}$ if the probability of finding a valid solution (i.e, a path from the root to a node on level $K$) approaches 1 asymptotically.

**Theorem 2.** *The RANSAC-Tree algorithm is probabilistically complete with respect to $T_{universal}$.*

*Proof.* Let $\{O_0^*, O_1^*, \ldots, O_K^*\}$ denote the object poses of a feasible solution in $T_{\text{universal}}$. Without loss of generality, assume that the object indices also correspond to the levels of the tree. Assume for the sake of contradiction that the algorithm is not probabilistically complete, i.e, it fails to publish any solution at all, despite the existence of one

in $T_{\text{universal}}$. This implies that the planner was unable to find a path from the root to a leaf on level $K$ in all episodes and expanded every node in the tree during the process. Now, since RANSAC uses uniform random sampling to pick 3 from the $m_i$ possible matches in each episode, the probability that it will pick the three which produce $O_i^*$ in any given episode (for object $i$) is non-zero, and approaches 1 as the number of episodes tends to infinity. Consequently, the probability that $\{O_o^*\}$ will be added as a successor to the root node tends to 1. Since the planner failed to return a solution in every episode, the probability that it expanded $\{O_0^*\}$ also tends to 1. This implies that $\{O_0^*, O_1^*\}$ will be added as a successor for $\{O_0^*\}$ with probability 1 (since every expanded state over all previous episodes is re-opened in a new episode). Recursing on the object index $i$, we can conclude that the probability that the planner generated $\{O_0^*, O_1^*, O_K^*\}$ also tends to 1, implying that a solution was found. This contradicts our original assumption. □

**Theorem 3.** *The RANSAC-Tree algorithm is asymptotically optimal (or bounded sub-optimal) with respect to $T_{universal}$.*

*Proof.* For simplicity, we will provide a proof sketch for asymptotic optimality when using optimal A* instead of FOCAL-MHA*. By construction of the RANSAC-Tree, there is always a prefix path of the optimal solution in the planner's OPEN list at the start of an episode (previously expanded states are added back to OPEN). Since A* expands nodes in the order of non-decreasing $g$-values (recall that the admissible $h$ is zero), the prefix path of the optimal solution is guaranteed to be expanded ahead of any other goal state (on the lines of A*'s optimality proof). Now, the probability that expanding the prefix path produces the next optimal successor approaches 1 as the number of episodes tends to infinity, following similar reasoning as in the proof for Theorem 2. Finally, using induction on the prefix path we can conclude that RANSAC-Tree is asymptotically optimal. □

### 6.3.2 PAC-type Bounds

Notwithstanding favorable asymptotic properties, we would like to understand the quality of every intermediate solution published for practical purposes. The Probably Approximately Correct (PAC) framework (Valiant, 1984) has been a popular choice in machine learning for relating sample count to high-probability bounds on low error rates. While ours is a different problem setup, we follow a similar analysis to establish a relationship between the number of batches or candidate object poses, and the quality of the found solution. Fortunately, the theoretical underpinning of RANSAC lends itself naturally to this analysis.

Let $t_i$ be the number of candidate object poses that have been generated for object $i$ at any given instant and let $r_i$ be the ratio of inliers (correct matches) to total number of matches for object $i$. While $r_i$ is not known precisely in practice, one can obtain an

estimate—for example, by looking at the discriminative learner's confidence or uncertainty in generating those matches. Let $p$ be the probability that the optimal solution (with respect to $T_{\text{universal}}$) is returned by the RANSAC-Tree algorithm in the current episode. The probability with which the optimal will *not* be returned is then the same as the probability with which at least one of the optimal object poses was not generated during a RANSAC call until the current episode:

$$
\begin{aligned}
1 - p &= \text{P}(\exists \text{ at least one } i \text{ such that } O_i^* \text{ was not generated}) \\
&\leq \sum_i \text{P}(O_i^* \text{ was not generated}) && \text{(union bound)} \\
&= \sum_i \text{P}(\text{none of the } t_i \text{ candidates include } O_i^*) \\
&= \sum_i \text{P}(\text{at least one of 3 selected matches is an outlier})^{t_i} && \text{(independent events)} \\
&= \sum_i (1 - \text{P}(\text{a randomly selected match is an inlier})^3)^{t_i} && \text{(independent events)} \\
&= \sum_i (1 - r_i^3)^{t_i} \\
p &\geq 1 - \sum_i (1 - r_i^3)^{t_i} && (6.1)
\end{aligned}
$$

Eq. 6.1 provides a handle on relating the confidence of discriminative learners (i.e, the inlier ratio) to the number of candidate object poses needed for guaranteeing a lower bound on the probability of returning the optimal solution. As an example, if we have a scene containing $5$ objects and a discriminative learner produces scene to model point matches of which $50\%$ are correct, then using $30$ candidate poses for each object will suffice to guarantee that we will find the optimal solution with probability at least $0.9$, or $47$ object poses for probability at least $0.99$. The number of candidates for each object could also be different, depending on the individual object's outlier ratio. Finally, we note that this result is valid only in case of using optimal A*, and that further analysis is required for the $w$-suboptimal solution case.

## 6.4 The LOV Dataset

The efficiency of the RANSAC-Tree algorithm is contingent on the ability to obtain a fairly accurate set of matches between scene and model points. Conventional approaches for obtaining scene-to-model point matches involve computing some kind of local feature descriptor (e.g., FPFH (Rusu, Blodow, and Beetz, 2009), Spin Images (Johnson and Hebert, 1999)) for scene and model points, and then matching those based on a distance metric.

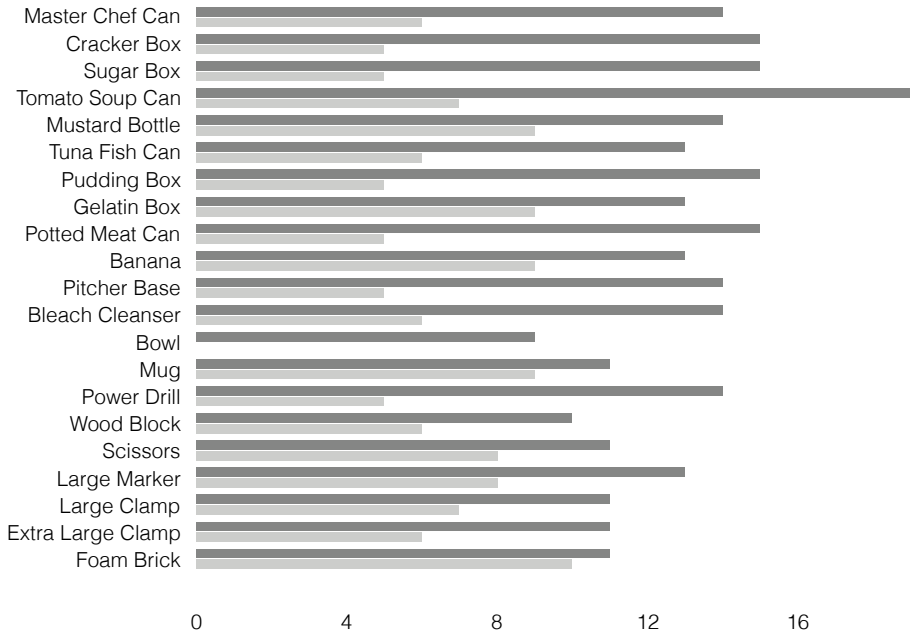



FIGURE 6.2: The number of occurrences of each object in the LOV dataset, with dark bars representing the UW portion and light bars representing the CMU portion.

Unfortunately, the manual design of feature descriptors is often cumbersome and brittle. In recent times, deep learning methods, especially convolutional neural networks, have enabled automatic feature learning for visual data. These methods however requires large-scale labeled datasets—which in our case would require knowing correspondences between scene and model points in all of the data, or equivalently, 6 DoF pose annotation for all objects.

Publicly available datasets for multi-object pose estimation, such as the LINEMOD dataset (Hinterstoisser et al., 2013), Imperial College London dataset (Tejani et al., 2014), Occlusion Dataset (Aldoma et al., 2012a), MIT-Princeton Amazon Picking Challenge dataset (Zeng et al., 2017), Rutgers APC dataset (Rennie et al., 2016) and the T-LESS dataset (Hodan et al., 2017) usually contain an insufficient number of frames or limited variability in the scenes (such as controlled lighting and environment), that makes them unsuitable for deep learning methods. While learning from synthetic datasets (based on rendering) is a promising area of research for augmenting labeled data, a clear methodology has not yet been established.

In the absence of existing large scale real-world datasets that contain 6 DoF annotated scenes comprising of multiple object instances, we prepared our own dataset using 21 objects from the YCB dataset (Calli et al., 2015). To do so, we collected a set of RGB-D videos, each of which contains a distinct scene composed of multiple object

FIGURE 6.3: Representative frames from the LOV dataset (odd rows), along with 2D overlays of the annotated 6 DoF pose (even rows). Scenes include large and small objects that occlude, touch, and rest on each other, all of which present challenges for state-of-the-art techniques.

instances and the camera capturing different perspectives of the scene. In all, we captured 92 video sequences—60 of them in the University of Washington (UW) campus, and 32 in the Carnegie Mellon University (CMU) campus. The total number of frames across all videos is 134,028. Finally, to obtain 6 DoF pose annotation for all objects in all frames, we manually annotate the initial frames of all videos by selecting point correspondences, perform global multi-object optimization based on the depth image to refine those poses, and finally use a state-of-the-art tracker DART (Schmidt, Newcombe, and Fox, 2014) to track the object across the whole video. The tracking step is also interleaved with global refinement of the camera poses at each frame, à la bundle adjustment. Figure 6.2 displays the number of occurrences of each object across the entire dataset, and Fig. 6.3 shows some representative frames from the videos, along with their annotations. This dataset contains some particularly challenging scenes that target the weaknesses of state-of-the-art pose estimation methods. Firstly, many of the scenes contain objects that are touching or resting on each other (including partly contained in the other), which causes problems for methods that rely on perfect segmentation of the scene into distinct clusters for each object. Secondly, the camera perspectives in the videos cover a wide range of angles leading to varying degrees of inter-object occlusions, ranging from no occlusion to completely occluded.

## 6.5 Experiment Details

### 6.5.1 Deep Learning for Dense Object Coordinate Regression

We first describe our pipeline for discriminative scene-to-model point matching, which is integral to the RANSAC-Tree algorithm. Rather than manual design of local geometric features for obtaining matches, we take advantage of the large scale LOV dataset and contemporary deep learning methods to directly learn mappings from raw image pixels to 3D vertex coordinates of an object model, in other words, doing “object coordinate regression". This approach has two distinct advantages over more traditional feature-matching based approaches: i) the discriminative system is trained directly to regress scene pixels to model coordinates (in the so called end-to-end fashion), rather than using a two-step procedure which requires defining a distance function on the feature space, and ii) the approach can regress every single image pixel to an object coordinate in a dense manner, as opposed to obtaining matches only on a sparse set of keypoints, as is typically done. Finally, in addition to the object coordinate regression, the learner is simultaneously trained to produce object class probabilities for all pixels.

Let $d$ be the number of object instances in the dataset, and $\mathcal{M}_i = \{(x, y, z)\}$ be the 3D model for object $i \in \{1, 2, \ldots, d\}$ represented by a set of 3D vertex coordinates (for example, the vertices of a polygon mesh model). Then formally, the regessor learns the mapping $f : \mathcal{I} \rightarrow \{\mathcal{V}, \mathcal{L}\}$, where $\mathcal{I}$ is a $h \times w \times 3$ RGB image (depth channel is not used), $\mathcal{V}$ is a $h \times w \times 3 \cdot d$ image where every pixel is essentially a concatenated vector of $(x, y, z)$ object coordinate predictions for the $d$ objects, and $\mathcal{L}$ is a $h \times w \times d$ label image,

**Algorithm 5** Dense RANSAC

**Inputs:**
Vertex coordinate predictions image $\mathcal{V}$
Label probabilities image $\mathcal{L}$
Depth image $\mathcal{D}$
Object models $\mathcal{M}_i \forall i \in \{1, 2, \ldots, d\}$
Maximum trials $T$

**Output:**
$m$ candidate poses for object $i$

**procedure** RANSAC($i, m$)
  **for all** $t \in 1 \ldots T$ **do**
    candidate_poses $\leftarrow \{\}$
    mask $\leftarrow (\text{argmax}_{j \in \{1,2,\ldots,d\}} \mathcal{L}) == i$
    probability_map $\leftarrow \mathcal{L}$[mask]
    probability_map $\leftarrow$ NORMALIZE(probability_map)
    integral_image $\leftarrow$ COMPUTEINTEGRALIMAGE(probability_map)
    matches $\leftarrow \{\}$
    **for all** $j \in 1, 2, 3$ **do**
      pixel_index $\leftarrow$ SAMPLE(integral_image)
      $s \leftarrow$ UNPROJECT(pixel_index, $\mathcal{D}$[pixel_index])
      $v \leftarrow \mathcal{V}$[pixel_index, $i$]
      matches $\leftarrow$ matches $\cup (s, v)$
    candidate_pose $\leftarrow$ FITPOSE(matches)
    candidate_poses $\leftarrow$ candidate_poses $\cup$ {candidate_pose}
  scores $\leftarrow$ PREDICTIONFITNESSSCORE(candidate_poses)
  **return** $m$ candidate poses with best scores

where every pixel is a vector of object probabilities. Here, $h$ and $w$ represent the height and width of the input RGB-D image. The LOV dataset contains labeled data for this regression: the 6 DoF annotated poses for each scene and known camera calibration allows us to render and compute the pixel-wise object coordinates and labels for the object class.

We use a Fully-Convolutional deep net architecture similar to the single stream network of Yu et al. (Xiang and Fox, 2017), except for an additional output layer that produces object coordinate predictions in addition the softmax probabilities for object classes. For training, we use 48 of the 60 UW videos, and 24 of the 32 CMU videos. All object coordinates (in meters) are normalized by a factor of 10 to reduce output magnitude. The publicly available Tensorflow framework (Abadi et al., 2016) was used for training the network.

### 6.5.2 RANSAC Details

The dense object coordinate regressor provides us for each pixel its most likely object class, the corresponding probability in relation to those for other objects, as well as the

3D object coordinate prediction for that pixel. A RANSAC procedure for generating $m$ object poses for an object $i$ in the scene is given in Alg. 5. Two key differences from a vanilla RANSAC implementation include using biased sampling (based on the object class probability) rather than uniform sampling of matches, as well as using a depth-image based fitness score for evaluating the candidate poses instead of counting inliers based on Euclidean distance between matched points. The former is done efficiently using an integral image of the probability map, and the latter is described below:

- Pre-compute a 3D distance field for every object model.
- Unproject all observed pixels that fall within the rendered mask of the predicted pose.
- Transform all unprojected points to the coordinate frame of the target object model.
- Compute the sum of scores for all transformed points using the pre-computed distance field.

### 6.5.3 Evaluation

The RANSAC-Tree algorithm was evaluated in two modes: the first is a one-shot setting where we take the first solution returned by the algorithm, and the second is the anytime setting where we analyze the solutions generated over time. Both settings have practical applications; the former is useful in situations where a solution is needed as quickly as possible, although not completely accurate (e.g., objects in the foreground might be better localized than the ones in the background), and the latter when more computation time is available for perception, such as when the robot is finishing execution of a previous task.

For the one-shot setting, we compare the results of RANSAC-Tree using different configuration parameters with a greedy baseline. The greedy baseline simply takes the best pose for each object according to the RANSAC fitness score, without doing any global search.

#### RANSAC-Tree Implementation

While the RANSAC-Tree implementation follows that of PERCH for the most part, we incorporate a few of the optimizations that were discussed in Ch. 4 in addition to a variation specific to this algorithm. First, we improve the efficiency of computing $J_{\text{rendered}}$ and $J_{\text{observed}}$ by using distance fields as discussed earlier. In addition, to compute $J_{\text{observed}}$ for an object $O$, we do not consider the observed points with the volume of the placed object $O$, but instead the observed points that fall within the 2D rendered mask (taking into account occlusions from other placed objects) of the object. It is easy to see that this modification preserves the monotone nature of the cost function. More

importantly however, the cost computation for an edge can now be entirely performed using the predicted and observed depth images, without having to consider a point cloud per se. This paves way for efficient parallelized computation of an edge cost in the future (in addition to evaluating multiple edges in parallel as is already done). The second variation is specific to RANSAC-Tree. Since the LOV dataset is contact-rich, candidate poses generated by RANSAC often tend to have small overlaps. Using the traditional occlusion-based pruning in the Monotone Scene Generation Tree therefore leads to poor results, as in most cases there is no collection of candidate poses that are collision-free. Consequently, we modify the successor generation routine in PERCH to allow occlusions of an existing object in the scene, thereby converting the tree to a directed acyclic graph (DAG). Again, it is is easy to convince oneself that if there exists a shortest path in the PERCH tree from root to a final level node, that would still be the shortest path in the DAG. Finally, the use of discriminative object coordinate regression to produce candidate poses instead of discretizing poses eliminates the need to perform ICP at every successor generation step.

**Accuracy Metric**

Predicted object pose accuracy is evaluated based on the Average Distance (AD) criterion (Hinterstoisser et al., 2013) used in recent works on 6 DoF object pose estimation. The AD criterion checks whether the average distance between nearest points in the predicted and true object pose (computed using the known 3D model of the object) is within a fraction $\eta$ of the object's diameter. If $\tilde{T}$ and $T^*$ represent the predicted and true poses (transformation matrices) for object $i$, the average distance is computed as:

$$e_i(\tilde{P}, P^*) = \frac{1}{|\mathcal{M}_i|} \sum_{p \in \mathcal{M}_i} \min_{p' \in \mathcal{M}_i} \|\tilde{T}p - T^*p'\|_2,$$

with the prediction considered correct if $e_i \leq \eta_i \cdot \text{DIAMETER}(\mathcal{M}_i)$. This metric is especially useful for evaluating object pose accuracies when objects have symmetric structure, where simple rotation and translation errors are difficult to characterize precisely. Further, the average distance can be computed reasonably efficiently by using a nearest neighbor datastructure for each model. Note however that the AD criterion is not ambiguity-invariant (Hodaň, Matas, and Obdržálek, 2016), meaning that the error could be non-zero even if the predicted and ground truth poses appear exactly identical in the camera image (for e.g., consider a situation where the coffee mug handle is self-occluded).

### 6.5.4 Results

Table 6.1 shows the object pose estimation accuracies for the UW test data. We used $\eta = 0.1$ in the AD criterion, suboptimality factor $w = 5$ and batch count $m = 5$ for the RANSAC-Tree. The first observation is the variance in accuracy for different objects.

TABLE 6.1: Object-wise pose estimation accuracies for RANSAC-Tree and the greedy baseline, for two different "correctness" thresholds ($\eta$).

| | $\eta = 0.075$ | | $\eta = 0.1$ | |
|---|---|---|---|---|
| **Object** | **RANSAC-Tree** | **Greedy** | **RANSAC-Tree** | **Greedy** |
| Master Chef Can | 90.00 | 87.50 | 100.00 | 97.50 |
| Cracker Box | 63.33 | 56.67 | 76.67 | 70.00 |
| Sugar Box | 88.00 | 88.00 | 96.00 | 98.00 |
| Tomato Soup Can | 80.00 | 76.67 | 91.67 | 88.33 |
| Mustard Bottle | 75.00 | 75.00 | 90.00 | 85.00 |
| Tuna Fish Can | 67.50 | 85.00 | 92.50 | 92.50 |
| Pudding Box | 100.00 | 100.00 | 100.00 | 100.00 |
| Gelatin Box | 100.00 | 100.00 | 100.00 | 100.00 |
| Potted Meat Can | 70.00 | 66.67 | 80.00 | 70.00 |
| Banana | 100.00 | 100.00 | 100.00 | 100.00 |
| Pitcher Base | 96.67 | 96.67 | 100.00 | 100.00 |
| Bleach Cleanser | 62.50 | 62.50 | 82.50 | 80.00 |
| Bowl | 65.00 | 55.00 | 65.00 | 70.00 |
| Mug | 95.00 | 90.00 | 95.00 | 100.00 |
| Power Drill | 100.00 | 100.00 | 100.00 | 100.00 |
| Wood Block | 10.00 | 0.00 | 30.00 | 20.00 |
| Scissors | 60.00 | 50.00 | 80.00 | 90.00 |
| Large Marker | 100.00 | 100.00 | 100.00 | 100.00 |
| Large Clamp | 45.00 | 45.00 | 55.00 | 50.00 |
| Extra Large Clamp | 35.00 | 35.00 | 40.00 | 40.00 |
| Foam Brick | 50.00 | 60.00 | 70.00 | 80.00 |

While a majority have greater than $90\%$ accuracy, few objects such as the bowl, scissors, wood block, clamp and foam brick have lower numbers. The reasons are two-fold: first, symmetric objects such as the bowl and wood block are challenging for RANSAC (scene-to-model matches need to account for the symmetry)) and produce poor-quality candidate poses. Second, objects such as the scissor and clamp often tend to lie flat on the resting surface, leading to low observed cost (points get explained by the surface) even when they are offset from their true poses. Next, we can see that the accuracy for RANSAC-Tree is better or about the same compared to the greedy baseline for most objects. The cases where an improvement is seen are especially in scenes containing occlusions, where we indeed expect global reasoning to play a part. Figure 6.4 shows some qualitative results of the approach.

Figure 6.5 demonstrates the anytime behavior of RANSAC-Tree. Each plot shows the average solution cost for a scene/video as a function of available time, where the average is computed over 10 uniformly sampled frames from the video. The shaded region represents one standard deviation error, and the dashed red line marks the mean time to find the first solution for that scene. Solution costs are normalized by the number of objects in the scene, and a maximum of 5 minutes was provided for each run. From the plots, it is evident that the time to find the first solution increases with the number of objects in the scene, and that solution cost improves over time, albeit at different rates depending on the scene composition.

