Inference and Task Planning over Spatially Complex Problems

by

Alex Cuellar

B.S. Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Electrical Engineering and Computer Science
August 31, 2022

Certified by .....
Julie Shah
H.N. Slater Professor, Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Inference and Task Planning over Spatially Complex Problems

by

Alex Cuellar

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

One core problem of robot viability in many sectors is retrainability; if a robot's task can change without changing code, automation becomes feasible for a wider set of applications. To advance robot retrainability, this thesis will introduce a learning from demonstrations (LfD) framework allowing a robot to learn and execute task-level plans in spatially complex environments. To achieve this goal, we introduce a propositional logic framework to encode spatial relationships between objects and an inference scheme to identify important relationships between defined object classes. Finally, we present a search-based algorithm to synthesize required class relationships into a task-level plan. As a representative problem for this of context, we focus on the problem of box packing, wherein the robot must learn specific rules surrounding how to place objects in a box according to a demonstrator's wishes.

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

Thesis Supervisor: Julie Shah
Title: H.N. Slater Professor, Aeronautics and Astronautics

Acknowledgments

This thesis would not have been possible without all my wonderful collaborators at the MIT Interactive Robotics Group and Lincoln Labs. I'd like to thank Julie Shah for her endless enthusiasm, guidance through this project, and for building a supportive lab community. Thanks to Hosea Siu for helping me turn my incoherent ramblings alchemy-like into workable ideas, teaching me so much about the research process, and providing insight into what Next House was like back in the old days. Also, thanks for the mug. Lastly, Thanks to Chris Fourie. Your support and advising helped me make the decision to apply to grad school, and led me to IRG. Everything here traces back to you.

To all my friends who offered support, encouragement, and pulled me away from work whenever I needed a break: this would have been a far worse process without you. Meghana, you are perhaps the best at providing such necessary distractions – thank you for making this scary first step in grad school life not only bearable, but a joy. And thanks to Rafael Fierro. Before you took me in, robots were just this cool thing I saw on TV. You showed me what working with robots is really like.

And of course I'd like to thank my family. To my Dad for helping me keep my eye on the prize, to my Mom for her endless encouragement and support, and to my brother for reminding me that sometimes an Akira Kurosawa marathon is the most important thing right now. It goes without saying that none of this would be possible without you three. It means so much that all of you are always in my corner no matter what.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past several years, the cost of purchasing robots has decreased dramatically, leading to wider accessibility. Given this advancement, one would expect a corresponding drive to utilize such technology in industry settings. However, recent studies show a relatively unchanging landscape of robotic usage: large manufactures with many resources (automotive manufactures for example) tend to utilize industrial robotics for repetitive tasks with high-volume output, while SMEs (Small to Medium Enterprises) lag behind [44]. In large part, the barrier between SMEs and robotic viability is robots' inflexibility with respect to changing task parameters. Unlike large companies, SMEs require "high-mix low-volume" output, meaning any specific task may be required a relatively small number of times before the task is changed. If each change in task parameters requires changes to code, using a robot becomes less economically viable.

One proposed tool to make robots viable for SMEs is Learning from Demonstration (LfD) techniques [25]. Here, the robot is able to infer the necessary steps or generalized policy for a task by watching a demonstration, thereby decreasing the cost of switching tasks dramatically in low-volume high-mix scenarios.

Naturally, many tasks in manufacturing require a robot to understand how parts relate to each other in space (e.g. in assembly tasks or part storage). However, to our knowledge, past research has not described a method to infer and replicate tasks based on the spatial relationships between objects in an environment. Therefore, in

this thesis, we introduce a set of methods intended to infer and replicate patterns in the spatial positions of objects in an environment via demonstrations.

This thesis is structured as follows: In Chapter 2, we provide background on relevant literature in order to ground and motivate the methods we present. Chapter 3 presents our formal problem formulation as well as the inference and planning algorithms that make up our technical contribution. Chapter 4 describes the evaluations of our methods, including a human pilot study and runtime analysis. Finally, chapter 5 gives a conclusion and describes potential future work.

1.1 Algorithms and Humans in Operational Settings

Rapid reprogramming of robotics for high-mix low-volume output is necessary for moving many industries and SMEs towards a more efficient and automated future. However, the problem does not start and end with task completion in a technically correct manner. Ultimately, these algorithms live and die by companies and workers choosing to utilize such technology – a choice based on far more than correct task completion. Many studies have shown worker aversion to using algorithms for decision making in contexts spanning box packing [49] to forecasting inventory needs [15]. Many have theorized that this so-called "algorithm aversion" stems from humans' belief that an algortihm will eventually make errors [18] and workers' lower tolerance for algorithmic errors than human errors [14]. Yet, people are more likely to use an algorithm if they have a say in it's functioning, resulting both from an increased trust of the agent and the user's ability to confer useful information known by the user but not the robot [15]. Therefore, in addition to rapid retraining, a learning from demonstrations approach allows us to capture the patterns and constraints the human prefers.

For example, imagine the task of placing objects into a box. There are many existing algorithms that can determine where to place objects to pack a box. However, there may be hidden constraints the human uses when packing a box. Some of these may help guide the algorithm to genuinely useful behaviors (e.g. placing boxes next

to each other can maintain maximal open space). Others may be more cosmetic, like packing alike items next to each other. While not strictly necessary to find a tractable set of object placements, these human-preference constraints can help the algorithm complete tasks that seems appropriate to the demonstrator.

Among techniques that allow humans to give input on an algorithm's functioning, learning from demonstration approaches provide an advantage over systems that only take linguistic explanations from the human. Past research has documented humans' tendency to underspecify a task (i.e. leave out important details for preferences and requirements) if their descriptions rely solely on language [22]. Learning from demonstration approaches avoid this issue by letting the system understand task requirements directly without the need of explicit, language-based specification.

1.2 Contributions

1.2.1 Spatial Logic Framework

In order to reason over the spatial relationships between objects, we first need a framework to describe how objects are related in space. For this purpose, we utilize a fragment of Region Connection Calculus (RCC) – a spatial language that formalizes a set of human-intuitive canonical relationships between regions of space. In Section 3.1, we describe and formalize an extension to this language that allows descriptions of objects' relative position in a coordinate system (i.e. if one object is North/South/East/West of another). We then use this extended fragment of RCC to formally introduce language around spatial constraints and how objects can satisfy such constraints.

1.2.2 Inference over Spatial Relationships

Given the Boolean logic framework to reason and describe constraints over objects in space, we then introduce a Bayesian inference algorithm to learn constraints over objects in space given a set of demonstrations in Section 3.2. This inference occurs

in two steps. First, it determines a set of "candidate" constraints which are satisfied in every provided demonstration. Then, a Bayesian algorithm determines whether each candidate constraint is likely to have been in the demonstrator's ground truth constraint set, or whether the demonstrations randomly satisfied the constraint. The algorithm outputs a set of constraints over the spatial relationships between objects with a high probability of being within the demonstrator's ground truth constraint set.

1.2.3 Object Placement Synthesis

Once a set of spatial constraints is inferred, we introduce an algorithm in Section 3.3 that can place objects in such a way as to satisfy the inferred constraints. This algorithm has two hierarchical components. The higher level system uses Monte Carlo Tree Search (MCTS) to search promising places an object can be placed in attempts to satisfy all constraints. Whenever the search tree needs a new promising object placement, it queries the lower level system, which takes in an existing set of object placements and determines the best position to place a new object.

1.2.4 Human Pilot Study

The three technical contributions of this thesis (spatial logic framework, the inference algorithm, and the synthesis algorithm) are intended to capture and recreate patterns of object placement from a human demonstrator. However, in order to prove its ability to capture human-intuitive constraints, we describe and carry out a pilot study to show the system's effectiveness in placing objects that a human finds satisfactorily reproduces a series of demonstrations. Additionally, we show that that human participants tend to underspecify the requirements inherent to their demonstrations.

Chapter 2

Background

2.1 Learning From Demonstrations

In recent years, the AI community has seen huge leaps in planning for complex domains like Atari games [35], Go [48], and Starcraft [50]. While these algorithms work extremely well in situations where the reward function of a task is clear (e.g. win the game), rapid robot retraining necessitates the robot understanding the definition of a properly completed task. Learning from Demonstration techniques ultimately allow a robot the ability to interpret a humans' reward function. Many of these techniques focus on imitating a demonstration, usually with respect to motion-level planning. While the specific goals behind these methods vary, imitation-based techniques generally attempt to minimize a distance metric between observed and executed trajectories. Despite being a powerful tool in motion, the goal of minimizing distance inherently limits imitation learning methods to situations where a robot needs to clone one specific task.

Such limitations inherent to imitation methods lead to the necessity of techniques that attempt to detect the attributes of desirable behavior separate from mimicry. Many methods rely on machine learning to construct high-dimensional surfaces in the feature space. However, purely deep learning methods lack the ability to provide insight or verifiable guarantees concerning the reward behavior learned. Additionally, they require large data sets, which can be costly or impossible to obtain in environ-

ments like high-mix low-volume manufacturing where tasks change quickly [2]. As an alternative, recent research has looked for ways to incorporate specifications via logic languages. These systems utilize parameters with explicit meaning to them and tend to require far fewer demonstrations than deep learning methods. In this thesis, we extend the existing literature on inferring and synthesizing behaviors via logical specifications into languages designed to encode human-intuitive object relations.

2.2 Formal Logic for Specification

Many formal logical languages have been described to capture information relevant to a system's behavior. All logical languages use the standard Boolean operators such as *or*, *and*, and *not* as well as language-specific operators to create a logical formula, or "specification" describing the desired behavior of a system. Research in inferring system specifications frequently utilize temporal logic systems, such as Linear Temporal Logic (LTL) [38]. LTL specializes in reasoning over a set of binary state variables with temporal operators including *before*, *eventually*, or *globally*. LTL naturally gives way to specifications over task-level plans for robots. For example, if one wants to describe making cookies, LTL can describe making dough and preheating the oven as prerequisites to putting the cookies in the oven. Shah et al introduced methods to both infer LTL specifications from demonstrations using a Bayesian framework [46] and generate task-level plans using a set of specification candidates [47]. Others have incorporated LTL into motion planning and control schemes to specify the ordering of locations or regions a robot needs to visit [27, 52]. While these methods bring LTL into contact with spatial planning, the spatial world is not described directly by the specifications; the LTL simply encodes the robot's existence within a pre-defined region as a binary variable.

Other languages more directly reason over continuous spaces. Signal Temporal Logic (STL) uses similar temporal operators to LTL, but reasons over continuous as opposed to binary variables [31]. This reasoning over continuous variables naturally means STL specifications can reason directly over positions in space. Past research

has utilized this fact to infer specifications defining desirable behavior over learned regions [39] and control for robot handovers [28]. Alsalehi et al extended STL to a system capable of determining signal correctness based on a Support Vector Machine (SVM) classification instead of standard linear $>$ and $<$ operators [4]. By combining this new STL-SVM language with convolutional neural networks, Alsalehi et al are able to reason over sequences of images in specification. Other languages focus more specifically on spatial reasoning to describe a system. For example, SpaTeL utilizes quad-trees to specify activity within areas of a region, such as output within a power-grid [23]. However, none of these languages focus on descriptions pertaining to human-intuitive relationships between regions.

To fill this gap in logic languages, researchers developed Qualitative Reasoning (QR), an AI subfield focused on making quantitative sense of the commonsense way humans see the world [53]. For several years, QR focused on describing scalar quantities including height, temperature, etc. Inspired by progress in QR, Randell et al expanded this notion to spatial reasoning with Region Connection Calculus (RCC), focusing on formalizing qualitative relations between objects [42]. With this QR perspective, RCC gives us the kind of human intuitive spatial language not otherwise present in prior the logic systems. While Randell described and formalized a set of canonical spatial relationships in RCC, no research to our knowledge has developed techniques to infer specifications using RCC (or other spatial languages based in QR) given demonstrations. In this thesis, we present a framework to infer specifications over a fragment of RCC most applicable to spatial relationships between objects.

2.2.1 Region Connection Calculus

The fundamental relation of RCC is $C(x, y)$, read 'x connects with y'. There exists two distinct but similar definitions of C [13, 42]. Though the choice of formalism is unimportant to the ideas presented in this thesis, we will say that $C(x, y)$ implies that the topological closure of regions x and y share at least one point. The two fundamental axioms for C are as follows:

RCC Relations		
Relation	Interpretation	Definition
$DC(x,y)$	x is disconnected from y	$\neg C(x,y)$
$P(x,y)$	x is a part of y	$\forall z[C(z,x) \rightarrow C(z,y)]$
$PP(x,y)$	x is a proper part of y	$P(x,y) \wedge \neg P(y,x)$
$EQ(x,y)$	x is identical with y	$P(x,y) \wedge P(y,x)$
$O(x,y)$	x overlaps y	$\exists z[P(z,x) \wedge P(z,y)]$
$DR(x,y)$	x is discrete from y	$\neg O(x,y)$
$PO(x,y)$	x partially overlaps y	$O(x,y) \wedge \neg P(x,y) \wedge \neg P(y,x)$
$EC(x,y)$	x is externally connected to y	$C(x,y) \wedge \neg O(x,y)$
$TPP(x,y)$	x is a tangential proper part of y	$PP(x,y) \wedge \exists z[EC(z,x) \wedge EC(z,y)]$
$NTPP(x,y)$	x is a non-tangential proper part of y	$PP(x,y) \wedge \neg\exists z[EC(z,x) \wedge EC(z,y)]$

Table 2.1: Description of relations included in Region Connection Calculus

$$\forall x[C(x,x)] \tag{2.1}$$

$$\forall x \forall y[C(x,y) \implies C(y,x)] \tag{2.2}$$

This fundamental relation is used to express the 10 canonical relations within RCC described in Table 2.1. From these relations, various fragments have been proposed with various properties, such a set of 8 relations which are exhaustive and pairwise disjoint known as RCC8. The 8 relations of RCC8 and their logical connection to the original RCC is depicted in figures 2-1 and 2-2 (both taken from [13]). Other similar spatial languages have been proposed, such as SU_4, which focuses on topological operations over the interior and closure of regions [3]. While RCC and SU_4 can express similar ideas, SU_4 reasons more directly over points within regions without the definition of several canonical relationships. While more direct reasoning over point sets allows SU_4 to express some ideas not available in RCC, our focus on recognizing and inferring common-sense relations means that the more codified set of relations given by RCC makes more sense for our applications.

