Masters Thesis

A Hybrid Planning Approach To Robot Construction Problems

by

Faseeh Ahmad

Submitted to the Graduate School of Engineering and Natural Sciences in partial fulfilment of the requirements for the degree of Master of Science

Sabancı University

Faculty of Engineering and Natural Sciences

Mechatronics Engineering

January 2019

A Hybrid Planning Approach To
Robot Construction Problems

APPROVED BY

Assoc. Prof. Dr. Volkan Patoğlu ..... 
(Thesis Supervisor)

Assoc. Prof. Dr. Esra Erdem Patoğlu ..... 
(Thesis Co-Supervisor)

Assoc. Prof. Dr. Ahmet Onat ..... 

Assist. Prof. Dr. Bekir Bediz ..... 

Assist. Prof. Dr. Orkunt Sabuncu ..... 

DATE OF APPROVAL: 7/1/2019

Acknowledgements

I would like to express my sincere gratitude to my thesis advisors Assoc. Prof. Dr. Volkan Patoğlu and Assoc. Prof. Dr. Esra Erdem Patoğlu for their support, and guidance. Without their continuous assistance and encouragement, this work have not been possible. I am greatly thankful for all the opportunities they have provided me. I learned the process and principles of doing research. It was a great opportunity for me to work under such great professors.

Furthermore, I would like to extend my thanks to the lab members of Cognitive Robotics Lab for their precious feedback. I would also like to express my regards to the jury members for their time and feedback.

Finally, I must express my very profound gratitude to God Almighty, and my family for providing me with continuous support and encouragement throughout my years of study. I would like to specially thank my wife Momina Rizwan for her help and support. This accomplishment would not have been possible without them. Thank you.

ABSTRACT

A HYBRID PLANNING APPROACH TO
ROBOT CONSTRUCTION PROBLEMS

Faseeh Ahmad

Mechatronics Engineering, Master of Science, 2019

Thesis Supervisor: Assoc. Prof. Dr. Volkan Patoğlu
Thesis Co-Advisor: Assoc. Prof. Dr. Esra Erdem Patoğlu

Keywords: Robot construction, Hybrid planning, Answer set programming

We study robot construction problems where multiple autonomous robots rearrange prefabricated components to build stable structures. Robot construction problems can play a vital role in construction industries where the tasks such as designing a desired structure, planning for the necessary actions, and constructing structures from available components can be performed by the robots. Robotic construction may especially be useful in places, such as disaster zones or the space, where it is not safe or feasible for humans to visit. In these unsafe or hard-to-reach places, robots can build necessary buildings, bridges or shelters using the surrounding materials.

We view robot construction problems as planning problems: find a plan (i.e., a sequence of actions) to obtain a final stable configuration of prefabricated objects satisfying some goal conditions, from a given initial configuration. These problems are challenging from the perspective of task planning

since they may need incorporation of preexisting structure into the final design, pre-assembly of movable substructures, and use of extra blocks as temporary supports or counterweights during construction. These problems are challenging from the perspective of geometric reasoning as well, since they need feasibility checks to ensure reachability of a block, to avoid collisions of blocks, and to ensure stability of complex structures.

We propose a formal hybrid planning framework to address these challenges using Answer Set Programming, and state-of-the-art feasibility checkers. This framework not only decides for a stable final configuration of the structure, but also computes the order of manipulation tasks for multiple autonomous robots to build the structure from an initial configuration, while simultaneously ensuring the stability, supportedness and other desired properties of the partial construction at each step of the plan.

We show the usefulness of our approach on a wide variety of robot construction tasks, including bridge building and overhang construction scenarios, and using different types of objects, including cylindrical ones.

We demonstrate the applicability of our approach through dynamic simulations and physical implementations with a bi-manual Baxter robot.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Motivation

1.1.1 Construction Industry

Nowadays robots are widespread helping humans in solving problems from simple day to day chores to complex tasks. There are still some areas where robots are not being used or their employment is low compared to their potential. One such area is *construction industry* that relies on manual labor as the main source of its productivity. Bock [4] shows that robots can dramatically improve the speed and quality of construction tasks by automatically performing the mundane and rigorous tasks.

With the advancement in automation and robotics, the efficiency of some construction tasks would certainly improve but not all, because we use robots only for their heavy lifting abilities, not for deciding how to perform a particular task or what kind of actions are necessary to complete a certain task. Designing the structure to be built, planning the robot motions and sequence of actions are still performed by humans. If robots could also do all these tasks automatically, it would not only assist humans, but would have a major impact on construction industry.

1.1.2 Disaster Zones

Another motivation of robot construction originates from *disaster zones*. Normally, when an earthquake occurs in a remote area, there is destruc-

tion everywhere and in these conditions people urgently need some kind of shelter. Imagine a team of search and rescue robots, that arrives at the location, automatically designs a structure for being used as a shelter, then selects the necessary blocks from the rubble near destroyed buildings, and creates a stable shelter out of them. Such an application of construction can save time and possibly lives in disaster zones. Similarly, those robots may be able to construct bridges to provide safety passages for the people to leave that area.

1.1.3 Space Exploration

Space exploration is another potential application. Humans have started traveling to other planets in search of resources. It might be useful to use robots with the ability to automatically perform some construction tasks to build habitable structures before humans arrive. A good example would be Mars exploration, where a search and rescue robot arrives at Mars and automatically sets up a camp using the resources around.

1.2 Challenges

We view robot construction problems as planning problems: find a plan (i.e., a sequence of actions) to obtain a final stable configuration of prefabricated objects satisfying some goal conditions, from a given initial configuration. Robot construction problems possess various challenges, from the perspective of hybrid planning:

- Finding stable goal configurations of blocks, that satisfy the desired conditions of a final construction.
- Handling ramifications of robot's actions.
- Construction and incorporation of sub-assemblies.
- Using counterweights/scaffolding temporarily to maintain stability.

- Handling concurrency of actions.
- Maintaining stability of the structure at all times.
- Ensuring feasibility of robot's actions.

Below these challenges are explained in detail.

1.2.1 Sub-Assembly Construction

A sub-assembly comprises of two or more blocks/boxes being manipulated together. In Figure 1, a sub-assembly consisting of the blocks $L1$, $S1$ and $S2$ is being manipulated at time step 4, as the robot picks the block $L1$ and places it on top of block $S3$. As part of hybrid planning, it is challenging to decide for sub-assembly construction, and to ensure the stability of the structures. Note that it is also challenging to represent effects of manipulating a sub-assembly, due to ramifications.

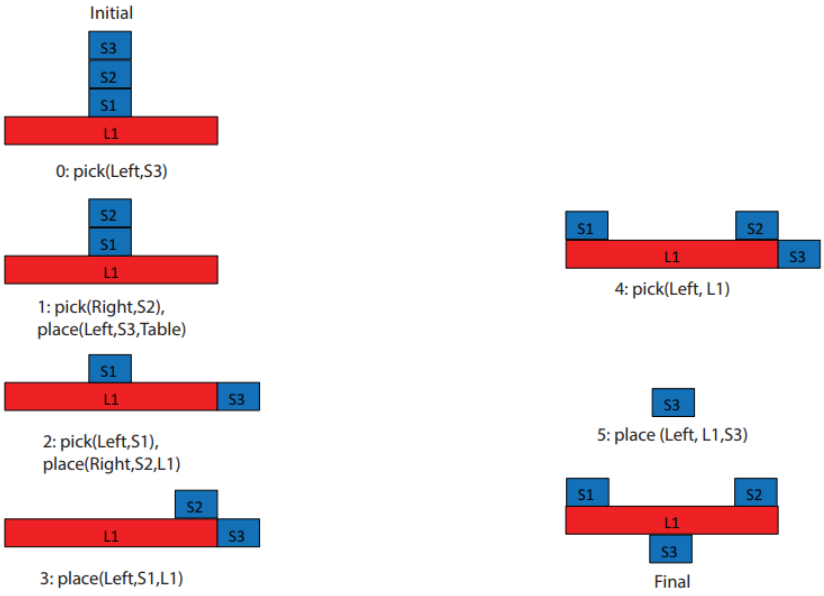


Figure 1: Sub-Assembly Manipulation

1.2.2 Temporary Counterweights

Counterweights may be required temporarily to balance the weight of the structure so that it remains stable during and at the end of the construction. In Figure 2, the block $M1$ is being used as a counterweight to balance the structure. So that the robot can place the blocks $S2$ and $S1$ on the ends of the block $L1$. Note that the counterweight (block $M1$) is not a part of the final configuration. It is used temporarily only. Therefore, deciding the use of counterweights as part of a hybrid plan is challenging.

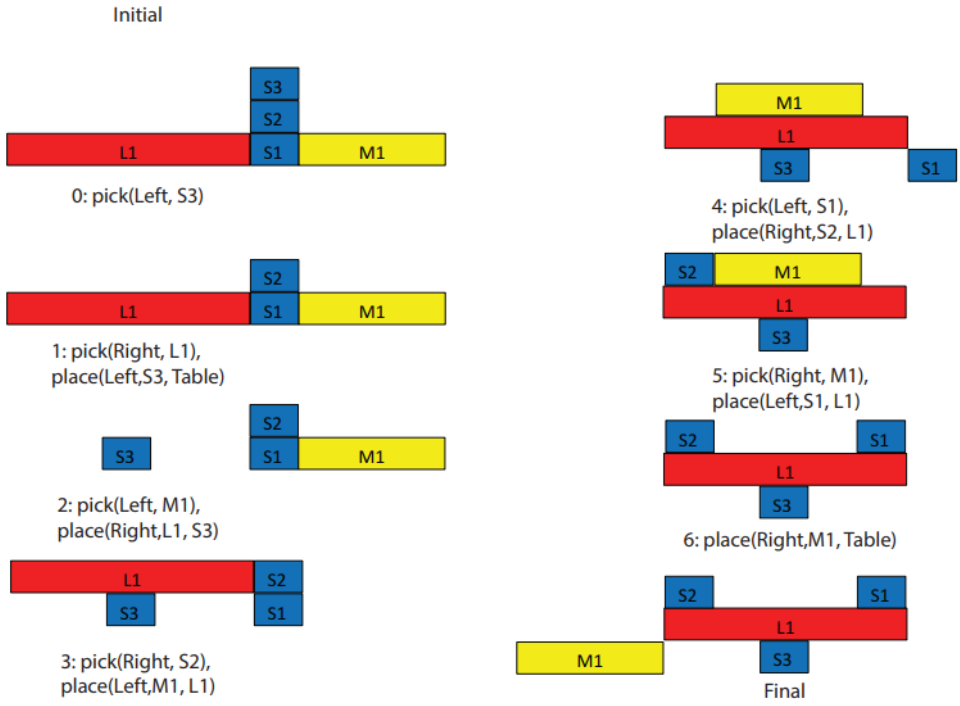


Figure 2: Counter Weights

1.2.3 Temporary Scaffolding

Similar to counterweights, scaffolding may be needed temporarily to complete the construction. In scaffolding, instead of supporting the structure from above by putting a heavy object, the structure is supported from below. In Figure 3, the block $S4$ is being used as a scaffold to support the structure below (time step 3 & 5). So that the robot can place the blocks $S1$ and $S2$ on top of the block $L1$, maintaining the stability. Deciding for temporary use of blocks as scaffolds is challenging.

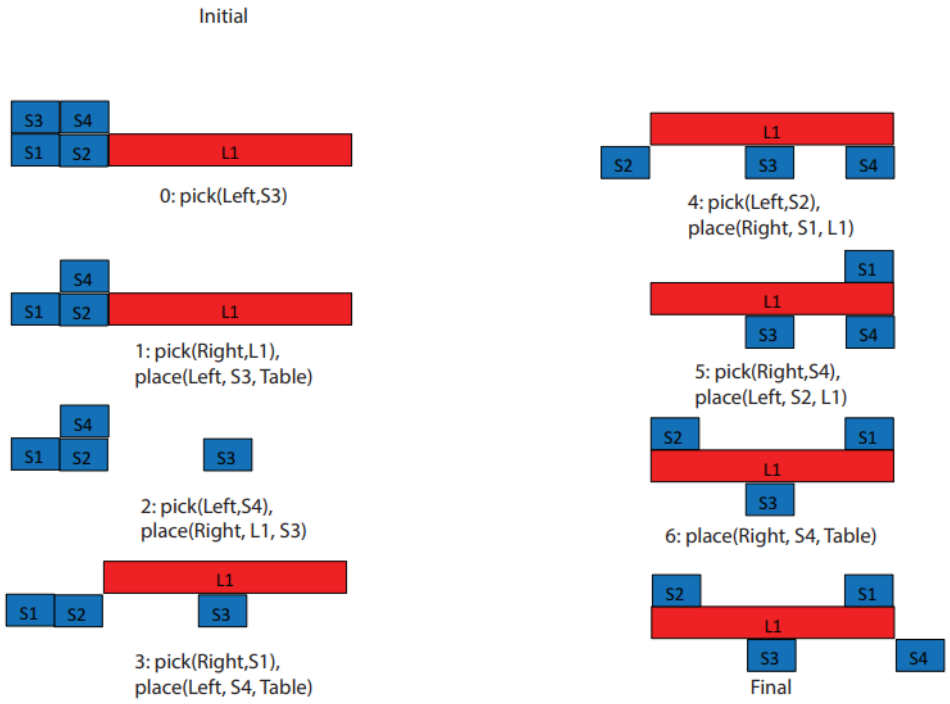


Figure 3: Scaffolding

1.2.4 Concurrency of Actions

For some robot construction problems multiple robots should be able to perform concurrent (or simultaneous) actions to achieve a task. In Figure 4, the blocks $S2$ and $S3$ are placed simultaneously on the block $L1$. Note that if they are not placed concurrently, the structure would lead to instability. Allowing true concurrency in planning is a challenging problem, from the perspectives of both representation and reasoning.