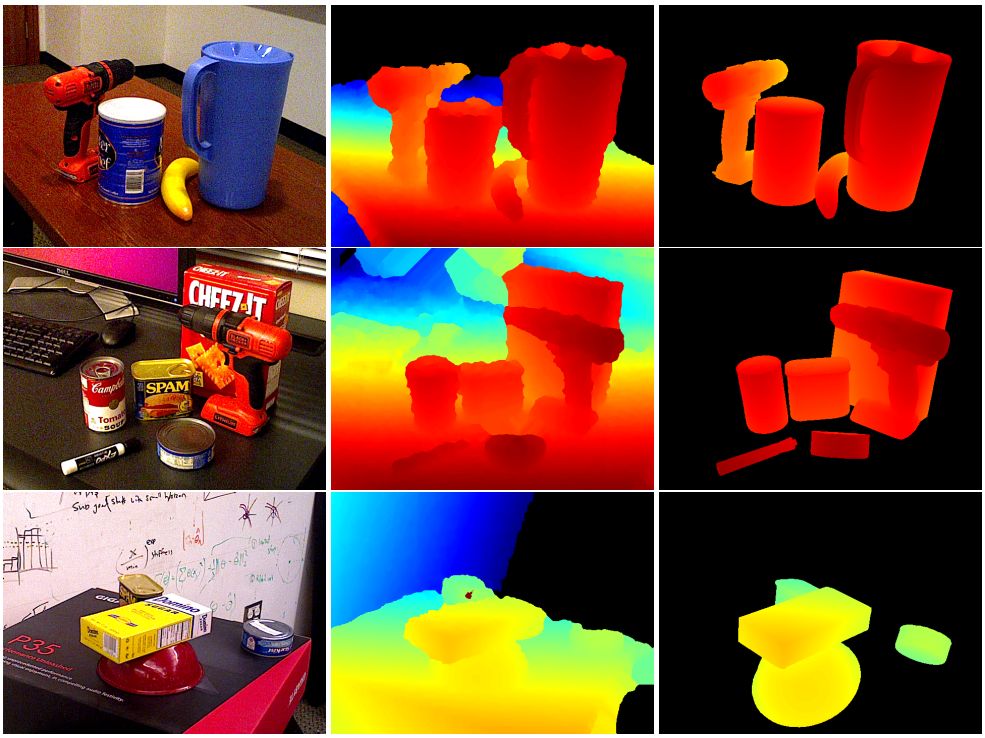



FIGURE 6.4: Qualitative results showing the performance of RANSAC-Tree on three scenes from the LOV dataset. *Left*: Input RGB image, *Middle*: Input depth image (preprocessed to remove "holes" by applying a median filter), and *Right*: Output depth image returned by the algorithm. Note that no local optimization such as ICP has been applied to the result.

## 6.6 Discussion

The work presented in this chapter was motivated by the neccecssity to scale Deliberative Perception to full 6 DoF multi-object pose estimation. The proposed approach was an anytime algorithm that asymptotically approached the best solution that could be obtained given the discriminative learner. While we showed that one could relate sample size to some notion of high-probability, high-quality solution, a lingering question remains: how do we determine which object to add more candidate poses for in successive episodes, and how many samples should we add? The present algorithm simply proceeds in a round-robin fashion and adds a fixed number of candidate object poses in each episode. However, this is a rather arbitrary choice.

The above question in fact leads to a more fundamental one: what should be the objective of an anytime algorithm? Do we want to see *some* improvement in every successive episode of the algorithm, or do we want something less myopic? This question, and a related problem are dealt with extensively in the following chapter.

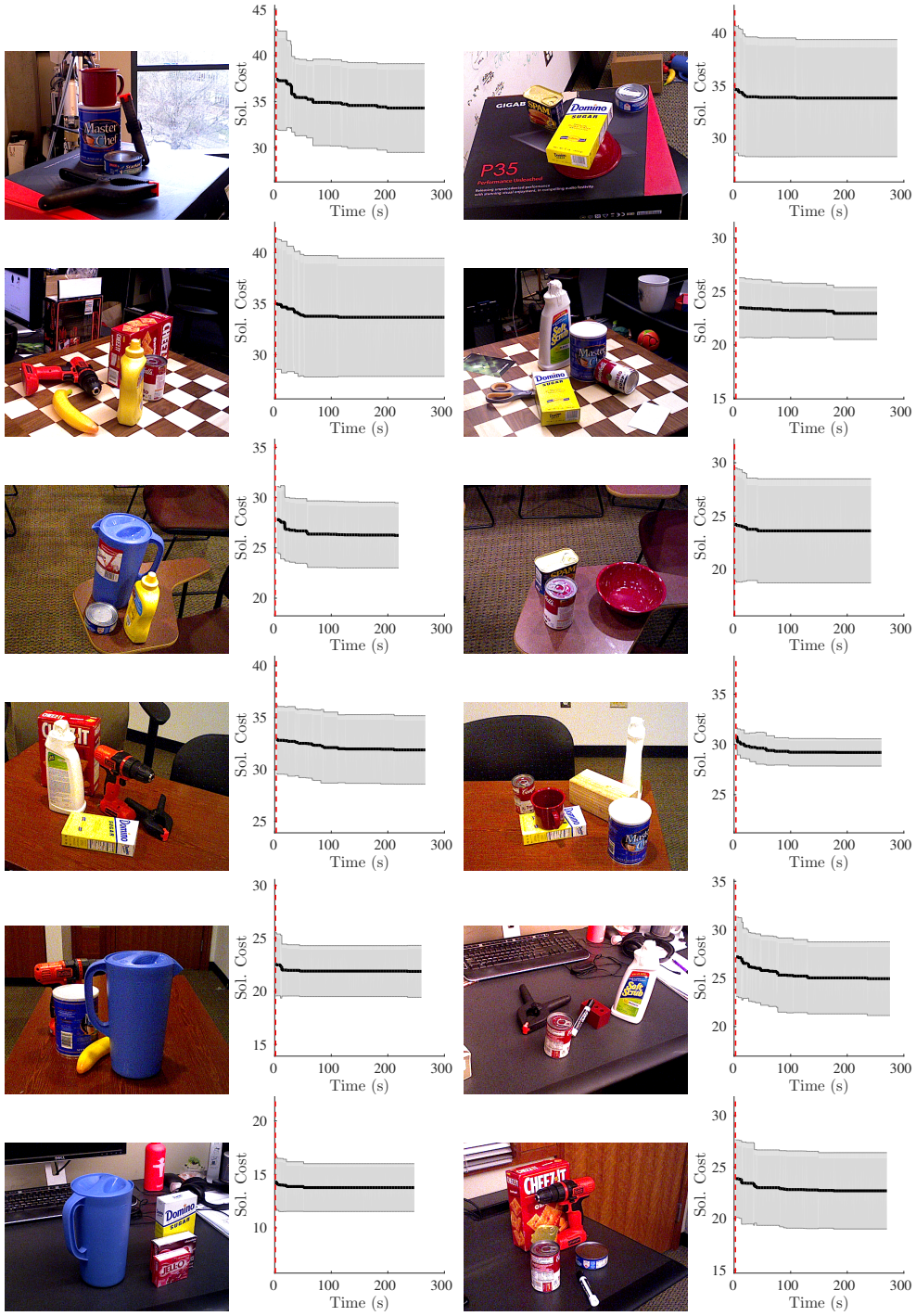



FIGURE 6.5: Anytime performance showing the average solution cost as a function of time for each scene in the UW test set. The shaded region represent one standard deviation, and the dashed red line shows the average time to find the first solution for that scene.

# Part III

# Bridging Heuristic Search and Learning

# Chapter 7

# Anytime Search on Graphs with Time-consuming Edge Evaluations

Chapter 6 introduced an anytime algorithm for continuously improving solution quality over time, by gradually increasing the set of candidate poses from the discriminative learner. However, there are two related, unanswered questions: what should be the behavior of the anytime algorithm?, and how can use discriminative guidance to further reduce the number of edges considered during the search? The latter has profound impact on the performance of the search, especially in the context of Deliberative Perception. The evaluation of edge costs is a multi-step process that involves rendering, ICP, and nearest neighbor calls, which together dominate the total run time of the algorithm.

In this chapter, we propose an approach to address the time-consuming edge evaluations by leveraging edge-existence probabilities provided by a third party (e.g., a statistical learner) to "skip" as many unnecessary edge evaluations (consequently, rendering and ICP calls) as we can while ensuring provably good solution qualities. We discuss a general formulation and algorithms for this problem as well as their theoretical properties. Further, we take a digression to the motion planning domain in which a similar problem of time-consuming edge evaluations arise.

## 7.1 Motivating Examples

Consider the problem of finding shortest paths in graphs where some edges have a prior probability of existence, and their existence can be verified during planning with time-consuming operations. Although Deliberative Perception is our target domain for this problem, we will consider robot motion planning as a proxy since it is

This chapter is based on material from Venkatraman Narayanan and Maxim Likhachev (2017b). "Heuristic Search on Graphs with Existence Priors for Expensive-to-Evaluate Edges". In: *International Conference on Automated Planning and Scheduling (ICAPS)*.

easier to grasp and shares several common features. Specifically, in real-world robot motion planning, edge existence is often expensive to verify (typically involves time-consuming collision-checking between the robot and world models), but edge existence probabilities are readily available.

Our goal is to develop an anytime algorithm that can return good solutions quickly by somehow leveraging the existence probabilities, and continue to return better quality solutions or provide tighter suboptimality bounds with more time. *En route* to this goal, we develop two algorithms applicable to generic graphs with probabilistic edges. They are: a) an algorithm for efficiently computing all *relevant* shortest paths in a graph with probabilistic edges, and as a by-product, the value of the expected shortest path cost, and b) an anytime algorithm for evaluating (verifying the existence of) edges in a collection of paths, which is *optimal* in expectation under a chosen distribution of the interruption time.

Many real world graph search problems involve expensive computation for determining the cost or existence of an edge in the graph. Aside from the expensive edge evaluation in the Monotone Scene Generation Tree, another example is that of robot motion planning, where states in the graph correspond to full configurations of the robot (e.g., a 7 degree-of-freedom manipulator's graph state would be a 7-dimensional vector) and edges between two states exist if the robot can get from the first configuration to the second in a collision-free and kinematically feasible fashion. This existence check requires expensive collision checking between the robot mesh model and the world representation, as well as kinematic feasibility checks, which often turn out to be the most time-consuming parts of the graph search.

In several scenarios however, we do have some prior probabilistic information about the existence of an edge. In the example of robot motion planning, a crude probabilistic model could compute an approximate distance between the robot and the world (without doing the full-blown collision checking) and use that to compute edge existence probabilities. Sophisticated schemes for learning edge existence probabilities online during the graph search are also possible. In this work, we seek to exploit any such probabilistic edge existence information to develop an anytime graph search algorithm that can return feasible solutions quickly. As a step towards this goal, we develop two principled algorithms that are applicable in a variety of related problems:

- **Shortest Paths in Graphs with Probabilistic Edges:** Our problem setup requires finding the shortest path in a graph where some edges have a prior probability of existence as well as a true state (measurable by evaluating those edges). This can be viewed as solving the shortest path problem on a graph which is drawn from a distribution over graphs defined by the edge existence probabilities. The first algorithm, Expected Shortest Paths* (ESP*), efficiently computes the set of all *unique* shortest paths that could result from this distribution, and consequently the expected shortest path cost. To our knowledge, this is the first algorithm

that can efficiently (defined more rigorously later) compute the expected shortest path cost for a graph distribution defined by independent Bernoulli edge existence priors.

- **Optimal Anytime Edge Evaluation:** Assume that we are presented with a set of candidate paths from start to goal that include probabilistic edges. Now at any time, we have the ability to query the true state of an edge, albeit by a time-consuming process. The second algorithm, Anytime Edge Evaluation (AEE*) answers the question of deciding how to evaluate these edges, so that the sub-optimality bound we provide on the solution quality is minimum in expectation whenever the algorithm is interrupted.

## 7.2 Background

The problem of time-consuming edge evaluation in robot motion planning has been touched upon in a number of recent works. These works adopt one of the following strategies to tackle slow collision-checking: i) lazily build the search graph as in Lazy PRM Bohlin and Kavraki, 2000, ii) use collision probabilities, which could be learnt online during planning Huh and Lee, 2016 to bias sampling-based motion planners, or heuristically guide search-based planners as in POMP Choudhury, Dellin, and Srinivasa, 2016, or iii) use lazy search techniques to defer as many edge evaluations as possible, as in LazySP Dellin and Srinivasa, 2016 or Lazy Weighted A* Cohen, Phillips, and Likhachev, 2014. Our work mostly falls under the last camp, where we use edge existence probabilities and estimated evaluation times to determine how to evaluate edges in a lazy fashion.

Other works related to minimizing edge evaluations include Partial Expansion A* (PEA*) Yoshizumi, Miura, and Ishida, 2000 and its enhancement EPEA* Felner et al., 2012, and BEAST Kiesel and Ruml, 2016. The former methods minimize unnecessary node insertions (and hence edge evaluations) when dealing with large branching factors, while the latter provides a method for online estimation and utilization of edge-existence probabilities in the context of abstraction-guided sampling-based motion planning. Another well-known problem that bears resemblance to ours is the Canadian Traveler Problem (CTP) Papadimitriou and Yannakakis, 1991. The major difference between the two is that in CTP, the existence status of an edge is known only when the agent physically moves to the location, as opposed to being able to evaluate an edge at any point in our case. Therefore, we deal with a deterministic shortest path problem (albeit with priors for edge existence), as opposed to the CTP—a deterministic Partially Observable Markov Decision Process (POMDP) Eyerich, Keller, and Helmert, 2010.

In comparison to existing lazy search methods, ours provides the ability to compute an optimal edge-evaluation strategy for a given set of candidate paths en masse (which could also be partial paths to the goal), as opposed to just one path. Moreover, our

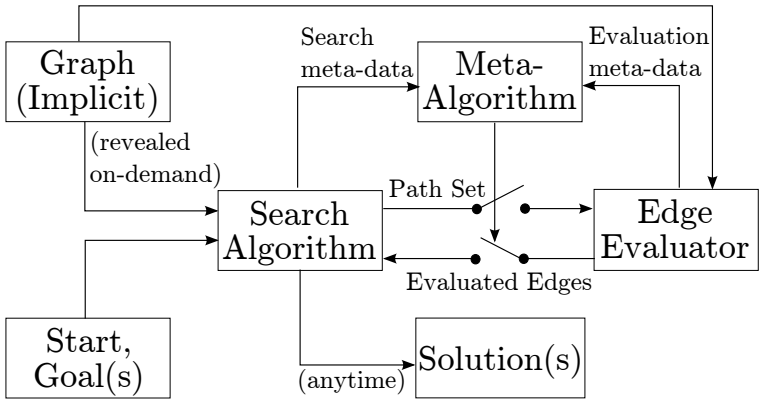



FIGURE 7.1: Strategy for interleaving search and edge evaluations. The search module provides a candidate set of paths (not necessarily from start to goal) to the edge evaluator whenever the meta-algorithm commands to. The evaluated statuses of the edges are then returned to the search algorithm for future use. Also, the search algorithm may choose to publish solutions in an anytime fashion as opposed to just once. Search and evaluation meta-data could include statistics such as time used, number of expansions made, number of paths found, or number of edges evaluated.

method provides greater flexibility in terms of determining when to evaluate edges and when to continue planning—either after a complete set of candidate paths has been found, after a single candidate path has been found, or using some other arbitrary interleaving strategy.

Finally, our approach has an attractive anytime property in that it is provably optimal under expectation with respect to stochasticity in both edge existences and the interruption time, assuming all interruptions occur only after the candidate set of paths has been found. This is a novel definition of optimal anytime behavior, distinct from the one proposed in Thayer, Benton, and Helmert, 2012, where an ideal anytime algorithm is defined as one that reduces the time between distinct solutions. On the other hand, our notion of ideal anytime behavior in AEE* includes both solution quality and the expected interruption time.

In addition to the above distinguishing features, ESP* is the first algorithm to our knowledge that can efficiently compute the exact expected shortest path cost for a distribution of graphs defined by Bernoulli edge existence probabilities.

## 7.3 Overview

The overarching goal of this work is to leverage edge existence probabilities to find solutions as quickly as possible. Right away, this leads to two questions: a) do we care only about the optimal solution or would we rather have an anytime algorithm that returns a feasible path very fast and continues to improve it with time, and b) should the objective be to minimize edge evaluations, or rather minimize the total planning

time, in which case we also need to consider the overhead of the search algorithm as well (i.e, operations such as inserting and deleting from a data structure). If we cared only about minimizing edge evaluations, we could find all possible paths between the start and goal, and then use an "optimal" edge evaluator on those paths for some notion of optimality. The other extreme would be to evaluate every edge that is being considered by the search algorithm, which reduces to a typical search implementation.

Our strategy (shown in Fig. 7.1) is to interleave search and edge evaluations, with flexibility in the choice of how to interleave them. This is similar to the LazySP approach Dellin and Srinivasa, 2016, but has two key novelties : i) the search algorithm can present an arbitrary set of paths to the edge evaluator rather than just 1 path from start to goal, and ii) the search and edge evaluation modules are moderated by a meta-algorithm that determines when each module should be active, instead of using a fixed interleaving policy. The intuition for the first is that the edge evaluator can make more informed decisions by considering paths en masse along with individual existence probabilities of edges. For instance, it might be worthwhile jointly considering the time required for evaluating a path and its probability of existence, if we are interested in an anytime setting. However, a complicated edge evaluator might lead to another time-consuming entity, and that motivates our meta-method. Finally, we assume that the run time of the meta-algorithm is negligible compared to either of the other two modules, so that the planning time is only a function of the search and edge evaluation modules. In the following sections, we will describe our contributions for each module.

## 7.4 Expected Shortest Paths* (ESP*)

The first algorithm, ESP*, is towards the search module whose job is to generate a set of (possibly partial) paths for the edge evaluator. Naturally, we would like these paths to be a diverse sampling of shortest, likeliest, and fastest-to-evaluate paths (which could all be different) so that the edge evaluator can hedge its bets. ESP* answers the question: what are *all* the paths we might care about in making the best edge evaluation decisions under uncertainty? This results in the problem of finding the shortest paths in the graph under all combinations of edge existences— any more information would be unnecessary to the edge evaluator (assuming all deterministic edges have the same evaluation time). Incidentally, an algorithm that achieves this could also compute the expected shortest path cost.

### 7.4.1 Problem Setup

Formally, we have

- $\Gamma = (X, E)$, a finite graph with vertices $X$ and *possible* edges $E$. We use $x_{start}$ and $x_{goal}$ to denote the start and goal vertices.

- $c : X \times X \rightarrow \mathbb{R}^+$, cost of an edge between two vertices ($\infty$ if an edge does not exist).
- $p : E \rightarrow [0, 1]$, Bernoulli prior for existence of $e \in E$.
- $E_b = \{e \in E | p(e) \neq 1\}$, the set of all edges in $\Gamma$ that are stochastic (i.e, there is some chance that the edge might not exist).
- $k = |E_b|$, the number of stochastic edges in $\Gamma$.

The Bernoulli prior $p$ implicitly defines a distribution $P(G)$ over the graphs $G$ that can be generated as sub-graphs of $\Gamma$. The problem we consider is that of finding the set of shortest paths across all ($2^k$) samples drawn from $P(G)$, and consequently the expected shortest path cost $c^*_\mu$ for this graph distribution. If we let $c^*(G)$ denote the cost of the shortest path on graph $G$,

$$c^*_\mu = \mathop{\mathbb{E}}_{G \sim P(G)}[c^*(G)] = \sum_{i=0}^{2^k - 1} c^*(G_i) P(G_i), \tag{7.1}$$

where $P(G_i)$ is the probability that graph $G_i$ is drawn from $P(G)$ (product of the existence and non-existence probabilities of the edges present and absent respectively). Note that $c^*_\mu$ is infinite if there is no deterministic path from start to goal.

### 7.4.2 Algorithm

While a trivial algorithm to compute $c^*_\mu$ would be to find $c^*(G)$ for every realization of $G$, it is impractical due to the exponential cardinality of the sample space. The insight we leverage here is that the set of *unique* shortest paths $S_\sigma$ across all the realizations has a much smaller cardinality than $2^k$ typically, and that it is sufficient to find this set to compute $c^*_\mu$. Clearly, the shortest path amongst all paths in $S_\sigma$ is the one where all stochastic edges are extant, and the longest path in the set is the one where all stochastic edges are nonexistent. The problem now reduces to finding the in-between paths.