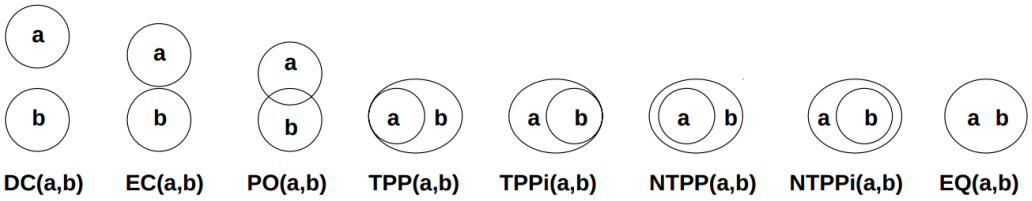

Figure 2-1: A depiction of the 8 spatial relations in the RCC fragment RCC8. Original figure from [13]

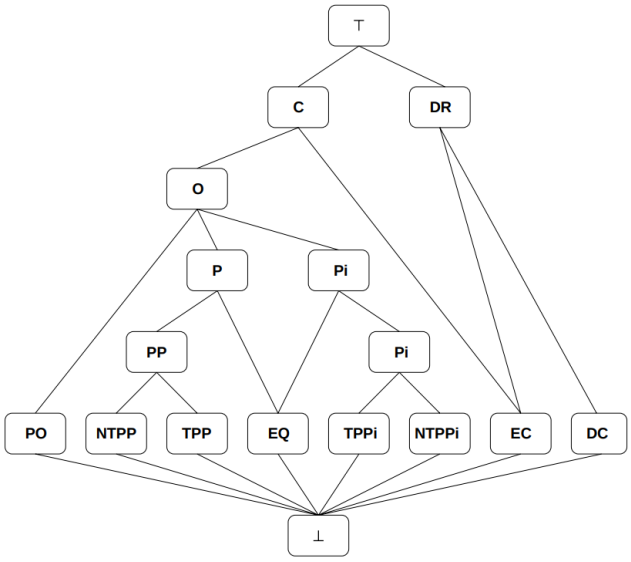

Figure 2-2: A subsumption lattice of various RCC relations. The 8 connected directly to $\perp$ are the RCC8 relations. Original figure from [13]

2.3 Synthesis

Once a set of demonstrations have been given and the necessary information inferred, the system must synthesize a plan or desired end state. The topic of task-level planning and synthesis is enormous, but for the purposes of this thesis we will focus on two aspects of liturature: synthesis from logical specifications, and synthesis for spatial problems with a focus on box packing.

2.3.1 Synthesis From Formal Specifications

Synthesis for formal specifications is the process of generating a demonstration which satisfies the specifications encoded in a formula. In order to satisfy a given formula, these demonstrations must be encoded in a way amenable to the given logical language. For example, synthesis for an LTL specification can only be given by a sequence of values for boolean state variables while synthesis for an STL specification can give a sequence of continuous state variables. Since this thesis is the first work we know of to generate synthesis for formulas using RCC, exact strategies from past work hold limited value. However, they serve as useful grounding for this thesis.

For temporal specification languages, many synthesis techniques encode a formula into a Markov Decision Process (MDP) [39, 47]. This approach suggests a more general pattern in synthesizing plans from logical specifications: instead of creating language-specific techniques, research attempts to encode the specification into a more well-understood planning tool.

Languages reasoning over continuous state variables often utilize various optimization techniques for synthesis. SpaTeL, the quad-tree spatio-temporal logic described in Section 2.2, performs synthesis using Partical Swarm Optimization (PSO) [23]. PSO is a non-gradient based probabilistic search technique wherein a large, randomly generated set of model parameters are randomly chosen, tested, and iteratively moved in space towards the best known point. Raman et al introduced a method for synthesis over STL specification by encoding the formulae as a Mixed Integer Linear Program (MILP) for Model Predictive Control [40]. As an alternative to MILPs, Pant et al

introduced a gradient based approach to synthesize signals for Metric Temporal Logic (MTL), a close cousin of STL [36]. Here, Pant creates an approximation of the "robustness" of a signal (a measure of how well a signal satisfies the specification) that is differentiable, and performs gradient optimization to maximize robustness.

With respect to synthesis, the closest research to our goals comes from Spatial Temporal Specification Language (STSL) [29]. STSL is an combination of STL and SU_4 that describes topological spaces over time. While no inference framework has been proposed over STSL, [29] provides a synthesis algorithm that utilizes simulated annealing. However, this algorithm is only shown to work in very low-dimensional systems with few decision variables, like setting the velocity and relative distance between two cars for a cruse control system. In contrast, this thesis presents a hierarchical search and optimization based algorithm designed to synthesize object placement over RCC descriptions of the pairwise relationship between multiple objects.

2.3.2 Box Packing Methods

In past decades, significant amounts of research have worked to refine techniques to solve the box packing problem. Introduced and first formally discussed by Martello et al, the packing problem involves n rectangular-shaped items of some dimension and one or multiple "bins" within which we must fit the objects without overlap [33]. There exist many variations on the problem (e.g. 2 vs 3-dimensional variations, level of freedom in orientation, shape of object, and constraints on weight distribution) and the resulting variety of solutions provides a good framing to discuss broad techniques for spatial planning with constraints.

Early algorithms in the world of bin packing reasoned very directly over specific strategies that may be useful in packing a box. Coffman et al for example, developed "strip packing" algorithms which place the one object at a particular box corner, and place following objects in rows based on heuristics of object size [12]. Later algorithms framed the box packing problem in terms of more general AI techniques, including search-based techniques [30, 32] or branch-and-bound methods [34].

State-of-the art techniques tend to focus on optimization methods to determine

object placement in the bin packing problem. Some early techniques explored optimization methods, such as Dowsland et al utilizing simulated annealing techniques to find object placements with minimal overlap [16]. Many later techniques moved away from gradient-based techniques in favor of Mixed Integer Linear Programming (MILP) in both 2- and 3-dimensional contexts [9]. MILP-based methods have found many variations for specific domains, including space operations [20] and cargo planes [37]. These works account for context-specific constraints such as particular non-rectangular container shapes or weight distribution that can be included in the MILP framework. Outside of context-specific research, others have added more general features to the fundamental optimization formulations. One common area of extension handles shapes other than rectangles. Santoro et al introduced a MILP-based technique to pack polygons and polyhedrons shaped out of slicing a rectangle or rectangular prism along planes [45]. Conversely, Erbayrak et al extended MILP object functions to favor configurations that place objects of the same pre-defined "family" close together given multiple bins [19].

While popular, MILP methods do not have a monopoly on modern box-packing methods. Edelkamp et al, for example, introduced a packing method using Nested Monte-Carlo Tree Search [17]. Here, the algorithm rapidly searches box packing rollouts and randomly chooses new successor states based on the success of prior packing attempts. While a more light-weight and simple implementation versus MILP algorithms, experiments have shown it to work better on discretized spaces. Conversely, Chernov et al introduced a non-MILP algorithm that specializes in packing ϕ objects: a class of objects based on the intersection of 2D or 3D objects containing most shapes a packing task will contain [11]. However, since the use of ϕ-objects removes the possibility of MILP optimization, [11] combines probabilistic search and gradient optimization to find viable object positions.

While the literature is extensive, few methods incorporate subjective human preference or constraints into object placement decisions. Methods that incorporate human preference are limited to a-priori assumptions over desirable states, such as families of objects being grouped together [19] or an ordering of object placement

based on object size [49]. In this thesis, we present a synthesis framework grounded in existing search and mixed integer optimization methods, but that satisfies logical specifications between objects defined in a fragment of RCC. This focus on synthesizing RCC specifications allows our framework to describe a much wider set of user preferences than prior work.

Chapter 3

Methods

In this chapter, we discuss our methods of inferring and planning patterns of object placement given human demonstrations. In Section 3.1, we formally describe objects, object relations, relational statements, and spatial contexts. In Section 3.2, we describe our inference method to find constraints given a set of human demonstrations. Finally in Section 3.3, we describe a synthesis method that plans a new set of object placements in accordance with the inferred constraints. The full pipeline for these methods is shown in Figure 3-1.

3.1 Problem Formulation

3.1.1 Objects

For this thesis, we define an object o to be a rectangle of width $o.w$ and length $o.l$ and to belong to an object "class" $o.c$ where all objects of the same class have the same width and lenght. Objects can either be placed or unplaced, as follows:

Definition 3.1.1 (placed object)**.** A "placed object" is an object o which has a defined x and y position in space, representing the location of the object's center. We label these attributes $o.x$ and $o.y$.

Definition 3.1.2 (un-placed object)**.** An "un-placed object" is an object o which has no defined placement in space, and therefore set $o.x$ and $o.y$ to $NULL$.

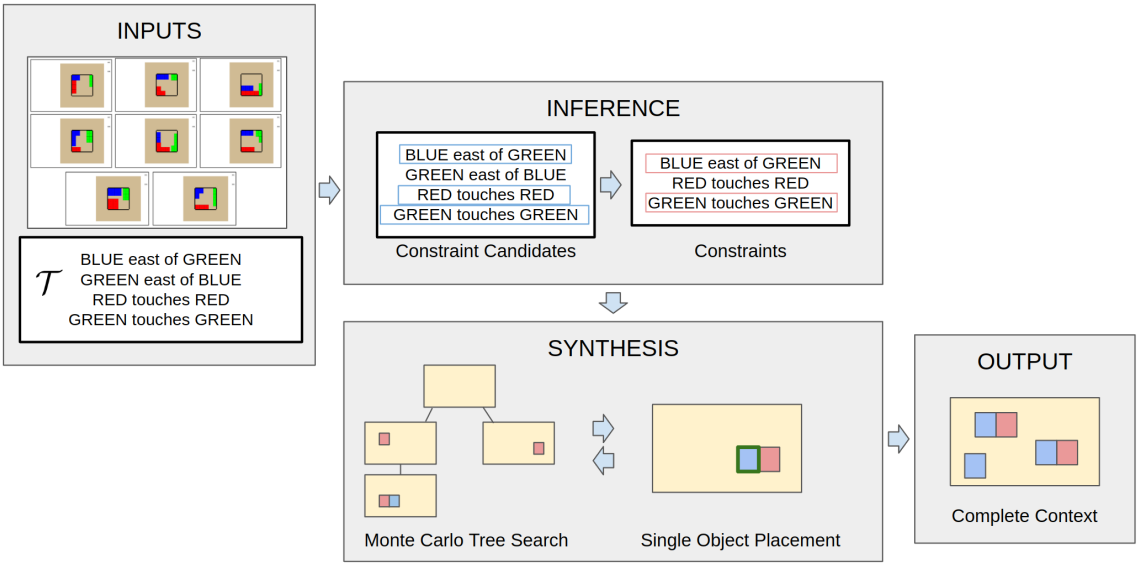

Figure 3-1: Pipeline of methods presented in this Thesis. This includes the inputs we expect from a demonstrator, the inference method, synthesis, out final output.

We will describe relations between objects using a fragment of the RCC language (presented in Section 2.2.1), and will therefore be describing objects under the RCC's definition of a region. All regions in RCC are defined solely by a "closure", or the closed set of points contained within a region. For a placed object o, the closure would be all points (x, y) such that

$$o.x - \frac{o.w}{2} \leq x \leq o.x + \frac{o.w}{2} \tag{3.1}$$

$$o.y - \frac{o.l}{2} \leq y \leq o.y + \frac{o.y}{2} \tag{3.2}$$

3.1.2 Relations

Throughout this thesis, our inference and planning algorithms will reason over RCC relations between placed objects. To define these relations, we utilize a fragment of RCC defined in Section 2.2.1. Specifically, we use only the relations 'discrete from' and 'externally connected to' (written $DR(x, y)$ and $EC(x, y)$ respectively). As described in Table 2.1, $DR(x, y)$ implies that the interiors of x and y do not overlap,

and $EC(x, y)$ implies that the exterior boundaries x and y touch. Of the remaining canonical relations described in Table 2.1 we decide not to include P, PP, EQ, O, PO, TPP and $NTPP$ because they describe some form of overlap between regions, which we specifically disallow given that our regions represent physical objects. Lastly, we do not include DC because it can already be easily described by our fragment via $DR(x, y) \wedge \neg EC(x, y)$.

While RCC as a whole can give expressive descriptions involving the overlap and contact between regions, it lacks an ability to describe relative position via a global coordinate system like (x, y). For example, no RCC relation can define "y is east of x". However, since our objects will always be rectangles with a position, width, and height, we can augment DR and EC to describe relative position within the (x, y) coordinate system.

Specifically, our use of non-overlapping rectangles mean that for any two objects a and b, it must be the case that either (1) all points in a have a greater x value than all points in b, (2) all points in a have a lower x value than all points in b, (3) all points in a have a greater y value than all points in b or (4) points in a have a lower y value than all points in b. We can notate these relative positional relations via cardinal directions $N/S/E/W$ augmenting the base DR and EC relations:

Definition 3.1.3 (position-augmented object relation)**.** position-augmented object relations are the basic DR and EC relations, but with a subscript indicating the relative position of one object with another. For DR relations, these include:

$$DR_N(a, b) \implies DR(a, b) \wedge a_y \geq b_y \quad \forall (a_x, a_y) \in a, (b_x, b_y) \in b \tag{3.3}$$

$$DR_S(a, b) \implies DR(a, b) \wedge a_y \leq b_y \quad \forall (a_x, a_y) \in a, (b_x, b_y) \in b \tag{3.4}$$

$$DR_E(a, b) \implies DR(a, b) \wedge a_x \geq b_x \quad \forall (a_x, a_y) \in a, (b_x, b_y) \in b \tag{3.5}$$

$$DR_W(a, b) \implies DR(a, b) \wedge a_x \leq b_x \quad \forall (a_x, a_y) \in a, (b_x, b_y) \in b \tag{3.6}$$

and for EC relations:

$$EC_N(a,b) \implies EC(a,b) \wedge a_y \geq b_y \quad \forall (a_x,a_y) \in a, (b_x,b_y) \in b \tag{3.7}$$

$$EC_S(a,b) \implies EC(a,b) \wedge a_y \leq b_y \quad \forall (a_x,a_y) \in a, (b_x,b_y) \in b \tag{3.8}$$

$$EC_E(a,b) \implies EC(a,b) \wedge a_x \geq b_x \quad \forall (a_x,a_y) \in a, (b_x,b_y) \in b \tag{3.9}$$

$$EC_W(a,b) \implies EC(a,b) \wedge a_x \leq b_x \quad \forall (a.x,a_y) \in a, (b_x,b_y) \in b \tag{3.10}$$

Notice that the existence of a position-augmented object relation implies the corresponding basic RCC DR or EC relation. For example, $DR_N(a,b) \implies DR(a,b)$. Additionally, for both DR and EC, the existence of the basic RCC relation implies one of the position-augmented object relations. For example, $DR(a,b) \implies DR_N(a,b) \vee DR_S(a,b) \vee DR_E(a,b) \vee DR_W(a,b)$.

In addition to the position-augmented object relations defined above, our algorithms must also reason over relations between classes. For this purpose, we define class relations:

Definition 3.1.4 (class relation)**.** Class relations use the same notation as position-augmented object relations, but take in two object classes. Given that $\mathcal{A}$ is the set of objects of class A and $\mathcal{B}$ is the set of objects of class B, our class DR relations are defined as:

$$DR_N(A,B) \implies DR_N(a,b) \quad \forall a \in \mathcal{A} \quad \forall b \in \mathcal{B} \tag{3.11}$$

$$DR_S(A,B) \implies DR_S(a,b) \quad \forall a \in \mathcal{A} \quad \forall b \in \mathcal{B} \tag{3.12}$$

$$DR_E(A,B) \implies DR_E(a,b) \quad \forall a \in \mathcal{A} \quad \forall b \in \mathcal{B} \tag{3.13}$$

$$DR_W(A,B) \implies DR_W(a,b) \quad \forall a \in \mathcal{A} \quad \forall b \in \mathcal{B} \tag{3.14}$$

and corresponding EC relations are:

$$EC_N(A,B) \implies EC_N(a,b) \quad \forall a \in \mathcal{A} \quad \exists b \in \mathcal{B} \tag{3.15}$$

$$EC_S(A,B) \implies EC_S(a,b) \quad \forall a \in \mathcal{A} \quad \exists b \in \mathcal{B} \tag{3.16}$$

$$EC_E(A,B) \implies EC_E(a,b) \quad \forall a \in \mathcal{A} \quad \exists b \in \mathcal{B} \tag{3.17}$$

$$EC_W(A,B) \implies EC_W(a,b) \quad \forall a \in \mathcal{A} \quad \exists b \in \mathcal{B} \tag{3.18}$$

3.1.3 Class Relational Statements

Throughout the thesis, we will use constraints to describe more complex relationships between classes of objects, which we call class relational statements:

Definition 3.1.5 (Class Relational Statement)**.** A class relational statement is a Boolean logic formula over class relations. We place two restrictions on the class relations within these formulas. First, each formula can only contain DR or EC class relations, not both. Second, all class relations in a formula must describe the same class's relationship to other objects. For example, a DR class relational statement for object class A might be $\phi = DR_N(A,B) \vee DR_N(A,C)$.