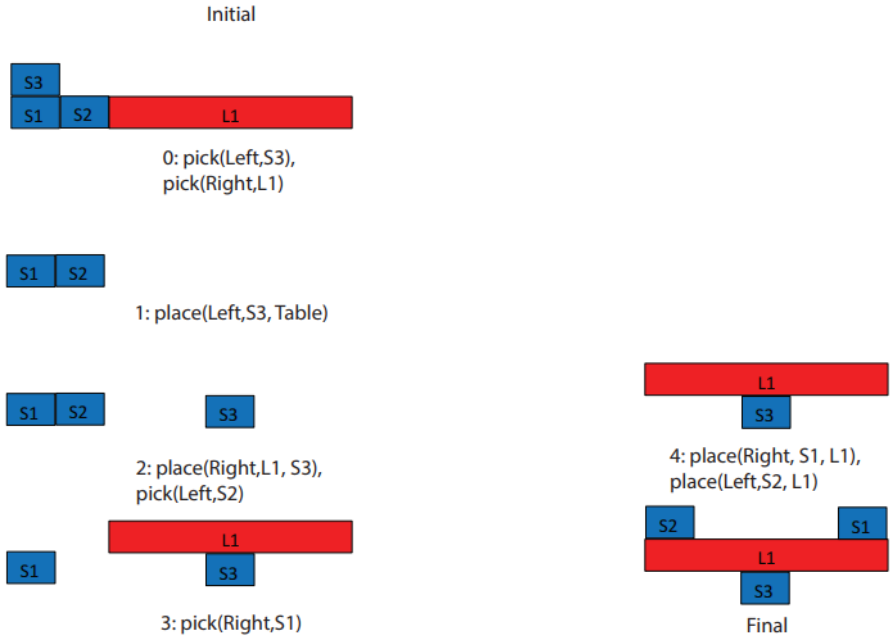


Figure 4: Concurrency of Actions

1.2.5 Finding a Stable Goal Configuration

There may be multiple goal configurations of blocks depending on the desired conditions about the final structure, and all of them may not be stable. In such cases, finding a stable goal configuration is a challenge itself. For instance, consider a construction task where the desired condition is to connect the both sides of a river to form a bridge. In such a bridge, as shown in Figure 5, all the counter blocks are properly placed to ensure a stable goal configuration. Finding such a stable goal state is important for planning. Note that stability checks are done in continuous space. Therefore, embedding these checks into discrete state constraints for planning is challenging.

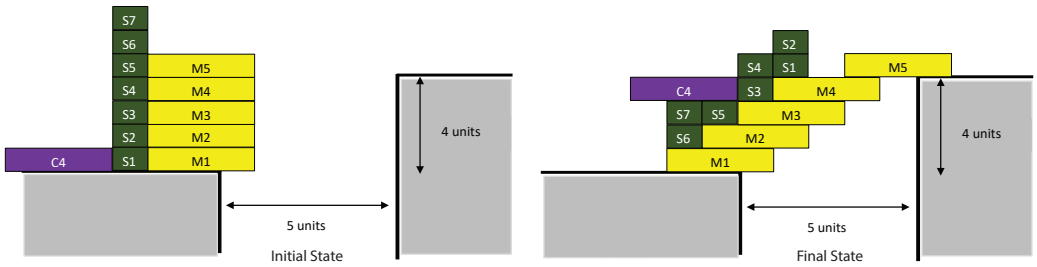


Figure 5: Stable Goal Configuration

1.2.6 Ramifications of Actions

Representing and reasoning about such indirect effect of actions or ramifications of actions have been challenging for planning. For instance, consider the construction task shown in Figure 6. As part of the plan block $C1$ is placed on block $C5$. As an indirect effect of this action block $C1$ becomes on top of block $M3$ as well.

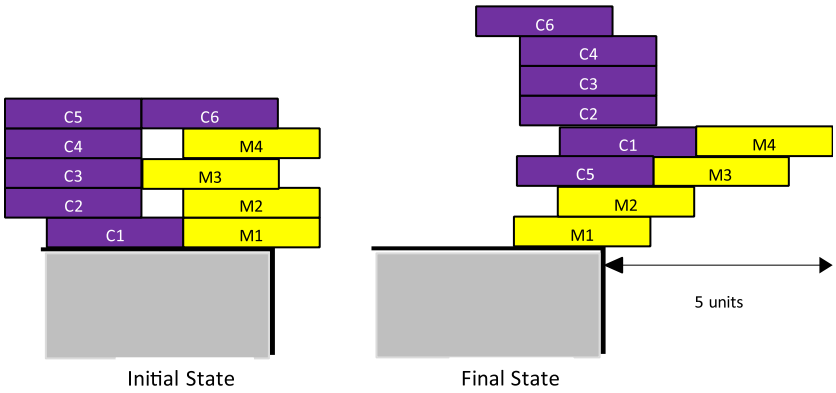


Figure 6: Ramifications of Actions

1.2.7 Embedding Feasibility Checks

Maintaining the stability of structure is of prime importance in construction tasks. The structure should be stable during all the steps of construction. Ensuring stability at all times is challenging from both planning and geometric point of view. For instance, consider Figure 7, stability is maintained by carefully balancing the weight distribution of the whole structure using counter blocks.

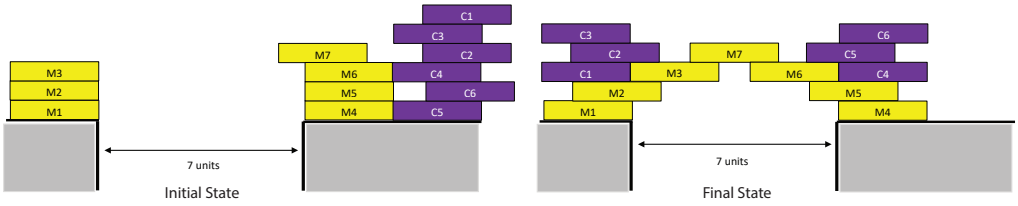


Figure 7: Determination of the order of actions

1.3 Our Contributions

We view robot construction problems as planning problems: find a plan (i.e., a sequence of actions) to obtain a final stable configuration of prefabricated objects satisfying some goal conditions, from a given initial configuration. These problems are challenging from the perspective of task planning since they may need incorporation of preexisting structure into the final design, pre-assembly of movable substructures, and use of extra blocks as temporary supports or counterweights during construction. These problems are challenging from the perspective of geometric reasoning as well, since they need feasibility checks to ensure reachability of a block, to avoid collisions of blocks, and to ensure stability of complex structures. We can summarize our contributions as follows:

- We have introduced a general formal planning framework for solving various robot construction problems from blocks with multiple robots, using Answer Set Programming (ASP). The framework proposes a hybrid planning approach by embedding feasibility checks (e.g., collision checks, stability checks) in logical formulas of ASP.

- We have extended our approach to solve construction problems that also involve cylindrical objects.

- We have designed and developed a collection of challenging construction benchmarks that include scenarios that necessitate sub-assembly manipulation, true concurrency of actions, scaffolding, use of counter-

weights, design of connected bridges and stable stack overhangs. Such a comprehensive collection of construction problems is useful for various studies, e.g., in robotics, planning, and knowledge representation and reasoning.

- We have implemented feasibility check algorithms to ensure stability of structures, and reachability of objects, utilizing state-of-the-art physics simulator PyBullet and the motion planner RRT*.

- We have illustrated the applicability of our hybrid approach by solving these benchmark problems, using the ASP solver DLVHEX with the feasibility checkers.

- We have illustrated the applicability of our hybrid approach in robotics by dynamic simulations of various scenarios of the benchmarks using a bi-manual Baxter robot. Furthermore, physical implementation of several scenarios have also been realized.

- We have also implemented an execution and monitoring algorithm to recover from failure during the execution of plan. The applicability of our algorithm has also been realized using physical implementation of several scenarios.

1.4 Thesis Outline

The organization of this document along with the summaries of upcoming chapters are provided below:

- In Chapter 2, related work has been presented. Various works related to the construction problems, such as blocks world, maximum overhang, image understanding, stability of assemblies, assembly planning and rearrangement planning have been discussed in detail.

- In Chapter 3, we formally describe the robot construction problems by discussing the input and output to the problem. Different types of goal conditions are discussed to further elaborate the problem.

- In Chapter 4, we provide the preliminaries to answer set programming (ASP) used to model robot construction problems.

- In Chapter 5, we discuss the modeling of construction problems using ASP. All the rules associated with the actions, preconditions of actions, fluents, ramifications, state constraints, concurrency constraints are provided in detail.

- Chapter 6 details the implementation of feasibility checks.

- Chapter 7 introduces the benchmarks scenarios for the robot construction problems. These benchmarks include scenarios that necessitate sub assembly manipulation, true concurrency of actions, scaffolding, counter weights, constructing bridges and overhangs.

- In Chapter 8, we introduce extension to our framework by introducing relevant ASP rules for cylindrical objects. Assumptions and constraints associated with the introduction of cylinders in the problem are discussed. Additional actions, preconditions, ramifications are provided in this chapter.

- Chapter 9 presents the solutions to all the benchmarks instances using our framework.
- Chapter 10 presents the implementation of dynamic and physical experiments.
- Chapter 11 concludes the thesis and elaborates on possible future research directions related to the construction problem.

Chapter 2

2 Related Work

To the best of authors' knowledge, this is the first robotic construction study that addresses a variety of multi-robot stack rearrangement planning problems for building stable structures of different sorts. In the literature, there exist several studies that focus on different specific aspects of the robotic construction task: deciding for the stability of a given structure (e.g., from an image obtained from Angry Birds), deciding for the existence of a specified stable structure (e.g., a maximum overhang) from a given set of identical blocks or an unspecified stack from a given set of different sizes of objects (e.g., like stones), planning for towers of identical blocks (e.g.. the blocks world) ignoring stability, etc. Let us go over them to better understand the challenges of the robot construction problems that we study.

2.1 The Blocks World

The well-known blocks world problems [78] have been widely studied by AI community [10]; it is proven to be NP-complete for polynomially bounded plans [23]. Blocks world problems are quite restricted compared to robot construction problems, since while proposing the problem, Winograd's interest was in language rather than in construction problems. For instance, the blocks world deals with identical blocks and allows a block to be placed on a flat surface or on another block, but not on multiple blocks as necessitated by the robot construction problems. It does not allow manipulation of sub assemblies, use of counterweights and scaffolds, or concurrent placements of

blocks, either. Also, there is no consideration of feasibility checks to ensure the stability of the stack at each step of a plan.

Later, Fahlman [17] has introduced a set of robot construction problems where the goal is for a robot to build specified structures out of simple blocks of different shapes and sizes. These problems allow incorporation of subassemblies into the final design, and the use of extra blocks as temporary supports or counterweights during construction; they also consider collisions of blocks and instability of the structures, but not motion planning. Since Fahlman's main interest was in maximizing common sense (rather than soundness, completeness or optimality), he implemented a planning system guided with heuristics to solve some of these problems. These problems have not been investigated with a formal approach since then.

2.2 Maximum Overhang Puzzle

Mathematicians and theoretical computer scientists have studied a classic puzzle that aims to determine the maximum overhang achievable by a stack of identical blocks [24, 50–52]. A relatively recent solution [50, 52] to this 150 year old puzzle, honored with the prestigious David P. Robbins Prize in mathematics, has introduced the use of blocks as counterbalance to improve upon the well-established solution. While the maximum overhang problem focuses on the determination of a stable and optimal final configuration of identical blocks, the planning aspects of the construction problem to attain the goal configuration is not considered within the scope of these studies.

2.3 Image Understanding and Qualitative Reasoning in Games

Applications in scene understanding from 2D pictures and computer games require inferring physical relations among objects [23, 29, 30, 57, 59]. Determination of stability of stacked objects and supportedness among objects

have been studied, commonly with qualitative reasoning approaches [70, 71]. Determination of stable final configuration of constructions has also been studied in computer games, like Angry Bird [7, 19, 20, 60]. These studies focus on the physical relations of a given final configuration and do not address the block rearrangement problem to build stable constructions.

2.4 Stability of Assemblies

In robotics, static stability [3, 38, 39, 41–43, 45, 55, 67, 74, 75, 79] and dynamic stability [49,58] of assemblies with and without friction have been thoroughly studied. The computational complexity of determining the assembly stability in 2D is established in [48]. The stability determination techniques have been utilized in several robotic applications, that include a Jenga playing robot [73], multiple robots building a ramp [46], an autonomous robot stacking a balancing vertical tower out of irregularly shaped stones [21], and a robot dry stacking irregular objects to build large piles [65]. Note that, in these studies, the challenging task planning aspect of construction planning has not been addressed. [40] focuses limit analysis of masonry brick-block system. This work formulates a mathematical programming problem with equilibrium constraints and proposes a solution by a sequence of linear mathematical programming problems.