Expected Shortest Paths* (ESP*) presented in Alg. 6 solves this task. At heart, it is very much simply A* search, albeit with two modifications: i) it operates on an augmented graph where every state $s$ in the graph is the original graph node $x$ augmented with a bit vector $b$ of length $k$ that represents which stochastic edges have been traversed by a path to $x$, and ii) it applies a state dominance relationship to prune partial paths that are guaranteed to not belong in $S_\sigma$. Intuitively, the augmented graph (Fig. 7.2) keeps track of partial paths along with the set of probabilistic edges that are part of those partial paths. This allows us to apply efficient dominance relationships: given two partial paths to an original graph vertex $x$ each of which traverses probabilistic edges $a, b$ and $a, b, c$ (in any order), and have costs 10 and 20 respectively, the latter partial path definitely cannot belong to $S_\sigma$. That is, the existence or non-existence of edge $c$ is

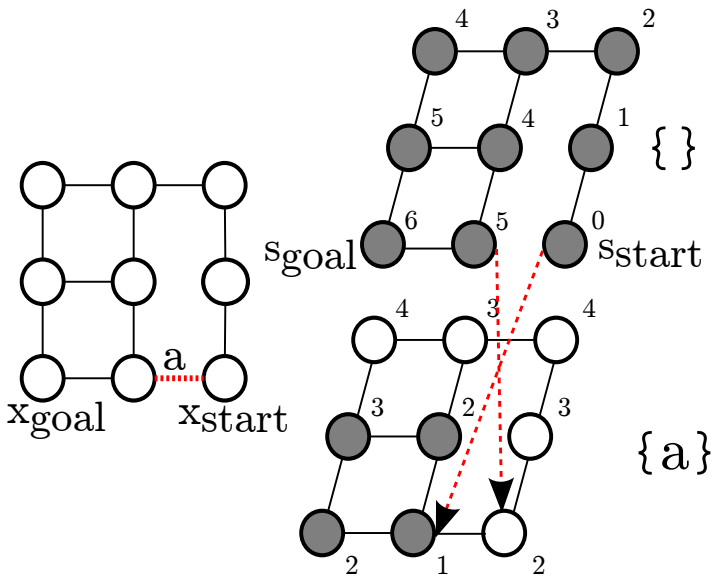



FIGURE 7.2: *Left*: An undirected graph with 1 stochastic edge $a$ and unit costs on all edges. *Right*: The augmented graph searched by ESP*. The upper layer corresponds to paths that have not traversed $a$, and vice versa for the lower one. The numbers next to the vertices represent their $g$-values and the shaded vertices are those expanded by ESP*, assuming a zero heuristic. Notice how some states in the augmented graph are "pruned" despite having smaller $g$-values than $g(s_{goal}) = 6$, through the dominance relationships. For e.g., assuming $(0, 0)$ is bottom-left, the state $[(2, 1), \{a\}]$ with a $g$-value of 3 is dominated by $[(2, 1), \{\})$ with a $g$-value of 1. Intuitively, this is because the path $(2, 0)$—$(1, 0)$—$(2, 0)$—$(2, 1)$ cannot be part of an optimal solution in any instantiation of the graph (sampled edge existences), given that we know the path $(2, 0)$—$(2, 1)$ is shorter and traverses only a subset (in this case empty set) of stochastic edges in the other path.

irrelevant to the shortest path from $x_{start}$ to $x_{goal}$ through state $x$, for *every* $G \sim P(G)$ in which $a$ and $b$ are existent.

**Notation.** In the algorithm, $g(s)$ represents the cost-to-come to $s$ from the start state $s_{start}$, $\tilde{c}(s, s') = c(s.x, s.x')$ is the cost of an edge between $s$ and $s'$, and $\tilde{h}(s) = h(s.x)$ is a consistent heuristic for state $s.x$ in $\Gamma$ (and hence also consistent for the augmented graph). In addition, $p(s)$ is the probability of arriving at state $s$ from $s_{start}$, and OPEN is a priority queue sorted by $f(s) = g(s) + \tilde{h}(s)$. The methods OPEN.TOP() and OPEN.MINKEY() return the best state in OPEN and its $f$-value respectively, while OPEN.EMPTY() returns true if the priority queue is empty. We will use the subset notation $b_1 \subset b_2$ on bit vectors $b_1$ and $b_2$ to indicate that the "set" (equal to 1) bits in $b_1$ are also set in $b_2$. The method NUMNONZEROS($b$) returns the number of set bits in $b$. Finally, we use an edge struct $e$ with three members: the parent vertex $e.first$, the child vertex $e.second$, and the index of the edge in $E_b$ denoted by $e.index$.

The algorithm proceeds like usual A* except for two main differences: Lines 25–27 codify the dominance pruning through checking if the set of traversed edges in a path

**Algorithm 6** ESP*

```
1: procedure ComputeExpectedCost(S_σ)
2:     // #Graph instances where all edges traversed by s_i are existent
           (augmented states s_i are paths in the original graph).
3:     ν[1 ... |S_σ|] = 0
4:     // Probability of getting s_i as the shortest path.
5:     ρ[1 ... |S_σ|] = 0
6:     for all i ∈ 1 : |S_σ| do
7:         ν[i] = 2^{k − NumNonZeros(s_i.b)}
8:         ρ[i] = p(s_i) · ν[i]
9:         for all j ∈ 1 : i − 1 do
10:            if s_i.b ⊂ s_j.b then
11:                ρ[i] = ρ[i] − p(s_j) · ν[j]
12:    return Σ_{i=1}^{|S_σ|} g(s_i) · ρ[i]
13: procedure Succ(s)
14:    S = ∅
15:    for all e ∈ OutEdges(s.x) do
16:        s' = [e.second, s.b]
17:        p(s') = p(e) · p(s)
18:        if e ∈ E_b and s'.b[e.index] ≠ 1 then
19:            s'.b[e.index] = 1                ▷ Update s.b if the new edge is stochastic.
20:        S = S ∪ {s'}
21:    return S
22: procedure ExpandState(s)
23:    Remove s from OPEN
24:    for all s' ∈ Succ(s) do
25:        C = {z ∈ CLOSED | z.x = s'.x ∧ z.b ⊆ s'.b}
26:        if |C| ≠ 0 then
27:            continue                         ▷ Prune path if it satisfies dominance condition
28:        if s' was not seen before then
29:            g(s') = ∞
30:        if g(s') > g(s) + c̃(s, s') then
31:            g(s') = g(s) + c̃(s, s')
32:            if s' ∉ CLOSED then
33:                Insert/Update s' in OPEN with priority g(s') + h̃(s')
34: procedure Main()
35:    b_empty = [0, 0, ..., 0]                 ▷ Bit-vector representing stochastic edges traversed
36:    s_start = [x_start, b_empty], s_goal = [x_goal, b_empty]
37:    p(s_start) = 1, p(s_goal) = 0
38:    g(s_start) = 0, g(s_goal) = ∞
39:    OPEN = ∅, CLOSED = ∅
40:    S_σ = ∅                                  ▷ Set of paths from x_start to x_goal ordered by cost
41:    Insert s_start in OPEN with priority h̃(s_start)
42:    while g(s_goal) > OPEN.MinKey() do
43:        if OPEN.Empty() then return (S_σ, ∞)
44:        s = OPEN.Top()
45:        CLOSED = CLOSED ∪ {s}
46:        if s.x = x_goal then
47:            S_σ = S_σ ∪ {s}
48:            // Prune paths from OPEN which traverse
49:            // all stochastic edges traversed by s
50:            OPEN = OPEN \ {o ∈ OPEN | o.b ⊇ s.b}
51:            continue
52:        ExpandState(s)
53:    S_σ = S_σ ∪ {s_goal}
54:    return (S_σ, ComputeExpectedCost(S_σ))
```

to $x$ is a superset of traversed edges in previously computed paths to state $x$. Lines 47–51 save a newly found path to $x_{goal}$, and prune any states in OPEN that cannot lead to a better solution given the edges traversed by the newly saved path. Note that the

SUCC method not only generates the successor states for an augmented graph state but also computes the arrival probabilities $p(s)$ to those successor states. Finally, the COMPUTEEXPECTEDCOST method processes the set of saved paths and returns the expected shortest path cost.

### 7.4.3 Theoretical Analysis

**Theorem 4** (Correctness)**.** *The* COMPUTEEXPECTEDCOST *function in ESP* (Line. 54) returns the expected shortest path cost* $c^*_\mu$ *(Eq. 7.1).*

*Proof.* (Sketch) A*'s monotone property Hart, Nilsson, and Raphael, 1968 guarantees that states are expanded in non-decreasing order of $f$-values, which for states $s$ where $s.x = x_{goal}$, are simply $g$-values. Thus, if we terminate the search either when OPEN is empty, or when we are about to expand the state $s$ with $s.x = x_{goal}$ and $s.b = [0, 0, \ldots, 0]$ (i.e, corresponding to a deterministic path), we are guaranteed to have found (put in CLOSED) every path to the original goal state $x_{goal}$ that is shorter than the shortest fully deterministic path. The pruning step only removes paths which are longer than an existing path with a subset of traversed stochastic edges, and is therefore admissible. Finally, COMPUTEEXPECTEDCOST simply partitions the $2^k$ possible graph instances into $|S_\sigma|$ groups with distinct shortest paths for each (in non-decreasing order of path costs) much like a priority encoder, and computes the probabilities of occurrence for each group. □

**Theorem 5** (Efficiency)**.** *A vertex $x$ in the underlying graph $\Gamma$ is expanded at most $2^k \cdot (1 - \frac{1}{2^l}) + 1$ times, where $l$ is the number of stochastic edges in the shortest path from $x_{start}$ to $x$ on $\Gamma$.*

*Proof.* (Sketch) Again, by A*'s monotone property, the first instance a state with underlying original vertex $x$ is expanded will be through a path with $l$ stochastic edges. The pruning step (Line 25) guarantees we will never expand a path to $x$ with a combination of stochastic edges that is a superset of the $l$ stochastic edges. The number of ways we can get to $x$ through a superset of those $l$ edges is $2^{k-l}$, implying that the maximum number of additional paths to $x$ that may not get pruned is $2^k - 2^{k-l}$. Including the first path with $l$ stochastic edges, we arrive at $2^k \cdot (1 - \frac{1}{2^l}) + 1$. □

**Corollary 1.** *If the shortest path from $x_{start}$ to $x$ on $\Gamma$ is deterministic (i.e, $l = 0$), ESP* expands at most one state corresponding to each vertex $x$ in the underlying graph.*

Finally, we note that these properties hold only when running optimal A* on the augmented graph. The investigation of these properties when using bounded suboptimal versions such as Weighted A*, as often the case in practice for large graphs, is left for future work.

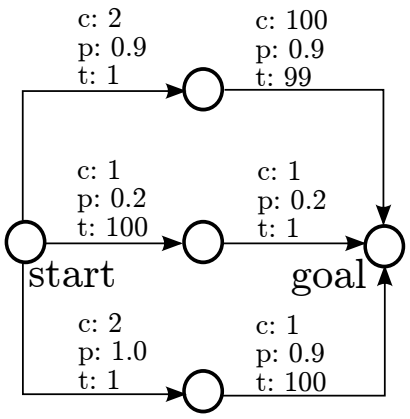



FIGURE 7.3: An anytime algorithm must jointly consider the edge costs, probabilities and evaluation times in deciding what edges to evaluate, as well as define explicitly what its desired "anytime" behavior is. In this example, evaluating the shortest ($c$) or likeliest ($p$) path will be suboptimal for an anytime algorithm that wants to return a feasible solution as fast as possible. On the other hand, evaluating the path with the smallest expected evaluation time ($t$) will be suboptimal for an anytime algorithm that tries to minimize its expected solution suboptimality at an arbitrary interruption time.

## 7.5 Optimal Policy for Edge Evaluation under Anytime Interruption

Let us now assume we have a set of paths $S_\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ between $x_{start}$ and $x_{goal}$ produced by ESP*, and $c(\sigma_i)$ be the cost of path $\sigma_i$, assuming all stochastic edges exist. Let $E = \{e \in \bigcup_{i=1}^{n} \sigma_i | p(e) \neq 1\}$ be the set of all probabilistic edges across the set of paths $S_\sigma$ and $|E| = k$. Let $t_i$ denote the (estimate of) time taken to evaluate the probabilistic edge $e_i \in E$. Clearly, there are multiple ways (Fig. 7.3) one can go about evaluating these edges. If all we care about is the optimal path, then we simply order the paths by increasing cost and evaluate edges in each until we find a path that is valid. If instead, we care only about a feasible solution, we would order the paths by likelihood of their existence.

Our next proposed algorithm, Anytime Edge Evaluation* (AEE*) strives to find a balance between conflicting objectives of finding good quality solutions and finding feasible solutions quickly. We formulate the edge-evaluation problem as a Markov decision problem where the objective is to minimize the expected suboptimality bound of the returned solution at any given interruption time. Formally, let a state $s \in \mathcal{S}$ (different from the $s$ used in ESP*) be a ternary $k$-vector $\{-1, 0, 1\}^k$, where the element $s[i]$ takes the value $1$ if edge $e_i$ is valid, $-1$ if invalid and $0$ if unknown (not yet evaluated). In other words, state $s$ captures information about what edges have been evaluated and what the outcomes were. An action $a_i \in \mathcal{A}$ corresponds to evaluating edge $e_i$. Let $I_\sigma = \{i \mid e_i \in \sigma \wedge p(e_i) \neq 1\}$ denote the index set of stochastic edges in a path $\sigma$. For

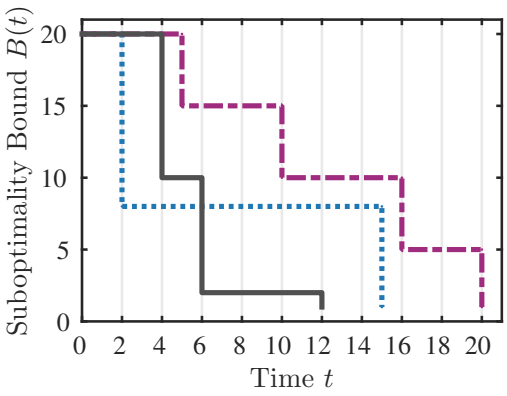



FIGURE 7.4: Illustration of AEE*'s optimization objective. Consider three hypothetical anytime profiles (the suboptimality bound as a function of time) corresponding to distinct edge-evaluation policies. AEE* finds the policy corresponding to the solid line, since it has the least area under its curve. Note that each curve is only a possible realization dependent on the edge existence probabilities, and that AEE*'s objective is to minimize the *expected* area under the curve.

any state $s$, define $B(s)$ as:

$$B(s) = \frac{\overline{B}(s)}{\underline{B}(s)} \qquad \begin{aligned} \overline{B}(s) &= \min_j c(\sigma_j) \text{ s.t. } \quad s[i] = 1 \; \forall i \in I_{\sigma_j} \\ \underline{B}(s) &= \min_j c(\sigma_j) \text{ s.t. } s[i] \neq -1 \; \forall i \in I_{\sigma_j}, \end{aligned}$$

where we assume the minimum of an empty set is $\infty$ and $\infty/\infty = 1$. Here, $\overline{B}(s)$ represents the lowest path cost amongst all paths which we know definitely exist (all stochastic edges have been evaluated as valid), and $\underline{B}(s)$ represents the lowest path cost amongst all paths that are not yet proven to be invalid (i.e., those paths may or may not exist based on the current edge evaluations). If the cost of the optimal path is $c^*$, then we have $\underline{B}(s) \leq c^* \leq \overline{B}(s)$. If the algorithm always returns the path corresponding to the argmin of $\overline{B}(s)$ with cost $c = \overline{B}(s)$ whenever it is interrupted, we have $c \leq B(s) \cdot c^*$, implying that $B(s)$ is a multiplicative suboptimality bound.

From the perspective of an anytime algorithm, we could have the algorithm be interrupted at any time $t$, which for simplicity is presently assumed to be drawn uniformly at random from $[0, T]$, with $T$ exceeding the time taken to evaluate all stochastic edges. At any such time $t$, the algorithm can provide a suboptimality bound corresponding to the last published solution before being interrupted. This suboptimality bound is denoted $B(t)$, with some abuse of notation. We now define the decision making problem (i.e, a mapping from state $s$ to action $a$) as that of choosing actions in a way to minimize the expected suboptimality bound, where the expectation is over the interruption time, as well as the inherent stochasticity in evaluating edges.

Let $p(s'|s,a)$ denote the transition probability, and a policy be denoted by $\pi : \mathcal{S} \to \mathcal{A}$. Then,

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \underset{t\sim U(0,T)}{\mathbb{E}}[B(t)] = \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \int_{t=0}^{T} B(t)dt. \tag{7.2}$$

Let $s_m$ represent the state at (discrete) step $m$ obtained by following the policy $\pi$, $t(s_m)$ represent the (continuous) time at which $s_m$ is reached and $t(a_m)$ represent the time taken to execute action $a_m = \pi(s_m)$. Let us use $s_{m+1}$ to denote the outcome of action $a_m$.

Now, the bound at time $t$, $B(t)$ is the suboptimality bound for state $s_m$, such that $t(s_m) \leq t < t(s_{m+1})$. However note that $s_m$ itself is a random variable drawn from the distribution over states obtained at step $m$ by following the policy $\pi$. If $[s_0, s_1, \ldots, s_m, \ldots, s_M]$ denotes a trajectory obtained by following policy $\pi$, and $T_\pi(s_m|s_{m-1}) = p(s_m|s_{m-1}, \pi(s_{m-1}))$ the transition probability between states $s_{m-1}$ and $s_m$ under policy $\pi$, we have,

$$B(t) = \underset{\substack{s_m\sim \\ T_\pi(s_m|s_{m-1})}}{\mathbb{E}} \left[ \sum_{m=0}^{\infty} B(s_m) \cdot \mathbb{1}(t(s_m) \leq t < t(s_{m+1})) \right],$$

where $\mathbb{1}$ is the indicator function which evaluates to $1$ when the conditional is true, and $0$ otherwise. Using the above in Eq. 7.2, and dropping the expectation distribution subscript,

$$\begin{aligned}
\pi^* &= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \int_{t=0}^{T} \mathbb{E}\left[ \sum_{m=0}^{\infty} B(s_m) \cdot \mathbb{1}(t(s_m) \leq t < t(s_{m+1})) \right] dt \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E}\left[ \sum_{m=0}^{\infty} B(s_m) \int_{t=0}^{T} \mathbb{1}(t(s_m) \leq t < t(s_{m+1}))dt \right] \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E}\left[ \sum_{m=0}^{\infty} B(s_m) \int_{t=t(s_m)}^{t(s_{m+1})} dt \right] \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E}\left[ \sum_{m=0}^{\infty} B(s_m)(t(s_{m+1}) - t(s_m)) \right] \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E}\left[ \sum_{m=0}^{\infty} B(s_m) t(a_m) \right].
\end{aligned}$$

Defining

$$C(s_m) = \frac{1}{T} B(s_m) t(\pi(s_m)), \tag{7.3}$$

the optimal policy $\pi^* = \underset{\pi}{\operatorname{argmin}}\, \mathbb{E}\left[ \sum_{m=0}^{\infty} C(s_m) \right]$.

Note that the expectation in this equation is only with respect to transition uncertainty

due to stochastic edges (the one due to interruption time has been integrated out). Therefore, the decision making problem has been reduced to a stochastic shortest path (SSP) problem, where we minimize the expected sum of future costs for the specific definition of cost (Eq. 7.3), and goal states given by $\{s : B(s) = 1\}$[1].

Intuitively, the term $B(s_m)t(\pi(s_m))$ is simply the area of the rectangle with those two corresponding sides, and we seek to find the policy that minimizes the area under the piece-wise constant curve which represents the suboptimality bound as a function of time, with transition-points on the curve corresponding to time-steps at which an edge was evaluated. Figure 7.4 illustrates this intuition.

In our experiments, we use LAO* Hansen and Zilberstein, 2001 to solve the resulting SSP optimally when it is tractable, or in an online fashion (with fixed time budget for policy computation) when the number of stochastic edges is intractably large. On a practical note, when the set of paths $S_\sigma$ are only partial paths from $x_{start}$ (not all the way to $x_{goal}$), we can still use AEE* on these paths by adding the admissible heuristic estimate of the last node on every path to its current cost.

## 7.6 Evaluation

We evaluate our approach on two domains with distinct properties: the first, a 11 degree-of-freedom mobile manipulation planning problem with dense stochastic edges and the second, a synthetic 2D grid navigation problem with sparse stochasticity. For both domains, we use the augmented graph construction of ESP*, AEE* for edge evaluation, and a meta-algorithm described shortly. An added advantage of using the augmented graph construction is that interleaved search and evaluation can be done without an incremental search algorithm—we only need to update the affected states in OPEN whenever new edge validity information is provided by the evaluator.

**Path-Set Selection.** While ESP* is capable of producing every relevant path we might care about in theory, it is impractical to wait until complete termination of ESP* because of the exponentially large state space. Consequently, we present two methods to obtain a candidate set of paths at any given time: a) select the set of distinct shortest paths from OPEN corresponding to unique sets of stochastic edges traversed, b) select the set of distinct shortest paths from OPEN according to multiple searches, each with its own heuristic (e.g, Fast Downward Stone Soup Helmert, Röger, and Karpas, 2011 or Multi-Heuristic A* Narayanan, Aine, and Likhachev, 2015). We use the first strategy for the grid navigation experiments, and the second for mobile manipulation planning. In the latter case, we specifically add an extra search with a heuristic that guides the search along the likeliest path to the goal. Note that in both cases, the set of paths are only partial paths to the goal, but can nevertheless be used in conjunction with AEE* as discussed.

[1] It is possible to generalize this result to arbitrary distributions for the interruption time, however with the added complexity that the cost function $C(s)$ is now dependent on the arrival time to $s$.

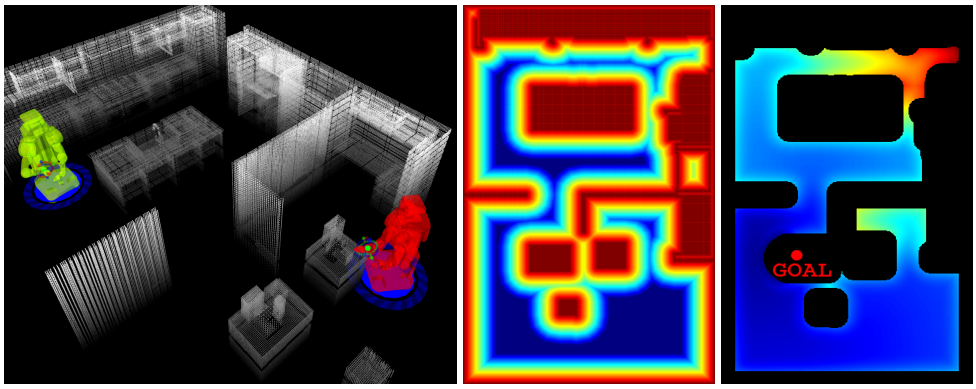



FIGURE 7.5: *Left:* The kitchen domain for 11 DoF mobile manipulation experiments with randomly positioned tables and randomized clutter on top of the tables. *Middle:* The distance transform $d(x, y)$ on the 2D map which is used to compute edge existence probabilities for mobile manipulation. *Right:* The probability heuristic $h_p$ which is used as part of MHA* to guide the search along likely-to-exist paths, for a particular end-effector goal $(x, y)$ location marked by a red dot. Red indicates highest heuristic value and dark blue the lowest value.