3.1.4 Spatial Context

In this thesis, we describe our inference and synthesis algorithm over a spatial "context" containing the various objects and relations between them:

Definition 3.1.6 (Spatial Context)**.** A spatial context is defined as the tuple $H = (\mathcal{S}, \mathcal{O}_p, \mathcal{O}_u, \boldsymbol{\alpha})$. Here, $\mathcal{S}$ is a 2 dimensional space of (x, y) points available for object placement. $\mathcal{O}_p$ is the context's set of placed objects. $\mathcal{O}_u$ is the context's set of unplaced objects. Lastly $\boldsymbol{\alpha}$ is the set of every position-augmented object relation between every pair of objects in $\mathcal{O}_p$.

Definition 3.1.7 (Complete Spatial Context)**.** A spatial context is considered a complete spatial context if $\mathcal{O}_u$ is empty (i.e. all objects in the spatial context have been assigned a placement within $\mathcal{S}$).

Throughout the thesis, we will use class relational statements to describe a context's relations $\boldsymbol{\alpha}$ in two distinct ways. First, we say that the entire relation set $\boldsymbol{\alpha}$ can satisfy a class relational statement ϕ. If ϕ is simply composed of one class relation, we check whether all objects in $\mathcal{O}_p$ hold to our definition of class relations in Eqn 3.11-3.18. For example, if $\phi = DR_N(A, B)$:

$$(\boldsymbol{\alpha} \implies \phi) \iff DR_N(a, b) \quad \forall a \in \mathcal{O}_p^A \quad \forall b \in \mathcal{O}_p^B \tag{3.19}$$

Where $\mathcal{O}_p^A$ is the subset of objects in $\mathcal{O}_p$ of class A and $\mathcal{O}_p^B$ is the subset of objects in $\mathcal{O}_p$ of class B. If ϕ is a more complex Boolean formula over individual class relations, we recursively search down the clauses of ϕ until we find individual relations which can be reasoned using eq 3.19.

Second, we say that an individual placed object o can satisfy a class relational statement ϕ. If ϕ is simply composed of one class relation, we check whether all position-augmented relations o has with other placed objects in $\boldsymbol{\alpha}$ holds according to Eqn 3.11-3.18. For example, if $\phi = DR_N(A, B)$:

$$(o \implies \phi) \iff DR_N(o, b) \quad \forall b \in \mathcal{O}_p^B \tag{3.20}$$

Once again, if ϕ is a more complex Boolean formula over individual class relations, we recursively search down the clauses of ϕ until we find individual relations which can be reasoned using eq 3.20.

We provide an example context H in Figure 3-2 to illustrate how we can use constraints. The context has 5 objects in $\mathcal{O}_p$, 2 of class B (objects b_1, and b_2) and 3 of class R (objects r_1, r_2, and r_3). In this context, $\boldsymbol{\alpha} \implies DR_S(B, R)$ since all objects of class B have the relation $DR_S(b, r)$ with all objects of class R. Additionally, $\boldsymbol{\alpha} \implies EC_S(B, R)$, since all objects of class B has the object relation $EC_S(b, r)$ with at least one object of class R. $\boldsymbol{\alpha} \not\implies EC_N(R, B)$ because object r_3 does not externally connect with an object of class B to the North. However, $r_1 \implies EC_N(R, B)$ since r_1 is externally connected to b_1 on the North. Similarly, $r_3 \implies DR_E(B, R)$ since r_3 has the relation $DR_E(r_3, b)$ with all objects $b \in \mathcal{O}_p^B$.

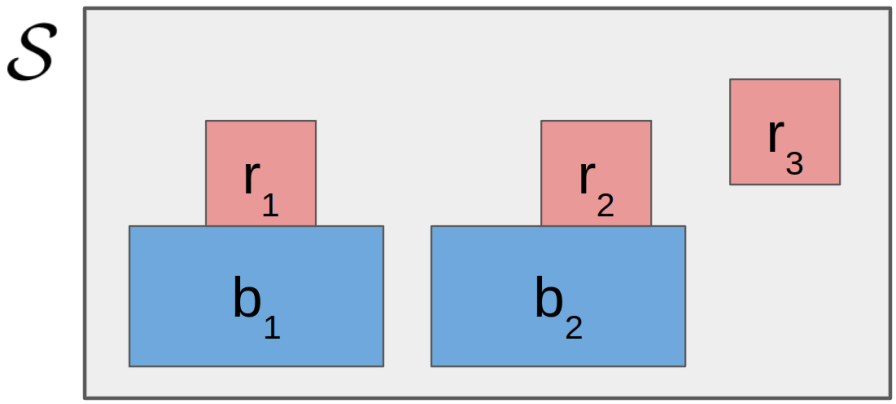

Figure 3-2: Example spatial context. The context space $\mathcal{S}$ is shown, as well as placed objects $\mathcal{O}_p$

3.1.5 Extended Example: Box Packing

To illustrate the ideas set forth in this thesis, we will frequently return back to the context of box packing. Here, the space $\mathcal{S}$ is a square table upon which objects can be placed. $\mathcal{O}_p$ always contains four objects of the wall class W, forming the walls of an open box on the table. In addition to wall objects are blue, red, and green objects of class B, R, and G which begin in $\mathcal{O}_u$, but must be given a position by our synthesis algorithm. A visualization of this problem is shown in Figure 3-3

3.2 Specification Inference

For our inference procedure, we will assume access to d demonstrations of a complete context state from a human collaborator $H_1 ... H_d$. We assume each H_i has the same space $\mathcal{S}$ and for which objects in $\mathcal{O}_p$ are subject to the same classes with the same set of underlying relational constraints. Note however, that different demonstrations can contain different numbers of each class. We also assume access to a set $\mathcal{T}$ of class relational statements. Our goal is to find the the subset of statements in $\mathcal{T}$ that best explain relational patterns in $\boldsymbol{\alpha}_1 ... \boldsymbol{\alpha}_d$. The inference process has 2 steps (1) find the subset $\bar{\mathcal{C}} \subset \mathcal{T}$ of class relational statements that are satisfied by $\boldsymbol{\alpha}_1 ... \boldsymbol{\alpha}_d$ (2) determine the likelihood that each statement in $\bar{\mathcal{C}}$ is satisfied by $\boldsymbol{\alpha}_1 ... \boldsymbol{\alpha}_d$ at random. A detailed

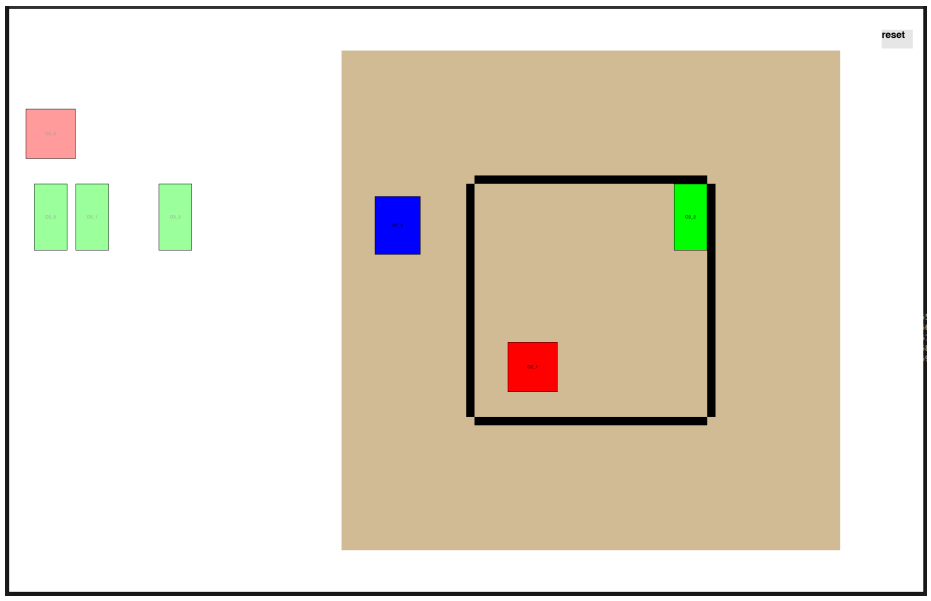

Figure 3-3: A visualization of the box packing problem we use as an extended example throughout the thesis. The large brown square represents the table on which objects can be placed. The four black rectangles represent the walls of a box. 3 placeable object classes are color-coded (red, green, and blue), three of which are placed on the table, and the rest of which are transparent on the left of the image. This image is taken from our packing interface used for experiments in chapter 4.

description of the process is in Algorithm 1.

3.2.1 Finding Candidate Constraints

The first step of our inference procedure involves finding the relation statements $\bar{\mathcal{C}} \subset \mathcal{T}$ that are satisfied by $\boldsymbol{\alpha}_1 ... \boldsymbol{\alpha}_d$. We call this subset of $\mathcal{T}$ candidate constraints:

Definition 3.2.1 (candidate constraint)**.** Given a set of demonstrations, candidate constraints are the set of statements in $\mathcal{T}$ such that:

$$\bar{\mathcal{C}} = \{\phi \in \mathcal{T} | \boldsymbol{\alpha} \implies \phi \; \forall \; \boldsymbol{\alpha} \in \{\boldsymbol{\alpha}_1 ... \boldsymbol{\alpha}_d\}\} \tag{3.21}$$

This process is enumerated in lines 2-5 of Algorithm 1.

3.2.2 Determining Candidate Constraint Relevance

Once we find $\bar{\mathcal{C}}$, we need to determine which candidate constraints $\phi \in \bar{\mathcal{C}}$ are relevant, and which are satisfied by the demonstrations but are not present in the unknown ground truth constraint set held by the demonstrator, which we label $\mathcal{C}_h$. To this end, we present a Bayesian formulation:

$$p(\phi \notin \mathcal{C}_h | H_1 ... H_d) \propto p(H_1 ... H_d | \phi \notin \mathcal{C}_h) p(\phi \notin \mathcal{C}_h) \tag{3.22}$$

and use a sampling based approach for inference. Given our lack of knowledge of the demonstrator, we use a uniform prior in our formulation. For the likelihood, we take $p(H_1 ... H_d | \phi \notin \mathcal{C}_h)$ to be the probability that each demonstration's object relations satisfy ϕ without ϕ being in the demonstrator's constraint set. If we assume that the probability of each $\boldsymbol{\alpha}_i$ satisfying ϕ when $\phi \notin \mathcal{C}_h$ is independent, we have:

$$p(H_1 ... H_d | \phi \notin \mathcal{C}_h) = p(\boldsymbol{\alpha}_1 \implies \phi, \ ... \ \boldsymbol{\alpha}_d \implies \phi | \phi \notin \mathcal{C}_h) \tag{3.23}$$

$$= \prod_{i=1}^{d} p(\boldsymbol{\alpha}_i \implies \phi | \phi \notin \mathcal{C}_h) \tag{3.24}$$

Furthermore, notice that $\boldsymbol{\alpha} \implies \phi$ if and only if $o \implies \phi$ for every $o \in \mathcal{O}_p$. Therefore, assuming the probability that each object satisfies ϕ is independent, we can further break down the likelihood as:

$$p(H_1 ... H_d | \phi \notin \mathcal{C}_h) = \prod_{i=1}^{d} \prod_{o \in \mathcal{O}_i^C} p(o \implies \phi | \phi \notin \mathcal{C}_h) \tag{3.25}$$

where we use C to notate the class over which ϕ reasons.

In order to approximate $p(o \implies \phi | \phi \notin \mathcal{C}_h)$, we take a sampling approach. Note that not all statements ϕ have the same probability of being satisfied randomly – EC relations, for example, technically have a 0 probability of appearing given continuous

space. Therefore, in order to approximate the likelihood p(o $\implies$ $\phi|\phi \notin \mathcal{C}_h$) we generate d_r randomized completed context states, which we label $H'_1...H'_{dr}$. While we give no definitive argument for the optimal value of d_r, all of our experiments use $d_r = 1000$. Each H'_i has the same $\mathcal{S}$ and object classes as the human demonstrations. However, the positions of objects in each $\mathcal{O}'_p$ have randomized x and y values within the context's given $\mathcal{S}$. We define our likelihood as the fraction of total objects from the random completed contexts that satisfy statement ϕ:

$$p(o \implies \phi|\phi \notin \mathcal{C}_h) = \max\left(\epsilon, \frac{\sum_{i=1}^{d_r}\sum_{o' \in \mathcal{O}_i'^C} \mathbf{1}(o' \implies \phi)}{\sum_{i=1}^{d_r}\sum_{o' \in \mathcal{O}_i'^C} 1}\right) \tag{3.26}$$

where ϵ is a small number that limits the likelihood from being 0 and $\mathbf{1}(o' \implies \phi)$ is an indicator variable set to 1 if object o' satisfies ϕ and 0 otherwise. We can now substitute this likelihood back into our Bayesian formulation, giving:

$$p(\phi \notin \mathcal{C}_h|H_1...H_d) \propto \prod_{i=1}^{d}\prod_{o \in \mathcal{O}_i^C} \max\left(\epsilon, \frac{\sum_{i=1}^{d_r}\sum_{o' \in \mathcal{O}_i'^C} \mathbf{1}(o' \implies \phi)}{\sum_{i=1}^{d_r}\sum_{o' \in \mathcal{O}_i'^C} 1}\right) \tag{3.27}$$

With this inferred posterior, we can define our set of constraints as all statements ϕ for which $p(\phi \notin \mathcal{C}_h|H_1...H_d)$ is under a given cutoff probability p_c, where $0 < p_c < 1$.

Definition 3.2.2 (constraint)**.** A constraint is a candidate constraint with a $p(\phi \notin \mathcal{C}_h|H_1...H_d)$ value under a cutoff probability p_c. Formally, this is:

$$\mathcal{C} = \{\phi \in \bar{\mathcal{C}}|p(\phi \notin \mathcal{C}_h|H_1...H_d) < p_c\} \tag{3.28}$$

The inference procedure is detailed in Algorithm 1.