Toussaint [69] has utilized stability checks for building some tallest stable tower from a set of unlabeled cylinders and blocks; no goal condition is specified. His method applies a restricted version of task planning to decide for the order of manipulation actions, based on simple Strips operators and Monte Carlo tree search, and considers a restricted form of stability check that depends on whether the objects are placed on support areas of other objects. Due to these restrictions, his method is limited to building towers with sequential plans.

Note that for sophisticated constructions that involve temporary scaffolding, counterweights, and sub assemblies, it is required to express ramifications

of actions as well as true concurrency. However, expressing ramifications directly by simple Strips operators is not possible [66] due to lack of logical inference. Also, expressing true concurrency is not possible unless the description is extended with exponential number of new operators, where each operator characterizes a concurrent action. Due to these theoretical results, other studies [15, 25, 68] that rely on simple Strips operators, do not present general methods for such sophisticated constructions either.

It is important to note that these methods do not cover sophisticated structures, like bridges or overhangs, since objects are not necessarily placed on support areas of other objects. Such sophisticated structures require definition of transitive closure to ensure supportedness or connectedness, but transitive closure is not definable in first-order logic [16] let alone Strips or PDDL.

2.5 Assembly Planning

In automated manufacturing, assembly plans aim to determine the proper order of assembly operations to build a coherent object. During assembly planning, the goal configuration is well-defined and the problem is generally approached by starting with the goal configuration and working backwards to disassemble all parts. Object stability has also been considered within this context [2, 5, 35, 44, 53, 54, 56, 72, 77]. The assembly planning problem is significantly different from the robotic construction problems: on the one hand, it allows assembly of irregular objects; on the other hand, the goal configuration is pre-determined and solutions are commonly restricted to monotone plans.

2.6 Rearrangement Planning

Geometric rearrangement with multiple movable objects and its variations (like navigation among movable obstacles [61, 62]) have been studied in lit-

erature. Since even a simplified variant with only one movable obstacle has been proved to be NP-hard [9,76], many studies introduce several important restrictions to the problem, like monotonicity of plans [1,8,11,33,34,47,63]. While a few can handle non monotone plans [28,32]; these studies do not allow stacking either. Recently, Han et al. [26] study rearrangement of objects in stack-like containers (by pushes and pops); these problems do not require stability checks.

2.7 Our Approach

Our approach is different from all the related works discussed. The differences have already been mentioned in the respective sub sections. We model robot construction problems as a planning problem where the goal is to achieve a final stable configuration; subject to some goal conditions; from an initial predefined configuration. The planner asks for a sequence of actions that can take the initial configuration to goal configuration. During planning the stability of all the states (intermediate and final) are ensured using stability checks. We introduce a general framework to solve robot construction problems using answer set programming.

Chapter 3

3 Description of Robot Construction Problems

Robot construction problem can be defined as a planning problem that asks for a final stable configuration of different types of prefabricated blocks stacked on each other that satisfy some goal conditions, and a feasible stack rearrangement plan to obtain that final configuration from a specified initial configuration of the blocks. Formally, it can in terms of input and output defined as:

Input: Initial configuration of blocks.

Output: A feasible stack rearrangement plan to obtain a stable final configuration satisfying some goal conditions from initial configuration of blocks.

Goal conditions can vary depending on the problem at hand. It can be defined in such that a particular block is on top of another block or it can be defined with more detail by specifying the actual goal location in terms of units e.g. block 1 should be on block 2 at unit space 3.

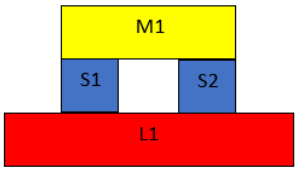


Figure 8: Initially all the blocks are on the table. Goal condition is expressed by specifying the relationship between relative blocks.

Figure 8 shows final state of 4 blocks. Initially all the blocks are on the table. Here the goal condition is defined by saying that $L1$ should be on $Table$, $S1$ and $S2$ should be on $L1$, $M1$ should be on both $S1$ and $S2$. Goal can also defined by considering the physical properties of the block e.g. small block contains something delicate so it should be placed on top of a medium block which is heavier so that the medium block does not damage the thing inside the small block.

Goal can also be defined in an abstract fashion such as maximizing the height of a block, maximizing the overhang or joining two sides of a river. For instance consider the **overhang** problem were the goal is to maximize the distance a block from the edge of the table [24, 50–52]. Figure 9 shows the initial and final state in an overhang problem. Initially all the blocks are stacked on the table and in the final state an overhang of 4 units is achieved.

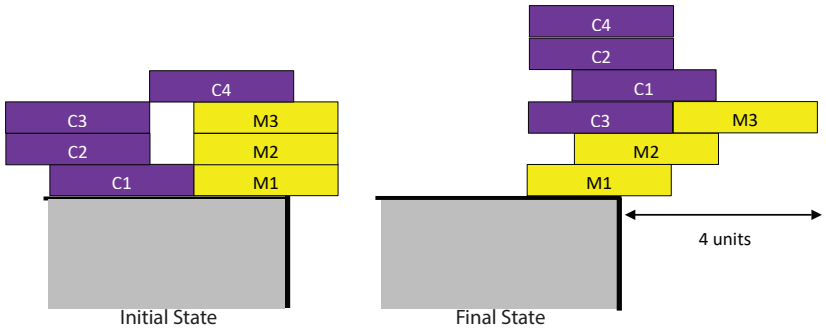


Figure 9: *Purple* blocks are heavier and are used as counter weights, while *yellow* blocks are the main blocks used to maximize the overhang. Overhang of 4 units is achieved.

Alternatively, a set of disconnected surfaces, similar to the banks or sides of a river can be given and as a goal both of these sides should be connected so that in the final state a bridge like structure is obtained. Figure 10 shows **bridge** like problem. Initially there are blocks on both sides of the river and in the final state a stable bridge is constructed so that both sides are connected.

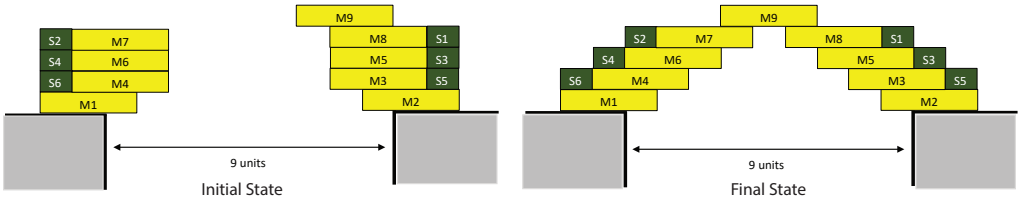


Figure 10: *Green* blocks are heavier and are used as counter weights, while *yellow* blocks are used to connect the sides of the river. The distance between the sides of the bridges is 9 units.

In some cases it is even possible, one side of the river is at some height like a **asymmetric bridge**. Figure 11 shows a scenario in which the right side of the river is 4 units higher than the left side. Initially all the blocks are on the left side of the river and in the final state the a stable stair like structure is obtained.

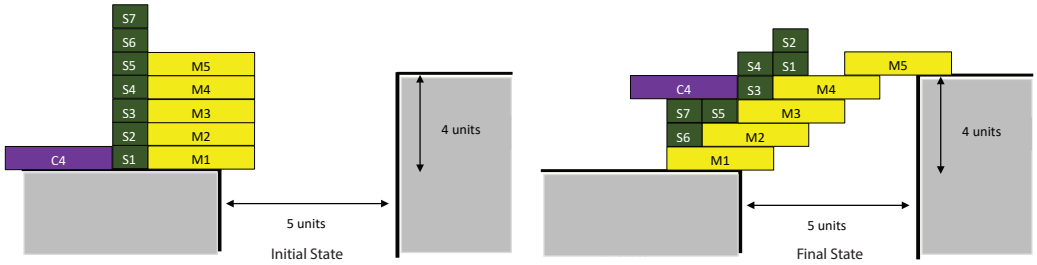


Figure 11: *Green* and *purple*blocks are heavier and are used as counter weights, while *yellow* blocks are used to connect the sides of the river. The distance between the sides of the bridges is 5 units and the height difference is 4 units.

Goal can also be specified to maximize or minimize the height of a particular block in **tower** formation. Figure 12 shows a tower like formation with 8 blocks where the goal was the maximize the height of $S3$. In figure 13 the goal was to minimize the height of $S3$. In both cases it is a condition that $S3$ should be part of the stack.

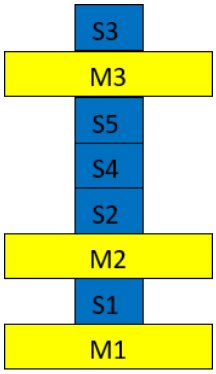


Figure 12: Maximizing the height of box s3 in a stack of 8 boxes

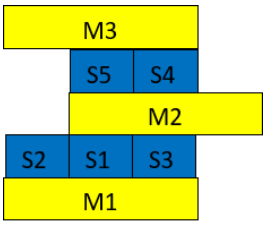


Figure 13: Minimizing the height of box s3 in a stack of 8 boxes

3.1 Assumptions

We use a hybrid planning approach to solve robot construction problems that involves planning an action sequence on a discrete level. Solving a continuous problem in discrete domain always requires some assumptions to solve the problem properly. Assumptions listed below enables the problem to be formally modeled.

- We consider a discrete model of robot construction problem where space on a box/surface or location of box is expressed in terms of unit space.
- A single unit space is set to be equal to the size of the smallest box.
- Without the loss of generality, we consider three size of boxes/blocks: *small* block occupies or has one unit space, *medium* block occupies or has three unit spaces and *large* block occupies or has five unit spaces.

- Width and height of all the boxes are assumed to be the same, but their weights may vary.
- There can be more than one robot with multiple manipulators.
- Orientation of the blocks remains the same during the planning.
- Weight distribution of blocks can be arbitrary.
- Without the loss of generality, we consider stability check as the only feasibility check; however, other checks, such as graspability, reachability can easily be added into our framework.

Chapter 4

4 Answer Set Programming (ASP)

ASP is a declarative paradigm to solve computationally complex problems especially NP hard problems [6, 22, 36, 37]. As robot construction problems are more harder than blocks world problem; which itself is an NP-Hard problem [23]; due to the challenges mentioned in chapter II, ASP can be used to solve these problems.

The basic idea is to model the hybrid domain into a set of logical rules called the ASP program P, whose model called answer set which satisfies all the rules in program P corresponds to the plan of the given problem instance. These answer sets can be computed by answer set solvers like, DLVHEX [12] and calls to relevant feasibility checkers can be made if necessary.

4.1 Input Language

We consider disjunctive ASP rules of the form:

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, not\ \beta_{n+1}, \ldots, not\ \beta_m$$

where $m, k \geq 0$, each α_i is an atom, and each β_i is an atom or an external atom. Intuitively, a rule expresses that if all β_i $(1 \leq i \leq n)$ holds but no β_i $(n+1 \leq i \leq m)$ holds then some α_i $(1 \leq i \leq k)$ holds as well. When $k = 0$, the rule is a constraint; when $n = m = 0$, it is a fact.

4.2 Negation

There are two types of negations in ASP:

Classical Negation Classical negation or strong negation of an atom holds only if it can be derived. Consider an example $-A$. The rule states that we know that the atom A does not hold. This kind of negation is expressed by connective $-$.

Default Negation Default negation or negation by failure means that if we cannot show the truth of an atom, it is assumed to be false. It is expressed by connective *not*. Consider an example $notA$. Here $notA$ is assumed to be true unless the atom A is derived to be true. We do not know if A holds.

4.3 Constraints

There are two types of constraints in ASP:

Hard Constraints The rules of the form

$$\leftarrow \beta_1, \ldots, \beta_n, not\ \beta_{n+1}, \ldots, not\ \beta_m$$

are known has hard constraints and intutively means that the *body* must not hold.

Weak Constraints The rules of the form

$$\leftarrow \beta_1, \ldots, \beta_n.\ [w@l, t_1, \ldots, t_n]$$

are known as weak constraints. Here w is weight and l is priority. Weak constraints are used to define preferences. Intutively they mean that it is preferred that the *body* should not hold but if it holds there is a cost w.

l is optional and is used to define priorities between more than one weak constraints.

4.4 Aggregates

Aggregates are used to express properties on specific set of elements. They are done by the following rules:

$$s_1 \prec_1 \alpha\{t_1 \ldots, t_n : L_1, \ldots, L_m\} \prec_2 s_2$$

Here t_i are terms, L_i are literals, α evaluates the numerical value of the aggregate function and $\prec_i$ are comparison predicates that compare the value with terms s_i. Some of the most common supoorted aggregate functions are $\#count$, $\#max$, $\#sum$ etc.

4.5 External Atom

An external atom:

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m)$$

is defined by its name g, input $Y_1, \ldots, Y_n$ and output $X_1, \ldots, X_m$. Intuitively, g takes the input $Y_1, \ldots, Y_n$, passes it to an external computation (like a stability checker), and conveys the results $X_1, \ldots, X_m$ into the rules.

Chapter 5

5 Modelling Robot Construction Problems

We use ASP to model robot construction problems. Our formal ASP description of the preconditions and effects of robot's pick and place actions, and the integration of reachability checks into preconditions of manipulation actions follow the guidelines described by Erdem et al. [13, 14] for hybrid planning problems. Let us explain, in particular, how the further challenges of robot construction problems are addressed using ASP. Details of modelling is provided as follows:

5.1 Fluents

We have two fluents in our domain description that represent the world state:

holding(a,b,t): robot's gripper a is holding box b at step t of the plan.

on(b,l,u,v,t): box b is at location l at time step t, in such a way that the unit space v of b is on the unit space u on l.