**Meta-Algorithm.** For both domains, we use a simple meta-algorithm to switch between searching and edge evaluation. If $t_{search}$ and $t_{eval}$ represent the cumulative times used thus far for searching and edge evaluation, we pick the operation with a smaller $t$ at any given instant[2]. This naturally balances time spent searching versus evaluating. More sophisticated methods that try to estimate the time required in the future for each operation might be possible, but as mentioned earlier, the meta-algorithm's run time needs to be negligible compared to other modules.

### 7.6.1 Mobile Manipulation Planning

Our first domain is 11 degree-of-freedom (DoF) full-body motion planning for the PR2 robot (a dual-arm mobile manipulation robot). The planner's task is to find a collision free motion for the robot to approach and pick up objects on cluttered tables in a kitchen environment, shown in Fig. 8.4. Specifically, the planner controls the position and orientation $(x, y, \theta)$ of the robot's base, the height of the prismatic spine which raises and lowers the torso, the 6 DoF pose of the gripper in the robot's body frame $(x_{hand}, y_{hand}, z_{hand}, roll_{hand}, pitch_{hand}, yaw_{hand})$, and the arm's "free angle" (which way the elbow is pointing).

Each vertex in the graph corresponds to a 11-dimensional robot configuration, while edges correspond to small kinematically feasible motion primitives that the robot can execute (for e.g., one motion primitive changes $roll_{hand}$ by 4 degrees). These edges are valid (existent) only if the motion is collision-free with the environment—which can be verified through expensive collision checking. The cost on an edge is the time

[2]The resolution for switching is either one expansion, or one complete run of the AEE* policy on the incumbent set of paths.

TABLE 7.1: Results for 11 DoF motion planning. Legend: A1: Probability Heuristic+AEE*, A2: Probability Heuristic+Lazy WA*, A3: Lazy WA* (all use Focal-MHA*). Plan times, expands and costs are averaged over instances on which all algorithms succeeded.

| | Success (%) | Plan Time (s) | Expands | Base Cost (m) | Arm Cost (rad) |
|---|---|---|---|---|---|
| A1 | 80 | 8.81 | 971.12 | 4.33 | 5.42 |
| A2 | 77 | 12.79 | 1173.19 | 3.84 | 4.70 |
| A3 | 65 | 10.22 | 1021.72 | 3.92 | 4.76 |

taken to execute the motion along that edge, based on nominal velocities for base and joint angle movements. The start state is fully specified in 11 DoF, while the goal state is underspecified as a 6 DoF gripper configuration to allow the robot to pick up the object from different $(x, y, \theta)$ base locations around the object. We use 16 fixed motion primitives, and 3 adaptive ones that allow the robot to tuck its arm, untuck its arm, and snap the end-effector to the goal end-effector pose when close to the goal region.

We compute the priors on edge existence as follows. Let $R_c$ denote the circumscribed radius of the robot at its fully outstretched configuration and $d(x, y)$ denote the 2D distance from the $(x, y)$ location to the closest obstacle. Let $\mathrm{base}(s)$ and $\mathrm{ee}(s)$ represent the 2D locations of the base and end-effector corresponding to the full robot state $s$. Define $p(x, y) = \min(d(x, y)/R_c, 1.0)$ and $p(s) = \min(p(\mathrm{base}(s)), p(\mathrm{ee}(s)))$. Finally, the probability of existence for edge $(s, s')$ is given by: $p(s, s') = \min(p(s), p(s'))$.

For the search module, we use Focal-MHA* on the augmented graph with three heuristics described in the same: a weighted version of the admissible heuristic , a base-heuristic to guide the robot to a location "behind" the goal with the correct gripper orientation, and another one to guide the base to the same location but with a tucked-arm configuration. We also add an additional probability heuristic to guide the robot along the most likely path to goal, which is computed by running a 2D Dijkstra search outward from the goal on the probability map $p(x, y)$. Additionally, we prioritize expanding from the probability heuristic queue[3] when it is making progress (measurable by monitoring $h$-values) rather than uniformly alternating between the heuristics.

For evaluation, we generated 100 random trials in which the tables are positioned differently every 10 trials, in addition to randomizing the clutter on the tables. For each trial, we choose a random staring configuration (11 DoF) for the robot, and a random pose (6 DoF) on one of the two tables for the gripper to reach. A trial is counted as successful if the planner returns a solution within a time limit of 2 minutes.

Table 7.1 compares AEE* and Lazy WA* (Cohen, Phillips, and Likhachev, 2014) (which is also edge-equivalent to LazySP with the forward edge selector (Dellin and Srinivasa, 2016)) edge evaluation schemes under different configurations. We use a suboptimality bound of $w = 100$ for all algorithms and report statistics only for the first solution found. The main takeaways are the significantly better success rates (solution found

[3]The probability queue also has restricted sharing—all other queues can expand states generated by every other queue, but the probability queue is only allowed to expand states it generated.

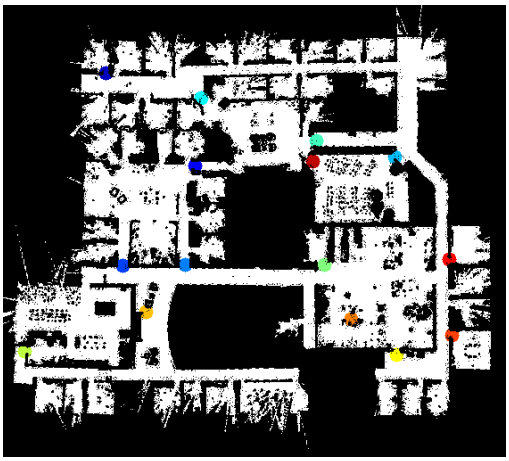


FIGURE 7.6: The 2D grid map used for synthetic benchmarking experiments. The 15 colored circles represent regions (and consequently edges) in the grid that exist with some probability, and are also expensive to evaluate.

within time limit) for the methods which produce a diverse path-set (using the probability heuristic) for edge evaluation, and an improved performance for AEE* due of its probabilistic edge-existence reasoning.

### 7.6.2 Synthetic Benchmarking

We also benchmarked our algorithm on a synthetic 2D grid navigation domain with artificially inflated evaluation times for certain edges. Figure 7.6 shows the 2D map used for the tests. The colored circles represent distinct probabilistic edge groups that may or may not exist in the graph. Further, these edges also have an artificial high evaluation time to verify their existence. To give an example of its relevance, imagine planning in $(x, y)$ state space while doing edge evaluations through full 11-DoF planning that is more expensive. This abstraction is similar to the one used in BEAST (Kiesel and Ruml, 2016), which focuses on online estimation of edge-existence probabilities, where edge evaluations are done using a sampling-based motion planner.

We compare ESP*+AEE* with Lazy WA* Cohen, Phillips, and Likhachev, 2014 with $W = 1$ (i.e., the optimal variant) under two control parameters: the time to evaluate existence of an edge group ($T_e$), and the probability of existence of an edge group ($P_e$). Both these parameters are held constant across all edge groups. We setup the problem such that the time to evaluate an edge in a specific group is large only for the first time, with successive evaluations of other edges in the group taking the same time as any deterministic edge. The motivation for this is twofold: first, this could represent spatial correlation between edges in the configuration space, and second, it simulates the abstraction described earlier in which an expensive edge-group evaluation represents solving a single subproblem. We chose Lazy WA* (with $W = 1$) for comparison since it was shown to perform consistently fast (with regard to total planning time)

TABLE 7.2: Comparison between Lazy WA* and our algorithm, ESP*+AEE* on the synthetic 2D grid navigation domain. Each data point is averaged over 100 trials (10 graph samplings $\times$ 10 random start-goal pairs) for the corresponding settings of edge-evaluation time ($T_E$) and edge existence probability ($P_E$). Legend: A1: ESP*+AEE*, A2: Lazy WA* ($W = 1$). Speedup values are geometric means (arithmetic means in parentheses).

| $P_E$ | | $T_E = 500\ ms$ | | $T_E = 100\ ms$ | | $T_E = 10\ ms$ | |
|---|---|---|---|---|---|---|---|
| | | A1 | A2 | A1 | A2 | A1 | A2 |
| 0.75 | Time (s) | 1.95 | 2.27 | 0.97 | 0.49 | 0.72 | 0.08 |
| | Speedup | 1.78 (12.07) | | 0.83 (3.03) | | 0.18 (0.40) | |
| 0.5 | Time (s) | 2.53 | 2.7 | 1.25 | 0.58 | 0.89 | 0.11 |
| | Speedup | 1.74 (13.5) | | 0.77 (2.45) | | 0.18 (0.38) | |
| 0.25 | Time (s) | 1.83 | 2.44 | 0.96 | 0.53 | 0.71 | 0.10 |
| | Speedup | 2.22 (15.05) | | 0.91 (2.80) | | 0.20 (0.44) | |

on different problems Dellin and Srinivasa, 2016 and also requires no expensive precomputation. For both algorithms, we use a 4-connected grid and compute an admissible heuristic by running Dijkstra's search outward from the goal on the optimistic map (where all edge groups are assumed to exist). Both algorithms are guaranteed to eventually return the optimal solution if one exists, or report failure otherwise.

We run each algorithm on 100 trials (10 random samplings of the graph and 10 random start-goal pairs), for each parameter combination. Table 7.2 presents the average planning times and speedups obtained by Lazy ESP* over Lazy WA*. We use geometric means (GM) to present speedups, as they are better suited than the arithmetic mean (AM) when "averaging" ratios. For instance, two speedup ratios of $2.0$ and $0.5$ have a GM of $1$ and an AM of $1.25$. Since the algorithm was twice faster in one instance and twice slower in the other, a mean of $1$ is more desirable than $1.25$.

Readily noticeable is how ESP* dominates Lazy WA* for an edge-evaluation time of $500\ ms$ and vice versa for $10\ ms$. This is expected: when the evaluation times are small compared to the overhead of just searching (specifically on the augmented graph as done by ESP*), using a complicated path generator or edge-evaluation strategy only hurts. However, the moment edge evaluation becomes a critical bottleneck, using a sophisticated edge evaluation scheme does provide benefits. The results are more balanced for an evaluation time of $100\ ms$, implying that the fewer edge evaluations just about pay for the search overhead. Another interesting observation is how both methods take longer planning times with $P_E = 0.5$ compared to $0.25$ or $0.75$, in some sense confirming the intuition that it is easier to plan under low-entropy distributions.

## 7.7 Discussion

In this chapter, we presented a) a general strategy for interleaving planning and edge evaluation, b) a search algorithm (ESP*) for finding expected shortest paths on a graph with probabilistic edges, and c) a policy for edge evaluation (AEE*) given a set of

candidate paths (possibly partial paths) with unevaluated edges and existence priors. We proved that ESP* can efficiently compute the expected shortest path cost, and that AEE* is optimal for edge evaluation in the anytime interruption sense. The experiments showed how the choice of edge-evaluation scheme is dependent on the problem setting at hand. Practically, we recommend using the ESP* + AEE* combination in domains where edge evaluations are expensive enough to justify the overhead of searching in the augmented graph. Typically, these tend to be domains where stochasticity is sparsely distributed, and where edge evaluation dominates the total planning time. In domains with many stochastic edges, alternative schemes for multiple path generation work well in conjunction with AEE*. Note that either of ESP* or AEE* could be used independently of the other, with a suitable complementary algorithm. For example, one could generate a set of candidate paths (without collision checking) from a sampling-based planner and use AEE* in an interleaved fashion. An important future extension would be to develop a bounded suboptimal version of ESP* to improve its tractability in domains with several stochastic edges. Finally, the integration of ESP* and AEE* into the perception domain and its performance, specifically in the context of RANSAC-Trees remains to be tested.

# Chapter 8

# Improved Multi-Heuristic A*

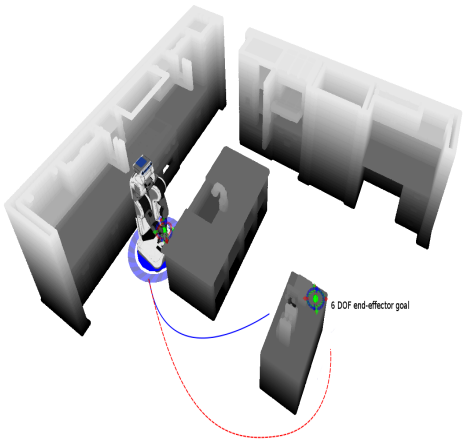
In the previous chapters, we treated multi-object localization as a tree search problem. While this makes the optimization somewhat tractable by obviating the need for exhaustive search over the joint object poses, a very large branching factor for the tree and the lack of an easily-constructible admissible heuristic makes tree search challenging. In this chapter, we introduce a novel graph search algorithm, Multi-Heuristic A* (MHA*), and an extension to it, Improved MHA*, which address the challenges of searching the Monotone Scene Generation Tree. Further, as shown in Ch. 5, these algorithms can be used to incorporate heuristics that arise from learning-based methods into global search, without compromising solution quality guarantees. Finally, these algorithms are applicable to arbitrary graphs, and we demonstrate the generality on various domains.

## 8.1 Motivation

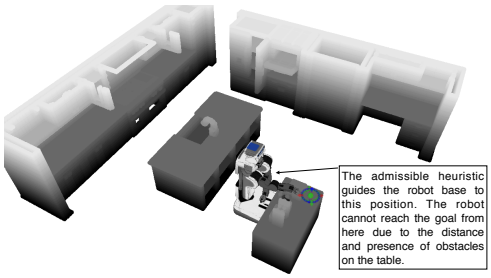
The quality of the heuristic makes or breaks informed search algorithms such as A* (Hart, Nilsson, and Raphael, 1968). An admissible heuristic guarantees optimality whereas an informative (goal-directed) heuristic leads to faster solutions. A common approach used in several high-dimensional state spaces is to use Weighted A* (WA*) (Pohl, 1970), which inflates an admissible heuristic by a factor $w \geq 1$ to give the search a depth-first or greedy flavor. Despite making the heuristic possibly inadmissible, WA* can guarantee bounded suboptimality—the cost of the returned solution is within $w$ of the optimal solution cost.

However, the performance of heuristic search algorithms deteriorates if the heuristic is misguiding or if it causes a *depressison region* (Hernández and Baier, 2012): a region of the state-space where the heuristic values are not correlated with the actual cost-to-go (ideal heuristic) values. This depression region (or "local minimum") causes

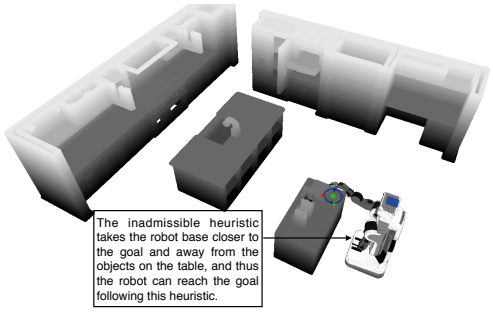
This chapter is based on material from Sandip Aine et al. (2016). "Multi-Heuristic A*". In: *IJRR* 35.1-3, pp. 224–243 and Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev (2015). "Improved Multi-Heuristic A* for Searching with Uncalibrated Heuristics". In: *Eighth Annual Symposium on Combinatorial Search (SoCS)*. This work was completed in collaboration with Sandip Aine, Siddharth Swaminathan, and Victor Hwang.




(a) Heuristics.




(b) Greedy search with the admissible heuristic.




(c) Greedy search with the inadmissible heuristic.

FIGURE 8.1: An example full-body (12D: $(x, y, yaw)$ for the base + spine height + 6 DoF object pose + 2 free angles for the arms) planning problem for the PR2 robot depicting the utility of multiple heuristics. Figure 8.1a shows two heuristic paths that correspond to greedily following the admissible (solid) and inadmissible (dotted) heuristics greedily. Greedily following the admissible heuristic guides the search to a depression region where the search gets stuck (Figure 8.1b), having to expand several states before finding a solution. In contrast, greedily following the inadmissible heuristic guides the search directly to a location from where the target object is graspable, and therefore expands much fewer states during the process (Figure 8.1c).

several unproductive states to be expanded before the search finds a way around. This problem is further aggravated when using inflated heuristics, since the depression region only grows in size.

A key observation we make is that designing a *single* heuristic that is admissible, informative *and* devoid of local minima is often challenging for high-dimensional state spaces. On the other hand, it might be easier to come up with several inadmissible heuristics, that provide complementary guiding powers in different parts of the state space.

Let us take the example of robot motion planning, where the task is to find a collision-free path from a given starting configuration to a goal configuration. Figure 8.1 shows a 12 degree of freedom (DoF) mobile manipulation scenario where the PR2 robot needs to drive and grasp an object on the table, marked by “end-effector goal". Note that the goal location for the robot base is not specified—the planner has to figure out where to move the base so that the object can be grasped by the end-effector.

The admissible heuristic function (path shown by the solid curve, Fig. 8.1a) guides the search to a local minimum region as the robot cannot reach the object from the left side of the table (Fig. 8.1b), which in turn causes the search to expand all states in that region before going around it and eventually finding a solution. However, we could perhaps do better by computing multiple inadmissible heuristics that guide the robot's base to different $(x, y)$ locations around the object to be grasped. In the example (Fig. 8.1a), we show one such additional inadmissible heuristic function that guides the search through a different route (shown by the dotted curve to the right side of the table). Using this heuristic, the search directly goes towards a base location that allows the robot to grasp the object, i.e., this heuristic does not have a local minimum (Fig. 8.1c).

When used in isolation, these additional heuristics provide little value, as they can neither guarantee completeness (because they can be arbitrarily inadmissible), nor can they guarantee efficiency (because each may have its own local minimum). We present an algorithmic framework called Multi-Heuristic A* (MHA*) that builds on the observation that different heuristics might be useful in different parts of the state-space, and together they might yield solutions faster if they can overcome local minima through a concerted effort. By using these multiple inadmissible heuristics in conjunction with a single consistent heuristic, MHA* provides guarantees on completeness and bounded suboptimality of the solution.

## 8.2 Background

### 8.2.1 Related Work

The utility of having multiple heuristics has been noted in many search applications including motion planning (Likhachev and Ferguson, 2008), searching with pattern database heuristics (Felner, Korf, and Hanan, 2004; Korf and Felner, 2002), AND/OR graphs (Chakrabarti, Ghose, and Sarkar, 1992) etc. For example, Likhachev and Ferguson (2008) use the maximum of two admissible heuristics (one mechanism-relative and another environment-relative), as it could guide the planner better when compared to either of the individual heuristics. In (Felner, Korf, and Hanan, 2004), it was shown that a more informative heuristic function can be created by adding multiple disjoint pattern database heuristics, and such a heuristic can substantially enhance search performance.

The procedure of deriving suboptimality bounds based on a consistent heuristic and using inadmissible estimates/constraints to guide the search has also been used in several other search algorithms (Aine, Chakrabarti, and Kumar, 2007; Chakrabarti et al., 1989; Pearl and Kim, 1982; Thayer and Ruml, 2011; Thayer et al., 2012). For example, the $A^*_\epsilon$ algorithm (Pearl and Kim, 1982) uses a distance-to-go estimate to determine the order of expansion among the states whose $f$ values (computed using a consistent heuristic) lie within the chosen bound of the minimum $f$ value in the open list. In

the bounded quality version of Anytime Window A* (Aine, Chakrabarti, and Kumar, 2007), the search is confined within a window of fixed size, as long as the $f$ values do not exceed the bound on the minimum $f$ value among the suspended states (states that are outside the current window). A more recent algorithm, Explicit Estimation Search (EES (Thayer and Ruml, 2011)), uses the same technique as in $A^*_\epsilon$ to provide suboptimality bounds, but improves search efficiency using an additional inadmissible distance function to guide the search.

Algorithms that are closest to Multi-Heuristic A* operationally are Multiple Heuristic Greedy Best-first Search (MH-GBFS) (Helmert, Röger, and Karpas, 2011), and Multiheuristic search via Subgoals (Isto, 1996). The former uses multiple search queues, each of which ranks states greedily with a distinct heuristic. Once a state is expanded, each of its successors is evaluated by every available heuristic, and put into the corresponding search queue.

In relation to the prior work, Multi-Heuristic A* (MHA*) is the first algorithm to our knowledge which can use multiple inadmissible heuristics (in conjunction with a single consistent one) to explore the state-space in a synergetic fashion while providing guarantees on a) completeness, b) solution quality, and c) bounded number of expansions.

### 8.2.2 Notation and Terminology

Consider a state-space planning problem represented as a graph-search problem. Let $S$ denote the finite set of states of the planning domain. For an edge between $s$ and $s'$, $c(s, s')$ denotes the cost of the edge, and if there is no such edge, then $c(s, s') = \infty$. The successor function $\text{SUCC}(s) := \{s' \in S | c(s, s') \neq \infty\}$, denotes the set of all reachable successors of $s$. An optimal path from state $s$ to $s'$ has cost $c^*(s, s')$ and the optimal path from $s_{start}$ to $s$ has cost $g^*(s)$. The best predecessor or backpointer for a state $s$, if computed, is denoted by $bp(s)$.

Let $g(s)$ denote the current best path cost from $s_{start}$ to $s$ and $h(s)$ denote the heuristic for $s$, typically an estimate of the best path cost from $s$ to $s_{goal}$. A heuristic is *admissible* if it never overestimates the best path cost to $s_{goal}$ and *consistent* if it satisfies $h(s_{goal}) = 0$ and $h(s) \leq h(s') + c(s, s')$, $\forall s, s'$ such that $s' \in \text{SUCC}(s)$ and $s \neq s_{goal}$. OPEN denotes a priority queue ordered by some priority function such as $g(s) + h(s)$ or $g(s) + w \cdot h(s)$ with $w \geq 1$.