3.2.3 Extended Example: Inference with Box Packing

To illustrate the inference process described above, we step through Algorithm 1 in tandem with an example from the box packing context described in Section 3.1.5. The

Algorithm 1: Specification Inference

Function `Inference`$(H_1 ... H_d, \mathcal{T})$**:**
- $\bar{\mathcal{C}} = \emptyset$
- **for** $\phi \in \mathcal{T}$ **do**
  - **if** ***SatisfiesAll***$(H_1 ... H_d, \phi)$ **then**
    - $\bar{\mathcal{C}}.add(\phi)$
- $H'_1 ... H'_{d_r} \leftarrow$ *Randomly generated complete contexts*
- $\mathcal{C} = \emptyset$
- **for** $\phi \in \bar{\mathcal{C}}$ **do**
  - $p(o \implies \phi | \phi \notin \mathcal{C}_h) = \max\left(\epsilon, \frac{\sum_{i=1}^{d_r} \sum_{o' \in \mathcal{O}'^C_i} \mathbf{1}(o' \implies \phi)}{\sum_{i=1}^{d_r} \sum_{o' \in \mathcal{O}'^C_i} 1}\right)$
  - $p(\phi \notin \mathcal{C}_h | H_1 ... H_d) = \prod_{i=1}^{d} \prod_{o \in \mathcal{O}^C_i} p(o \implies \phi | \phi \notin \mathcal{C}_h)$
  - **if** $p(\phi \notin \mathcal{C}_h | H_1 ... H_d) < p_c$ **then**
    - $\mathcal{C}.add(\phi)$
- **return** $\mathcal{C}$

Function `SatisfiesAll`$(H_1 ... H_d, \phi)$**:**
- **for** $H \in \{H_1 ... H_d\}$ **do**
  - **if** $H.\boldsymbol{\alpha} \not\Longrightarrow \phi$ **then**
    - **return** *False*
- **return** *True*

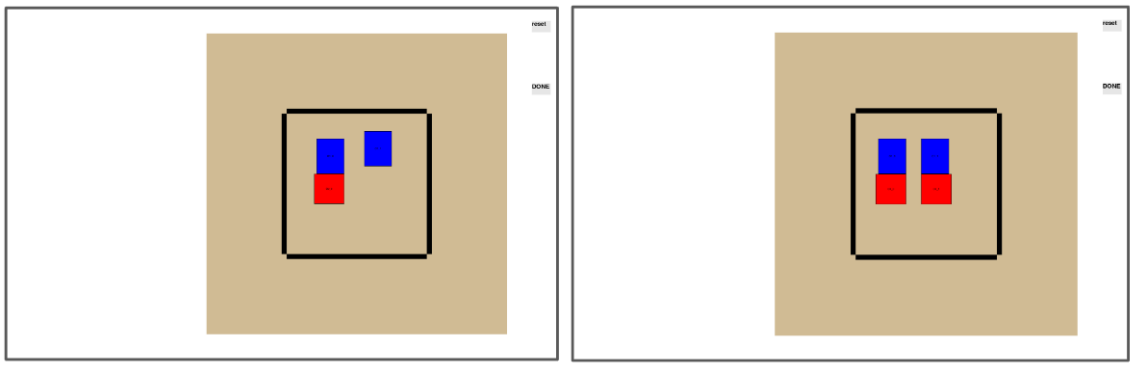

Figure 3-4: Example demonstrations in our box packing context used in Section 3.2.3. Demonstration 1 is on the left and demonstration 2 in on the right.

inference function takes in demonstrations $H_1...H_d$, and a set of relational statements $\mathcal{T}$ (line 1). In our example, we have two demonstrations H_1 and H_2 shown in Figure 3-4. Additionally, we define $\mathcal{T}$ with 4 statements:

- $\phi_1 = DR_N(B, R)$ – Blue objects are North of Red objects
- $\phi_2 = EC_N(B, R)$ – Blue objects are externally connected and North of Red objects
- $\phi_3 = EC_S(R, B)$ – Red objects are externally connected and South of Blue objects
- $\phi_4 = (\neg DR_i(B, N)) \wedge (\neg DR_i(B, S)) \wedge (\neg DR_i(B, E)) \wedge (\neg DR_i(B, W))$ – Blue objects are not North, East, South, or West of all Wall objects (i.e. Blue objects are within the box walls).

In line 2, we initialize candidate constraints as an empty set. In line 3, we iterate through all statements $\phi \in \mathcal{T}$. In line 4, we check if all demonstrations satisfy ϕ, by calling the `SatisfiesAll` function starting on line 14. In this function, we iterate through all demonstrations (line 15) and check if its set of object relations satisfies ϕ (line 16). If any demonstration's object relations do not satisfy ϕ, return False (line 17) otherwise return True (line 18). In our example, both demonstrations satisfy ϕ_1, ϕ_2, and ϕ_4. However, $H_2.\boldsymbol{\alpha} \not\Longrightarrow \phi_3$ because one Blue object does not connect with a Red object on the North side. Each constraint for which `SatisfiesAll` returns True is added to the candidate constraints $\bar{\mathcal{C}}$. Thus, in our example, $\bar{\mathcal{C}} = \{\phi_1, \phi_3, \phi_4\}$.

We then generate random demonstrations $H'_1...H'_{dr}$, which we won't show explicitly for our example (line 6). We then initialize an empty set for our constraints $\mathcal{C}$ (line 7). We then iterate over each constraint candidate $\phi \in \bar{\mathcal{C}}$ (line 8). For each ϕ, we determine the likelihood that an object o satisfies ϕ, using equation 3.26 (line 9). In our example, imagine our random demonstrations gave likelihoods of .4 for ϕ_1, .01 for ϕ_3, and .3 for ϕ_4. In line 10, we use this object likelihood to determine the full probability that $\phi \notin \mathcal{C}_h$. Since there are a total of 4 Blue objects across all demonstrations, $p(\phi_1 \notin \mathcal{C}_h | H_1...H_d) = .4 * .4 * .4 * .4 = 0.0256$ and

$p(\phi_3 \notin \mathcal{C}_h | H_1 ... H_d) = .01 * .01 * .01 * .01 = 1.0e-8$. Since there are a total of 3 Red objects across all demonstrations, $p(\phi_4 \notin \mathcal{C}_h | H_1 ... H_d) = .3 * .3 * .3 = 0.027$. We then check if the posterior $p(\phi \notin \mathcal{C}_h | H_1 ... H_d) < p_c$ (line 11), and if so add ϕ to constraints $\mathcal{C}$ (line 12). If we have $p_c = .01$, only ϕ_3 makes the cutoff, and thus $\mathcal{C} = \{\phi_3\}$. Finally, we return the set of constraints in $\mathcal{C}$ (line 18).

3.3 Synthesis

Once our inference algorithm finds a set of constraints $\mathcal{C}$, our synthesis algorithm must find placements for unplaced objects such that all constraints in $\mathcal{C}$ are satisfied. This planning has two hierarchical parts. At a high level, we describe a Monte-Carlo tree approach to search over potential placements for each object in $\mathcal{O}_u$ such that all constraints in $\mathcal{C}$ are met [7]. In order to find the best next object placement to expand a node in the search tree, the Monte Carlo Tree Search algorithm queries the lower level system for a promising object placement. In this thesis, we present a Mixed Integer Quadratic Program (MIQP) based approach for the lower level system to find the position an object $o \in \mathcal{O}_u$ that maximally satisfies constraints in $\mathcal{C}$ given an existing context. We intentionally separate the higher and lower level aspects of this algorithm because there are contexts in which a mixed integer optimization is not the appropriate choice. For example, in domains with discretized space or non-rectangular environments $\mathcal{S}$, Nested Monte Carlo search or non-linear optimization respectively have been shown effective [6, 17].

3.3.1 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a randomized search algorithm commonly used in single and multiplayer games [8]. The fundamental algorithm includes four main steps.

- *Node selection:* Starting from the root node, the algorithm traverses down the existing tree, and chooses a node to expand.

- *Expansion:* Starting from the chosen node, the algorithm creates a child node representing the next move taken in the process.
- *Simulation:* The child node is expanded down a default policy until hitting a terminal node.
- *Backup:* The resulting terminal node is backed up to the root, and visited nodes' statistics are updated.

For the Monte Carlo Tree Search, we will use these steps to search object placements satisfying constraints $\mathcal{C}$. To achieve this, we represent each node on the tree as a tuple $Q = (H, C_u, C_p, n, v)$, defined by the following:

- H is a context associated with a node
- C_u is a mapping from each object $o \in \mathcal{O}_u$ to all constraints in $\mathcal{C}$ that reason over the object o's class (which we will notate as $\mathcal{C}[o.c]$)
- C_p is a mapping from each object $o \in \mathcal{O}_p$ to constraints $\phi \in \mathcal{C}[o.c]$ such that $o \not\Rightarrow \phi$
- n is the number of leaf nodes in a node's subtree
- s is the average value of leaf nodes in a node's subtree as determined by the `Simulate` function in Algorithm 2

For our MCTS, we take in a context H, C_p, and C_u, and return a completed context state that best satisfies all constraints in C_p and C_u.

During synthesis, there may be situations where not all constraints are satisfiable. Therefore, we require a metric for the overall "score" of a context to compare the relative desirability of two context states, defined here:

Definition 3.3.1 (context score)**.** For a context H and constraints $\mathcal{C}$, we define a context score as the weighted ratio of constraints satisfied by each object in $\mathcal{O}_p$ and total constraints possible to satisfy. In this score, we place more importance on constraints

which we are more confident were not satisfied randomly in the demonstrations (i.e. constraints with a lower $p_h(\phi)$ are weighed with higher importance). We therefore weigh each constraint by $-\ln(p_h(\phi))$:

$$score(H) = \frac{\sum_{o \in \mathcal{O}_p} \sum_{\phi \in \mathcal{C}[o.c]} -\ln(p_h(\phi))(1 - \mathbf{1}(o \implies \phi))}{\sum_{o \in \mathcal{O}_p} \sum_{\phi \in \mathcal{C}[o.c]} -\ln(p_h(\phi))} \tag{3.29}$$

Now that we have a defined a scoring mechanism, we will describe our synthesis method. The first step in Monte Carlo Tree Search is node selection. One popular selection approach, referred to as UTC, uses the Upper Confidence Bound (UCB) algorithm to select branches down which to traverse the search tree [26]. If a node down the tree has unexplored possible moves, the algorithm expands it into a child node. This variant provides a structured way to balance exploration (i.e. ensure that we explore many routes down any particular state) and exploitation (i.e. explore routes down the tree that have given the most promising scores in the past).

Despite the exploration/exploitation advantages of UTC, its rule to expand any node with possible unexplored children means it only works well in situations with a small branching factor. However, since object placement occurs in the continuous spatial domain, we cannot realistically expand a node to every possible child (i.e. place every unplaced object in every conceivable position). Therefore, we modify the node selection process as follows. Starting at the root node, we decide to select a node for expansion with probability p_e (lines 13 and 14 of Algorithm 2). If the current node is not chosen for expansion, we choose to traverse down its children as determined by the UCB algorithm (line 15 of Algorithm 2). This probabilistic method allows us to retain the exploration/exploitation property of UCB while not requiring every possible child of every node to be explored. Additionally, as long we require no two branches of a node to place the same object in the same location, as time reaches infinity, we retain the completeness property inherent to MCTS algorithms since every possible object placement will eventually be explored.

Once a node v is chosen for expansion, one unplaced object $o \in \mathcal{O}_u$ is randomly selected and the algorithm queries the single object placement algorithm to find a

position to place o. The placement of o at position (x, y) generates a new node v_n with o in $\mathcal{O}_p$ instead of $\mathcal{O}_u$ (lines 5 and 6 of Algorithm 2).

Next, we simulate placing the remainder of objects in $\mathcal{O}_u$ in the context of the newly created node v_n (line 7 of algorithm 2). This process involves iteratively selecting an object $o \in \mathcal{O}_u$ and querying the single placement algorithm to generate a new node with o as a placed object (lines 18-20 of Algorithm 2). When $\mathcal{O}_u$ is empty, we return the node with a complete context and check if its score is 1. If so, we return the node, as it satisfies all constraints in $\mathcal{C}$. This process is detailed in lines 8-9 in Algorithm 2. Note that none of the contexts generated during simulation are added to the tree.

Lastly, we "backup" the search tree starting at node v_n (line 10 of Algorithm 2). We update both the value of the node (line 24 of Algorithm 2) and number of leaf nodes (line 25 of Algorithm 2). Then, we recurse up the tree and update $v.s$ and $v.n$ until reaching the root node (line 23 of Algorithm 2). Once the backup step is finished, we select a new node and begin the process over again.

3.3.2 Finding Singular Object Placement

Whenever the Monte Carlo Tree Search algorithm requires a new child from node v, we need a way to provide a promising position to place an object o. For this thesis, we will call the object being placed the "decision object". In this Section, we describe an MIQP-based placement algorithm that finds the position for the decision object that maximizes $score(v.H)$.

First, we need a way to place an object a such that it has a position-augmented object relation with another object b. Since all objects are restricted to rectangles, the bounds on object a's position are linear, and thus amenable to a mixed integer

Algorithm 2: Synthesis

Function `MCTS(`H, C_p, C_u`)`**:**
  $v0 = (H, \mathcal{O}_p, C_p, 0, 0)$
  while *not timeout* **do**
    $v_s =$ **Select** (v_0)
    $o \leftarrow RandomChoice(v_s.\mathcal{O}_u)$
    $v_n =$ **GetChild**(v_s, o)
    $v_t =$ **Simulate**(v_n)
    if *score*$(v_t.H) = 1$ **then**
      return v_t
    Backup$(v_n, score(v_t.H))$
Function `Select(`v`)`**:**
  while v *not terminal* **do**
    if *with probability* p_e **then**
      return v
    $v = \underset{v' \in v_{children}}{\arg\max}\, v.value + c\sqrt{\frac{\log(v.visited)}{v'.visited}}$
  return v
Simulate v**:**
  while v *not terminal* **do**
    $v' =$ **GetChild**(v)
    $v = v'$
  return v
Backup `Backup(`v, s_t`)`**:**
  while v *not root* **do**
    $v.s = \frac{v.s \cdot v.n}{v.n+1} + \frac{s_t}{v.n+1}$
    $v.n++$
    $v = v.parent$

problem. The bounds for a DR class relations are as follows :

$$DR_N(a,b): \quad a.y \geq b.y + \frac{b.l}{2} + \frac{a.l}{2} \tag{3.30}$$

$$DR_S(a,b): \quad a.y \leq b.y - \frac{b.l}{2} - \frac{a.l}{2} \tag{3.31}$$

$$DR_E(a,b): \quad a.x \geq b.x + \frac{b.w}{2} + \frac{a.w}{2} \tag{3.32}$$

$$DR_W(a,b): \quad a.x \leq b.x - \frac{b.w}{2} - \frac{a.w}{2} \tag{3.33}$$

And our EC bounds are:

$$EC_N(a,b): \quad a.y = b.y + \frac{a.l}{2} + \frac{b.l}{2} \tag{3.34}$$

$$a.x \geq b.x - \frac{a.w}{2} - \frac{b.w}{2} \tag{3.35}$$

$$a.x \leq b.x + \frac{a.w}{2} + \frac{b.w}{2} \tag{3.36}$$

$$EC_S(a,b): \quad a.y = a.y - \frac{a.l}{2} - \frac{b.l}{2} \tag{3.37}$$

$$a.x \geq b.x - \frac{a.w}{2} - \frac{b.w}{2} \tag{3.38}$$

$$a.x \leq b.x + \frac{a.w}{2} + \frac{b.w}{2} \tag{3.39}$$

$$EC_E(a,b): \quad a.x = b.x + \frac{a.w}{2} + \frac{b.w}{2} \tag{3.40}$$

$$a.y \geq b.y - \frac{a.l}{2} - \frac{b.l}{2} \tag{3.41}$$

$$a.y \leq b.y + \frac{a.l}{2} + \frac{b.l}{2} \tag{3.42}$$

$$EC_W(a,b): \quad a.x = b.x - \frac{a.w}{2} - \frac{b.w}{2} \tag{3.43}$$

$$a.y \geq b.y - \frac{a.l}{2} - \frac{b.l}{2} \tag{3.44}$$

$$a.y \leq b.y + \frac{a.l}{2} + \frac{b.l}{2} \tag{3.45}$$

When choosing an object's position, we want the decision object to satisfy $a \implies \phi$ for every $\phi \in C_u[a.c]$. Recall that ϕ is composed of class relations, which have different definitions for DR and EC, as defined in equations 3.11-3.14 and 3.15-3.18 respectively. If ϕ is composed of one DR class relation such as $DR_N(A,B)$,

$a \implies DR_N(A, B)$ requires a to have the object relation $DR_N(a, b)$ with every object of class B. Therefore, when using our mixed integer process to find the position for a, we simply include the constraints from Eqn 3.30-3.33 and 3.34-3.45 for every object of class B.