5.2 Actions

We consider two actions which effect the world state:

pick(a,b,t): pick the box b with the gripper a at step t.

placeOn(a,b,l,u,v,t): place the box b being held by the gripper a such that the unit space v of b is on the unit space u of l.

5.2.1 Pick action

Either a *pick* action can occur or it may not occur. This is defined by the occurrence of *pick* action using disjunctive rules.

Occurrence of the pick action is defined as:

$$pick(a, b, t) \vee \neg pick(a, b, t) \leftarrow$$

The direct effect of *pick* action is that the object is being held the gripper at next time step.

Direct effect of pick action is defined as:

$$holding(a, b, t + 1) \leftarrow pick(a, b, t)$$

Preconditions of *pick* action are as following:

- A robot cannot pick a box b with its gripper a if it is already holding some box b' with it:

$$\leftarrow pick(a, b, t), holding(a, b', t)$$

- A robot cannot pick a box b with its gripper a if it is already holding the box b with another gripper:

$$\leftarrow pick(a, b, t), holding(a', b, t)$$

- A robot cannot place a box b with gripper a at location l which is not

supported by table

$$\leftarrow place(a, b, t), not\ supported(b, Table, t).$$

5.2.2 Place action

Either a *place* action can occur or it may not occur. This is defined by the occurrence of *place* action using disjunctive rules.

Occurrence of the place action is defined as:

$$placeOn(a, b, l, u, v, t) \vee \neg placeOn(a, b, l, u, v, t) \leftarrow$$

The direct effect of *place* action is that the object is *on* another location at next time step.

Direct effect of pick action is defined as:

$$on(b, l, u, v, t + 1) \leftarrow placeOn(a, b, l, u, v, t), holding(a, b, t)$$

Preconditions of *place* action are as following:

- A robot cannot place a box on location l if its gripper is empty:

  $$\leftarrow place(a, l, t), \#count\{b' : box(b'), holding(a, b', t)\} = 0$$

  where *place* is obtained from *placeOn* by projection:

  $$place(a, l, t) \leftarrow placeOn(a, b, l, u, v, t)$$

- A robot cannot place a box b with its gripper a if the target box is already being held:

  $$\leftarrow place(a, b, t), holding(a', b, t)$$

- A robot cannot pick a box b with gripper a if it is not supported by table

$$\leftarrow pick(a, b, t), not\ supported(b, Table, t).$$

5.3 Ramifications

There a lot of interesting ramifications or indirect effects of actions in robot construction problems. Following are the necessary ramifications:

- If a box b is placed on some location l, then as a direct effect of this action b becomes on l; as an indirect effect, the robot's gripper becomes empty:

$$\neg holding(a, b, t) \leftarrow onAux(b, l, t).$$

- If the unit space v of box b is on the unit space u of location l, then (b, v) is not on any other unit (l', u'):

$$\neg on(b, l', u', v, t) \leftarrow on(b, l, u, v, t)$$

- If a robot's gripper a picks a box b, then as its direct effect a is holding b; as an indirect effect, b is not on any box or the table:

$$\neg on(b, l, u, v, t) \leftarrow holding(a, b, t)$$

- If a robot's gripper a picks a box b, then as its direct effect a is holding b; as an indirect effect, the gripper a is not holding any other box b' $(b \neq b')$,

$$\neg holding(a, b', t) \leftarrow holding(a, b, t)$$

- If a robot's gripper a picks a box b, then as its direct effect a is holding b; no other gripper a' is holding b $(a \neq a')$.

$$\neg holding(a', b, t) \leftarrow holding(a, b, t)$$

An interesting ramification occurs when a longer box b is placed on top of another box: after the robot places the box b being held its gripper a onto the location l, so that unit space v of b is placed on which unit u of l, as an indirect effect the box b occupies as many unit spaces as its size allows on l. We represent the ramifications of placing a longer box b is on top of another box l as follows. Suppose that b occupies the right part of l, starting from the unit space u of l (Figure 14). This can be expressed by the following rule:

$$on(b, l, u + i, v + i, t) \leftarrow on(b, l, u, v, t)$$

where i ranges between 1 and $min\{size(b) - v, size(l) - u\}$. Similarly, a rule is added for the left part of l being occupied by b.

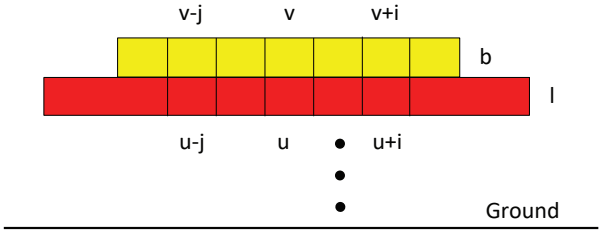


Figure 14: As indirect effects of placing unit v of b on unit u of l, unit $v + i$ (resp. $v - j$) of b is on unit $u + i$ (resp. $u - j$) of l.

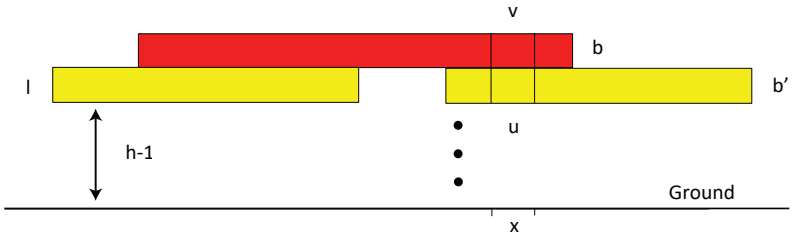


Figure 15: As indirect effects of placing b on l, unit v of b becomes on unit u of b', and unit $v + i$ (resp. $v - j$) of b becomes on unit $u + i$ (resp. $u - j$) of b'.

Another interesting ramification occurs when a longer box b is placed on top of another box, but as a ramification it is also placed on a neighboring box b' that is not too far (Figure 15). Such a sophisticated ramification is represented as follows. First, we introduce some auxiliary atoms of the

form $above(h, b, v, x, t)$ to define the global locations of the boxes on the table, taking the leftmost side of the table as a reference; $above(h, b, v, x, t)$ expresses that the unit space v of the box b at time step t is at a global location that is x units to the right of the leftmost side of the table and at h units high from the surface of the table. This predicate is defined with double recursion. The first recursion defines the global location of unit v of a box b vertically within a tower that is located x units to the right of the leftmost side of the table:

$$\begin{aligned} &above(1, b, v, x, t) \leftarrow on(b, \mathit{Table}, x, v, t) \\ &above(h, b, v, x, t) \leftarrow above(h{-}1, b', u, x, t), \\ &\quad on(b, b', u, v, t). \end{aligned} \tag{1}$$

The second recursion defines the locations of other units of box b horizontally to the right and to the left of that tower:

$$\begin{aligned} &above(h, b, v{+}1, x{+}1, t) \leftarrow \\ &\quad above(h, b, v, x, t) \quad (v{<}size(b)) \\ &above(h, b, v{-}1, x{-}1, t) \leftarrow \\ &\quad above(h, b, v, x, t) \quad (v{>}1) \end{aligned} \tag{2}$$

The rule below is used to decide if a neighboring box is on another box or not:

$$on(b, b', u, v, t) \leftarrow above(h, b, v, x, t), above(h{-}1, b', u, x, t)$$

5.4 State Constraints

There are some uniqueness constraints associated with the boxes in our domain description:

- There can only one box at a particular unit of a location:

$$\leftarrow \#count\{b : box(b), on(b, l, u, , t)\} > 1$$

- A robot should not be holding the same box with both arms:

$$\leftarrow \#count\{a : arm(a), holding(a, b, t)\} > 1$$

5.5 Concurrency Constraints

Unless specified otherwise, the ASP modeling of the construction problem allows concurrent actions

- The concurrency of two pick actions of the same box but with different grippers is not allowed with the following formula:

$$\leftarrow \#count\{a : arm(a), pick(a, b, t)\} > 1$$

- The concurrency of two pick actions of different boxes with the same gripper is not allowed with the following formula:

$$\leftarrow \#count\{b : box(b), pick(a, b, t)\} > 1$$

- The concurrency of two place actions with the same gripper is not allowed with the following formula:

$$\leftarrow \#count\{l : objloc(l), place(a, l, t)\} > 1$$

- The concurrency of two place actions at the same location is not allowed with the following formula:

$$\leftarrow \#count\{a : arm(a), place(a, l, t)\} > 1$$

- A box b cannot be picked by a gripper a while another gripper a' $(a \neq a')$ is placing a box on it:

$$\leftarrow pick(a, b, t), place(a', b, t)$$

5.6 Supportedness Constraints

At any state of the world, no box is supported by itself (i.e., no circular configurations). For that, we recursively define supportedness by the table:

$$\begin{aligned} supported(b, l, t) &\leftarrow onAux(b, l, t) \\ supported(b, l, t) &\leftarrow onAux(b, l', t), \\ &\quad supported(l', l, t) \quad (b{\neq}l') \end{aligned}$$

where $onAux$ is obtained from on by projection:

$$onAux(b, l, t) \leftarrow on(b, l, u, v, t)$$

After that, we add a constraint to ensure that no box b is supported by itself:

$$\leftarrow supported(b, b, t)$$

5.7 The Commonsense of Law of Inertia

We represent the commonsense law of inertia to address the frame problem, by the following rules:

- For *holding* fluent

$$holding(a, b, t+1) \leftarrow holding(a, b, t), not\ \neg holding(a, b, t+1)$$
$$\neg holding(a, b, t+1) \leftarrow \neg holding(a, b, t), not\ holding(a, b, t+1)$$

- For *on* fluent

$$on(b, l, u, v, t+1) \leftarrow on(b, l, u, v, t), not\ \neg on(b, l, u, v, t+1)$$
$$\neg on(b, l, u, v, t+1) \leftarrow \neg on(b, l, u, v, t), not\ on(b, l, u, v, t+1)$$

5.8 Integrating Stability Check

As discussed earlier, we use stability checks as the only feasibility check for the problem but other checks such as reachability, graspability etc can be easily added to our framework. Stability checks are quite important in robot construction problems because they link the discrete model of the problem with continuous domain calculations. Integration of stability check is done with the help of external atom construct of ASP by the following rule:

$$\leftarrow not\ \&stable[on, t]()$$

The external atom takes *on* predicate and time step t as its input.

A similar stability check is also applied for sub-assembly manipulation which makes sure that the sub-assembly being manipulated is stable or not. This is done by the following rule:

$$\leftarrow holding(a, b, t), onAux(b', b, t),$$
$$not\ \&hStable[holding, on, t]().$$

The important thing to note here is that these feasibility checks are applied as state constraints, not as preconditions to actions which makes sure that our structure or sub-assembly is going to be stable at every time step.

5.9 Bridge Construction

In bridge construction scenarios, one of the required conditions about a final structure is that there exists a block x on the left side of the bridge and another block y on the right hand side of the bridge such that x and y are connected to each other. For this reason, we recursively define connectedness of blocks using an auxiliary atom of the form $connected(x, y, t)$ (block x is supported by block y, or vice versa) similar to the recursive definitions that we have seen above, and add a constraint to express the required condition above for the goal (i.e., last time T):

$$\begin{aligned} \leftarrow \#count\{x, y : connected(x, y, T), \\ side(x, Left, T), side(y, Right, T)\} = 0. \end{aligned}$$

where *side* is defined as:

$$side(x, Left, t) \leftarrow on(x, Table, u, \_, t), leftsideunits(u)$$
$$side(x, Right, t) \leftarrow on(x, Table, u, \_, t), rightsideunits(u)$$

In order to define *connectedness*, we need to define a bidirectional connectedness graph between boxes as shown in the figure below. This is done by the following rules:

$$connected(x, y, t) \leftarrow supported(x, y, t)$$
$$connected(x, y, t) \leftarrow connected(y, x, t)$$
$$connected(x, y, t) \leftarrow connected(x, z, t), connected(z, y, t)$$

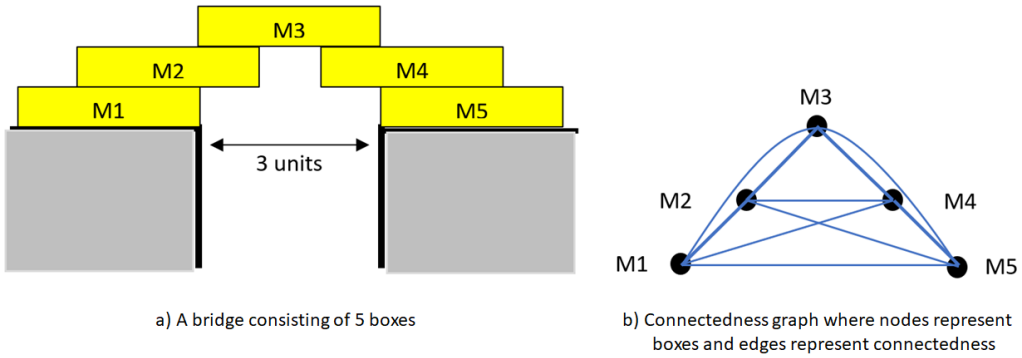


Figure 16: Connectedness Graph

5.10 Asymmetric Bridge Construction

In order to solve scenarios involving asymmetric bridges, heights of both sides are defined separately by the following rule:

$$
\begin{aligned}
&above(H, Left, x, x, t) \leftarrow \\
&above(H', Right, x, x, t) \leftarrow \\
&above(H+1, b, v, x, t) \leftarrow on(b, Left, x, v, t) \\
&above(H'+1, b, v, x, t) \leftarrow on(b, Right, x, v, t)
\end{aligned}
$$

Here H and H' are constants representing height of $Left$ and $Right$ side of the bridge.