## 8.3 Multi-Heuristic A*

While two variants of Multi-Heuristic A* (Independent and Shared) are presented in (Aine et al., 2016), we will discuss only the Shared variant here since it is the more

**Algorithm 7** Multi-Heuristic A* (MHA*)

```
1:  procedure KEY(s, i)
2:      return g(s) + w_1 · h_i(s)
3:  procedure EXPANDSTATE(s)
4:      Remove s from OPEN_i ∀ i = 0, 1, ..., n
5:      for all s' ∈ SUCC(s) do
6:          if s' was never generated then
7:              g(s') = ∞; bp(s') = null
8:          if g(s') > g(s) + c(s, s') then
9:              g(s') = g(s) + c(s, s'); bp(s') = s
10:             if s' ∉ CLOSED_anchor then
11:                 Insert/Update s' in OPEN_0 with KEY(s', 0)
12:                 if s' ∉ CLOSED_inad then
13:                     for i = 1, 2, ..., n do
14:                         if KEY(s', i) ≤ w_2 · KEY(s', 0) then
15:                             Insert/Update s' in OPEN_i with KEY(s', i)
16: procedure MAIN()
17:     g(s_start) = 0; g(s_goal) = ∞
18:     bp(s_start) = bp(s_goal) = null
19:     for i = 0, 1, ..., n do
20:         OPEN_i ← ∅
21:         Insert s_start in OPEN_i with KEY(s_start, i)
22:     CLOSED_anchor ← ∅
23:     CLOSED_inad ← ∅
24:     while OPEN_0.MINKEY() < ∞ do
25:         for i = 1, 2, ..., n do
26:             if OPEN_i.MINKEY() ≤ w_2 · OPEN_0.MINKEY() then
27:                 if g(s_goal) ≤ OPEN_i.MINKEY() then
28:                     if g(s_goal) < ∞ then
29:                         Terminate and return path pointed by bp(s_goal)
30:                 else
31:                     s ← OPEN_i.TOP()
32:                     EXPANDSTATE(s)
33:                     Insert s in CLOSED_inad
34:             else
35:                 if g(s_goal) ≤ OPEN_0.MINKEY() then
36:                     if g(s_goal) < ∞ then
37:                         terminate and return path pointed by bp(s_goal)
38:                 else
39:                     s ← OPEN_0.TOP()
40:                     EXPANDSTATE(s)
41:                     Insert s in CLOSED_anchor
```

relevant one to Deliberative Perception. Henceforth, we will use MHA* to refer to the Shared MHA* variant.

### 8.3.1 Algorithm

MHA* assumes that there is one consistent heuristic $h_0$ and $n$ possibly inadmissible heuristics $h_i, i = 1, 2, \ldots, n$. It maintains an *anchor* search (OPEN$_0$), where states are sorted by $f_0(s) = g(s) + w_1 \cdot h_0(s)$ and $n$ inadmissible searches (OPEN$_i, i = 1, 2, \ldots, n$), where states are sorted by KEY$(s, i) = f_i(s) = g(s) + w_1 \cdot h_i(s)$. A key point to note is that the $g$-values for all states are shared across the searches, allowing the algorithm to share paths found by different searches—i.e, we could find a path from $s_{start}$ to state $s$ using heuristic $h_1$, and a path from $s$ to $s_{goal}$ using $h_2$, without explicitly encoding any switching behaviors. Furthermore, path sharing ensures MHA* expands each state at most *twice*—once from an inadmissible search, and once from the anchor search.

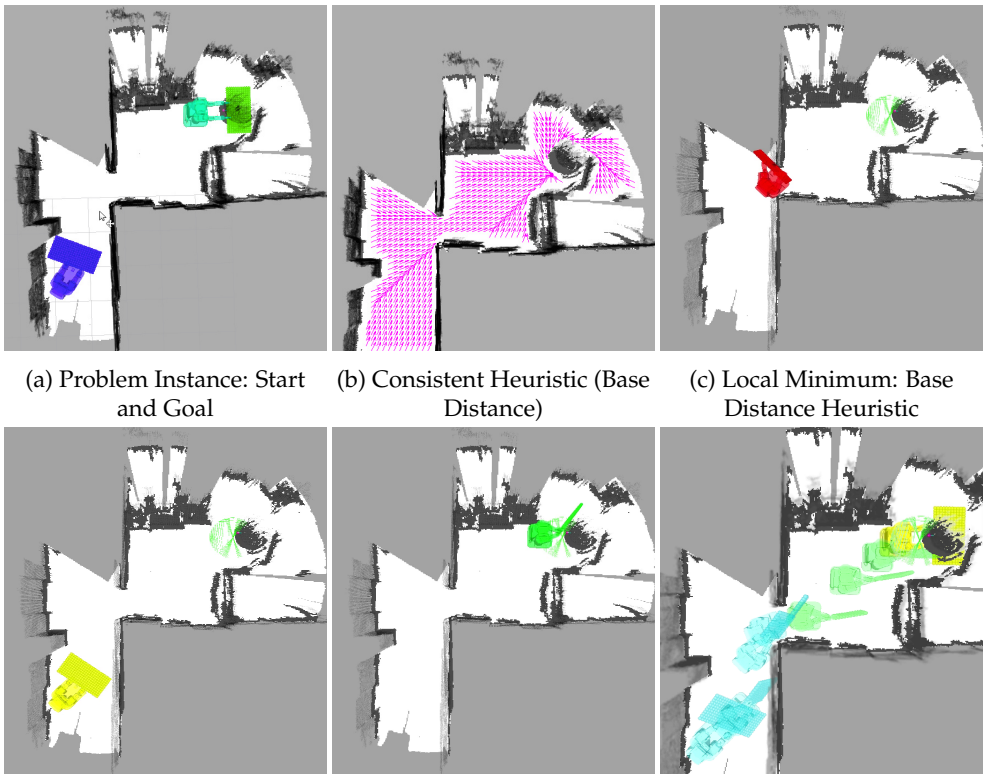
MHA* (Algorithm 7) cycles through each of the inadmissible searches in a round-robin fashion and expands the state at the top of OPEN$_i$ if the condition $\min_{s \in \text{OPEN}_i} f_i(s) \leq w_2 \cdot \min_{s \in \text{OPEN}_0} f_0(s)$ is met, otherwise making an expansion from the anchor search.

When a state $s$ is expanded, its children ($s' \in$ SUCC$(s)$) are simultaneously updated in all the priority queues if $s'$ has not yet been expanded (lines 13-15). If $s'$ has been expanded in any of the inadmissible searches ($s' \in$ CLOSED$_{inad}$) but not in the anchor search (i.e., $s' \notin$ CLOSED$_{anchor}$, check at line 10), it is inserted only in OPEN$_0$. A state $s'$ that has been expanded in the anchor search ($s' \in$ CLOSED$_{anchor}$) is never re-expanded and thus, never put back into any of the priority queues.

The only exception to the simultaneous update (for a state $s'$ not yet expanded) is the optimization at line 14 which ensures that $s'$ is not put into OPEN$_i$ if KEY$(s', i) >$ $w_2 \cdot$ KEY$(s', 0)$, because such a state would never be expanded from OPEN$_i$ anyway (check at line 26). The ExpandState routine also removes $s$ from all OPEN$_i$ (line 4) making sure that it is never re-expanded again in any of the inadmissible searches, and not re-expanded in the anchor search unless its $g$-value is lowered.

MHA* terminates when $g(s_{goal})$ becomes the minimum key value in any of the searches (anchor or inadmissible), and returns a solution whose cost is within the $w_1 \cdot w_2$ bound. The solution path can be reconstructed by greedily following the *bp* pointers from $s_{goal}$ to $s_{start}$. In the event of no solution existing, MHA* terminates when OPEN$_0$ becomes empty.

Figure 8.1 provides an illustration of MHA*'s ability to use different heuristics in a synergetic fashion. By sharing partial paths found by different heuristics, it can overcome nested local minima that would be problematic for any single heuristic.



(a) Problem Instance: Start and Goal

(b) Consistent Heuristic (Base Distance)

(c) Local Minimum: Base Distance Heuristic

(d) Local Minimum: Goal Orientation Heuristic

(e) Local Minimum: Vertical Orientation Heuristic

(f) Solution Path Produced by MHA*

FIGURE 8.2: Illustration of using MHA* for a 12 DoF mobile manipulation planning problem where no individual heuristic can escape all the depression regions by itself (nested depression regions). Here the task for the robot (PR2) is to carry a large object (picture frame) through a narrow corridor and a doorway, and then to put it down on a table. The problem instance is shown in (a), with left and right poses depicting the start and end configurations, respectively. Figure (b) shows the vector field for a consistent heuristic ($h_0$), computed by performing a backward (from goal to start) 2D search for the PR2 base. Figure (c) shows how a search using $h_0$ gets stuck (at the door) as it cannot orient the end-effector correctly, and thus ends up expanding a large number of states at the shown position without moving toward the goal (deep local minimum) before running out of time (1 minute). To address this, we compute two additional (inadmissible) heuristics by including the orientation information for the end-effector, one targeting the goal orientation ($h_1$) and another targeting a vertical orientation ($h_2$). Unfortunately, none of these heuristics are powerful enough to take the search to the goal state as they both suffer from their own depression regions (the searches using $h_1$ and $h_2$ gets stuck at different positions as shown in Figures (d) and (e)). However, as shown in Figure (f), MHA* can efficiently find a plan by using partial paths from different heuristics. It uses the base and vertical orientation heuristics (in parts) to go through the corridor and the door, and then switches to the goal orientation heuristic to align the end-effector to the goal, while not requiring any explicit encoding of the switching behavior.

### 8.3.2 Theoretical Analysis

We present the main theoretical properties of MHA* here, with proof sketches. Complete proofs are available in (Aine et al., 2016).

**Theorem 6.** *At line 25, for any state $s$ with* $\textsc{Key}(s, 0) \leq \textsc{Key}(u, 0)\ \forall u \in OPEN_0$*, it holds that* $g(s) \leq w_1 \cdot g^*(s)$*, and consequently* $OPEN_0.\textsc{MinKey}() \leq w_1 * g^*(s_{goal})$.

*Proof.* (Sketch) At an intuitive level, we can observe that the *anchor* search, $\text{OPEN}_0$ in MHA* is a superset of the OPEN list that would result from running WA* without re-expansions. This is due to the fact that whenever a state $s$ is expanded in any of the searches of MHA*, its children are put into $\text{OPEN}_0$, and it thus includes states from different searches. On the other hand, although $s$ is deleted from $\text{OPEN}_0$ at this point (line 4), it can be re-inserted later if a better path to it is discovered (lowered $g$ value), as long as it is not expanded in the anchor search. The statement of the theorem then results from the fact that the minimum of any set is always less than or equal to the minimum of a subset. □

**Theorem 7.** *When MHA* terminates,* $g(s_{goal}) \leq w_1 \cdot w_2 \cdot g^*(s_{goal})$*, i.e., the solution cost is bounded by a* $w_1 \cdot w_2$ *suboptimality factor.*

*Proof.* MHA* can terminate successfully in lines 37 (anchor search) or 29 (inadmissible search), or it can terminate without a solution in line 24.

If the anchor search terminates at line 37, i.e., $\textsc{Key}(s_{goal}, 0) \leq \textsc{Key}(u, 0)$, $\forall u \in \text{OPEN}_0$, from Theorem 6 we have,

$$\begin{aligned} g(s_{goal}) &\leq w_1 \cdot g^*(s_{goal}) \\ &\leq w_1 \cdot w_2 \cdot g^*(s_{goal}) \\ &\quad \text{(since } w_2 \geq 1.0 \text{ by assumption)} \end{aligned} \tag{8.1}$$

If an inadmissible search (say $i^{th}$ search) terminates in line 29, then from lines 26 and 29, we have,

$$\begin{aligned} g_i(s_{goal}) &\leq w_2 \cdot \text{OPEN}_0.\textsc{MinKey}() \\ &\leq w_2 \cdot w_1 \cdot g^*(s_{goal}) \text{ (From Theorem 6)} \end{aligned} \tag{8.2}$$

Therefore, in both the above mentioned cases, i.e., if either the anchor terminates or an inadmissible search terminates, we have the solution cost to be within $w_1 \cdot w_2$ factor of the optimal solution cost.

On the other hand, if the search terminates unsuccessfully at line 24 (while condition is not satisfied), from Theorem 6 we know $\text{OPEN}_0.\textsc{MinKey}() \leq w_1 * g^*(s_{goal})$. Now, $\text{OPEN}_0.\textsc{MinKey}() \geq \infty \implies g^*(s_{goal}) \geq \infty$, i.e., there is no finite cost solution.

□

**Theorem 8.** *During the execution of MHA*, a) no state is expanded more than twice, b) a state expanded in the anchor search is never re-expanded, and c) a state expanded in an inadmissible search can only be re-expanded in the anchor search if its $g$-value is lowered.*

*Proof.* In MHA*, a state $s$ can only be expanded when it is selected as the top state of OPEN$_i$ in either line 31 or 39.

If $s$ is selected for expansion in line 31, the very next call is to the function ExpandState (line 32), which removes this selected state from OPEN$_i, \forall i = 0..n$ (line 4). Also, after the ExpandState call, $s$ is inserted in CLOSED$_{inad}$ (line 33). Now, a state (other than $s_{start}$) can only be inserted in OPEN$_i$ ($i \neq 0$) in line 15. If a state $s$ has already been expanded in any of the inadmissible searches (i.e., $s \in$ CLOSED$_{inad}$), the check at line 12 will ensure that $s$ is not inserted again in OPEN$_i$ ($i \neq 0$).Therefore, a state can only be expanded once in the inadmissible searches.

Now, when a state $s$ is expanded in the anchor search, similar to the earlier case, here also, $s$ is removed from all OPEN$_i$ (line 4) and inserted to CLOSED$_{anchor}$. Thus, $s$ can only be expanded again either in inadmissible searches or in anchor search, if it is re-inserted in any of the OPEN$_i$, which can only be done in lines 11 or 15. However, as $s \in$ CLOSED$_{anchor}$, the check at line 10 will never be true, meaning control will never reach lines 11 or 15, i.e., $s$ will never be re-expanded. Therefore, statement b) is true. Since $s$ can be expanded at most once in the anchor search and at most once in the inadmissible searches, i.e., $s$ cannot be expanded more than twice, statement a) is true.

Finally, a state $s$ that has been expanded in an inadmissible search, can only be expanded in the anchor search later if it is re-inserted in OPEN$_0$. A state can only be inserted in OPEN$_0$ (any OPEN$_i$, for that matter) if the check at line 8 is true, i.e., if its $g$-value is less than its earlier $g$-value. Thus, a state $s$ whose $g$ has not been lowered after its expansion in any inadmissible search will never satisfy the condition in line 8, and will not be re-inserted in OPEN$_0$—implying that it can never be expanded in the anchor search. Therefore, statement c) is true. □

### 8.3.3 Evaluation

We evaluated the performance of MHA* for the following domains: 12 DoF mobile manipulation planning for the PR2 robot, 3 DoF $(x, y, orientation)$ navigation for single and multiple goal domains, and sliding tile puzzles. Here, we present results only for the 12 DoF mobile manipulation domain, and refer the reader to (Aine et al., 2016) for the other domains.

#### Mobile Manipulation Planning

We evaluate MHA* on a mobile manipulation planning domain for the PR2 robot. The PR2 mobile manipulation robot is a dual-arm robot (7 degrees of freedom each) with

an omni-directional base and a prismatic spine. In our experiments, we used a state space representation similar to that used in (Cohen, Chitta, and Likhachev, 2010). We represent robot state with $12$ degrees of freedom: a $6$ DoF object pose, $2$ redundant arm joints, 2D Cartesian coordinates for the base, an orientation of the base, and the prismatic spine height. The planner was provided the initial configuration of the robot as the start state. The goal state was specified as a $6$ DoF position of the object, which made it inherently under-specified—i.e, there are no constraints on the position of the robot base or the redundant joint angles. The actions used to generate successors for states were a set of motion primitives, which are small, kinematically feasible motion sequences that move the object in 3D Cartesian space, rotate the redundant joint, or move the base in a typical lattice-type manner (Likhachev and Ferguson, 2008). The prismatic spine was also allowed to adjust its height in small increments.

**Heuristics.** We compute the admissible heuristic for the anchor search by taking the maximum value of the end-effector heuristic and the base heuristic, defined next. The end-effector heuristic is obtained by a 3D Dijkstra search initialized with the ($x$,$y$,$z$) coordinates of the goal and with all workspace obstacles inflated by their inner radius, while the base heuristic is obtained using a 2D Dijkstra search for the robot base where the goal region is defined by a circular region centered around the ($x$,$y$) location of the $6$ DOF goal. The purpose of this circular region is to maintain an admissible heuristic despite having an incomplete search goal. As the set of possible goal states must have the robot base within arm's reach of the goal, we ensure that the heuristic always underestimates the actual cost to goal by setting the radius of the circular region to be slightly larger[1] distance than the maximum reach of the robot arm.

We generate our inadmissible heuristics as follows: First, we randomly select $2$ points from the base circle around the goal with valid inverse kinematic (IK) solutions for the arm to reach the goal (end-effector location) and run 2D Dijkstra backward searches (to the start state) starting from these $2$ points. This gives us two different base distances. Next, we compute an orientation distance heuristic as the angular difference between the current base orientation (at a given state) and the desired orientation, where the robot faces the end-effector goal. These distances (base and orientation) were then added to the end-effector heuristic to compute the final heuristic values. Note that this informative heuristic is clearly inadmissible, as the desired orientation and locations for the base might not be the optimal ones.

Finally, we augment this set of heuristics with the base (2D Dijkstra + orientation) and the end-effector heuristics (3D Dijkstra) as two additional ones. Since MHA* can share paths found by different heuristics, we might potentially benefit from not combining two heuristics into a single one, and hence the augmentation with the individual heuristics.

[1] We use a slightly larger distance to be conservative and thereby avoid any discretization errors which may make the heuristic inadmissible. However, if perfect measurements are available, the exact value of the maximum reach of the robot arm can be used as the radius.

| | WA* | MH-GBFS | MPWA* | EES | MHA* |
|---|---|---|---|---|---|
| SR | 31% | 76% | 36% | 27% | 81% |
| SE | 1.08 | 0.78 | 3.84 | 1.54 | 1.0 |
| RT | 0.99 | 0.91 | 2.82 | 1.54 | 1.0 |
| SC | 0.95 | 1.57 | 0.97 | 0.93 | 1.0 |

TABLE 8.1: Comparison between WA*, MH-GBFS, MPWA*, EES and MHA* for PR2 manipulation planning in kitchen environments. The first row (SR) shows the percentage of total problem instances solved by each planner. The other rows include the results as a ratio between the algorithm marked in the column heading and the corresponding MHA* numbers, when both of them solved an instance. Legend: SR - success rate, SE - state expansion ratio, RT - runtime ratio, SC - solution cost ratio.

For each trial of the experiment, we randomly generated a full robot configuration anywhere in the kitchen for the start state, while generating a valid goal state that lies above the tabletops containing clutter. We generated $15$ such environments by randomly changing the object positions and for each such environment we used $10$ different start and goal configurations. All the experiments were performed on an Intel $i7-3770$ ($3.40GHz$) PC with $16GB$ RAM.

**Comparison with Graph-Search Methods.** We primarily benchmarked MHA* against WA* without re-expansions (as in ARA* (Likhachev, Gordon, and Thrun, 2004)/RWA* (Richter, Thayer, and Ruml, 2010)), multiple heuristic greedy best first search (MH-GBFS) (Röger and Helmert, 2010), multiple parameter WA* (MPWA*) (Valenzano et al., 2010) and Explicit Estimation Search (EES) (Thayer and Ruml, 2011) when applicable.

Since MHA* uses two suboptimality bounds ($w_1$ and $w_2$) in comparison to one $w$ used by WA*/MPWA*/EES, we set $w_2 = min(2.0, \sqrt{w})$ and $w_1 = w/w_2$, for all our experiments (and the examples), so that the solutions are guaranteed to be within the $w$-suboptimality bound.

For MH-GBFS, we used the same heuristics as used for MHA*. For MPWA*, we used the admissible heuristic with $5$ different weights ($0.2 \times w$ to $1.0 \times w$, with $0.2$ gap; where $w \geq 10.0$). For EES, we used an inadmissible distance measure, one inadmissible heuristic function (from the set used for MHA*) and the admissible heuristic. We ran WA*, MPWA* and MH-GBFS without state re-expansions as re-expansions can significantly increase planning time. Both WA* and MPWA* can provide solution quality bounds without re-expansions, while MH-GBFS does not guarantee any such bounds.

In Table 8.1, we include the results comparing WA*, EES, MPWA*, MH-GBFS with the MHA*. We used $w = 50$ for all the algorithms. Each planner was given a maximum of 60 seconds to compute a plan. The results clearly show that MHA* and MH-GBFS perform much better than WA*/MPWA*/EES, highlighting the efficacy of using multiple heuristics over a single heuristic function, which often suffers from local minima due to the robot's orientation, presence of obstacles, etc.

| | PRM | RRT-Connect | RRT*(First) | RRT*(Final) | MHA* |
|---|---|---|---|---|---|
| SR | 74% | 98% | 100% | 100% | 81% |
| RT | 2.07 | 0.18 | 5.39 | 8.48 | 1.00 |
| BD | 1.93 | 1.88 | 1.36 | 1.34 | 1.00 |
| ED | 1.87 | 1.68 | 1.27 | 1.24 | 1.00 |

TABLE 8.2: Comparison between MHA* and sampling-based planners for PR2 manipulation in kitchen environments. All the results are presented as a ratio between the algorithm marked in the column heading and the corresponding MHA* numbers. For sampling-based planners, the distances are obtained after post-processing. Since RRT* is an anytime algorithm, we include the results for the first solution reported (RRT*-First) and the solution obtained at the end of 60 secs (RRT*-Final). Legend: SR - success rate, RT - runtime ratio, BD - base distance ratio, ED - end-effector distance ratio.