If ϕ is composed of one EC class relation such as $EC_N(A, B)$, $a \implies EC_N(A, B)$ requires a to have the relation $EC_N(a, b)$ with at least one object of class B. To encode this requirement, we use a strategy known as big M notation, where we create an indicator variable $\mathbf{1}(b)$ for each object of class B. We then make a copy of the bounds from Eqn 3.34-3.45 for object of class b, and modify the bounds such that every inequality of the form $g < h$ becomes $g \leq h + M \cdot \mathbf{1}(b)$ and M is chosen to be large enough such that any reasonable values g or h will still satisfy the inequality. We then constrain the binary variables such that:

$$|\mathcal{O}_p^B| - 1 \geq \sum_{b \in \mathcal{O}_p^B} \mathbf{1}(b) \tag{3.46}$$

to enforce at least 1 binary variable to be 0, and thus force the EC bounds to only act upon 1 object of class B.

Constraints ϕ are not necessarily single DR or EC class relations, but are recursively defined over the basic Boolean "and", "or", and "not" operators. We use separate strategies to handle each in a mixed integer framework.

"And" Operators: Given an "And" constraint $\phi = \bigwedge_{i=1}^{|\phi|} \phi_i$, we can simply recursively apply constraints to each ϕ_i individually until reaching single DR or EC class relations. Here, we use the notation $|\phi|$ to refer to the number of clauses the "And" relation involves.

"Or" Operators: Given an "Or" constraint $\phi = \bigvee_{i=1}^{n_\phi} \phi_i$, we need to ensure that at least one ϕ_i is satisfied. Once again, we utilize big M notation by assigning each ϕ_i a binary indicator variable $\mathbf{1}(\phi_i)$ and relaxing inequalities from all bounds in each clause $g < h$ to $g < h + M \cdot \mathbf{1}(\phi_i)$. To ensure that at least one ϕ_i is satisfied, we

enforce:

$$|\phi| - 1 \geq \sum_{i=1}^{|\phi|} \mathbf{1}(\phi_i) \tag{3.47}$$

"Not" Operators: Unlike "And" and "Or" operators, "Not" acts over only one clause. Therefore, our handling of "Not" lies in how it affects the "And", "Or", or single class relation clause over which it operates. Not operators acting over "And" and "Or" use de Morgan's laws:

$$\neg(\bigwedge_{i=1}^{|\phi|} \phi_i) = \bigvee_{i=1}^{|\phi|} \neg\phi_i \tag{3.48}$$

$$\neg(\bigvee_{i=1}^{|\phi|} \phi_i) = \bigwedge_{i=1}^{|\phi|} \neg\phi_i \tag{3.49}$$

which can then be handled by the rules over "And" and "Or" operators. When "Not" acts over a single class relation, we can constrain the object placement by similar bounds as eq 3.30-3.45. However, we modify these bounds in 3 ways. First, we inverse all $\geq$ and $\leq$ to $<$ and $>$ respectively. Second, we need to ensure that at least one of the new bounds is satisfied. To achieve this, we once again use big M notation. For example, if we had an atomic EC_N constraint, the bounds described above would modify to:

$$b.y < a.y + \frac{a.l}{2} + \frac{b.l}{2} + m_1 \cdot M \tag{3.50}$$

$$b.y > a.y + \frac{a.l}{2} + \frac{b.l}{2} - m_2 \cdot M \tag{3.51}$$

$$b.x < a.x - \frac{a.w}{2} - \frac{b.w}{2} + m_3 \cdot M \tag{3.52}$$

$$b.x > a.x + \frac{a.w}{2} + \frac{b.w}{2} - m_4 \cdot M \tag{3.53}$$

$$3 > m_1 + m_2 + m_3 + m_4 \tag{3.54}$$

Notice that we can negate an equality constraint via a greater than and less than constraint. Lastly, we reverse our handling of class-based relations for DR and EC

relations. Thus, $\neg DR$ statements use big M notation to act over at least one object of another class while $\neg EC$ statements constrain all objects of the related class.

Discussion thus far in this Section seeks to satisfy constraints relevant to the decision object. In addition to these, we also choose the place such that it satisfies as many constraints from C_p as possible. The strategies employed to satisfy these constraints follow naturally from the constraint handling described above, and is not explained in detail.

Beyond learned relational constraints, we apply 3 more types of constraints to the MIQP. First, we constrain no two objects to overlap. MILP-based box packing algorithms use Big M notation to ensure that no two objects overlap. For each placed object o, we constrain the decision object's x and y positions as:

$$a.y \geq o.y + \frac{a.l}{2} + \frac{o.l}{2} + M \cdot m_1 \tag{3.55}$$

$$a.y \leq o.y - \frac{a.l}{2} - \frac{o.l}{2} + M \cdot m_2 \tag{3.56}$$

$$a.x \geq o.x + \frac{a.w}{2} + \frac{w_o}{2} + M \cdot m_3 \tag{3.57}$$

$$a.x \leq o.x - \frac{a.w}{2} - \frac{w_o}{2} + M \cdot m_4 \tag{3.58}$$

$$3 \geq m_1 + m_2 + m_3 + m_4 \tag{3.59}$$

Secondly, we constrain all objects' x and y values to be within bounds x_{lb}, x_{ub}, y_{lb}, and y_{ub}. Lastly, we check whether other children of the node v placed the same decision object as the one being expanded in this iteration. This process is detailed in lines 2 - 5 of Algorithm 3. If so, we need the MIQP to choose a new position for the decision object in order to avoid redundancy in the tree. Therefore, we impose a constraint similar to the non-overlapping constraints above around a small ϵ x ϵ box:

$$y \geq o.y + \epsilon + M \cdot m_1 \tag{3.60}$$

$$y \leq o.y - \epsilon + M \cdot m_2 \tag{3.61}$$

$$x \geq o.x + \epsilon + M \cdot m_3 \tag{3.62}$$

$$x \leq o.x - \epsilon + M \cdot m_4 \tag{3.63}$$

$$3 \geq m_1 + m_2 + m_3 + m_4 \tag{3.64}$$

In order to encourage more diversity in object placement, we define an objective that maximizes the overall distance between where we place an object and where it has been placed in previous children of the MCTS node being expanded:

$$max_{x,y} \sum_{o \in v.children} (x - o.x)^2 + (y - o.y)^2 \tag{3.65}$$

where we use the notation $o \in v.children$ to the refer to the decision object's corresponding object in each child of v.

We cannot expect the decision object's chosen position to always satisfy all constraints. Therefore, we find the subset of all unsatisfied constraints with maximum overall score that is satisfiable. The overall process is described in Algorithm 3. Note that we call a sub-process **MIQP**, which we do not write out in full. It represents solving the MIQP problem described throughout this Section. **MIQP** takes as input the node being expanded, the decision object, all constraints as a set of tuples (o, ϕ) where we require $o \implies \phi$, and the positions the decision object must avoid.

During this process, we first create a set $\mathcal{L}$ of tuples (o, ϕ) where we require $o \implies \phi$. In lines 7-8, we collect constraints relevant the decision object, and in lines 9-11, we collect unsatisfied constraints from placed objects C_p. Then, we iterate through $\mathcal{L}$ and attempt to find a position for the decision object with each constraint applied one at a time. If any cannot be satisfied, we remove them from $\mathcal{L}$ (lines 12-14 of Algorithm 3). Then, we form every pairwise subset of constraints from $\mathcal{L}$, and check which pairwise relations are unsatisfiable (lines 15 - 18 of Algorithm 3). Then, we generate all subsets of tuples $\mathcal{L}' = 2^{\mathcal{L}}$, and eliminate any that include an infeasible pair of constraints (lines 19 - 23 in Algorithm 3). Then, we sort $\mathcal{L}'$ by the total sum of their constraint's metric discussed in Definition 3.3.1:

$$\sum_{(o,\phi) \in l} -\ln(p_r(\phi)) \tag{3.66}$$

Where l is one of the subsets of $\mathcal{L}$ contained in $\mathcal{L}'$ (line 24 of Algorithm 3). Lastly, we iterate through the sorted $\mathcal{L}'$ and attempt to find a placement for the decision object given each subset of constraint tuples (lines 25 - 26 of Algorithm 3). Since we sort $\mathcal{L}'$, the first l for which the **MIQP** succeeds in finding a position for the decision object will produce the position with the maximum possible score for the context H. We then create a new node v' with the decision object placed and return the node (lines 27 and 28 of Algorithm 3).

3.4 Summary

In this chapter, we have introduced a set of methods to help robots and other automation better understand and reproduce human-intuitive patterns of object placement. To show the efficacy of this pipeline, we design and perform a human pilot study in Section 4.2. Here, an individual is instructed to give 8 demonstrations of a task in the box packing context described in 3.1.5 which are used as input to our inference algorithm. The learned constraints then inform our synthesis algorithm, which finds a set of object placements satisfying the learned constraints. The aims of this study are two-fold. First, we aim to show that the demonstrator is satisfied with the synthesized complete context without knowing any of the specifics of the algorithm presented here. Second, we hope the experiment will show that our method is able to capture constraints not specified by the participants in natural language. This result would show that our algorithms capture aspects of the participant's intuition not present in a natural language description.

Algorithm 3: Single Object Placement

Function `GetChild`(v, o_d)**:**
- $Avoid \leftarrow \emptyset$
- **for** $v' \in v.children$ **do**
  - **if** $o_d \in v'.\mathcal{O}_p$ **then**
    - $Avoid \leftarrow (o.x, o.y)$
- $\mathcal{L} = \emptyset$
- **for** $\phi \in v.\mathcal{C}_u[o_d.c]$ **do**
  - $\mathcal{L}.add((o_d, \phi))$
- **for** $o \in v.\mathcal{O}_p$ **do**
  - **for** $\phi \in v.\mathcal{C}_u[o.c]$ **do**
    - $\mathcal{L}.add((o, \phi))$
- **for** $(o, \phi) \in \mathcal{L}$ **do**
  - **if** ***MIQP***$(v, \{(o, \phi)\}, Avoid)$ *fails* **then**
    - $\mathcal{L}.remove(\phi)$
- $Infeasible \rightarrow \emptyset$
- **for** $(o_1, \phi_1), (o_2, \phi_2) \in \mathcal{C}'$ **do**
  - **if** ***MIQP***$(v, \{(o_1, \phi_1), (o_2, \phi_2)\}, Avoid)$ *fails* **then**
    - $Infeasible.add(\{\phi_1, \phi_2\})$
- $\mathcal{L}' = 2^{\mathcal{L}}$
- **for** $l \in \mathcal{L}'$ **do**
  - **for** $I \in Infeasible$ **do**
    - **if** $I \subset l$ **then**
      - $\mathcal{L}'.remove(l)$
- *Sort subsets* $l \in \mathcal{L}'$ *by* $\sum_{(o,\phi)\in l} -\ln(p_r(\phi))$
- **for** $l \in \mathcal{L}'$ **do**
  - **if** ***MIQP***$(v, \{l\}, Avoid)$ *succeeds* **then**
    - $v' = v.place(o_d)$
    - *return v'*

Chapter 4

Evaluation

With our problem statement, inference, and synthesis algorithm described, we move on to evaluations of our system. For the thesis, we evaluate the system in two ways. First, we test the practical run-time statistics of our synthesis algorithm. Second, we perform a human pilot study to understand whether the inference algorithm captures and recreates patterns important to human participants.

4.1 Synthesis Runtime Analysis

In this section, we present a runtime evaluation of our synthesis algorithm. If the system takes a prohibitive amount of time to find a satisfactory set of object placements given a set of class constraints, its use cases are limited. For this evaluation, we test the runtime as the function of two variables: number of objects to place and number of constraints per object class.

4.1.1 Experimental Setup

For our run-time evaluation, we conduct tests in the same box packing python environment as described in Section 3.1.5. However, we set all place-able object classes (i.e. non-wall objects) to be 20 x 20 pixel squares. Within this context, we vary the number of objects and the number of constraints per object class. Specifically, we test

every combination 2-6 place-able objects and 1-5 constraints per object class. The object constraints are chosen from the relational statements $\mathcal{T}$ that we will use in our human study (Section 4.2). We test each combination of #objects and #constraints per object class with 10 different sets of constraints.

If chosen randomly, some combinations of constraints may be contradictory. For example, if $DR_N(G, B)$ is chosen for the constraints of object class G and $DR_N(B, G)$ is chosen for the constraints of object class B, no possible placement of B and G objects can satisfy both constraints. Furthermore, the possible contradictions between constraints are often difficult to determine a-priori. Therefore, to choose a set of viable constraints, we randomly choose 5 constraints per object class from $\mathcal{T}$ (the maximum tested in this evaluation) and attempt to satisfy a context with 2 objects of each class R, G, and B. If the context can be satisfied before timing out, we keep the set of 5 constraints per object class as a viable constraint set, $\mathcal{C}^1$. We repeat this process until we find 10 viable constraint sets $\mathcal{C}^1 ... \mathcal{C}^{10}$. Then, for every test involving n constraints per object class, we randomly chose n constraints per class from one of the viable constraint sets. For example, for the 5th of 10 tests involving 4 objects and 3 constraints per object class, we randomly choose 3 constraints per class from $\mathcal{C}^5$ and attempt to satisfy them with 4 objects, each of which is randomly assigned a class.

For this experiment, we used the MIT Supercloud supercomputer, and evaluate runtime based on 2 metrics [43]. First, we simply measure the time taken to find a solution. However, since runtime varies from processor to processor, we also evaluate based on the algorithm's total number of calls to the MIQP Gurobi optimizer, which will remain consistent across platforms.