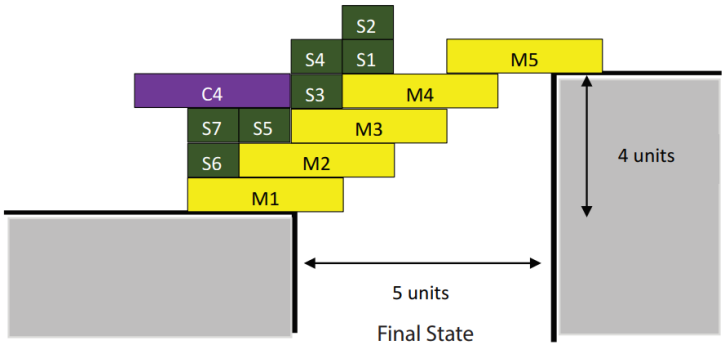


Figure 17: The difference between the height of bridges is 4 units

5.11 Overhang Construction

In overhang scenarios, one of the required conditions about a final structure is that there exists a block x supported by *table* and the difference between the maximum overhang z and size of *table* is equal to the global unit space x of unit space v of box b.

$$\leftarrow \#count\{b : supported(b, Table, t), above(\_, b, v, x, t),$$
$$x = z - size(Table), overhang(z)\} = 0.$$

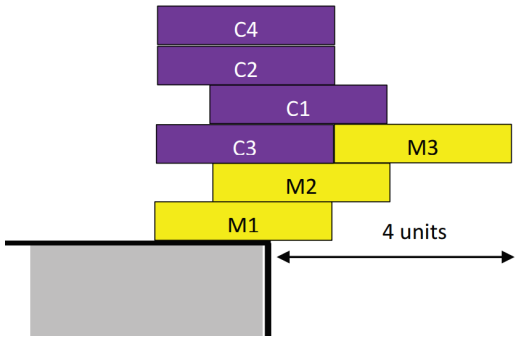


Figure 18: Overhang of 4 units

Chapter 6

6 Implementation of Feasibility Checks

In this chapter, implementation of feasibility checks will be discussed. As discussed earlier, ASP provides a formal framework to solve computationally complex problems. It allows us to solve planning problems with a defined initial and goal configurations. These problems are normally modeled in discrete domain using ASP formulation. Some of these problems require continuous domain calculations to reason about them in the discrete domain. The robot construction problem is such a problem. In order to determine the stability of the structure, continuous domain stability check needs to be done to ensure the stability. Similarly, reachability check that allow us to find out if a particular object is reachable or not, needs to be done in the continuous domain. In order to perform these continuous domain checks, ASP provides external atom construct to allow these calculations. An external atom is defined as:

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m)$$

where g is the name of feasibility check, inputs are defined by $Y_1, \ldots, Y_n$ and outputs are defined by $X_1, \ldots, X_m$. Intuitively, g takes the input $Y_1, \ldots, Y_n$, passes it to an external computation (like a stability checker), and conveys the results $X_1, \ldots, X_m$ into the rules. Let us consider stability check as an example:

6.1 Stability Check

Stability check in our domain description is defined as:

$$\leftarrow not\ \&stable[on, t]()$$

Here the name of the check is *stable* and it takes the *on* predicate and time step t as input and based on this information decides if the structure is stable or not. *on* predicate holds the state information of all the boxes/blocks and t indicates the current time step. This *stable* external atom is implemented as a Python plugin. In the Python implementation there is a function with the same name *stable* and the same inputs. In this Python function some pre-computation is done to convert the incoming data from ASP into such a format that it can be passed to a physics simulator to test if the current state is stable or not. After pre-computation, the data representing the current state is passed to a physics simulator.

We use Pybullet 2.4.1 for dynamic simulation. Pybullet physics simulator takes the input state and generates the state accordingly. After generating the structure, it is tested for some time under real physics parameters, such as gravity and disturbance forces. All the physical parameters such as size, shape, gravity, disturbance forces, time of simulation, step time are decided before hand and can be changed if needed. During the simulation displacements of all the boxes or blocks are measured along 2-dimensions (no depth) and if the displacements cross an empirically measured threshold, the structure is considered unstable, otherwise stable. This information regarding stability is passed to the ASP program which uses it to reason more intelligently.

The important thing note here is that we use stability check as a state constraint not as a precondition to some action. The reason behind this is to ensure that only those states are allowed in which stability is guaranteed.

We also have a similar check to ensure the stability of sub-assemblies and

it is defined as:

$$\leftarrow holding(a, b, t), onAux(b', b, t),$$
$$not\ \&hStable[holding, on, t]().$$

In this check we pass an additional input *holding* which gives the information of boxes in hand and *onAux* predicate ensures that the check runs for sub assemblies.

6.2 Reachability Check

Reachability check ensures if a particular box is reachable by a particular robot gripper without colliding with any of the other boxes. Since the other boxes are movable obstacles, such a reachability check is challenging.

We implement reachability checks in Python. Our algorithm first converts the discrete high-level information of objects stored in 'on' predicate into location of object in continuous domain. Since, in our case we already have discretization of table in terms of units at high level. We utilize this information to find location of boxes. A small box occupies one unit space, so we use this as the basic unit of our grid and assign locations to boxes according to the units occupied. Then, the algorithm finds inverse kinematic solutions from the end-effector of the robot to possible grip locations of the box under consideration. If a solution exists, then the box is reachable; otherwise, it is not. Next, our algorithm employs a motion planner to find a collision-free trajectory from the end-effector to a reachable grip location on the box, to make sure that there is no collision. We use the motion planner based on an asymptotically optimal variant of Rapidly-exploring Random Trees (RRT*) [31] from OMPL library [64].

This check facilitates the overall system in two ways. 1) It determines which objects are reachable from which particular arm. There might be some objects which are reachable through one arm only. 2) It also makes sure that while reaching an object the arm is not going to collide with any obstacle.

We embed these reachability checks into the formal model of previous section, by formalizing a precondition of the pick action that prevents the occurrence of the action in a plan if the box is not reachable:

$$\leftarrow pick(a, b, t), not\&reachable[on, a, b]()$$

Note that here the external atom $\&reachable[on, a, b]()$ gets as input a robotic gripper a (left one or right one), a box b (to be picked), and the extent of the predicate on, that describes the current configuration of the boxes (i.e., which box is on top of which box(es), and which box is on the table). Place action involves a similar check.

Chapter 7

7 Benchmark Scenarios

In this section, we propose benchmark scenarios associated with robot construction problems. Some of these benchmarks were introduced by Fahlman [17] where the goal of the robot is construct some specified structures from a given initial configuration. We extend these scenarios to present a comprehensive set of benchmarks.

7.1 Sub-assembly Manipulation

Incorporating pre-existing structures is an important challenge in construction tasks. Moving a sub assembly means moving a group of objects all together. This involves sophisticated ramifications which allows the objects on top of the object picked to be moved together. In Figure 19, there are pre-existing stable structures already available and they need to be placed at appropriate locations so that structure does not fall.

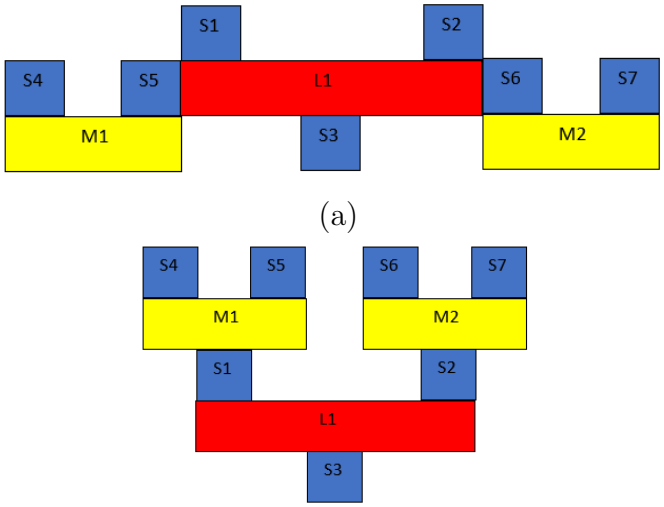


Figure 19: [17][Fig. 1.8]: (a) initial state and (b) goal state.

Similarly, in Figure 20, considering a robot with one manipulator, it is not possible to achieve the goal state without creating a stable subassembly of boxes and then incorporating it into the main structure.

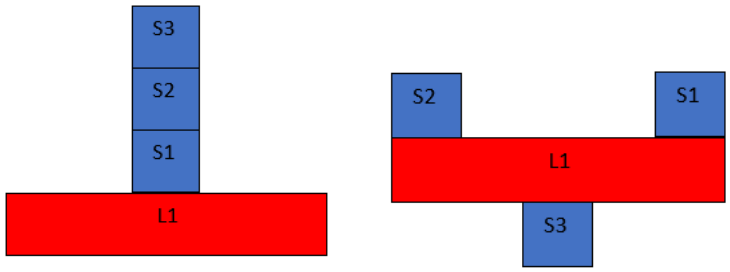


Figure 20: [17][Fig. 1.4]: (a) initial state and (b) goal state.

7.2 Disassembly

Disassembly is also an important challenge because sometimes when creating a new structure it is important to disassemble the already built structure but while doing so the overall the structure should be stable. It is not always simple to disassemble a structure by simply disassembling it one by one. In Figure 21, it is not possible to get to goal state by putting one block on the table at a time. If the structure disassembled one by one the structure would fall.

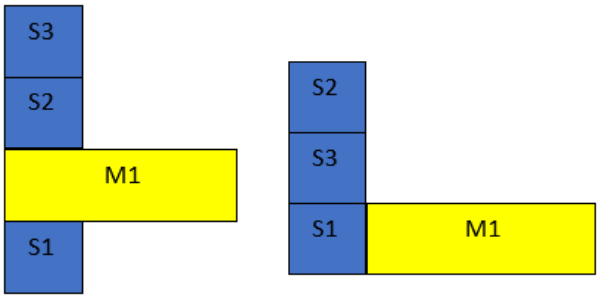
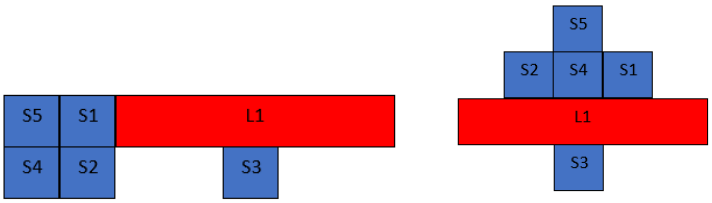


Figure 21: [17][Fig. 1.9]: (a) initial state and (b) goal state.

7.3 CounterWeights

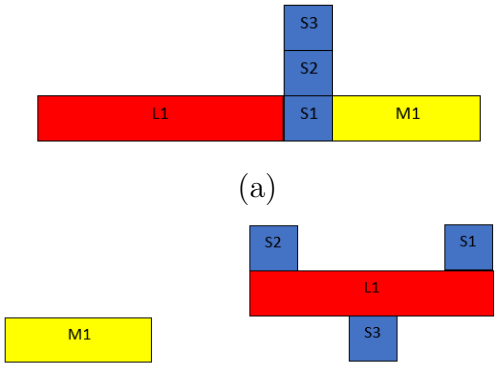
Another challenge in construction problems is utilizing the fact that blocks can act as counter weights. These counter weights basically help in balancing the whole structure. The motivation of this challenge comes from the fact that a robot should be able to utilize material from its surroundings in order to create a stable structure. In Figure 22, again considering a robot with a single manipulator, the goal state cannot be achieved without first placing $S4$ along with $S5$ on $L1$ to act as a counter weight when placing $S2$ and $S1$ one by one.



(a) (b)

Figure 22: A construction problem: (a) initial state and (b) goal state.

Similarly, in Figure 23 the goal state resembles the goal state in Figure 20 but in this scenario there is an extra *medium* block available. Again considering a single manipulator and also that the robot cannot create sub assembly, the robot has to use the heavier block as a counter to weight to put other blocks one by one.



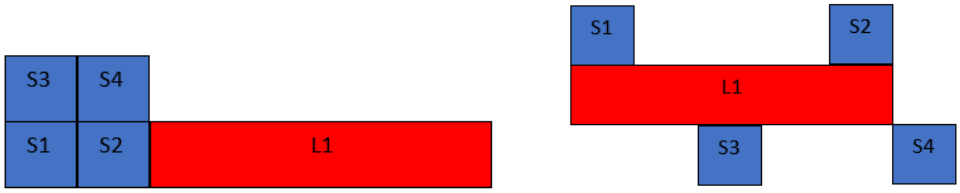
(a)

(b)

Figure 23: A construction problem: (a) initial state and (b) goal state.