MPWA* performs slightly better than WA* indicating that the size of a local minimum can depend on the weights used. However, it still gets stuck in most of the cases, since it uses the same heuristic (albeit with different weights) for each search. EES performs poorly when the inadmissible distance function has a large depression. Also, the inadmissible and admissible searches in EES do not use weighted heuristics and thus, often get trapped in some cost plateau.

MHA* (and MH-GBFS) is less prone to suffer from heuristic depression regions as they can converge in time if any of the heuristics can lead the search to the goal. MH-GBFS performs comparably to MHA* in terms of number of instances solved and slightly better in terms of convergence time. However, the solution costs obtained for MH-GBFS are significantly worse than MHA*, as noted in Solution Cost ratio in Table 8.1. This highlights the utility of the anchor search, which ensures better quality solution by intelligently controlling the inadmissible expansions.

**Comparison with Sampling-based Planners.** In Table 8.2, we include the results comparing MHA* with 3 sampling-based motion planning algorithms: PRM (Kavraki et al., 1996), RRT-Connect (Jr. and LaValle, 2000), and RRT* (Karaman and Frazzoli, 2010)). The first constructs a roadmap by randomly sampling the configuration space and then searches it for a solution, while the other two build a tree from start to goal through random sampling and extension. RRT* is also an anytime algorithm, and converges asymptotically to the optimal solution.

For the sampling-based algorithms we used the standard OMPL (Şucan, Moll, and Kavraki, 2012) implementation. Since the sampling-based planners do not directly report solution costs, in this table we include the results in terms of base and end-effector distances covered by the robots (after post-processing). All the results are presented as a ratio over the corresponding MHA* numbers for episodes where that planner and MHA* were both successful in finding a solution within 60 seconds).

The results show that MHA* performs comparably to sampling-based planners. Its runtime is better than both PRM (5x) and RRT* (8x) but worse than RRT-Connect (5x).

In terms of solution quality, MHA* results are noticeably better than all the sampling-based planners. However, both RRT-Connect and RRT* can solve more instances, mainly due to the facts that a) they are not bound by discretization choices and b) they do not use any heuristic function that may lead to local minima. Overall, the results show that MHA* is a reasonable alternative for planning in such high-dimensional state spaces, especially when we are looking for predictable and bounded suboptimal (with respect to the discretization choice) planning solutions.

## 8.4 Improved Multi-Heuristic A*

While MHA* can handle multiple inadmissible heuristics, it suffers from a *calibration problem* when the heuristics are completely unrelated to cost-to-go estimates. Since MHA* computes priorities for states in queue $i$ as $g(s) + w \cdot h_i(s)$, we could be mixing two fundamentally different quantities—$g$, which is an estimate of the cost-to-come and $h_i$, the heuristic which might have nothing to do with the cost-to-go. We term this the calibration problem since $g$ and $h$ could be operating on different units or scales. The following example better illustrates the calibration problem. Consider a robotics motion planning problem where a dual-arm personal robot needs to move from one room to another, and we want to minimize the distance traveled by the robot. Each state in the graph represents the configuration of the robot: its position in the world and the configuration of its arms. Suppose that the robot starts out with its arm spread out and that the search uses an admissible heuristic computed as the Euclidean distance from the current $(x, y)$ location to the goal $(x, y)$ location of the robot base. While the search could start out well by expanding states towards the goal it might get stuck at a doorway because its arms are spread out, thereby unable to generate successor states that pass through the doorway. A typical heuristic one might try to get around this problem is to say, for states near the door, prefer expanding those states where the arms are closer to being 'tucked-in' as they are more likely to get the robot through the doorway. Such a heuristic has no connection to the actual cost being minimized (the distance traveled) and therefore it means very little to compute quantities such as $g(s) + h(s)$.

The above example calls for a method to handle *uncalibrated heuristics* in a non-additive fashion and yet provide good quality solutions. To this end, we present a second algorithm, Improved MHA*, that is philosophically similar to MHA*, but algorithmically different. It solves the calibration problem posed by inadmissible heuristics while providing theoretical guarantees on completeness, solution quality and bounds on number of expansions.

**Algorithm 8** Improved MHA*: Requires instantiations of TERM-CRITERION$(s)$, PRIORITY$(s)$, P-CRITERION$(s)$

**Inputs:**

The start state $s_{start}$ and the goal state $s_{goal}$

Suboptimality bound factor $w$ ($\geq 1$)

One consistent heuristic $h$ and $n$ arbitrary (possibly inadmissible, uncalibrated) heuristics $h_1, h_2, \ldots, h_n$.

**Output:**

A path from $s_{start}$ to $s_{goal}$ whose cost is within $w \cdot g^*(s_{goal})$.

```
1:  procedure EXPANDSTATE(s)
2:      Remove s from OPEN
3:      for all s' ∈ SUCC(s) do
4:          if s' was not seen before then
5:              g(s') ← ∞
6:          if g(s') > g(s) + c(s, s') then
7:              g(s') ← g(s) + c(s, s')
8:              if s ∉ CLOSED_a then
9:                  Insert/Update s' in OPEN with PRIORITY(s')
10: procedure MAIN()
11:     OPEN ← ∅
12:     CLOSED_a ← ∅, CLOSED_u ← ∅
13:     g(s_start) ← 0, g(s_goal) ← ∞
14:     Insert s_start in OPEN with PRIORITY(s_start)
15:     while not TERM-CRITERION(s_goal) do
16:         if OPEN.EMPTY() then return null
17:         P-SET ← {s : s ∈ OPEN
                      ∧ s ∉ CLOSED_u
                      ∧ P-CRITERION(s)}
18:         for i = 1, ..., n do
19:             s_i ← argmin_{s∈P-SET} RANK(s, i)
20:             EXPANDSTATE(s_i)
21:             CLOSED_u ← CLOSED_u ∪ {s_i}
22:         s_a ← OPEN.TOP()
23:         EXPANDSTATE(s_a)
24:         CLOSED_a ← CLOSED_a ∪ {s_a}
25:     return solution path
```

### 8.4.1 Algorithm

Like MHA*, Improved MHA*, has access to one consistent (and hence admissible) heuristic, and several inadmissible heuristics. However, instead of using separate queues for each inadmissible heuristic, Improved MHA* uses just a single queue, and interleaves admissible and inadmissible expansions. Further, it uses the uncalibrated inadmissible heuristics in a greedy fashion, never additively combining them with the cost-to-come as in the original MHA*. To ensure that the greedy selection does not violate bounds on solution quality, Improved MHA* restricts the inadmissible heuristics to select only from "promising" states in OPEN, and permits every state to be

expanded a second time by the admissible search to maintain completeness.

We first define an "uncalibrated heuristic" before delving into the operation of the algorithm.

**Definition 2** (Uncalibrated Heuristic)**.** *An uncalibrated heuristic $h^u : S \to \mathbb{R}$ is a heuristic that induces a ranking for a set of states, i.e, state $s_i$ is ranked higher than state $s_j$ by the uncalibrated heuristic $h^u$ if $h^u(s_i) > h^u(s_j)$. Note that the uncalibrated heuristic has no relation to the cost-to-go for a state and does not have non-negativity constraints.*[2]

For most parts of the algorithm (Alg. 8), Improved MHA* resembles Weighted A* (Pohl, 1970) without re-expansions (Likhachev, Gordon, and Thrun, 2004). In fact, if we remove lines 18-21 in Alg. 8, then the algorithm is identical to weighted A* (WA*). The difference arises in the fact that we interleave WA* expansions with 'inadmissible' expansions by each of the $n$ inadmissible heuristics. As in WA*, we maintain an OPEN list sorted by some priority (such as $f(s) = g(s) + w \cdot h(s)$). Then the algorithm repeatedly performs the following until a termination condition is satisfied: a subset of the OPEN list called the potential set (abbreviated as P-SET) is constructed (line 17). As explained earlier, states in the P-SET are likely to lead towards a bounded suboptimal solution. Then, each available heuristic $h_i$ selects a state for expansion from the P-SET according to a ranking function $\textsc{Rank}(s, i)$ (line 19). For an uncalibrated heuristic, this is simply $h_i(s)$, whereas for a calibrated heuristic, the ranking function is $g(s)+w\cdot h_i(s)$. These states are expanded and marked as expanded 'inadmissibly' (line 21). Finally, the state at the top of OPEN ($s_a$) is expanded 'admissibly' and marked as closed for the *anchor search* (lines 22-24), following the same terminology as in (Aine et al., 2016). The rationale behind making an admissible expansion during every execution of the while loop (line 15) is that we might obtain a tighter (larger) lower bound on the $w$-optimal solution cost, which in turn provides more freedom for the inadmissible heuristics to 'explore' the state space.

As noted in the algorithm pseudocode, we need to provide instantiations for the OPEN list priority, the termination criterion, and the criterion for membership in the P-SET. We present three variants of Improved MHA* based on different instantiations of the said methods.

#### MHA*++

MHA*++ uses the following instantiations:

$$\textsc{Priority}(s) : g(s) + w \cdot h(s)$$
$$\textsc{Term-Criterion}(s) : g(s) \leq \max_{s \in \text{CLOSED}_a} \textsc{Priority}(s)$$
$$\textsc{P-Criterion}(s) : g(s) + h(s) \leq \max_{s \in \text{CLOSED}_a} \textsc{Priority}(s)$$

[2]More generally, the uncalibrated heuristic could be a mapping from $X \in 2^S$, the power set of states, to some $s \in X$.

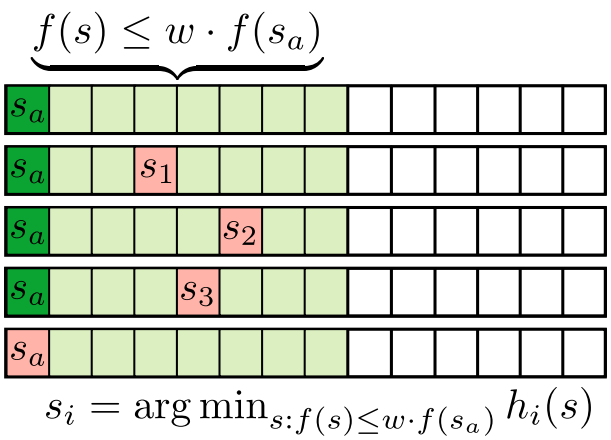



FIGURE 8.3: Illustration showing the operation of Focal-MHA*. Each row depicts the OPEN list and the states selected for expansion during an iteration over the available heuristics.

MHA*++ uses a weighted A* search as its anchor and evaluates P-SET membership by comparing the unweighted priority ($g(s)+h(s)$) of a state with the maximum weighted priority ($g(s)+w\cdot h(s)$) of any state expanded admissibly. Although typically one uses the minimum priority from OPEN to obtain bounds, we use the maximum priority from CLOSED since the anchor is a WA* search and the priorities ($f$-values) need not be monotonically non-decreasing as in regular A*. This way, we can maintain a monotonically non-decreasing lower bound on the $w$-optimal solution cost and provide maximum latitude for inadmissible expansions.

**Focal-MHA***

Focal-MHA* uses the following instantiations:

$$\text{PRIORITY}(s) : g(s) + h(s)$$
$$\text{TERM-CRITERION}(s) : g(s) \leq w \cdot \min_{s\in\text{OPEN}} \text{PRIORITY}(s)$$
$$\text{P-CRITERION}(s) : g(s) + h(s) \leq w \cdot \min_{s\in\text{OPEN}} \text{PRIORITY}(s)$$

Focal-MHA* is so named because of its direct connection with the $A^*_\epsilon$ family of algorithms and their use of the FOCAL list (Pearl and Kim, 1982). Here, the anchor search is an optimal A* search and the P-SET is simply the FOCAL list, i.e, it is a prefix of the OPEN list and contains states whose $f$-values are within $w$ of the best $f$-value in OPEN. Figure 8.3 shows the operation of Focal-MHA*. During every execution of the while loop (line 15 in Alg. 8), every heuristic selects and expands a state from the FOCAL list according to the RANK function, and finally the anchor state ($s_a$) itself is expanded.

**Unconstrained-MHA***

As suggested by the name, Unconstrained-MHA* imposes no restrictions for membership in the P-SET:

$$\text{PRIORITY}(s) : g(s) + w \cdot h(s)$$
$$\text{TERM-CRITERION}(s) : g(s) \leq \max_{s \in \text{CLOSED}_a} \text{PRIORITY}(s)$$
$$\text{P-CRITERION}(s) : \textbf{true}$$

This algorithm is similar to the multi-heuristic greedy best-first search proposed in (Röger and Helmert, 2010) in that it uses each inadmissible heuristic to run an unconstrained greedy search. However, the use of the anchor search in our case enables us to guarantee bounds on solution quality.

### 8.4.2 Theoretical Analysis

All variants of Improved MHA* have guarantees similar to MHA*: the suboptimality of the solution found is bounded by $w$ times the cost of the optimal solution, and no state is expanded more than twice (at most once by the anchor search and at most once across all inadmissible searches). In addition, MHA*++ and Focal-MHA* provide the guarantee that if the search currently does not have a $w$-optimal solution through a particular state $s$ in OPEN (i.e., the state is not 'promising'), then $s$ will not be expanded inadmissibly. This property is novel to Improved MHA* and distinguishes it from MHA*. These properties are formalized below:

**Theorem 9.** *At any point during the execution of Improved MHA* (for all its variants),* $\text{PRIORITY}(s_a) \leq w \cdot g^*(s_{goal})$*, where* $s_a = \text{argmin}_{s \in OPEN} \text{PRIORITY}(s)$.

*Proof.* (Sketch) For MHA*++ and Unconstrained-MHA*, the proof for this theorem follows in a manner similar to the proof for WA* without re-expansions (Likhachev, Gordon, and Thrun, 2004). What makes it different from WA* without re-expansions is that states can be expanded 'out-of-order' by the inadmissible heuristics, possibly violating the invariants maintained by WA*. However, by allowing any state to be re-expanded a second time by the anchor search, we can show that the anchor search can rectify the $g$-value of any state $s$ if $g(s) > w \cdot g^*(s)$. This essentially proves that the invariant maintained by WA* without re-expansions still holds for Improved IMHA*, i.e, the priority of state $s_a$ at the top of OPEN is a lower bound on $w \cdot g^*(s_a)$ and $w \cdot g^*(s_{goal})$. A rigorous proof for this theorem would be identical to the proofs provided for Shared MHA* (Aine et al., 2016).

For Focal-MHA*, the proof is identical to the above except that we have a stronger bound: $\text{PRIORITY}(s_a) \leq g^*(s_{goal})$. This follows from the fact that the anchor search is an optimal A* search. However, because $w \geq 1$, the theorem is trivially true for Focal-MHA* too. □

**Corollary 2.** *At any point during the execution of Improved MHA* (for all its variants),* $\text{PRIORITY}(s_a) \leq w \cdot g^*(s_{goal})$*, where* $s_a = \text{argmax}_{s \in CLOSED_a} \text{PRIORITY}(s)$.

*Proof.* $\text{CLOSED}_a$ contains states that have been expanded by the anchor search, i.e, those states that were at the top of OPEN at some point during the search. From Theorem 9, we know that every state in $\text{CLOSED}_a$ has a priority that lower bounds the $w$-optimal solution cost. Specifically, the maximum priority of any of those states $\max_{s \in \text{CLOSED}_a} \text{PRIORITY(S)}$ is also a lower bound on $w \cdot g^*(s_{goal})$. □

**Theorem 10** (Bounded re-expansions)**.** *No state is expanded (opened) more than twice by any variant of Improved MHA*, i.e, a state can be re-expanded (re-opened) only once.*

*Proof.* For any state $s$ to be expanded from OPEN, it must first be inserted into OPEN and this happens only when either $s \notin \text{CLOSED}_u$ or $s \notin \text{CLOSED}_a$ (lines 8 and 17). Further, every expanded state $s$ is added to either $\text{CLOSED}_u$ or $\text{CLOSED}_a$ (lines 21 and 24). Thus, it immediately follows that a state can be expanded at most twice before it is added to both $\text{CLOSED}_u$ and $\text{CLOSED}_a$. In fact, if a state is added first to $\text{CLOSED}_a$ before it is added to $\text{CLOSED}_u$, it will not be expanded a second time at all (line 8). □

**Theorem 11** (Bounded suboptimality)**.** *All variants of Improved MHA* terminate and when they do, the solution returned (if one exists) has a cost which is at most $w$ times the cost of the optimal solution. In other words, when Improved MHA* terminates, $g(s_{goal}) \leq w \cdot g^*(s_{goal})$.*

*Proof.* The search terminates either on line 16 or line 25. Termination on line 25 occurs only when TERM-CRITERION is satisfied. Using Theorem 9 and Corollary 2 with the termination criterion for each variant, we see that the search terminates only when $g(s_{goal}) \leq w \cdot g^*(s_{goal})$, thus proving the theorem.

For the case when no solution exists, OPEN will be empty (line 16) once every state in the graph has been expanded at most twice (Theorem 10) and the search terminates. □

**Theorem 12** (Efficiency)**.** *For MHA*++ and Focal-MHA*, any state $s$ with $g(s) + h(s) > w \cdot g^*(s_{goal})$ will not be expanded inadmissibly.*

*Proof.* For MHA*++, the membership criterion for a state $s$ in the P-SET requires $g(s) + h(s) \leq \max_{s \in \text{CLOSED}_a} \text{PRIORITY}(s)$. From Corollary 2, we see that states in the P-SET satisfy $g(s) + h(s) \leq w \cdot g^*(s_{goal})$. For Focal-MHA*, the anchor search is an optimal A* search which satisfies the invariant $\min_{s \in \text{OPEN}} \text{PRIORITY}(s) \leq g^*(s_{goal})$. Using this in the P-SET membership criterion, we see that all states in the P-SET satisfy $g(s) + h(s) \leq w \cdot g^*(s_{goal})$. Thus, in MHA*++ as well as Focal-MHA*, a state with $g(s) + h(s) > w \cdot g^*(s_{goal})$ cannot belong to the P-SET, and can thus never be expanded inadmissibly. □

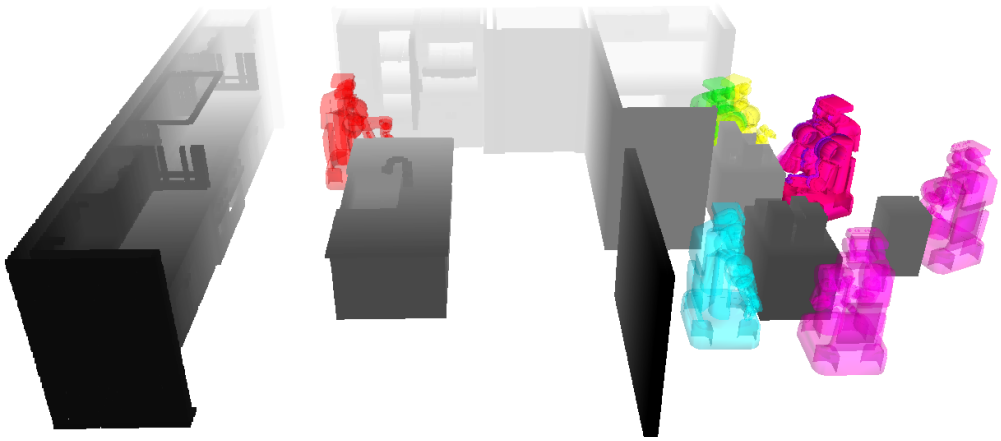


FIGURE 8.4: The kitchen domain for our experiments. Note the randomly placed tables on the right with randomized clutter on top. In the middle is a narrow door that separates the two rooms. The different colored robots show node expansions from each of the 20 different heuristics we used.

### 8.4.3 Evaluation

#### Mobile Manipulation Planning

We first evaluate the performance of the Improved MHA* on the mobile manipulation planning domain discussed earlier. We use a kitchen environment similar to that used in MHA*, with an additional room and a doorway that the robot needs to pass through. Figure 8.4 shows the environment we ran our experiments in. The domain is challenging due to the high dimensionality of the problem, cluttered tables, and narrow passages which must be crossed (the robot's base barely fits through the doorway and only if the arm is tucked in). Determining a single heuristic which can guide the base and arm toward the goal and around obstacles is challenging. It becomes especially hard when there are conflicting ideas, like wanting to extend the arm when reaching for the goal, but wanting to tuck it when going through a door. A multi-heuristic search is perfect for dealing with these heuristics with multiple (and at times conflicting) components.

**Heuristics.** We designed 20 heuristics (19+1 anchor) to help guide the search. 16 of these guide the base's $(x, y)$ position while requiring different fixed base headings and a tucked arm. These heuristics help navigation in tight spaces, but can't reach for the goal. There are then 3 other heuristics which guide the arm to the goal with or without guiding the base to a specific pose within arm's reach of the goal. One guides the base to a pose "behind" the goal (so that the gripper faces forward when the robot gets there), the second focuses on gripper orientation, while the third only tries to pull the gripper to the proper position and orientation without influencing the base position. Note that almost all of these heuristics are inadmissible and uncalibrated. They serve as ranking functions rather than estimates of cost-to-go—e.g., the tuck-arm heuristic

TABLE 8.3: Comparison of different Improved MHA* variants with the original MHA* algorithm, MH-GBFS and RRT-Connect for full-body motion planning.

| | $w = 100$ | | | | $w = 10$ | | | | $w = 5$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ++ | Focal | Uncons | Orig. | ++ | Focal | Uncons | Orig. | ++ | Focal | Uncons | Orig. | MH-GBFS | RRT-C |
| Success (%) | 84 | 75 | 84 | 61 | 83 | 74 | 83 | 0 | 66 | 74 | 60 | 0 | 85 | 45 |
| States Expanded | 2415 | 3058 | 2415 | 5179 | 3293 | 3086 | 3227 | - | 2378 | 3086 | 2472 | - | 2752 | n/a |
| Plan Time (s) | 36.89 | 47.44 | 36.98 | 75.63 | 47.60 | 47.88 | 46.96 | - | 37.92 | 48.08 | 38.31 | - | 41.98 | 21.95 |
| Base Cost (m) | 5.33 | 5.47 | 5.33 | 5.47 | 5.32 | 5.52 | 5.33 | - | 4.77 | 5.52 | 4.49 | - | 5.45 | 5.22 |
| Arm Cost (rad) | 6.32 | 6.45 | 6.32 | 6.02 | 6.17 | 6.49 | 6.17 | - | 5.70 | 6.49 | 5.41 | - | 6.60 | 8.26 |

merely prefers to expand states where the robot's arm is tucked in as opposed to other states.