4.1.2 Results and Discussion

Both runtime metrics we use for analysis (number of queries to the MIQP optimizer and runtime in seconds) increase rapidly both with respect to number of place-able objects and number of constraints per class. In Figure 4-1, we show the median of both metrics as a function of the number of place-able objects. Each individual

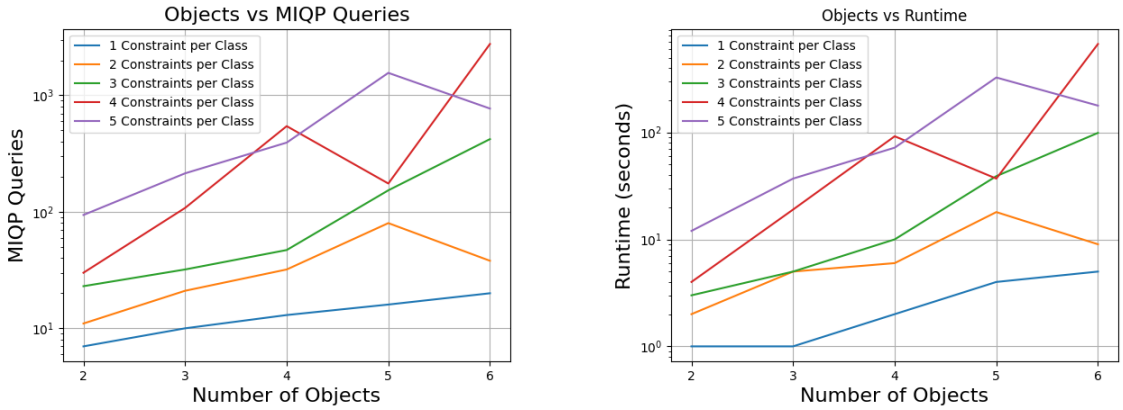

Figure 4-1: Median runtimes of our trials as a function of the number of place-able objects in each trial. The left figure uses the number queries to the MIQP solver as a runtime metric, and the right figure uses seconds passed during the trial as a runtime metric. Each plot shows 5 lines, which represent the median runtime of trials for a different number of constraints per object class. The y axis of both plots are in a logarithmic scale.

plot contains 5 lines, representing a different number of constraints per class used in synthesis. In these figures, we see that experiments involving the fewest objects and number of constraints take only a couple seconds. However, experiments with more objects and constraints take a median of hundreds of seconds to complete.

Each plot in Figures 4-2 and 4-3 focus on trials involving the same number of constraints per object class. Here, we show the runtime for all 10 trials of each combination of number of constraints and number of place-able objects along with the median line. In each graph, the points shown in blue represent tests that found a solution while points shown in red represent tests that exceeded our limit of 500 nodes explored in the search tree.

4.2 Human Pilot Study

4.2.1 Background and Hypotheses

As described in Section 2.1, large efforts in robotics research have focused on extracting human task specifications for automation. In this section, we seek to understand

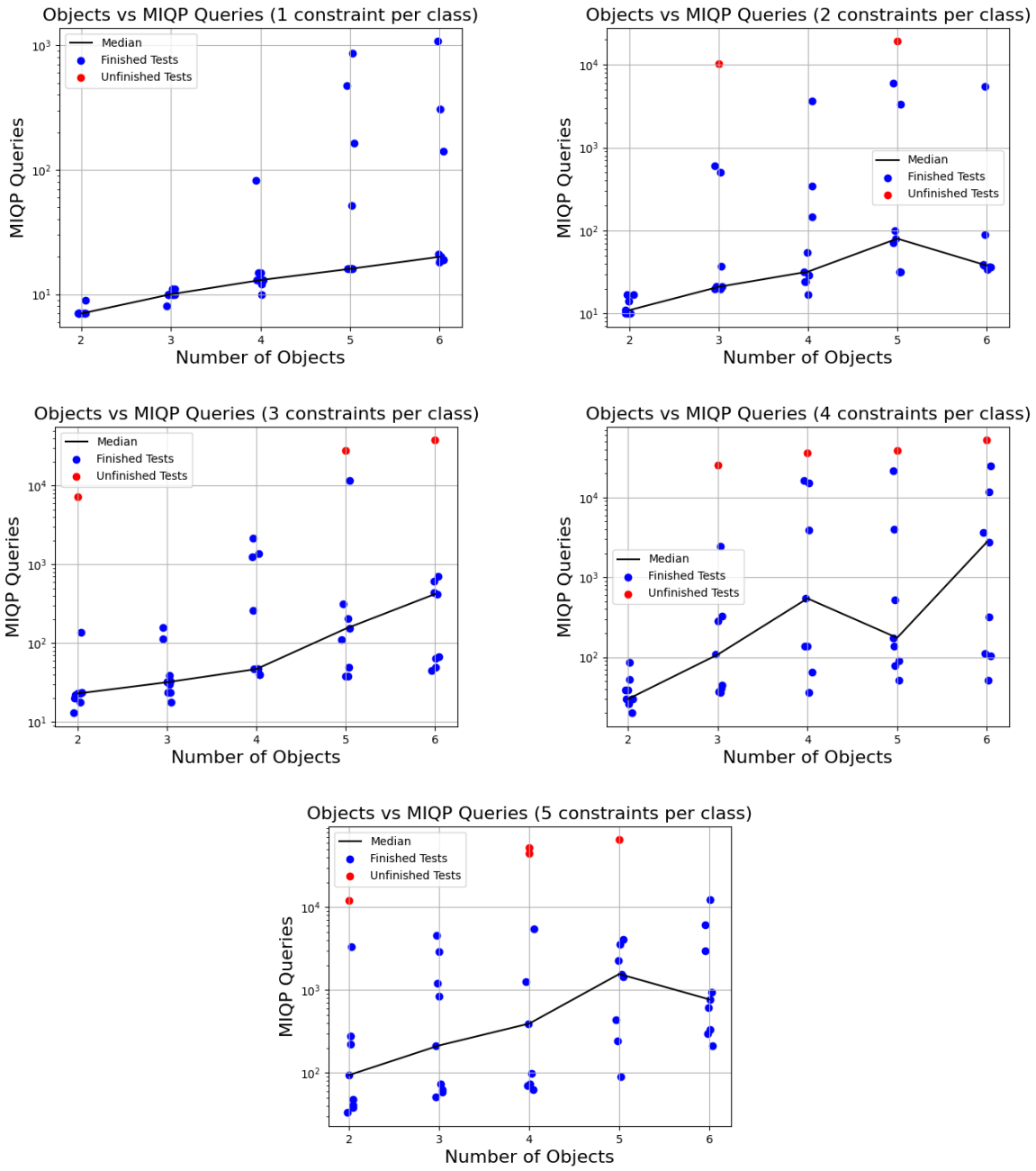

Figure 4-2: These plots show the number of queries our synthesis algorithm makes to the MIQP optimizer for each trial in our test suite. Each plot contains trials with the same number of constraints per object class, and shows the number of queries as a function of the number of objects to place. Each blue point represents a trial for which a solution is found, and each red point represents a trial for which the solution is not found within the maximum search tree size of 500 nodes. In addition to the individual points, a black line shows the median queries across the all 10 trials with the same number of place-able objects. Note that we artificially shift all points slightly along the x direction to avoid visual overlap, but in reality each point should exactly align with its number of place-able objects. The y axis of all plots are in a logarithmic scale.

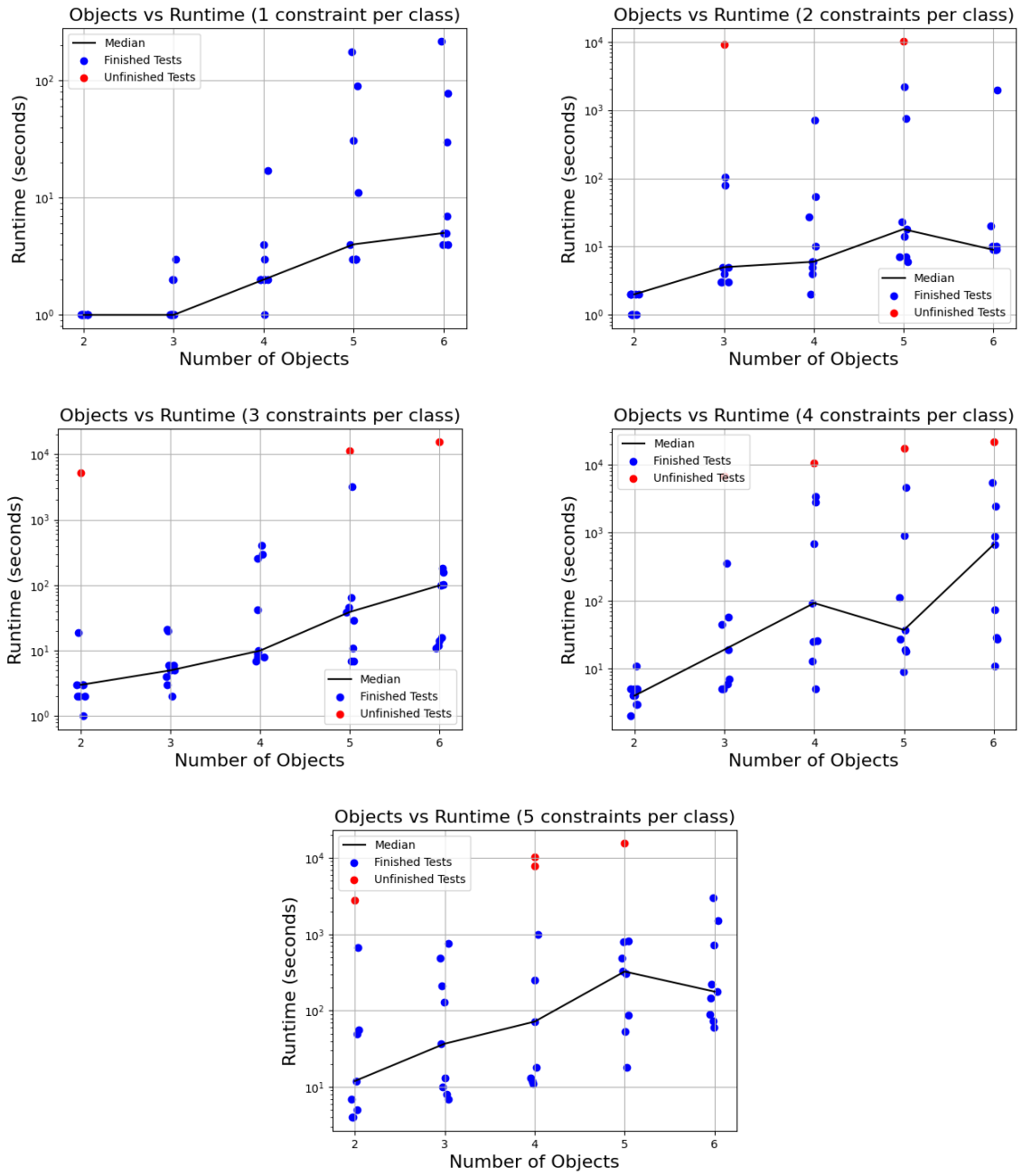

Figure 4-3: These plots show the runtime in seconds of each trial in our test suite. Each plot contains trials with the same number of constraints per object class, and shows the runtime as a function of the number of objects to place. Each blue point represents a trial for which a solution is found, and each red point represents a trial for which the solution is not found within the maximum search tree size of 500 nodes. In addition to the individual points, a black line shows the median runtime across the all 10 trials with the same number of place-able objects. Note that we artificially shift all points slightly along the x direction to avoid visual overlap, but in reality each point should exactly align with its number of place-able objects. The y axis of all plots are in a logarithmic scale.

how our methods can contribute to this body of research in two specific ways.

First, we want to understand weather the RCC-based spatial logic approach presented in this thesis accurately captures the human's intuitive understanding of tasks involving the spatial relationship between objects. Learning from demonstration systems focusing on inferring logical specifications have either focused purely on temporal systems [46] or use Signal Temporal Logic to define control sequences for a robot within space [1, 39]. The few approaches that reason over regions of space directly are designed for applications in physical system dynamics such as reaction-diffusion systems and power grids rather than human intuition [21, 23]. To our knowledge, no algorithm has been presented to infer specifications for a spatial language focused on Quantiataive Spatial Reasoning, such as RCC or SU_4. However, since these languages are specifically designed to describe human-intuitive relationships between regions, we expect that the rules inferred and demonstrated by our synthesis algorithm will capture implicit human patterns and constraints in object relations.

H1: The inference and planning systems presented in Chapter 3 will capture and reproduce object placement patterns that demonstrators rate as more satisfactory than not.

In addition, we hope to show our method's advantage over non learning from demonstrations methods of specification. Namely, past research has investigated using Natural Language Processing (NLP) to provide specification to robots [5, 10, 24, 51]. One such method even uses a natural language input to provide Linear Temporal Logic specifications, and gives feedback if the specifications are infeasible [41]. However, past research has shown natural language descriptions to be insufficient for defining tasks, as humans frequently underspecify [22]. Therefore, in this pilot study we qualitatively compare the constraints inferred by our algorithm and descriptions of spatial constraints provided by participants. One can imagine constraints inferred vs specified by the demonstrator as two overlapping sets. One is the set of constraints inferred by our algorithm and the other is the set of constraints the demonstrator specifies. An un-specified constraint is in the set inferred by our algorithm, but not in the human-specified constraints. Conversely, the set of constraints specified by

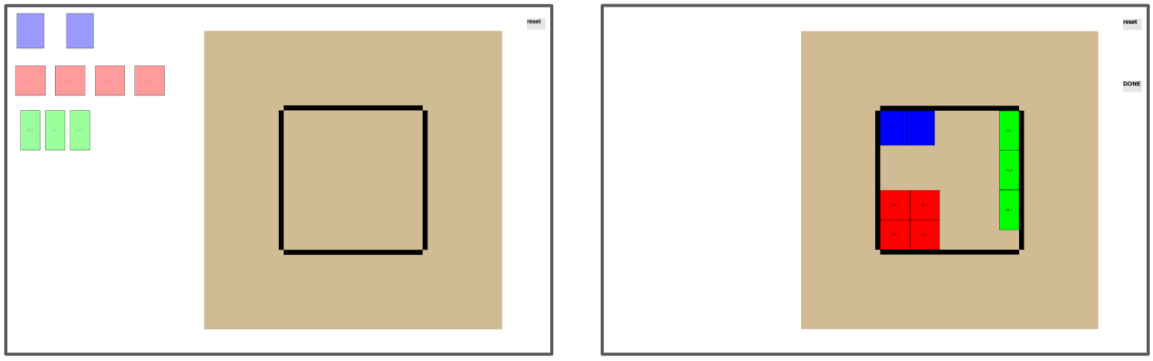

Figure 4-4: The demonstration interface used in our experiment. The initialization state is shown on the left, and a completed interface is shown on the right.

the demonstrator but not inferred by the algorithm is important to understanding aspects of human intuition not captured in our algorithm. However, while some such constraints are identified in our discussion, it is not the main focus of our analysis.

H2: Our inference system will capture spatial constraints between objects that are not explicitly specified by human demonstrators.

4.2.2 Experimental Setup

For the pilot study, we utilize the box-packing environment described in 3.1.5 to both record human demonstrations and present an object placement generated by our synthesis system. The environment initializes with a brown square representing a "table" (acting as the context space) and four already-placed objects representing the walls of an open box on the table. Outside the table are 2 to 4 objects of each class R (for red objects), G (for green objects), and B (for blue objects) which make up the un-placed objects in the context. As a visual indicator, unplaced objects are slightly transparent, while placed objects are entirely opaque. To provide a demonstration, the user may click an unplaced object to select it, move the unplaced object anywhere on the virtual table, and press "Enter" to place the object in the given position. If the placed position is not allowed (i.e. not on the table or overlapping an existing object) the object returns to its initialized position as an unplaced object. As the participant drags and drops an object, the object's position will always snap to the nearest 10th pixel in each direction. Note that the user cannot move an object once

Figure 4-5: Pre-generated completed contexts initially shown to pilot study participants.

placed – if they are unhappy with a placement, they must reset the entire context to its initial state. Once all objects are placed on the table, the pygame environment presents a "done" button, which will record an image of the environment as well as object positions and relations for all placed objects. An example initial and final state of this environment is shown in Figure 4-4.