7.4 Scaffolds

Another challenge similar to counter weights is scaffolding which requires supporting the structure from beneath the structure. This might be important in cases where there is no heavy block which can act as a counter weight. A particular example is given in Figure 24. Considering an single manipulator and no sub assembly manipulation, there is no way the robot can achieve the goal state without using scaffolding, because all the blocks except the large one are light weight and cannot act as counter weights. To achieve the goal state the structure has to be supported from beneath in the intermediate states.



(a) (b)

Figure 24: A construction problem: (a) initial state and (b) goal state.

7.5 True Concurrency

Normally, more than one robots can work on construction problems and in that case a problem arises which is related to the concurrent or simultaneous actions of multiple robot. This is also a major challenge while working with multiple robots. It helps to solve some problems which may not be solved using a single manipulator. In Figure 25, there are no extra blocks available and the robot does not know sub assembly manipulation. The robot has to use true concurrency to achieve goal state.

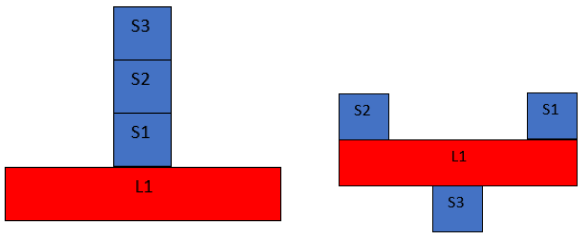


Figure 25: A construction problem: (a) initial state and (b) goal state.

7.6 Overhang

The maximum overhang benchmark used for evaluation is a 150 old challenging puzzle that has been studied in the fields of computer science and mathematics. A recent solution to this problem using counterbalance blocks has been awarded by David P. Robbins Prize in mathematics in 2011.

The goal in this problem is to maximize the distance a block from the edge of the table [24, 50–52]. Figure 26 shows the initial and final state in an overhang problem. Initially all the blocks are stacked on the table and in the final state an overhang of 4 units is achieved.

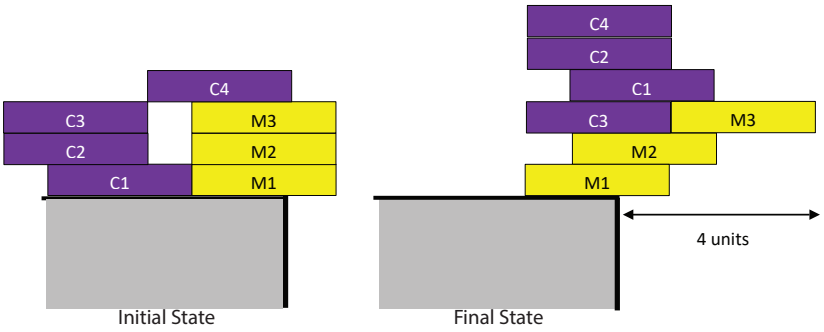


Figure 26: *Purple* blocks are heavier and are used as counter weights, while *yellow* blocks are the main blocks used to maximize the overhang. Overhang of 4 units is achieved.

7.7 Bridges

Another challenge is construction problems is the construction of a stable bridge. In this challenge the goal is to connect two sides o f a river in such a way that a stable bridge is obtained. Here the biggest challenge is the stability of the bridge at every step. Figure 27 shows **bridge** like problem. Initially there are blocks on both sides of the river and in the final state a stable bridge is constructed so that both sides are connected.

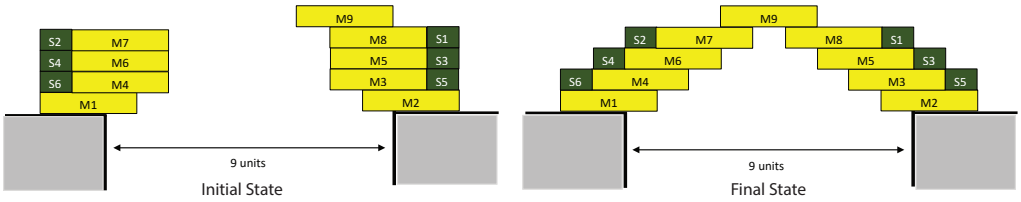


Figure 27: *Green* blocks are heavier and are used as counter weights, while *yellow* blocks are used to connect the sides of the river. The distance between the sides of the bridges is 9 units.

7.8 Asymmetric Bridges

Asymmetric bridges involves the same goal conditions as symmetric bridges but there is a difference in height of both sides of the river. Figure 28 shows a scenario in which the right side of the river is 4 units higher than the left side. Initially all the blocks are on the left side of the river and in the final state the a stable stair like structure is obtained.

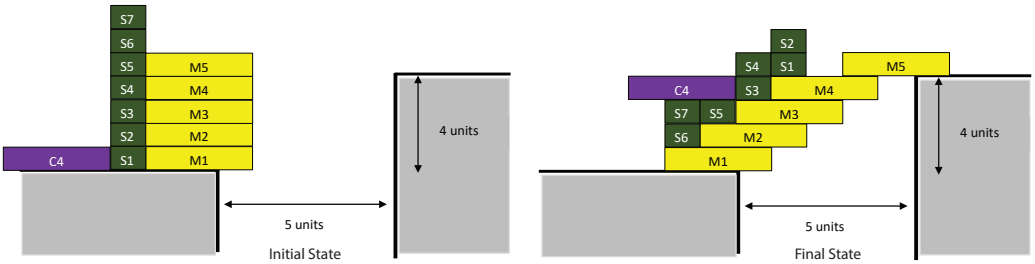


Figure 28: *Green* and *purple* blocks are heavier and are used as counter weights, while *yellow* blocks are used to connect the sides of the river. The distance between the sides of the bridges is 5 units and the height difference is 4 units.

7.9 Tower Stacking

Tower stacking benchmarks allow us to maximize or minimize the height of a particular block in **tower** formation. Figure 29 shows a tower like formation with 8 blocks where the goal was the maximize the height of $S3$. In Figure 30 the goal was to minimize the height of $S3$. In both cases it is a condition that $S3$ should be part of the stack.

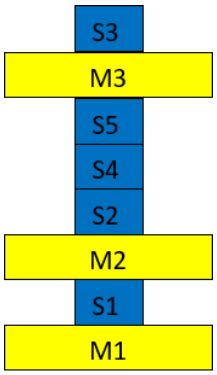


Figure 29: Maximizing the height of box s3 in a stack of 8 boxes

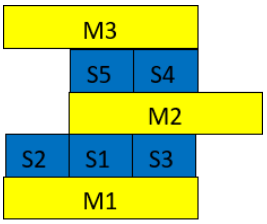


Figure 30: Minimizing the height of box s3 in a stack of 8 boxes

Chapter 8

8 Cylinder Extensions

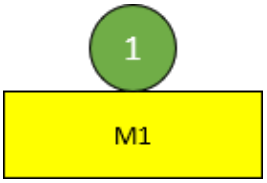
Rules introduced in Chapter 4 are sufficient for solving robot construction problems involving rectilinear objects. Of course, in reality objects are not perfectly rectangular in shape, but they may be assumed rectangular by creating a bounding box around them. Alternatively, if we have complex shaped object, it may be sub divided in to small rectangular objects and the rules introduced in the previous chapters may be used to reason about the the stability of the whole structure.

In this chapter, we will consider another extension to the construction problems, that is, the introduction of cylindrical shaped objects or simply cylinders in the problem. Cylinders introduce interesting challenges to the problem due to the following assumptions:

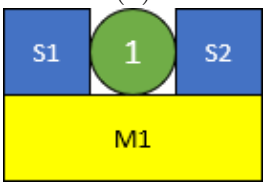
Assumptions

- A cylinder must be supported by objects on its sides. The reason behind this assumption lies on the fact that if a cylinder is not supported by objects on its sides, it may roll over and in that case the location of cylinder becomes uncertain. Refer to Figure 31 for both invalid and valid configurations.
- The diameter of cylinder is equal to the length of the small block.
- Cylinder has point contact with an object. This assumption comes from the geometrical aspect of cylinder.

- A cylinder can be on top of another cylinder in three different states: *top left*, *top mid* or *top right*. Refer to Figure 32 for all these configurations.
- A cylinder can also be on a particular unit of a box or table.
- A unit space of a particular box can be on top of a cylinder provided the cylinder is supported by objects on both sides (see Figure 36).

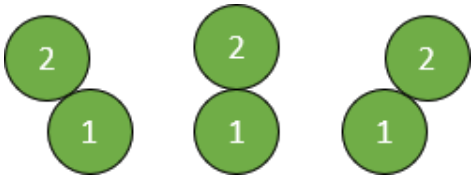


(a)



(b)

Figure 31: (a) Invalid state because cylinder is not supported by objects on sides (b) Valid state because cylinder is supported by objects on sides



(a) (b) (c)

Figure 32: (a) cylinder 2 is on top left of cylinder 1 (b) cylinder 2 is on top mid of cylinder 1 (c) cylinder 2 is on top right of cylinder 1

Figure 33: Unit space *2* of box *M1* is on cylinder *1*

8.1 Modelling

The rules mentioned in the previous chapters are sufficient to solve robot construction problems involving rectangular objects or boxes, but they are not enough when cylinders are concerned due to the following reasons:

- We do not have a fluent to express that a cylinder is on a particular unit of a location.
- We do not have a fluent to express if a cylinder is on top of another cylinder.
- We do not have a fluent to express the left and right adjacency of a cylinder or in other words, we cannot express whether a cylinder is supported on sides by another object or not.
- Similarly, we need some new actions to cause the relative changes in the world state expressed in the above reasons.

In order to model the rules required to solve scenarios related to cylinders, consider a sample scenario in Figure 34

8.2 Fluents

We introduce some new fluents to express the world states including cylinders:

on_cyl(c,l,v,t): cylinder c is on v unit space of location l at time step t. Here location $l \in \{Table, boxes\}$.

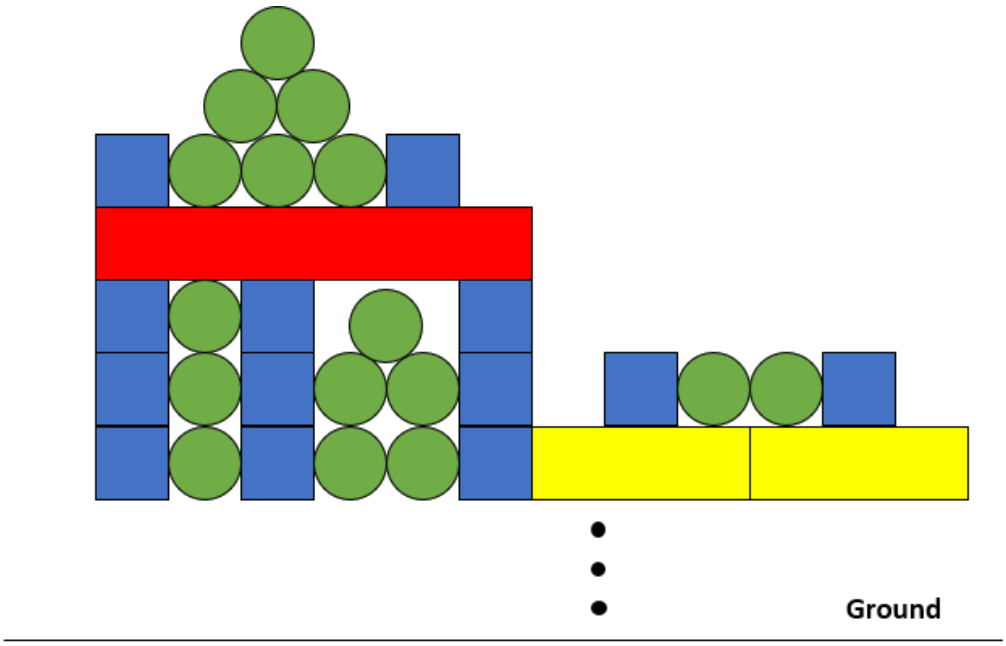


Figure 34: Sample scenario containing boxes and cylinders

top(c,c',s,t): cylinder c is on top of cylinder c' in state s at time step t. Here state $s \in \{Top\_Left, Top\_Mid, Top\_Right\}$.

box_on_cyl(b,v,c,t): v unit space of box b is on cylinder c at time step t.

Apart from that we also introduce two more fluents to describe the adjacency and global location of cylinders:

adj(c,c',d,t): cylinder c is adjacent to cylinder c' in direction d at time step t. Here direction $d \in \{Right, Left\}$.

above_cyl(h,c,x,t): cylinder c is at height h and global location x at time step t.

8.3 Actions

We also consider three new actions:

place_cyl_on(a,c,l,v,t): place cylinder c on v unit space of location l with gripper a at time step t.

place_cyl_top(a,c,c',s,t): place cylinder c on cylinder c' in state s with gripper a at time step t.

push(a,b,d,t): push box b in direction d with gripper a at time step t.

8.3.1 place_cyl_on action

Either a *place_cyl_on* action can occur or it may not occur. This is defined by the occurrence of *place_cyl_on* action using disjunctive rules.

Occurrence of the place action is defined as:

$$place\_cyl\_on(a, c, l, v, t) \vee \neg place\_cyl\_on(a, c, l, v, t) \leftarrow$$

The direct effect of *place_cyl_on* action is that the cylinder becomes *on* another location at next time step.