**Comparison with Baselines.** We generated 100 random trials. The two tables in the kitchen are randomly positioned differently every 10 trials as is the clutter on top of them. Each of the 100 trials is created by choosing a random pose on one of the two tables for the gripper to reach and the starting configuration for the robot is randomly generated as well. A trial is deemed successful if the planner can find a $w$-optimal solution within a time limit of 5 minutes, and unsuccessful otherwise.

Table 8.3 compares the three variants, MHA*++, Focal-MHA* and Unconstrained-MHA* with the original MHA* algorithm for different suboptimality bounds $w$, as well as the multi-heuristic greedy best-first search (MH-GBFS) (Röger and Helmert, 2010) and RRT-Connect (Jr. and LaValle, 2000). To uniformly compare across all methods, the solution quality of the generated paths is measured by the distance traveled by the robot base and the arm (joint angles). The reported statistics for a method are average values across its successful trials.

Unlike the Improved MHA* variants, the original MHA* algorithm requires two suboptimality factors $w_1$ and $w_2$, for the inflation and anchor respectively. As recommended in (Aine et al., 2016), we set $w_2 = \min(2.0, \sqrt{w})$ and $w_1 = w/w_2$ for all our comparisons to get the same desired suboptimality bounds for each case. The Improved MHA* methods significantly outperform the original MHA* algorithm for lower suboptimality bounds; in fact the original MHA* algorithm fails to succeed on any trial at all. This is expected, since MHA* essentially reduces to weighted A* with a single heuristic when the inadmissible heuristics are out-of-scale (several orders of magnitude greater) with the consistent heuristic, or equivalently when the anchor suboptimality factor ($w_2$) is too small. For a large suboptimality bound ($w = 100$) however, MHA* provides a reasonable success rate as was shown in (Aine et al., 2016).

MH-GBFS performs comparably to Unconstrained-MHA* for larger values of the suboptimality bound as expected, since they both run unconstrained greedy searches. The high success rate of these approaches can be attributed to the fact that the inadmissible heuristics designed for this problem are all useful at some point or another, thereby not really requiring the 'control' provided by MHA*++ and Focal-MHA* when operating at higher suboptimality bounds. However, when we desire lower suboptimality bounds, it becomes essential to control the inadmissible searches, as can be seen from Table 8.3 for $w = 5$. Characteristic of greedy search, MH-GBFS has higher solution costs especially when compared to MHA*++ and Unconstrained-MHA*.

RRT-Connect (Jr. and LaValle, 2000) is a popular sampling-based motion planning algorithm in robotics. While it is known to quickly generate plans for high dimensional problems, it suffers from a 'narrow passage' problem. In our experiments, the doorway in the kitchen creates a narrow passage in the 11-DoF configuration space, thereby affecting RRT-Connect's success rate. Moreover, RRT-Connect does not explicitly minimize a cost and therefore the paths generated typically have high cost (Table 8.3).

**Sliding Tile Puzzles**

To demonstrate the generality of Improved MHA*, we evaluate the different variants on a sliding tile puzzle domain—a traditional test domain for heuristic search algorithms. Here, we present the experimental results for large sliding tile puzzles ($8 \times 8$, $9 \times 9$ and $10 \times 10$). For each size, we create $100$ random (solvable) puzzle instances to build our test suite. We evaluate the performance of the Improved MHA* variants and MH-GBFS over the entire test suite. In each case, we run the planner for a time limit of $5$ minutes.

**Heuristics.** For this domain, we used a set of $9$ heuristics ($8$ inadmissible + $1$ anchor). We used the Manhattan distance plus linear conflicts as the consistent heuristic (anchor). The inadmissible heuristics were computed in the following manner: for a given puzzle size, we generate a database of $1000$ different solved configurations by performing a random walk of $k$ steps from the goal state, where $k$ is a random number between $2$ and $10$ times the puzzle size. For each configuration, we store the path to goal and store $k$ as the cost to goal.

We cluster this database in 8 parts using the heuristic difference between two configurations as the distance metric. For a given instance to solve (say with configuration $s_c$), we pick one target configuration ($tc_i$) from each cluster, such that the heuristic distance between $s_c$ and $tc_i$ is minimum. Once a target configuration $tc_i$ is chosen, inadmissible heuristic $h_i$ for any state $s$ was computed by $h_i(s) = w \cdot h_0(s, tc_i) + cost(tc_i)$ (note that this heuristic includes inflation), where $w$ is the desired suboptimality bound. For the original MHA* algorithm we set $w_2 = \min(2.0, \sqrt{w})$ and $w_1 = w/w_2$ (we use $h_i(s) = w_1 \cdot h_0(s, tc_i) + cost(tc_i)$). It may be noted that unlike the full-body planning domain, the inadmissible heuristics for this domain are not really uncalibrated as these are computed using the same function as the consistent heuristic (albeit with different target configurations). Therefore in this case, we use $g + h$ ranking (we do not inflate the heuristics here, as they are already inflated) for the Improved MHA* variants (line 19 in Alg. 8), as it takes into account the impact of $g$ values.

**Comparison with Baselines.** We include the results for this domain in Table 8.4. The first thing to note is that original MHA* does not perform as poorly as in the full-body planning domain. This is expected, as the heuristics used in this domain are not really out-of-scale. However, even in this case, MHA*++ consistently performs better/equivalent to the original on most trials, and the improvement gets more pronounced with larger puzzle size and lower desired suboptimality bounds. As examples, for the $9 \times 9$ puzzle with $w = 5$, original MHA* solves $61$ instances whereas MHA*++ solves $87$, and for $10 \times 10$ with $w = 5$, original MHA* solves $21$ instances whereas MHA*++ solves $42$. This highlights the fact that even if we have heuristics that are not out-of-scale, MHA*++ can dominate original MHA* due to its improved control, ranking, and expansion policies. Considering the other variants, we observe that MHA*++ and original MHA* tend to do better than others in most cases indicating that when the heuristic is not out-of-scale, weighted best-first ranking is probably

TABLE 8.4: Comparison of different Improved MHA* variants with the original MHA* algorithm and MH-GBFS for sliding tile puzzle problems.

| Size | | $w = 50$ | | | | $w = 10$ | | | | $w = 5$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ++ | Focal | Uncons | Orig. | ++ | Focal | Uncons | Orig. | ++ | Focal | Uncons | Orig. | MH-GBFS |
| 8×8 | Success (%) | 95 | 91 | 91 | 97 | 100 | 92 | 96 | 100 | 100 | 66 | 61 | 83 | 94 |
| | Plan Time (s) | 18.16 | 38.56 | 27.74 | 20.70 | 18.23 | 28.03 | 22.65 | 18.14 | 19.56 | 39.43 | 42.95 | 26.31 | 28.45 |
| | Sol. Cost | 1617.4 | 1883.6 | 1704.0 | 1573.1 | 1552.5 | 1848.9 | 1656.4 | 1438.4 | 1415.6 | 1642.5 | 1345.2 | 1302.7 | 1889.8 |
| 9×9 | Success (%) | 94 | 69 | 85 | 93 | 97 | 61 | 86 | 77 | 87 | 46 | 28 | 61 | 71 |
| | Plan Time (s) | 44.25 | 53.52 | 60.49 | 43.15 | 46.99 | 60.31 | 53.16 | 62.16 | 58.78 | 49.18 | 66.26 | 89.33 | 55.59 |
| | Sol. Cost | 2127.6 | 2374.8 | 2350.6 | 2030.8 | 1988.6 | 2428.2 | 2196.8 | 1914.0 | 1750.2 | 2084.6 | 1560.2 | 1630.7 | 2447.0 |
| 10×10 | Success (%) | 41 | 24 | 35 | 44 | 51 | 27 | 34 | 36 | 42 | 19 | 10 | 21 | 24 |
| | Plan Time (s) | 89.84 | 81.62 | 88.50 | 88.33 | 93.62 | 90.00 | 92.78 | 98.42 | 100.12 | 88.28 | 91.00 | 114.40 | 99.22 |
| | Sol. Cost | 2523.1 | 2951.8 | 2800.3 | 2411.4 | 2449.9 | 2859.7 | 2571.5 | 2261.0 | 2251.9 | 2533.3 | 2075.4 | 1994.2 | 2663.4 |

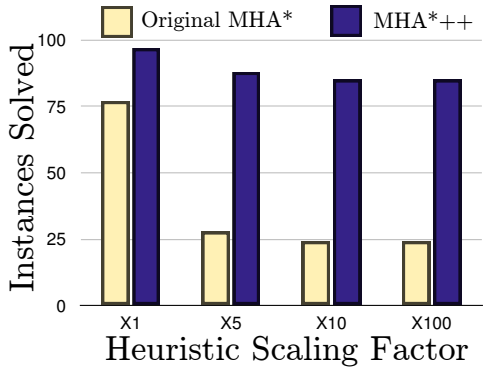



FIGURE 8.5: Comparison between the original MHA* and MHA*++ for $9 \times 9$ puzzles ($w = 10$) for different scalings of the heuristic. The $x$-axis shows the factors with which the inadmissible heuristics are multiplied (e.g., $\times 5$ denotes multiplication by $5$) and the $y$-axis shows the number of instances solved (out of 100), within a time limit of 5 minutes.

a better choice than greedy ranking. However, there are cases where the opposite is true (as seen for $9 \times 9$ with $w = 10$).

To understand the impact of out-of-scale heuristics in the puzzle domain, we conducted the following experiment: we multiplied the inadmissible heuristics by a chosen factor ($5$, $10$ and $100$) and ran original MHA* and MHA*++ for the $9 \times 9$ puzzle with $w = 10$. The results (Fig. 8.5) clearly depict the impact of out-of-scale heuristics on original MHA*; its performance degrades considerably as we make the heuristics more out-of-scale, and after a point ($10$ and above) it reduces to weighted A* (with additional overhead of multiple queue updates). In contrast, MHA*++ remains robust to heuristic scaling and outperforms MHA* by a significant margin.

In summary, we presented a framework for searching with multiple inadmissible heuristics, Multi-Heuristic A*, and an improvement to it which can handle uncalibrated heuristics. On the theoretical front, these algorithms provide guarantees on completeness, bounded suboptimality and bounded number of expansions, while on the experimental side, they achieve state-of-the-art performance on an inherently continuous domain—robot motion planning, and an inherently discrete one—sliding tile puzzles.

Next, we will return back to the domain of multi-object instance localization, and observe how MHA* can help in searching the Monotone Scene Generation Tree.

# Chapter 9

# Conclusions

## 9.1 Summary of Contributions

This thesis introduced the notion of Deliberative Perception, wherein multi-object recognition and pose estimation is formulated as a search for the best explanation of the scene. The presented algorithms combine classical AI techniques with modern learning approaches to inherit benefits of both camps: robustness and solution quality guarantees of the former, and speed and data-driven discriminative power of the latter. To recap, Perception via Search (PERCH) introduced an efficient tree search formulation for optimizing the scene explanation objective and Clutter-PERCH (C-PERCH) extended the formulation to be applicable even when 3D models for extraneous objects in the scene are not available. Then, D2P and RANSAC-Trees demonstrated how discriminative learners, specifically deep learning methods, could be used to improve global search efficiency by either employing them as heuristics, or directly baking them into the construction of the search tree. Finally, we presented two general-purpose graph search algorithms: AEE* introduced the idea of an optimal anytime algorithm in the context of time-consuming edge evaluations, and Improved Multi-Heuristic A* provided a framework for using arbitrary, inadmissible heuristics that are unrelated to the edge costs for guiding search.

## 9.2 Directions for Future Work

We strongly believe that this thesis is only the start of what could be a long and fruitful exploration of the joint Classical AI and Robot Perception frontier. As regards to future directions for Deliberative Perception, there are both well-defined avenues, and open-ended questions. Some of these are discussed next.

### 9.2.1 Exploiting Object Independences

While combinatorial search is a powerful tool for dealing with arbitrary cost functions defined over a discrete set of object instances, a naive implementation does not scale

FIGURE 9.1: An example scene from the LOV dataset with 5 objects. Notice that the scene can be easily segmented, say by connected component analysis, into 3 clusters that are independent of each other, i.e, we can guarantee that an object in one does not not occlude an object in another. Such independences can be exploited to develop an algorithm that decomposes joint object pose estimation for all 5 objects into multiple subproblems. Note that the composition of each cluster still remains unknown.

well with the number of objects. A recurring theme in the combinatorial search literature, as well as domains such as multi-robot motion planning, is the emphasis on discovering and using *independences* to decouple as many state variables as possible to reduce the joint search space. In the context of PERCH, there is a similar interpretation: if we can partition the input scene into independent groups or clusters, such that an object in one cluster does not occlude an object in another, one could run solve multiple PERCH problems of smaller sizes. That said, there is often no straightforward way to know which objects belong in which cluster. Consequently, we are faced with an algorithm design problem for factoring the knowledge of multiple clusters into account without sacrificing the properties of the vanilla PERCH solution. An added feature of this problem is the scope for parallelization: since the smaller PERCH problems deal with independent clusters, they could be solved in parallel.

### 9.2.2 Physically-based Reasoning

The use of physics-based constraints and reasoning is an under-explored direction in this thesis. Reasoning about the stability of object poses, i.e., whether a given configuration of object poses is under static equilibrium, and detailed checking of non-penetration constraints would be useful extensions of the current framework. Although this adds complexity to edge evaluations (i.e, running a physics simulator), there are potential benefits in pruning large chunks of the search space. For example, the number of 6 DoF configurations for an object where it can be in stable equilibrium would typically be much smaller than the full 6 DoF configuration space. The research questions that need to be addressed are: a) how do we incorporate physically-based simulation to produce "feasible" configurations of objects in an efficient manner? (i.e, without using rejection sampling), and b) can we ensure provably high-quality solutions despite the use of possibly imprecise simulators? We hypothesize that the answer

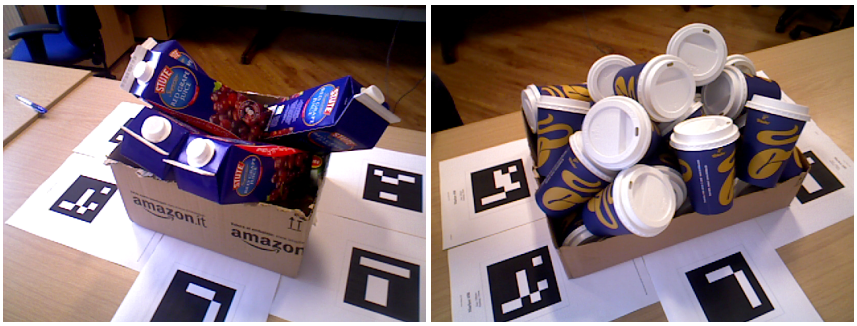



FIGURE 9.2: Examples from the dataset in (Doumanoglou et al., 2016) where consideration of physics constraints is required to reliably estimate 6 DoF poses of all objects in the scene.

to the first question lies in obtaining a realistic approximation of the "feasible"-space manifold. For example, if we assume all objects rest exclusively on a support surface, we could obtain an approximation of the contact-manifold (Koval, Pollard, and Srinivasa, 2015) which we can then use in the search procedure.

A second approach might be to use a suite of simulators with different fidelities (and consequently different computation times) to perform coarse-to-fine scene validity checking. This approach could be incorporated into the algorithm described in Sec. 7 for taking advantage of coarse edge validity priors to minimize expensive high-fidelity checking. A similar coarse-to-fine method might start off by using low-resolution 3D models for the object to reduce rendering time, and consequently use high-resolution models in a lazy fashion. Finally, physics-based reasoning can also be employed to learn better discriminative learners by generating realistic and plausible synthetic training data (Mitash, Bekris, and Boularias, 2017).

### 9.2.3 Color and Lighting

Color imagery is a rich source of information, and modern computer vision methods make use of copious amounts of such data to obtain impressive results. However, much of the work in this thesis, barring Ch. 6, has exclusively used depth data. While one part of the reason is the computational complexity of rendering photorealistic colored scenes, a second part is that the source and type of lighting is also unknown. To truly fold in color into the deliberative reasoning framework, one needs to augment the state-space with lighting parameters (position and type of lighting) and define a cost function that captures both color and depth. A naive joint search over lighting and object poses is likely to be extremely inefficient. Our conjecture is that the state-space can be compressed by carefully studying the physics of light and color interaction, with the favorable result that most states end up being "dominated" by canonical ones. To achieve truly robust perception, seeing through transparent objects, dealing with specularity etc., the ability to jointly reason about color, lighting, object pose, and material properties would be necessary—an ambitious target indeed.

### 9.2.4 Deformable and Intertwined Objects

Deliberative Perception is based on the premise that we can "generate" or render scenes corresponding to hypothesized configurations of objects. However, deformable objects with their infinitely many degrees of freedom present a hindrance. While one can still render scenes containing deformable objects by modeling those as textured spring-mass systems (Schulman et al., 2013), the number of configurations that need to be considered is intractable. While this is a completely open-ended problem, our first thoughts on a solution involves two components: a) learning a series of increasingly low-dimensional representations for deformable objects (for e.g., modeling an object with fewer and fewer control points), and b) using an adaptive-dimensional search (Gochev et al., 2011) that uses low-dimensional representations as often as it can, and moving to the full dimensional space only when necessary to satisfy solution quality bounds.

Another problem that we have sidestepped in this thesis is that of objects being contained in another, and scenes where the reverse painter's algorithm assumption fails to hold, i.e, a cycle of occlusions is present: A occludes B, B occludes C, and C occludes A. While this problem does not seem to appear as much in practice, a potential solution exists: we can dynamically detect occlusion cycles during the search and compute the edge cost as a sum of the usual positive cost, and a negative cost for occluding an existing object. Then, we could compute a conservative estimate of the largest possible negative cost (for e.g., by reasoning about the object sizes in relation to the camera) that could be encountered. Finally, we modify the termination criterion for A* or FOCAL-MHA* such that the incumbent goal's $g$-value needs to be lesser than the minimum $f$-value of OPEN by the conservative margin. Note that if the solution quality bounds are not important, the last two steps can be dropped.

### 9.2.5 Active Deliberation

While the focus of this thesis was on multi-object instance recognition for static scenes, the presented methods allow for a natural extension to the dynamic case, either when objects are moving, or when the camera is allowed to move as in the case of a mobile robot. The latter opens up possibilities for deliberative active perception, where the sequence and cost of robot motions can be integrated into the optimization formulation. Subsequently, questions of finding the best explanation of the scene, the associated uncertainty, cost and time-constraints on robot motion can be jointly explored. Yet another possible extension is with regard to localizing articulated objects such as doors or drawers. In this case, the state-space for the objects can be augmented to include the articulation parameters, such as translation along a prismatic axis or rotation angle about an axis.

## 9.3 Parting Notes

We conclude the thesis with a discussion of frequently asked questions and lessons learned.

**Why is multi-object pose estimation essential?**

Often times robots manipulate only a single object at a time or look for a particular target item as opposed to a collection of objects. However, multi-object pose estimation still remains important for two reasons: a) knowledge of another object's pose can improve recognition of the target object (a simple example is the process of recognition by elimination), and b) having a complete understanding of the scene enables sophisticated planning and decision-making: robots can determine what is the optimal order in which to move objects to access a target one, as opposed to using simple greedy heuristics. The latter is especially true in a number of real-world situations, including what we have observed from our participation in the Amazon Robotics Challenge (ARC). In the process of extracting a target item from a cluttered tote containing 20 objects (which was exactly the task in ARC 2017), knowledge of other objects in the tote and their spatial relation to the target is paramount for minimizing execution time of the robot, and consequently improving warehouse automation efficiency.

**Under what circumstances should the different methods presented in the thesis be used?**

The tools provided in this thesis span the entire deliberation-discrimination spectrum. In situations where discriminative methods are not easily available, say there is not enough labeled training data, or sufficient time for training, PERCH would be an ideal candidate as it can operate with no training at all (assuming that 3D models for objects are already available). On the other hand, PERCH has the lowest test time efficiency as it searches through a naively discretized state space with no informative guidance. When test time efficiency is critical and there *is* time available for learning a discriminative model, D2P and RANSAC-Tree are well-suited. Notice that RANSAC-Tree reduces to a purely discriminative method when the batch count and number of episodes are set to 1.

**What scenes are the most difficult for PERCH?**

The use of weighted heuristic search is a double-edged sword: while on one hand it allows for faster termination of the search, on the other hand it can inadvertently introduce deep local minima for the search. Scenes where this effect is pronounced are those that contain several objects, such as in Fig. 3.7, and in which small objects tend to be in the background and occluded by larger objects in the front. The search proceeds by placing the smaller objects first, since they tend to incur lesser cost, and then the larger objects. The use of a depth-first heuristic in combination with a large inflation factor for weighted A* reduces the chance of early back-tracking (the larger objects need to be placed first as per the monotone ordering), thereby causing the search to expand many states before discovering the correct ordering. While the use

of normalized edge costs mitigates this problem to some extent, more sophisticated heuristics for exploring correlated orderings of objects could be beneficial.

**The optimization objective: careful what you wish for.**

Throughout this thesis work, the lesson that was evident time and again is the careful selection of the optimization objective, especially when using optimal or close-to-optimal algorithms. Unlike other domains such as robot motion planning where the cost being optimized is truly representative of the robot's performance, the cost function in our work is only a proxy for object pose accuracy. Consequently, an optimal solution returned by the algorithm could be far from what we had expected to see. In the context of depth images, this work and a few others such as DART (Schmidt, Newcombe, and Fox, 2014) have independently arrived at similar ideas for the cost function. Practice has shown that both parts—the rendered explanation cost (a.k.a. model cost or positive information), and the observed explanation cost (a.k.a. negative information) are necessary for obtaining "good" solutions. Further, when combining cost functions for multiple objects, care must be taken to normalize for object size to prevent artifacts. For e.g., the algorithm could optimize the pose of a large object at the expense of poorly localizing a smaller one. Apart from the said artifact, having low variance in edge costs of the tree often results in better branch-and-bound behavior.