The study involves 6 steps. First, we introduce participants to the demonstration environment – what the various objects represent (i.e. table, walls, objects to place) and demonstrate how to select and place objects in the context space. Second, we present the participant with a screencapture of 8 pre-generated example completed contexts shown in Figure 4-5. These examples are designed to incorporate constraints able to be expressed using the position-augmented RCC framework discussed in Section 3.1 (e.g. red objects are south of blue objects, red objects are externally connected to red objects, etc). Third, we ask the participant to give 8 demonstrations via the pygame environment, such that they follow the same patterns of object placement seen in the 8 pre-generated examples. Throughout this process, the participant is

allowed to refer back to the pre-generated examples, though the random number of each object class presented to the participant means they are unable to simply copy example contexts in every demonstration. Fourth, we ask the participant to write down how objects are placed on the table, as if describing the task to a friend. We allot a maximum of 5 minutes to record this description. Fifth, we show the participant a completed context generated by our synthesis algorithm using the constraints inferred from the participant's demonstrations. We ask the participant to rate the level to which the generated image match patterns provided by their demonstrations on a 1-5 Likert scale (where 1 is a very unsatisfactory object placement and 5 is a very satisfactory object placement). In addition to the Likert scale, we ask the participant to give an open-ended explanation of their score. Sixth and lastly, we ask the participant to describe patterns they used in placing objects. However, instead of using natural language, we ask them to provide rules based on the relationships between object colors. For inspiration, we provide several illustrative examples of a context and possible object relationships that one can describe. We are specific that these illustrative examples are not related to the rules from the pre-generated examples, and that the participant is not limited to the object relations in the given examples. The examples are only shown to inspire ideas of the kinds of patterns we want the participant to describe.

The eight pre-generated examples shown to the participant include the following position-augmented RCC constraints. While these constraints do not uniquely describe the eight examples, they are the ground truth constraints used when generating the examples:

1. $DR_N(B, R)$ – Blue objects are North of Red objects.

2. $DR_E(G, R)$ – Green objects are East of Red objects.

3. $DR_E(G, B)$ – Green objects are East of Blue objects.

4. $\bigvee_{i \in \{N,S,E,W\}} EC_i(B, B)$ – Blue objects are externally connected to another blue object on any side.

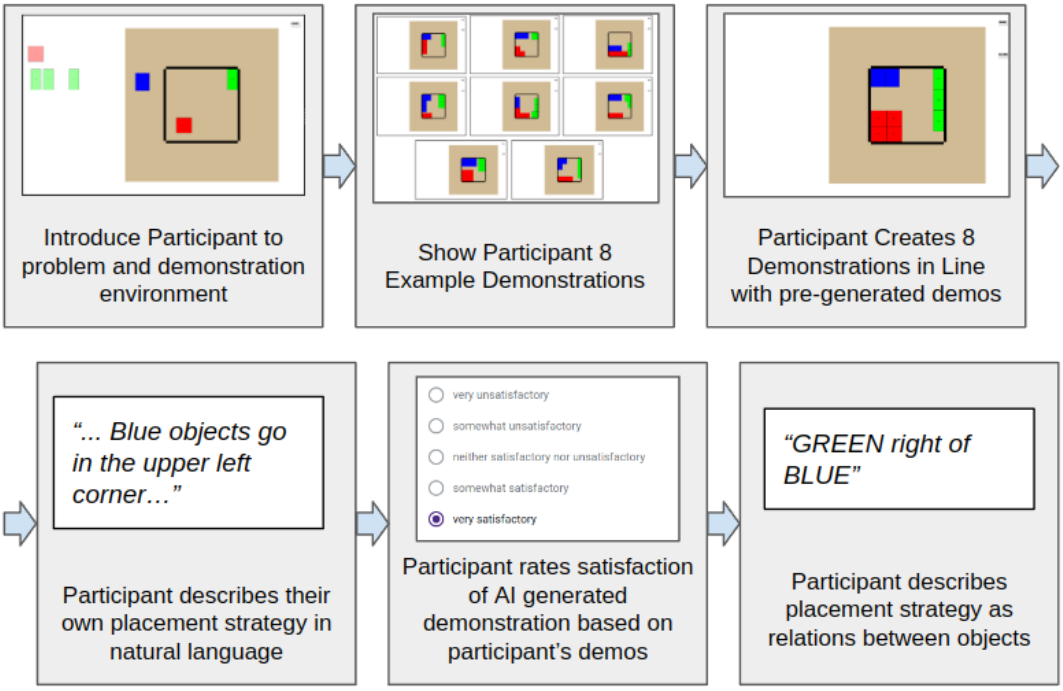

Figure 4-6: A flowchart of the user study.

5. $\bigvee_{i\in\{N,S,E,W\}} EC_i(R,R)$ – Red objects are externally connected to another red object on any side.

6. $\bigvee_{i\in\{N,S,E,W\}} EC_i(G,G)$ – Green objects are externally connected to another green object on any side.

7. $\bigvee_{c_1\in\{R,G,B,W\}} \bigvee_{c_2\in\{R,G,B,W\}} (EC_N(B,c_1)\wedge EC_W(B,c_2))\vee(EC_W(B,c_1)\wedge EC_S(B,c_2))\vee (EC_S(B,c_1)\wedge EC_E(B,c_2))\vee(EC_E(B,c_1)\wedge EC_N(B,c_2))$ – Blue objects are externally connected to any object on two adjacent sides (i.e. on the North and West side, West and South side, South and East side, or East and North side).

8. $\bigvee_{c_1\in\{R,G,B,W\}} \bigvee_{c_2\in\{R,G,B,W\}} (EC_N(R,c_1)\wedge EC_W(R,c_2))\vee(EC_W(R,c_1)\wedge EC_S(R,c_2))\vee (EC_S(R,c_1)\wedge EC_E(R,c_2))\vee(EC_E(R,c_1)\wedge EC_N(R,c_2))$ – Red objects are externally connected to any object on two adjacent sides (i.e. on the North and West side, West and South side, South and East side, or East and North side).

9. $\bigvee_{c_1\in\{R,G,B,W\}} \bigvee_{c_2\in\{R,G,B,W\}} (EC_N(G,c_1)\wedge EC_W(G,c_2))\vee(EC_W(G,c_1)\wedge EC_S(G,c_2))\vee$

$(EC_S(G, c_1) \wedge EC_E(G, c_2)) \vee (EC_E(G, c_1) \wedge EC_N(G, c_2))$ – Green objects are externally connected to any object on two adjacent sides (i.e. on the North and West side, West and South side, South and East side, or East and North side).

10. $\bigwedge_{i \in \{N,S,E,W\}} \neg DR_i(B, W)$ – Blue objects are not North, South, East, or West of all wall objects (i.e. inside the box).

11. $\bigwedge_{i \in \{N,S,E,W\}} \neg DR_i(R, W)$ – Red objects are not North, South, East, or West of all wall objects (i.e. inside the box).

12. $\bigwedge_{i \in \{N,S,E,W\}} \neg DR_i(G, W)$ – Green objects are not North, South, East, or West of all wall objects (i.e. inside the box).

4.2.3 Data Collection

For the study, we recruited 5 participants (2 males, 3 females aged 18 - 26). All participants have earned or are pursuing a bachelor's degree in a STEM field. In each case, all written aspects of the survey (i.e. natural language description, evaluation of AI-generated demonstration, and relation-based description) were completed on the participant's personal computer via a Google form (and Google sheet in the case of relational descriptions). The pygame demonstration environment and AI-generated demonstrations were completed on a separate computer.

In addition to the written portions of the survey, we record the demonstrations from each participant and the AI-generated example created using participant demonstrations. Given the scope of this experiment as a pilot study with limited participants, our goal is not to obtain statistically significant results, but to gather information useful in designing future experiments.

4.2.4 Results and Discussion

In discussing the results of our survey, we will begin with high-level observations concerning which constraints from the 8 pre-generated examples translated to the participants' 8 demonstrations. All participants' demonstrations maintain that R, G,

Participant Satisfying vs Demonstrating Constraints													
Participant		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
P1	Demonstrated				✓	✓	✓	✓	✓	✓	✓	✓	✓
	Specified				✓	✓	✓				✓	✓	✓
P2	Demonstrated	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓
	Specified	✓	✓	✓	✓	✓	✓				✓	✓	✓
P3	Demonstrated	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
	Specified	✓	✓	✓	✓	✓	✓				✓	✓	✓
P4	Demonstrated	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Specified	✓	✓	✓	✓	✓	✓				✓	✓	✓
P5	Demonstrated				✓	✓	✓	✓	✓	✓	✓	✓	✓
	Specified				✓	✓	✓	✓	✓	✓			

Table 4.1: A representation of whether participants demonstrated and specified each of the 12 ground truth constraints used to generate the 8 pre-generated examples. Notice that every participant did not explicitly specify some subset of the constraints they demonstrated.

and B objects must externally connect to another object of its own class (i.e. constraints 4-6 listed above). Additionally, all participants provided examples such that all objects were placed within the box walls (i.e. constraints 10-12 listed above). Three participants (P2, P3, and P4) noticed that G objects should be East of R and B objects, and R objects are South of B objects (constraints 1-3 above). P1, P4, and P5 gave demonstrations following constraints 7-9 above (that each class of object should be externally connected to another object in two adjacent directions). However, the participants who did not follow constraints 7-9 came very close. P3 only broke constraint 8 with one object and P4 only broke constraint 7 with one object and constraint 9 with two objects over all 8 demonstrations. A summary of these findings is shown in Table 4.1

Discussion on *H1*

The Likert scale is our main quantitative measure of participants' satisfaction with the AI generated demonstration. A higher satisfaction score from the generated example implies that our RCC framework and inference system captured the object placement patterns important to the participant. Across the 5 participants, the average Likert

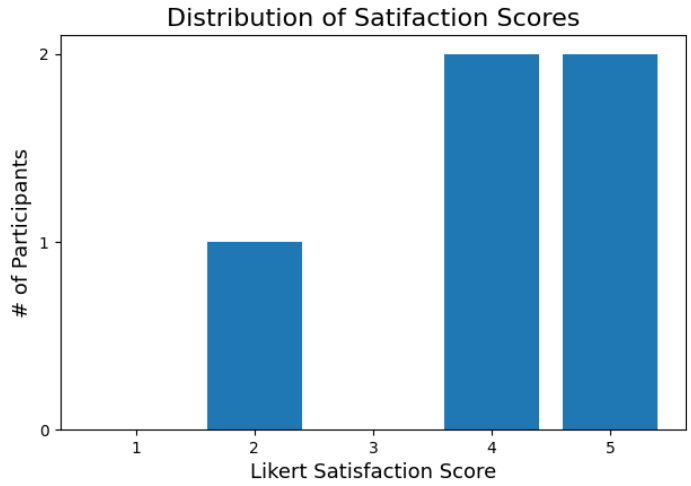

Figure 4-7: Distribution of satisfaction scores for human study participant. A score of 1 is "very unsatisfactory" and a score of 5 is "very satisfactory".

scale is 4, with two participants giving a score of 5, two giving a score of 4, and one giving a score of 2. These results are shown in Figure 4-7

To understand why the majority of participants rated the generated demonstration less than "very satisfactory" we refer to their explanation for the Likert score. Here, we see three common through-lines in the study's results. First, some participants' explanations heavily focused on whether objects were edge-aligned. P2 and P3 are the best examples of this trend. Neither P2 nor P3 followed the "object contact on adjacent sides" constraints (i.e. constraints 7 - 9) for at least one class of object. Therefore, the AI generated a demonstration where objects would be externally connected to another object only one one side. However, unlike P2 and P3, the AI-generated demonstration does not place connected objects in alignment, as shown in in Figure 4-8. P2 cited this lack of object alignment as the sole reason for providing a 4 on the likert scale, and P3 cited this lack of alignment in their score of 2 on the Likert scale.

However, it is also worthy to note instances where a participants' focus on object alignment did not result in an imperfect score on the Likert scale. P1 references the importance of object alignment in both their natural language explanation and relation-based rules. However, unlike P2 and P3, P1's demonstrations perfectly follow

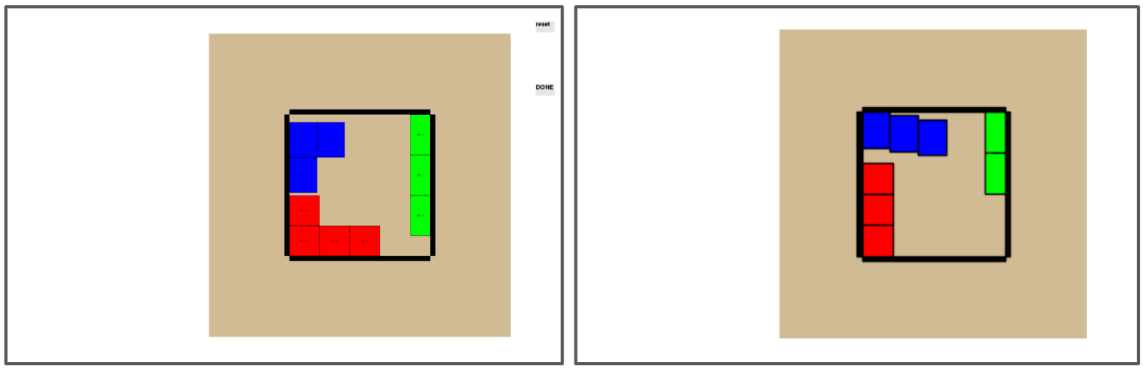

Figure 4-8: An example from study participant that demonstrates blue objects being aligned (Left) and the AI generated demonstration with blue objects not aligned (Right).

the "object contact on adjacent sides" constraints (rules 7-9). These constraints, along with constraints requiring objects of the same class being connected (rules 4 - 6) led the AI to generate a demonstration where connected objects are aligned. This case study provides an important observation: even if a person's specified rules do not appear in the algorithm's inferred constraints directly, inferred constraints may still lead the synthesis algorithm to produce to the same patterns as the person's specified rules.

The second through-line is the importance of composite shapes created by each object class (e.g. whether objects of a particular class form a straight line, an "L"-shape or are arranged in a square-like pattern). P4 cited undesirable composite shapes as their reason for giving a 4 on the Likert scale, specifically stating that none of the provided demonstrations had "non-square" L-shaped class groupings for both the red and blue object classes, while the AI-generated demonstration did. In contrast, P5 gave the AI generated demonstration a 5 on the Likert scale, and approved of the generated composite shapes.

We also see composite shapes mentioned outside the Likert scale explanation. P3 referenced composite shapes in their natural language description of rules, stating "...preferring horizontal repetitions to vertical repetitions, aligning edges exactly without space in between of the same color blocks, and preferring a 2x2 grid over 4x1 for example." Additionally, informal conversations with P2 and P4 post-experiment

revealed that their goal to preserve desired composed shapes seen in examples led to their lack of adherence to constraints 7 - 9. Though unspoken, their near-perfect adherence to these constraints may suggest that the tendency to follow constraints 7-9 was present, but this was overwritten by the desire to maintain desired composite shapes.

Outside the Likert scale, we find one more through-line in the rules provided by various participants: a tendency to describe patterns based on "at least one object" rather than all objects of a class. In the relation-based rules section of the survey, both P1 and P3 used language that mentioned constraints involving not all objects of a class. Specifically, P1 described "(at least one) Red" in a rule, and P3 described rules as "one red box" and "remaining red boxes."

Ultimately, the average Likert score of 4 gives merit to $H1$, though the results here are certainly not conclusive. The algorithm clearly did not capture some constraints involving object alignment and composite shapes which brought the satisfaction score of the majority of participants below the maximum of 5.

Discussion on *H2*

Our analysis of $H2$ involves comparing the natural language and relation-based rules provided by participants with rules inferred from their demonstrations. Using Table 4.1, we can see that within the subset of constraints used as ground truth in the 8 pre-generated examples, every participant underspecified (i.e. gave demonstrations consistent with a constraint, but did not specify the constraint in their explanations). On average, participants only specified 71.3% of the constraints inferred by our algorithm.

Of course, natural language will often imply constraints without stating them outright. For example, participant P2 wrote "Blue pieces go roughly in the top left corner of the box ... . Red pieces go in the bottom left corner" in their natural language explanation. While this doesn't explicitly say "B is North of R", their explanation implies the constraint $DR_N(B, R)$, and thus we count $DR_N(B, R)$ as a specified constraint in table 4.1.

There are other instances when the translation of natural language into our specification is less clear. For example, P3 followed constraint 7 (all blue objects should contact another object on two adjacent sides). In their natural language explanation, they say "Blue in the upper left corner of the box, then place all the blues in that corner, preferring horizontal repetitions to vertical repetitions, aligning edges exactly without space in between of the same color blocks, and preferring a 2x2 grid over 4x1 for example." This explanation, while clearly distinct from rule 7 in its framing, will most of the time (though not always) lead to satisfying constraint 7. Therefore, it's difficult to say whether this description counts as an under-specification. However, since, there are some configurations following P3's explanation that violate constraint 7, we leave this as unspecified in the table.

Despite such ambiguity, this experiment does clearly show evidence of human under-specification, and our algorithm's ability to interpret constraints that go unspecified by participants.

4.2.5 Recommendations for a Full Experiment

Given the results of this pilot study, there are several changes we propose for a full experiment. When participants found that AI-generated demonstrations do not satisfy the constraints given in their demonstrations, the cited reason often involved patterns not able to be represented by RCC. The most noteworthy of these are the tendency to focus on object alignment and composite shapes formed by objects of the same class. While these trends show some limitation in our methodology, it seems likely that our choices to (1) use only 2-4 objects per class and (2) make all objects of the same class the same shape, exacerbated the repetition of composite shapes in our pre-generated examples. One participant even noted that seeing our demonstrations immediately reminded them of Tetris blocks. Since scaling up the number of objects may not be feasible, we have two recommendations to mitigate the problem. First, we can intentionally inject examples into our pre-generated demonstrations that do not form expected shapes and do not align with other objects. Second, we can randomize the shapes of the given objects. While this modification does require minor changes

to our methodology, it would prevent the formation of a few, predictable composite shapes.

Additionally, all participants are in the process of earning or have earned degrees in STEM. Since this research is intended to inspire systems for domain experts in fields outside engineering, the results we find may not be representative of the target population. Therefore, future studies should look into ways to expand the participant pool outside STEM students and professionals.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we present a series of techniques intended to further robots' ability to understand and reproduce human-intuitive spatial patterns in object placement. In Chapter 1, we motivate the need for systems capable of understanding and reproducing tasks demonstrated by domain experts. Many small to medium manufacturing firms need to increase efficiency to stay afloat, but require the ability to rapidly retrain robots without the assistance of a programmer. In Chapter 2, we provide a background on research applicable to the problem, including learning from demonstrations as a broad field, learning specifications over logic languages, formal spatial languages designed for human-intuitive reasoning, and methods of spatial synthesis with an emphasis on bin packing problems.

Chapter 3 presents this thesis's three core technical contributions. In Section 3.1, we present a formal problem formulation involving the relationships between objects in a spatial "context" via Region Connection Calculus. Then in Section 3.2, we define a Bayesian inference method to capture spatial constraints between objects in a context based on demonstrations from a human. Lastly in Section 3.3, we present a hierarchical synthesis algorithm designed to place objects in a context satisfying the constraints learned by our inference method. This synthesis involves a high-level Monte Carlo Tree Search to explore promising object placements, and a lower level

mixed integer optimization to provide promising object placements given a partially completed context.

In Chapter 4, we present two evaluation tools for determining the viability of our methods in the intended workflows. In Section 4.1, we test the synthesis algorithms' runtime across (1) number of objects placed and (2) number of constraints per object class. Then in Section 4.2, we describe and perform a human pilot study to test our algorithm's ability to capture spatial constraints relevant to a human and whether the captured constraints are frequently un-specified by human explanations.

5.2 Future Work

There are three primary areas of future work. First, we want to augment the constraint inference algorithm to not require a set of pre-defined relational statements. Second, we hope to incorporate an active learning component into the algorithm pipeline. Lastly, we consider the possibility of learning constraint priors for inference in situations where our algorithm is given demonstrations for multiple different tasks.

5.2.1 Expanding Constraint Inference

The constraint inference algorithm presented in Section 3.2 requires a set of pre-defined relational statements $\mathcal{T}$ from which our algorithm infers the relevant set of constraints $\mathcal{C} \subset \mathcal{T}$. While a statement set $\mathcal{T}$ may be available in some applications, the assumption of it's availability presents a major restriction. Therefore, our next steps in this project involve inference methods not dependent on the pre-defined set $\mathcal{T}$.

5.2.2 Active Learning

One major advantage of inferring over logical specifications not leveraged in our methods is the ability to provide a set of verifiable rules to the user. While our inference algorithm captures a set of constraints, we have not yet formulated a strategy to

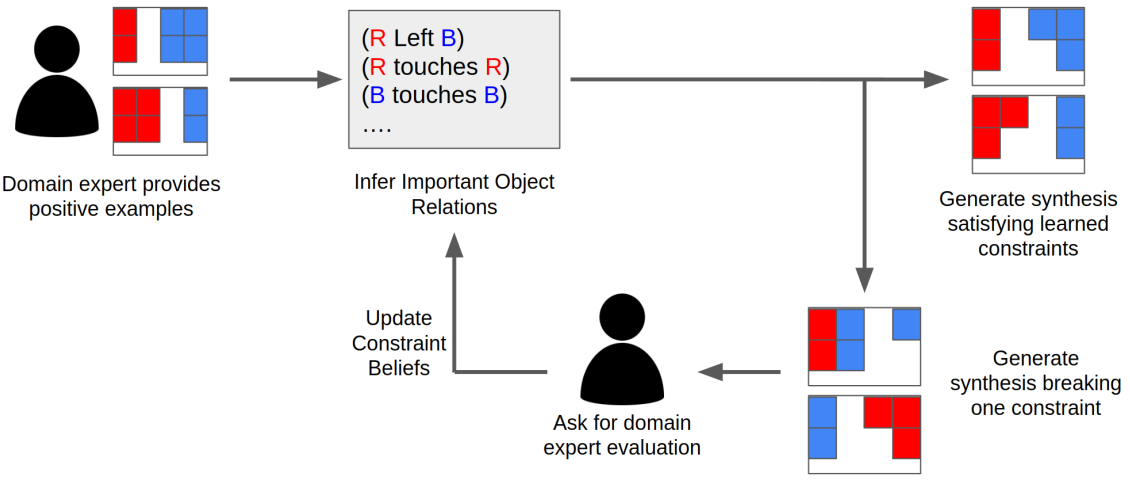

Figure 5-1: General pipeline of future work incorporating active learning as an inference component.

allow a human to check the learned constraints and approve or deny them. There are several ways to incorporate this type of verification into an algorithm pipeline. The simplest is to present the learned constraints to the user, and allow them to change or delete any that are not relevant. However, this strategy assumes the domain expert will be able to interpret these Boolean constraints, which is often not the case.

Therefore, we hope to incorporate a pipeline wherein the algorithm will intentionally break or ignore a subset of learned constraints during synthesis and ask for the user's approval. The human response can then be used to update the set of learned constraints. The pipeline for this active learning approach is shown in Figure 5-1.

5.2.3 Non-Unifrom Constraint Priors

In our current inference method, we assume a uniform prior over the probability that any relational statement will be in a demonstrator's ground truth constraint set (i.e. $p(\phi \notin \mathcal{C}_h)$). However, in reality, there are likely to be some relational statements that are more likely to be constraints across a wide variety of tasks. For example, in a box packing scenario, the relative positions of different object classes may vary from demonstrator to demonstrator, but rules about objects of the same class being in contact may remain consistent.

Therefore, if we imagine a situation in which a variety of people demonstrate a variety of tasks to the robot, the algorithm can learn which constraints show up over and over. It may be possible to utilize this knowledge by changing a statement's prior.

Bibliography

[1] Erfan Aasi, Mingyu Cai, Cristian Ioan Vasile, and Calin Belta. Time-incremental learning from data using temporal logics. *arXiv preprint arXiv:2112.14300*, 2021.

[2] Pulkit Agrawal. The task specification problem. In *Conference on Robot Learning*, pages 1745–1751. PMLR, 2022.

[3] M. Aiello and J. van Bentham. Logical patterns in space. *First CSLI Workshop on Visual Reasoning,*, 2000.

[4] Suhail Alsalehi, Erfan Aasi, Ron Weiss, and Calin Belta. Learning spatio-temporal specifications for dynamical systems. *arXiv preprint arXiv:2112.10714*, 2021.

[5] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. *arXiv preprint arXiv:1806.01946*, 2018.

[6] Mouaouia Cherif Bouzid and Said Salhi. Packing rectangles into a fixed size circular container: Constructive and metaheuristic search approaches. *European Journal of Operational Research*, 285(3):865–883, 2020.

[7] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

[8] Guillaume Maurice Jean-Bernard Chaslot Chaslot. *Monte-carlo tree search*, volume 24. Maastricht University, 2010.

[9] CS Chen, Shen-Ming Lee, and QS Shen. An analytical model for the container loading problem. *European Journal of operational research*, 80(1):68–76, 1995.

[10] Kevin Chen, Junshen K Chen, Jo Chuang, Marynel Vázquez, and Silvio Savarese. Topological planning with transformers for vision-and-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11276–11286, 2021.

[11] N Chernov, Yu Stoyan, and Tatiana Romanova. Mathematical model and efficient algorithms for object packing problem. *Computational Geometry*, 43(5):535–553, 2010.

[12] Edward G Coffman, Jr, Michael R Garey, David S Johnson, and Robert Endre Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[13] Anthony G Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997.

[14] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Algorithm aversion: people erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology: General*, 144(1):114, 2015.

[15] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Overcoming algorithm aversion: People will use imperfect algorithms if they can (even slightly) modify them. *Management Science*, 64(3):1155–1170, 2018.

[16] Kathryn A Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68(3):389–399, 1993.

[17] Stefan Edelkamp, Max Gath, and Moritz Rohde. Monte-carlo tree search for 3d packing with object orientation. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 285–296. Springer, 2014.

[18] Hillel J Einhorn. Accepting error to make less error. *Journal of personality assessment*, 50(3):387–395, 1986.

[19] Seda Erbayrak, Vildan Özkır, and U Mahir Yıldırım. Multi-objective 3d bin packing problem with load balance and product family concerns. *Computers & Industrial Engineering*, 159:107518, 2021.

[20] Giorgio Fasano. Cargo analytical integration in space engineering: a three-dimensional packing model. In *Operational Research in Industry*, pages 232–246. Springer, 1999.

[21] Ebru Aydin Gol, Ezio Bartocci, and Calin Belta. A formal methods approach to pattern synthesis in reaction diffusion systems. In *53rd IEEE Conference on Decision and Control*, pages 108–113. IEEE, 2014.

[22] Stephanie Gross, Brigitte Krenn, and Matthias Scheutz. Multi-modal referring expressions in human-human task descriptions and their implications for human-robot interaction. *Interaction Studies*, 17(2):180–210, 2016.

[23] Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Gros, and Calin Belta. Spatel: a novel spatial-temporal logic and its applications to networked systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 189–198, 2015.

[24] Michael Janner, Karthik Narasimhan, and Regina Barzilay. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 6:49–61, 2018.

[25] Wilson Kien Ho Ko, Yan Wu, Keng Peng Tee, and Jonas Buchli. Towards industrial robot learning from demonstration. In *Proceedings of the 3rd international conference on human-agent interaction*, pages 235–238, 2015.

[26] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1, 2006.

[27] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[28] Alap Kshirsagar, Hadas Kress-Gazit, and Guy Hoffman. Specifying and synthesizing human-robot handovers. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5930–5936. IEEE, 2019.

[29] Tengfei Li, Jing Liu, JieXiang Kang, Haiying Sun, Wei Yin, Xiaohong Chen, and Hui Wang. Stsl: A novel spatio-temporal specification language for cyber-physical systems. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 309–319. IEEE, 2020.

[30] Andrea Lodi, Silvano Martello, and Daniele Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357, 1999.

[31] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[32] Silvano Martello, Michele Monaci, and Daniele Vigo. An exact approach to the strip-packing problem. *INFORMS journal on Computing*, 15(3):310–319, 2003.

[33] Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations research*, 48(2):256–267, 2000.

[34] Silvano Martello and Daniele Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management science*, 44(3):388–399, 1998.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[36] Yash Vardhan Pant, Houssam Abbas, and Rahul Mangharam. Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1235–1240. IEEE, 2017.

[37] Célia Paquay, Michael Schyns, and Sabine Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23(1-2):187–213, 2016.

[38] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[39] Aniruddh Gopinath Puranic, Jyotirmoy V Deshmukh, and Stefanos Nikolaidis. Learning from demonstrations using signal temporal logic in stochastic and continuous domains. *IEEE Robotics and Automation Letters*, 6(4):6250–6257, 2021.

[40] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87. IEEE, 2014.

[41] Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton CT Lee, Mitchell P Marcus, and Hadas Kress-Gazit. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language. In *Robotics: Science and Systems*, volume 2, pages 2–1. Citeseer, 2013.

[42] David A Randell, Zhan Cui, and Anthony G Cohn. A spatial logic based on regions and connection. *KR*, 92:165–176, 1992.

[43] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[44] Lindsay Sanneman, Christopher Fourie, and Julie A Shah. The state of industrial robotics: Emerging technologies, challenges, and key research directions. *arXiv preprint arXiv:2010.14537*, 2020.

[45] Miguel Cezar Santoro and Felipe Kesrouani Lemos. Irregular packing: Milp model based on a polygonal enclosure. *Annals of Operations Research*, 235(1):693–707, 2015.

[46] Ankit Shah, Pritish Kamath, Julie A Shah, and Shen Li. Bayesian inference of temporal task specifications from demonstrations. *Advances in Neural Information Processing Systems*, 31, 2018.

[47] Ankit Shah, Shen Li, and Julie Shah. Planning with uncertain specifications (puns). *IEEE Robotics and Automation Letters*, 5(2):3414–3421, 2020.

[48] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[49] Jiankun Sun, Dennis J Zhang, Haoyuan Hu, and Jan A Van Mieghem. Predicting human discretion to adjust algorithmic prescription: A large-scale field experiment in warehouse operations. *Management Science*, 68(2):846–865, 2022.

[50] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[51] Sida I Wang, Percy Liang, and Christopher D Manning. Learning language games through interaction. *arXiv preprint arXiv:1606.02447*, 2016.

[52] Yanwei Wang, Nadia Figueroa, Shen Li, Ankit Shah, and Julie Shah. Temporal logic imitation: Learning plan-satisficing motion policies from demonstrations. *arXiv preprint arXiv:2206.04632*, 2022.

[53] Daniel S Weld and Johan De Kleer. *Readings in qualitative reasoning about physical systems*. Morgan Kaufmann, 2013.