While "soft" costs seem attractive, attempts to incorporate more sophisticated objectives other than the simple binary explanation cost have always resulted in reverting back to the original one. My intuition is that the binary cost is intrinsically robust and insensitive to outliers, and is therefore a natural fit for noisy depth data. The use of this highly discontinuous objective is possible because of combinatorial search, which assumes very little about the cost, i.e., no differentiability or Lipschitz-continuity requirements that gradient-based methods typically require.

# Bibliography

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467*.

Aine, Sandip, Chakrabarti, P. P., and Kumar, Rajeev (2007). "AWA* - A Window Constrained Anytime Heuristic Search Algorithm". In: *IJCAI*, pp. 2250–2255.

Aine, Sandip, Swaminathan, Siddharth, Narayanan, Venkatraman, Hwang, Victor, and Likhachev, Maxim (2016). "Multi-Heuristic A*". In: *IJRR* 35.1-3, pp. 224–243.

Aldoma, Aitor, Vincze, Markus, Blodow, Nico, Gossow, David, Gedikli, Suat, Rusu, Radu Bogdan, and Bradski, Gary (2011). "CAD-model Recognition and 6DOF Pose Estimation using 3D Cues". In: *ICCV Workshops*. IEEE.

Aldoma, Aitor, Marton, Zoltan-Csaba, Tombari, Federico, Wohlkinger, Walter, Potthast, Christian, Zeisl, Bernhard, Rusu, Radu Bogdan, Gedikli, Suat, and Vincze, Markus (2012a). "Point Cloud Library". In: *IEEE Robotics & Automation Magazine* 1070.9932/12.

Aldoma, Aitor, Tombari, Federico, Rusu, Radu Bogdan, and Vincze, Markus (2012b). "OUR-CVFH–Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation". In: *DAGM*.

Aldoma, Aitor, Tombari, Federico, Di Stefano, Luigi, and Vincze, Markus (2012c). "A Global Hypotheses Verification Method for 3D Object Recognition". In: *ECCV*, pp. 511–524.

Aldoma, Aitor, Tombari, Federico, Prankl, Johann, Richtsfeld, Andreas, Di Stefano, Luigi, and Vincze, Markus (2013). "Multimodal Cue Integration through Hypotheses Verification for RGB-D Object Recognition and 6DOF Pose Estimation". In: *ICRA*. IEEE, pp. 2104–2111.

Bohlin, Robert and Kavraki, Lydia E (2000). "Path planning using lazy PRM". In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. Vol. 1. IEEE, pp. 521–528.

Butzke, Jonathan, Gochev, Kalin, Holden, Benjamin, Jung, Eui-Jung, and Likhachev, Maxim (2016). "Planning for a ground-air robotic system with collaborative localization". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 284–291.

Calli, Berk, Walsman, Aaron, Singh, Arjun, Srinivasa, Siddhartha, Abbeel, Pieter, and Dollar, Aaron M (2015). "Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set". In: *IEEE Robotics & Automation Magazine* 22.3, pp. 36–52.

Chakrabarti, P. P., Ghose, Sujoy, and Sarkar, S. C. De (1992). "Generalized best first search using single and multiple heuristics". In: *Inf. Sci.* 60.1-2, pp. 145–175.

Chakrabarti, P. P., Ghose, Sujoy, Pandey, A., and Sarkar, S. C. De (1989). "Increasing Search Efficiency Using Multiple Heuristics". In: *Inf. Process. Lett.* 30.1, pp. 33–36.

Chen, Yung and Medioni, Gérard (1991). "Object Modeling by Registration of Multiple Range Images". In: *ICRA*.

Choudhury, Shushman, Dellin, Christopher M, and Srinivasa, Siddhartha S (2016). "Pareto-Optimal Search over Configuration Space Beliefs for Anytime Motion Planning". In: *IROS*.

Cohen, Benjamin, Phillips, Mike, and Likhachev, Maxim (2014). "Planning Single-arm Manipulations with N-Arm Robots". In: *Robotics: Science and Systems*.

Cohen, Benjamin J, Chitta, Sachin, and Likhachev, Maxim (2010). "Search-based planning for manipulation with motion primitives". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, pp. 2902–2908.

Collet, Alvaro, Martinez, Manuel, and Srinivasa, Siddhartha S (2011). "The MOPED framework: Object recognition and pose estimation for manipulation". In: *The International Journal of Robotics Research*, p. 0278364911401765.

Correll, Nikolaus, Bekris, Kostas E., Berenson, Dmitry, Brock, Oliver, Causo, Albert, Hauser, Kris, Okada, Kei, Rodriguez, Alberto, Romano, Joseph M., and Wurman, Peter R. (2016). "Lessons from the Amazon Picking Challenge". In: *arXiv preprint arXiv:1601.05484*.

Dellin, Christopher M and Srinivasa, Siddhartha S (2016). "A Unifying Formalism for Shortest Path Problems with Expensive Edge Evaluations via Lazy Best-First Search over Paths with Edge Selectors". In: *ICAPS*.

Doumanoglou, Andreas, Kouskouridas, Rigas, Malassiotis, Sotiris, and Kim, Tae-Kyun (2016). "Recovering 6D Object Pose and Predicting Next-Best-View in the Crowd". In: *CVPR*.

Drost, Bertram, Ulrich, Markus, Navab, Nassir, and Ilic, Slobodan (2010). "Model Globally, Match Locally: Efficient and Robust 3D Object Recognition". In: *CVPR*. IEEE, pp. 998–1005.

Eitel, Andreas, Springenberg, Jost Tobias, Spinello, Luciano, Riedmiller, Martin, and Burgard, Wolfram (2015). "Multimodal Deep Learning for Robust RGB-D Object Recognition". In: *IROS*.

Eyerich, Patrick, Keller, Thomas, and Helmert, Malte (2010). "High-quality Policies for the Canadian Traveler's Problem". In: *Third Annual Symposium on Combinatorial Search (SoCS)*.

Fan, Gaojian, Müller, Martin, and Holte, Robert (2017). "The two-edged nature of diverse action costs". In: *Proceedings of the 27th International Conference on Automated Planning and Scheduling*.

Felner, A., Korf, R. E., and Hanan, S. (2004). "Additive Pattern Database Heuristics". In: *J. Artif. Intell. Res. (JAIR)* 22, pp. 279–318.

Felner, Ariel, Goldenberg, Meir, Sharon, Guni, Stern, Roni, Beja, Tal, Sturtevant, Nathan R, Schaeffer, Jonathan, and Holte, Robert (2012). "Partial-Expansion A* with Selective Node Generation." In: *AAAI*.

Figueiredo, Rui Pimentel de, Moreno, Pablo, Bernardino, Alexandre, and Santos-Victor, José (2013). "Multi-Object Detection and Pose Estimation in 3D Point Clouds: A Fast Grid-based Bayesian Filter". In: *ICRA*. IEEE, pp. 4250–4255.

Fischler, Martin A and Bolles, Robert C (1981). "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6, pp. 381–395.

Fouhey, David F, Collet, Alvaro, Hebert, Martial, and Srinivasa, Siddhartha (2012). "Object recognition robust to imperfect depth data". In: *European Conference on Computer Vision*. Springer, pp. 83–92.

Glover, Jack and Popovic, Sanja (2013). "Bingham Procrustean Alignment for Object Detection in Clutter". In: *IROS*. IEEE, pp. 2158–2165.

Goad, Chris (1987). "Special Purpose Automatic Programming for 3D Model-based Vision". In: *Readings in Computer Vision*, pp. 371–381.

Gochev, Kalin, Cohen, Benjamin, Butzke, Jonathan, Safonova, Alla, and Likhachev, Maxim (2011). "Path planning with adaptive dimensionality". In: *Fourth annual symposium on combinatorial search*.

Grimson, W Eric L and Lozano-Perez, Tomas (1987). "Localizing Overlapping Parts by Searching the Interpretation Tree". In: *PAMI* 4, pp. 469–482.

Hager, Gregory D and Wegbreit, Ben (2011). "Scene parsing using a prior world model". In: *The International Journal of Robotics Research* 30.12, pp. 1477–1507.

Hansen, Eric A and Zhou, Rong (2007). "Anytime heuristic search". In: *Journal of Artificial Intelligence Research* 28, pp. 267–297.

Hansen, Eric A and Zilberstein, Shlomo (2001). "LAO*: A heuristic search algorithm that finds solutions with loops". In: vol. 129. 1. Elsevier, pp. 35–62.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Hatem, Matthew and Ruml, Wheeler (2014). "Simpler bounded suboptimal search". In: *Proceedings of Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press.

Helmert, Malte, Röger, Gabriele, and Karpas, Erez (2011). "Fast Downward Stone Soup: A Baseline for Building Planner Portfolios". In: *ICAPS 2011 Workshop on Planning and Learning*, pp. 28–35.

Herbst, Evan, Henry, Peter, Ren, Xiaofeng, and Fox, Dieter (2011). "Toward object discovery and modeling via 3-d scene comparison". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, pp. 2623–2629.

Hernández, Carlos and Baier, Jorge A (2012). "Avoiding and escaping depressions in real-time heuristic search". In: *Journal of Artificial Intelligence Research* 43, pp. 523–570.

Hinterstoisser, Stefan, Lepetit, Vincent, Ilic, Slobodan, Holzer, Stefan, Bradski, Gary, Konolige, Kurt, and Navab, Nassir (2013). "Model Based Training, Detection and Pose Estimation of Texture-less 3D Objects in Heavily Cluttered Scenes". In: *ACCV*, pp. 548–562.

Hodaň, Tomáš, Matas, Jiří, and Obdržálek, Štěpán (2016). "On evaluation of 6D object pose estimation". In: *Computer Vision–ECCV 2016 Workshops*. Springer, pp. 606–619.

Hodan, Tomáš, Haluza, Pavel, Obdržálek, Štepán, Matas, Jiri, Lourakis, Manolis, and Zabulis, Xenophon (2017). "T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects". In: *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, pp. 880–888.

Hsiao, Edward and Hebert, Martial (2014). "Occlusion reasoning for object detectionunder arbitrary viewpoint". In: *IEEE transactions on pattern analysis and machine intelligence* 36.9, pp. 1803–1815.

Huh, Jinwook and Lee, Daniel D (2016). "Learning high-dimensional Mixture Models for fast collision detection in Rapidly-Exploring Random Trees". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 63–69.

Isto, Pekka (1996). "Path Planning By Multiheuristic Search Via Subgoals". In: *Proceedings of the 27th International Symposium on Industrial Robots, CEU*, pp. 71272–6.

Johnson, Andrew E and Hebert, Martial (1999). "Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes". In: *PAMI* 21.5, pp. 433–449.

Jr., James J. Kuffner and LaValle, Steven M. (2000). "RRT-Connect: An Efficient Approach to Single-Query Path Planning". In: *ICRA*. IEEE, pp. 995–1001. ISBN: 0-7803-5889-9.

Karaman, S. and Frazzoli, E. (2010). "Incremental Sampling-based Algorithms for Optimal Motion Planning". In: *Robotics: Science and Systems*. Zaragoza, Spain: The MIT Press.

Kavraki, Lydia E., Svestka, Petr, Latombe, Jean-Claude, and Overmars, Mark H. (1996). "Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces". In: *IEEE T. Robotics and Automation* 12.4, pp. 566–580.

Kiesel, Scott and Ruml, Wheeler (2016). "A Bayesian Effort Bias for Sampling-based Motion Planning". In: *Planning and Robotics Workshop (ICAPS-PlanRob16)*.

Kocsis, Levente and Szepesvári, Csaba (2006). "Bandit based monte-carlo planning". In: *ECML*. Vol. 6. Springer, pp. 282–293.

Koenig, Sven, Likhachev, Maxim, and Furcy, David (2004). "Lifelong planning A*". In: *Artificial Intelligence* 155.1-2, pp. 93–146.

Korf, R. E. and Felner, A. (2002). "Disjoint pattern database heuristics". In: *Artif. Intell.* 134.1-2, pp. 9–22.

Koval, Michael C, Pollard, Nancy S, and Srinivasa, Siddhartha S (2015). "Pose estimation for planar contact manipulation with manifold particle filters". In: *The International Journal of Robotics Research* 34.7, pp. 922–945.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E (2012). "Imagenet Classification with Deep Convolutional Neural Networks". In: *NIPS*, pp. 1097–1105.

Krull, Alexander, Brachmann, Eric, Michel, Frank, Ying Yang, Michael, Gumhold, Stefan, and Rother, Carsten (2015). "Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images". In: *ICCV*.

Lai, Kevin Kar Wai (2014). "Object Recognition and Semantic Scene Labeling for RGB-D Data". PhD thesis.

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press.

Lavalle, Steven M., Kuffner, James J., and Jr. (2000). "Rapidly-Exploring Random Trees: Progress and Prospects". In: *Algorithmic and Computational Robotics: New Directions*, pp. 293–308.

Likhachev, M. and Ferguson, D. (2008). "Planning Long Dynamically-Feasible Maneuvers For Autonomous Vehicles". In: *Proceedings of Robotics: Science and Systems (RSS)*. Cambridge, USA.

Likhachev, M., Gordon, G. J., and Thrun, S. (2004). "ARA*: Anytime A* with Provable Bounds on Sub-Optimality". In: *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Lowe, David G (1987). "The Viewpoint Consistency Constraint". In: *IJCV* 1.1, pp. 57–72.

MacAllister, Brian, Butzke, Jonathan, Kushleyev, Alex, Pandey, Harsh, and Likhachev, Maxim (2013). "Path planning for non-circular micro aerial vehicles in constrained environments". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, pp. 3933–3940.

Marton, Zoltan-Csaba, Pangercic, Dejan, Blodow, Nico, and Beetz, Michael (2011). "Combined 2D–3D Categorization and Classification for Multimodal Perception Systems". In: *IJRR*.

Meger, David, Wojek, Christian, Little, James J, and Schiele, Bernt (2011). "Explicit Occlusion Reasoning for 3D Object Detection." In: *BMVC*, pp. 1–11.

Mitash, Chaitanya, Bekris, Kostas E, and Boularias, Abdeslam (2017). "A Self-supervised Learning System for Object Detection using Physics Simulation and Multi-view Pose Estimation". In: *IROS*.

Narayanan, Venkatraman, Aine, Sandip, and Likhachev, Maxim (2015). "Improved Multi-Heuristic A* for Searching with Uncalibrated Heuristics". In: *Eighth Annual Symposium on Combinatorial Search (SoCS)*.

Narayanan, Venkatraman and Likhachev, Maxim (2016a). "PERCH: Perception via Search for Multi-Object Recognition and Localization". In: *ICRA*. IEEE.

Narayanan, Venkatraman and Likhachev, Maxim (2016b). "Discriminatively-guided Deliberative Perception for Pose Estimation of Multiple 3D Object Instances". In: *Robotics: Science and Systems*.

Narayanan, Venkatraman and Likhachev, Maxim (2017a). "Deliberative Object Pose Estimation in Clutter". In: *ICRA*.

Narayanan, Venkatraman and Likhachev, Maxim (2017b). "Heuristic Search on Graphs with Existence Priors for Expensive-to-Evaluate Edges". In: *International Conference on Automated Planning and Scheduling (ICAPS)*.

Narayanan, Venkatraman, Phillips, Mike, and Likhachev, Maxim (2012). "Anytime safe interval path planning for dynamic environments". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 4708–4715.

Narayanan, Venkatraman, Vernaza, Paul, Likhachev, Maxim, and LaValle, Steven M (2013). "Planning under topological constraints using beam-graphs". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, pp. 431–437.

Papadimitriou, Christos H and Yannakakis, Mihalis (1991). "Shortest Paths without a Map". In: *Theoretical Computer Science* 84.1, pp. 127–150.

Papazov, Chavdar and Burschka, Darius (2010). "An efficient ransac for 3d object recognition in noisy and occluded scenes". In: *Asian Conference on Computer Vision*. Springer, pp. 135–148.

Pearl, Judea and Kim, Jin H (1982). "Studies in semi-admissible heuristics". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 4, pp. 392–399.

Phillips, Mike and Likhachev, Maxim (2011). "Sipp: Safe interval path planning for dynamic environments". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, pp. 5628–5635.

Phillips, Mike, Narayanan, Venkatraman, Aine, Sandip, and Likhachev, Maxim (2015). "Efficient Search with an Ensemble of Heuristics". In: *IJCAI*, pp. 784–791.

Pohl, I. (1970). "First Results on the Effect of Error in Heuristic Search". In: *Machine Intelligence* 5. Ed. by B. Meltzer and D. Michie, pp. 219–236.

Ren, Shaoqing, He, Kaiming, Girshick, Ross, and Sun, Jian (2015). "Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks". In: *NIPS*, pp. 91–99.

Rennie, Colin, Shome, Rahul, Bekris, Kostas E, and De Souza, Alberto F (2016). "A dataset for improved rgbd-based object detection and pose estimation for warehouse pick-and-place". In: *IEEE Robotics and Automation Letters* 1.2, pp. 1179–1185.

Richter, Silvia, Thayer, Jordan Tyler, and Ruml, Wheeler (2010). "The Joy of Forgetting: Faster Anytime Search via Restarting". In: *ICAPS*, pp. 137–144.

Röger, Gabriele and Helmert, Malte (2010). "The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning". In: *ICAPS'10*, pp. 246–249.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li (2015). "" In: *IJCV* 115.3, pp. 211–252.

Rusu, Radu Bogdan, Blodow, Nico, and Beetz, Michael (2009). "Fast Point Feature Histograms (FPFH) for 3D Registration". In: *ICRA*. IEEE.

Rusu, Radu Bogdan, Bradski, Gary, Thibaux, Romain, and Hsu, John (2010). "Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram". In: *IROS*. IEEE.

Schmidt, Tanner, Newcombe, Richard A, and Fox, Dieter (2014). "DART: Dense Articulated Real-Time Tracking." In: *Robotics: Science and Systems*.

Schulman, John, Lee, Alex, Ho, Jonathan, and Abbeel, Pieter (2013). "Tracking deformable objects with point clouds". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, pp. 1130–1137.

Schwarz, Max, Schulz, Hannes, and Behnke, Sven (2015). "RGB-D Object Recognition and Pose Estimation Based on Pre-trained Convolutional Neural Network Features". In: *ICRA*. IEEE.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Stevens, Mark R and Beveridge, J Ross (2000a). "Localized Scene Interpretation from 3D Models, Range, and Optical Data". In: *Computer Vision and Image Understanding*.

Stevens, Mark R and Beveridge, J Ross (2000b). "Integrating Graphics and Vision for Object Recognition". In: vol. 589. Springer Science & Business Media.

Şucan, Ioan A., Moll, Mark, and Kavraki, Lydia E. (2012). "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4, pp. 72–82.

Tejani, Alykhan, Tang, Danhang, Kouskouridas, Rigas, and Kim, Tae-Kyun (2014). "Latent-class Hough Forests for 3D Object Detection and Pose Estimation". In: *ECCV*, pp. 462–477.

Thayer, Jordan Tyler, Benton, J, and Helmert, Malte (2012). "Better Parameter-Free Anytime Search by Minimizing Time Between Solutions". In: *Fifth Annual Symposium on Combinatorial Search (SoCS)*, pp. 120–128.

Thayer, Jordan Tyler and Ruml, Wheeler (2011). "Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates". In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 674–679.

Thayer, Jordan Tyler, Stern, Roni, Felner, Ariel, and Ruml, Wheeler (2012). "Faster Bounded-Cost Search Using Inadmissible Estimates". In: *ICAPS*.

Valenzano, Richard Anthony, Sturtevant, Nathan R., Schaeffer, Jonathan, Buro, Karen, and Kishimoto, Akihiro (2010). "Simultaneously Searching with Multiple Settings: An Alternative to Parameter Tuning for Suboptimal Single-Agent Search Algorithms". In: *ICAPS*, pp. 177–184.

Valiant, Leslie G (1984). "A theory of the learnable". In: *Communications of the ACM* 27.11, pp. 1134–1142.

Vernaza, Paul, Narayanan, Venkatraman, and Likhachev, Maxim (2012). "Efficiently finding optimal winding-constrained loops in the plane". In: *Robotics: Science and Systems*.

Wilt, Christopher Makoto and Ruml, Wheeler (2011). "Cost-based heuristic search is sensitive to the ratio of operator costs". In: *Fourth Annual Symposium on Combinatorial Search*.

Wohlkinger, Walter and Vincze, Markus (2011). "Ensemble of Shape Functions for 3D Object Classification". In: *ROBIO*. IEEE, pp. 2987–2992.

Wu, Zhirong, Song, Shuran, Khosla, Aditya, Yu, Fisher, Zhang, Linguang, Tang, Xiaoou, and Xiao, Jianxiong (2015). "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: *CVPR*, pp. 1912–1920.

Xiang, Yu and Fox, Dieter (2017). "DA-RNN: Semantic Mapping with Data Associated Recurrent Neural Networks". In: *Proceedings of Robotics: Science and Systems*. Cambridge, Massachusetts. DOI: 10.15607/RSS.2017.XIII.013.

Xiang, Yu and Savarese, Silvio (2013). "Object detection by 3D aspectlets and occlusion reasoning". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 530–537.

Yoshizumi, Takayuki, Miura, Teruhisa, and Ishida, Toru (2000). "A* with Partial Expansion for Large Branching Factor Problems." In: *AAAI/IAAI*, pp. 923–929.

Zeng, Andy, Yu, Kuan-Ting, Song, Shuran, Suo, Daniel, Walker Jr, Ed, Rodriguez, Alberto, and Xiao, Jianxiong (2017). "Multi-view Self-supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge". In: *ICRA*.