Direct effect of place_cyl_on action is defined as:

$$on\_cyl(c, b, v, t + 1) \leftarrow place\_cyl\_on(a, c, l, v, t), holding(a, c, t)$$

Preconditions of *place_cyl_on* action are described as following:

- A robot cannot place a cylinder on location l if its gripper is empty:

$$\leftarrow place(a, l, t), \#count\{c : cylinder(c), holding(a, c, t)\} = 0$$

where *place* is obtained from *place_cyl_on* by projection:

$$place(a, l, t) \leftarrow place\_cyl\_on(a, c, l, v, t)$$

- A robot cannot place a cylinder c with its gripper a if the target cylinder is already being held:

$$\leftarrow place(a, c, t), holding(a', c, t)$$

- A robot cannot place a cylinder c on unit space v of box b with gripper a if the box is not supported by table

$$\leftarrow place\_cyl\_on(a, c, b, v, t), not\ \ supported(b, Table, t).$$

8.3.2 place_cyl_top action

Either a *place_cyl_top* action can occur or it may not occur. This is defined by the occurrence of *place_cyl_top* action using disjunctive rules.

Occurrence of the place action is defined as:

$$place\_cyl\_top(a, c, c', s, t) \vee \neg place\_cyl\_top(a, c, c', s, t) \leftarrow$$

The direct effect of *place_cyl_top* action is that the cylinder becomes *top* on another cylinder at next time step.

Direct effect of place_cyl_on action is defined as:

$$top(c, c', s, t + 1) \leftarrow place\_cyl\_top(a, c, c', s, t), holding(a, c, t)$$

Preconditions of *place_cyl_top* action are described as following:

- A robot cannot place a cylinder c on another cylinder c' if its gripper

is empty:

$$\leftarrow place\_cyl\_top\_aux(a, c, t), \#count\{c : cylinder(c), holding(a, c, t)\} = 0$$

where *place_cyl_top_aux* is obtained from *place_cyl_top* by projection:

$$place\_cyl\_top\_aux(a, c, t) \leftarrow place\_cyl\_top(a, c, c', s, t)$$

- A robot cannot place a cylinder c with its gripper a if the target cylinder is already being held:

$$\leftarrow place(a, c, t), holding(a', c, t)$$

- A robot cannot place a cylinder c on top of cylinder c' with gripper a if the c' is not supported by table

$$\leftarrow place\_cyl\_top(a, c, c', s, t), not\ \ supported(c', Table, t).$$

8.3.3 push action

Either a *push* action can occur or it may not occur. This is defined by the occurrence of *push* action using disjunctive rules.

Occurrence of the push action is defined as:

$$push(a, b, d, t) \vee \neg push(a, b, d, t) \leftarrow$$

The direct effect of *push* action is that the box moves one unit space in the direction the action is performed.

Direct effect of *push* action is defined as:

$$on(b, l, u-1, v, t+1) \leftarrow push(a, b, Left, t), on(b, l, u, v, t)$$
$$on(b, l, u+1, v, t+1) \leftarrow push(a, b, Right, t), on(b, l, u, v, t)$$

where $0 < u - 1 \leq size(l)$, $0 < u + 1 \leq size(l)$

Preconditions of *push* action are described as following:

- A robot cannot push a box if it is already holding something:

$$\leftarrow push(a, \_, \_, t), holding(a, \_, t).$$

- A robot cannot push a box b with its gripper a if the target box is already being held:

$$\leftarrow push(a, b, \_, t), holding(a', b, t)$$

- A robot cannot push a box b if the b is not supported by table

$$\leftarrow push(a, b, \_, t), not\ \ supported(b, Table, t).$$

8.4 Ramifications

As noted before, there are interesting ramifications when cylinders are considered in the problem. We need to define adjacency of cylinders and in order to do that we need an *above* predicate similar to boxes defined before. The *above* predicate will allow us to reason about adjacency in an easier way. For cylinders, we call this predicate as *above_cyl* and it is defined by the following recursive rules:

$$
\begin{aligned}
&above\_cyl(1, c, v, t) \leftarrow on\_cyl(c, Table, v, t).\\
&above\_cyl(h+1, c, x, t) \leftarrow on\_cyl(c, b, v, t)\\
&\qquad above(h, b, v, x, t).\\
&above\_cyl(h+1, c, x, t) \leftarrow top(c, c', TopMid, t)\\
&\qquad above\_cyl(h, c', x, t).
\end{aligned}
$$

First rule says that if a cylinder c is on v unit space of table, then it is globally on v unit space and at height 1.
Second rule says that if a cylinder c is on v unit space of box b and v unit space of box b is globally at x unit space and box b is at height h, then cylinder c is globally at x unit space and height $h+1$.
Third rule says that if a cylinder c on top of c' in $TopMid$ state and cylinder c' is globally at x unit space and height h, then cylinder c will be at globally x unit space and height $h+1$.

Using the rules just mentioned, we have proper definition of *above_cyl* for cylinders and this can be used to define the adjacency for cylinders. A cylinder can be adjacent to another cylinder or it can be adjacent to a box. For cylinders we use the following rules:

$$adj(c, c', Left, t) \leftarrow adj(c', c, Right, t).$$
$$adj(c, c', Right, t) \leftarrow above\_cyl(h, c, x, t)$$
$$\quad above\_cyl(h, c', x + 1, t).$$
$$adj(c, c', Right, t) \leftarrow top(c, c'', TopLeft, t)$$
$$\quad top(c', c'', TopRight, t).$$

First rule says that if a cylinder c' is *right adjacent* to cylinder c, then cylinder c is $leftadjacent$ to cylinder c'.
Second rule says that if a cylinder c is globally at x unit space, height h and a cylinder c' is globally at $x+1$ unit space, height h, then c is $rightadjacent$ to c' .
Third rule says that if a cylinder c on top of c'' in $TopLeft$ state and a cylinder c' on top of c'' in $TopRight$ state, then c is $rightadjacent$ to c'.

Adjacency of a cylinder to box can be expressed by the following rules in the similar way:

$$adj(c, b, Right, t) \leftarrow above\_cyl(h, c, x, t)$$
$$\quad above(h, b, 1, x + 1, t).$$
$$adj(c, b, Left, t) \leftarrow above\_cyl(h, c, x + 1, t)$$
$$\quad above(h, b, size(b), x, t).$$

There is also another important ramification which allows us to deal with the stack of cylinders. Consider the arrangement of cylinders as shown in Figure 35. Let us assume that cylinder 5 is placed on cylinder 2 in $TopRight$ state, we can use ramification rules to find out if it is on top of cylinder 3 or not:

$$top(c, c'', TopRight, t) \leftarrow top(c, c', TopLeft, t), adj(c', c'', Left, t)$$
$$top(c, c'', TopLeft, t) \leftarrow top(c, c', TopRight, t), adj(c', c'', Right, t)$$

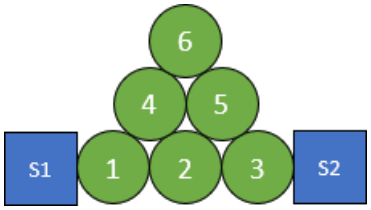


Figure 35: Cylinders arranged in a stack

Ramification rules also help us express the state when a unit space of v of box b is on cylinder c. This state is expressed in Figure 36:

$$box\_on\_cyl(b, c, v, t) \leftarrow above\_cyl(h, c, x, t), above(h + 1, b, v, x, t)$$

Figure 36: Unit space *2* of box *M1* is on cylinder *1*

Apart from the ramifications discussed above there are some other ramifications:

- If a cylinder c is placed on some location, then as a direct effect it is *on* that location but as an indirect effect the gripper is empty:

$$\neg holding(a, c, t) \leftarrow on\_cyl(c, \_, \_, t)$$

- Similarly if a cylinder c is *on* unit space u of location l, then the gripper is not holding cylinder c:

$$\neg on\_cyl(c, l, u, t) \leftarrow holding(\_, c, t)$$

- If a cylinder c is placed on top of another cylinder, then as a direct effect it is on top of that cylinder, but as an indirect effect the gripper is empty:

$$\neg holding(a, c, t) \leftarrow top(c, \_, \_, t)$$

- Similarly if a cylinder c is on top of cylinder c' in state s, then the gripper is not holding cylinder c:

$$\neg top(c, c', s, t) \leftarrow holding(\_, c, t)$$

8.5 State Constraints

Following are the necessary state constraints:

- A cylinder should be adjacent to two objects when it is on top of another cylinder in *topmid* state:

$$\leftarrow top(c, c', TopMid, t), \#count\{o : object(o), adj\_aux(c, o, t)\} < 2.$$

  Here *adj_aux* is defined as:

$$adj\_aux(c, o, t) \leftarrow adj(c, o, Left, t).$$
$$adj\_aux(c, o, t) \leftarrow adj(c, o, Right, t).$$

- A cylinder should be adjacent to two object when it is on another location:

$$\leftarrow on\_cyl(c, \_, \_, t), \#count\{o : object(o), adj\_aux(c, o, t)\} < 2.$$

- In order to avoid states shown in figure 37, following rules are added:

$$\leftarrow top(c, c', TopLeft, t), adj(c', b, Left, t).$$
$$\leftarrow top(c, c', TopRight, t), adj(c', b, Right, t).$$

(a) (b)

Figure 37: Invalid state because cylinder cannot be on a box and another cylinder

There are some existence and uniqueness constraints associated with the cylinders are stated below:

- There can be one cylinder at a particular unit of a location:

$$\leftarrow \#count\{c : cylinder(c), on\_cyl(c, l, y, t)\} > 1.$$

- A cylinder and a box cannot be at the same location:

$$\leftarrow on(b, l, u, \_, t), on\_cyl(c, l, u, t).$$

- A cylinder cannot be on top of another cylinder in more than one states:

$$\leftarrow \#count\{c : cylinder(c), top(c, c', s, t)\} > 1.$$

- A cylinder c cannot be on top of another cylinder c'' if there is already a cylinder c' which is on top of c'' in state $TopMid$:

$$\leftarrow top(c, c', \_, t), top(c'', c', TopMid, t).$$

8.6 Concurrency Constraints

Following are the concurrency constraints necessary after the introduction of three new actions:

place_cyl_on:

- The concurrency of two place_cyl_on actions with the same gripper is not allowed with the following formula:

$$\leftarrow \#count\{l : objloc(l), place(a, l, t)\} > 1$$

- The concurrency of two place actions at the same location is not allowed with the following formula:

$$\leftarrow \#count\{a : arm(a), place(a, l, t)\} > 1$$

  In the above two rules are the same as that of original place action.

- A box b cannot be picked by a gripper a while another gripper a' $(a \neq a')$ is placing a cylinder on it:

$$\leftarrow pick(a, b, t), place(a', b, t)$$

place_cyl_top:

- The concurrency of two place_cyl_top actions with the same gripper is not allowed with the following formula:

$$\leftarrow \#count\{c' : cylinder(l), place\_cyl\_top\_aux(a, c', t)\} > 1$$

- The concurrency of two place actions at the same location is not allowed

with the following formula:

$$\leftarrow \#count\{c' : arm(a), place\_cyl\_top\_aux(a, c', t)\} > 1$$

In the above two rules *place_cyl_top_aux* is obtained by projection.

$$place\_cyl\_top\_aux(a, c', t) \leftarrow place\_cyl\_top(a, c, c', s, t)$$

- A cylinder c cannot be picked by a gripper a while another gripper a' ($a \neq a'$) is placing a cylinder on it:

$$\leftarrow pick(a, c, t), place(a', c, t)$$

push:

- The concurrency of two push actions with the same gripper is not allowed with the following formula:

$$\leftarrow \#count\{b' : box(b), push(a, b, \_, t)\} > 1$$

- The concurrency of two push actions for the same box is not allowed with the following formula:

$$\leftarrow \#count\{a : arm(a), push(a, b, \_, t)\} > 1$$

- The concurrency of two push actions of same box in different directions is not allowed with the following formula:

$$\leftarrow \#count\{d : direction(d), push(a, b, d, t)\} > 1$$

Apart from these individual concurrency constraints of actions, there are some constraints within different actions

- Cannot push a box b if place action is being performed on that box:

$$\leftarrow push(a, b, \_, t), place(a', b, t), a \neq a'.$$

- Cannot push a box b if that box is being picked:

$$\leftarrow push(a, b, \_, t), pick(a', b, t), a \neq a'.$$

- pick and push actions cannot be performed with the same gripper:

$$\leftarrow pick(a, b, t), push(a, b', \_, t).$$

- pick and place_cyl_top_aux actions cannot be performed with the same gripper:

$$\leftarrow pick(a, \_, t), place\_cyl\_top\_aux(a, \_, t).$$

8.7 Supportedness Constraints

At any state of the world, no cylinder is supported by itself (i.e., no circular configurations). For that, we recursively define supportedness by the table:

$$\begin{aligned} &\mathit{supported\_cyl}(c, l, t) \leftarrow \mathit{on\_cyl\_aux}(c, l, t) \\ &\mathit{supported\_cyl}(c, l, t) \leftarrow \mathit{on\_cyl\_aux}(c, l', t), \\ &\qquad \mathit{supported}(l', l, t) \quad (c{\neq}l') \end{aligned}$$

where *on_cyl_aux* is obtained from *on_cyl* by projection:

$$\mathit{on\_cyl\_aux}(c, l, t) \leftarrow \mathit{on\_cyl}(c, l, v, t)$$

Since, *top* fluent also describes cylinder so we need to define supportedness using that fluent also:

$$\begin{aligned} &\mathit{supported\_cyl}(c, c', t) \leftarrow \mathit{top\_aux}(c, c', t) \\ &\mathit{supported\_cyl}(c, c'', t) \leftarrow \mathit{top\_aux}(c, c', t), \\ &\qquad \mathit{supported\_cyl}(c', c'', t) \quad (c{\neq}c') \end{aligned}$$

where *top_aux* is obtained from *top* by projection:

$$\mathit{top\_aux}(c, c', t) \leftarrow \mathit{top}(c, c', s, t)$$

After that, we add a constraint to ensure that no cylinder c is supported by itself:

$$\leftarrow \mathit{supported}(c, c, t).$$

8.8 The Commonsense of Law of Inertia

We represent the commonsense law of inertia to address the frame problem, by the following rules:

- For *on_cyl* fluent:

$$on\_cyl(c, l, v, t+1) \leftarrow on\_cyl(c, l, v, t), not\ \neg on\_cyl(c, l, v, t+1)$$
$$\neg on\_cyl(c, l, v, t+1) \leftarrow \neg on\_cyl(c, l, v, t), not\ on\_cyl(c, l, v, t+1)$$

- For *top* fluent:

$$top(c, c', s, t+1) \leftarrow top(c, c', s, t), not\ \neg top(c, c', s, t+1)$$
$$\neg top(c, c', s, t+1) \leftarrow \neg top(c, c', s, t), not\ top(c, c', s, t+1)$$

- For *box_on_cyl* fluent:

$$box\_on\_cyl(b, v, c, t+1) \leftarrow box\_on\_cyl(b, v, c, t), not\ \neg box\_on\_cyl(b, v, c, t+1)$$
$$\neg box\_on\_cyl(b, v, c, t+1) \leftarrow \neg box\_on\_cyl(b, v, c, t), not\ box\_on\_cyl(b, v, c, t+1)$$

Chapter 9

9 Experimental Evaluations

We consider the robot construction planning problems introduced in the previous chapters, where the aim is for a robot to build specified structures out of simple rectangular blocks of different sizes. These problems are challenging both from the perspective of high-level knowledge representation and reasoning and from the perspective of low-level geometric reasoning. Let us illustrate these challenges by some examples.

9.1 Experimental Setup

All experiments are conducted on a PC workstation running Ubuntu 14.04 on 16 2.4GHz Intel E5-2665 CPU cores with 64GB memory. For reasoning over ASP programs, we use DLVHEX version 2.5. We use Python 2.7, Pybullet 2.4.1 and RRT* from OMPL library for feasibility checks.

9.2 Sub-assembly Manipulation

Scenario 1 (Figure 38) This construction problem involves incorporation of the existing structures into the final design. Initial state of all the boxes is specified by the following facts:

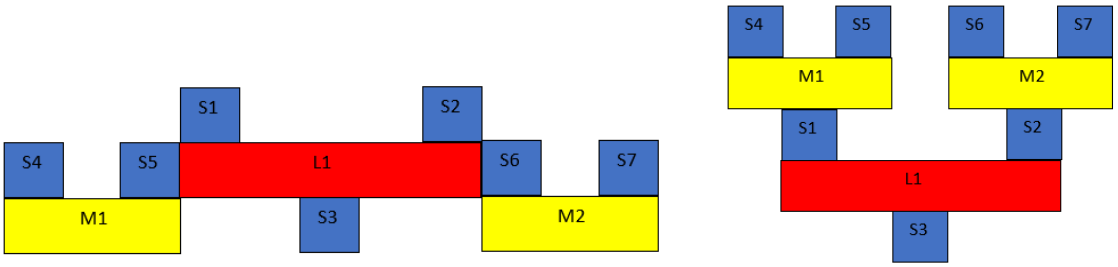
$init(M1, Table, 1, 1).init(M1, Table, 2, 2).init(M1, Table, 3, 3).init(S4, M1, 1, 1).$
$init(S5, M1, 3, 1).init(S3, Table, 6, 1).init(L1, S3, 1, 3).init(S1, L1, 1, 1).$
$init(S2, L1, 5, 1).init(M2, Table, 9, 1).init(M2, Table, 10, 2).init(M2, Table, 11, 3).$
$init(S6, M2, 1, 1).init(S7, M2, 3, 1).$

Goal conditions for a final configuration are specified by the set of following facts:

$goal(S3, Table).goal(L1, S3).goal(S1, L1).goal(S2, L1).goal(M1, S1).$
$goal(M2, S2).goal(S4, M1).goal(S5, M1).goal(S6, M2).goal(S7, M2).$

For such an instance, a plan for a bimanual robot, like Baxter, to achieve the goal configuration from the initial state involves the following actions:

$pick(Left, M1, 0), pick(Right, M2, 0).$
$placeOn(Left, M1, S1, 1, 2, 1), placeOn(Right, M2, S2, 1, 2, 1).$



(a) (b)

Figure 38: (a) initial state and (b) goal state.

Scenario 2 (Figure 39) This problem requires first the pre-assembly of a movable stable substructures on the table. Initial state of all the boxes is specified by the following facts:

$$init(L1,\, Table, 1, 1).init(L1,\, Table, 2, 2).init(L1,\, Table, 3, 3).init(L1,\, Table, 4, 4).$$
$$init(L1,\, Table, 5, 5).init(S1, L1, 3, 1).init(S2, S1, 1, 1).init(S3, S2, 1, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3,\, Table).goal(L1, S3).goal(S1, L1).goal(S2, L1).$$

For such a problem instance, our planner generates the following plan:

$$pick(Left, S3, 0).$$
$$placeOn(Left, S3, Table, 6, 1, 1), pick(Right, S2, 1).$$
$$pick(Left, S1, 2), placeOn(Right, S2, L1, 1, 1, 2).$$
$$placeOn(Left, S1, L1, 5, 1, 3).$$
$$pick(Left, L1, 4).$$
$$placeOn(Left, L1, S3, 1, 3, 5).$$

Note that special attention needs to be paid as to where blocks are placed on $L1$ to ensure stability.

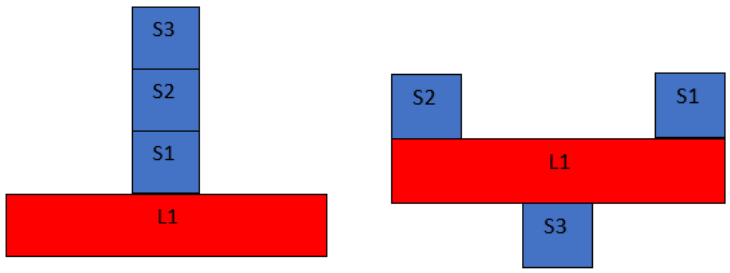


Figure 39: [18][Fig. 1.4]: (a) initial state and (b) goal state.

9.3 Disassembly

Scenario 3 (Figure 40) This construction problem cannot be solved by moving one block at a time as in the Blocks World, since the stability of the overall structure needs to be preserved while executing the plan. It requires first moving the block $M1$ and the blocks above it. Initial state of all the boxes is specified by the following facts:

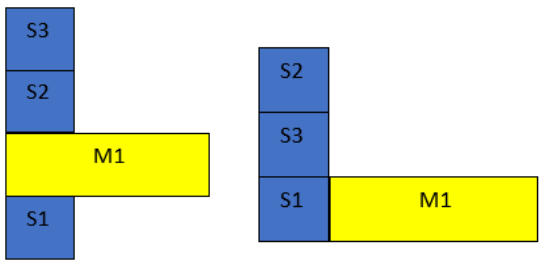
$$init(S1, Table, 1, 1).init(M1, S1, 1, 1).init(S2, M1, 1, 1).init(S3, S2, 1, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(M1, Table).goal(S1, Table).goal(S3, S1).goal(S2, S3).$$

Generated plan is as following:

$$pick(Left, M1, 0).$$
$$placeOn(Left, M1, Table, 2, 1, 1).$$
$$pick(Right, S3, 2).$$
$$placeOn(Right, S3, S1, 1, 1, 3), pick(Left, S2, 3).$$
$$placeOn(Left, S2, S3, 1, 1, 4).$$



(a) (b)

Figure 40: [18][Fig. 1.9]: (a) initial state and (b) goal state.

9.4 CounterWeights

Scenario 4 (Figure 41) This construction problem requires some kind of counter weight to balance out the whole structure. Initial configuration is expressed by the following set of facts:

$$init(S4,\, Table, 1, 1).init(S2,\, Table, 2, 1).init(S3,\, Table, 5, 1).init(S5, S4, 1, 1).$$
$$init(S1, S2, 1, 1).init(L1, S3, 1, 3).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3,\, Table).goal(L1, S3).goal(S2, L1).$$
$$goal(S4, L1).goal(S1, L1).goal(S5, S4).$$

Generated plan is as following:

$$pick(Left, S4, 0).$$
$$placeOn(Left, S4, L1, 3, 1, 1), pick(Right, S1, 1).$$
$$pick(Left, S2, 2), placeOn(Right, S1, L1, 4, 1, 2).$$
$$placeOn(Left, S2, L1, 2, 1, 3).$$

It is interesting that the block $S4$ (and the block $S5$ above it) is moved onto $L1$ as a counterweight, so that the blocks $S2$ and $S1$ can be moved onto $L1$ appropriately.

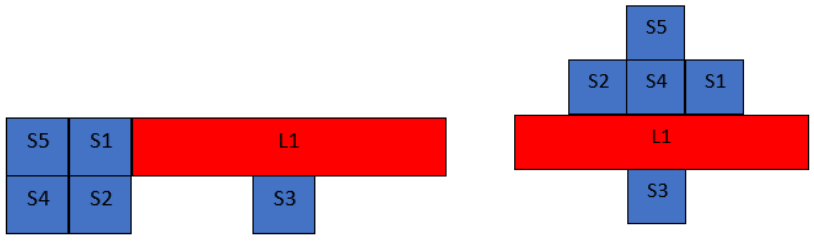


Figure 41: A construction problem: (a) initial state and (b) goal state.

Scenario 5 (Figure 42) For this problem, again counter weight needs to be used to balance the whole structure. Initial configuration is expressed by the following set of facts:

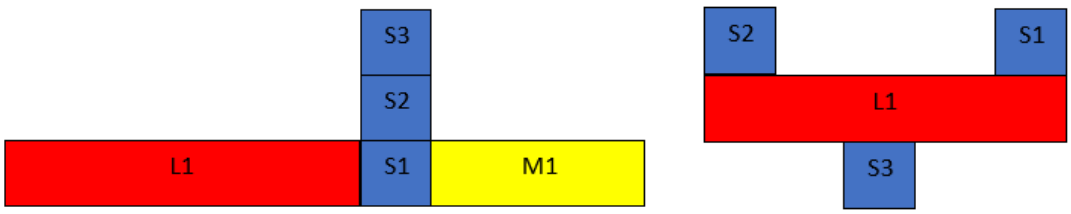
$$init(L1,\ Table, 1, 1).init(L1,\ Table, 2, 2).init(L1,\ Table, 3, 3).init(L1,\ Table, 4, 4).$$
$$init(L1,\ Table, 5, 5).init(S1,\ Table, 6, 1).init(M1,\ Table, 7, 1).init(M1,\ Table, 8, 2).$$
$$init(M1,\ Table, 9, 3).init(S2, S1, 1, 1).init(S3, S2, 1, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3,\ Table).goal(L1, S3).goal(S2, L1).$$
$$goal(S1, L1).goal(M1,\ Table).$$

Generated plan is:

$$pick(Left, S3, 0).$$
$$placeOn(Left, S3, Table, 10, 1, 1), pick(Right, L1, 1).$$
$$pick(Left, M1, 2), placeOn(Right, L1, S3, 1, 3, 2).$$
$$placeOn(Left, M1, L1, 2, 1, 3), pick(Right, S2, 3).$$
$$pick(Left, S1, 4), placeOn(Right, S2, L1, 1, 1, 4).$$
$$pick(Right, M1, 5), placeOn(Left, S1, L1, 5, 1, 5).$$
$$placeOn(Right, M1, Table, 3, 1, 6).$$



(a) (b)

Figure 42: A construction problem: (a) initial state and (b) goal state.

9.5 Scaffolds

Scenario 6 (Figure 43) This construction problem requires scaffolding because in this scenario we do not have a heavy box or multiple boxes that can be used as a counter weight. In this scenario scaffolding technique is used to support the structure. Initial configuration is expressed by the following set of facts:

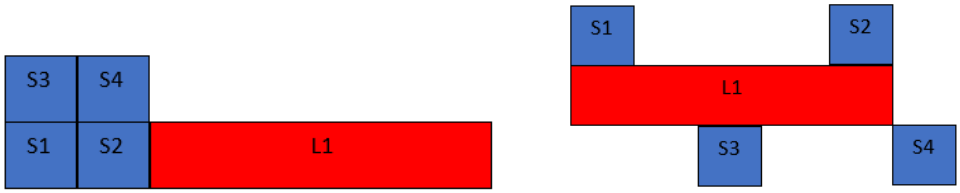
$init(S1,\ Table, 1, 1).init(S2,\ Table, 2, 2).init(L1,\ Table, 3, 1).init(L1,\ Table, 4, 2).$
$init(L1,\ Table, 5, 3).init(L1,\ Table, 6, 4).init(L1,\ Table, 7, 5).init(S3, S1, 1, 1).$
$init(S4, S2, 1, 1).$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3,\ Table).goal(L1, S3).goal(S2, L1).$$
$$goal(S1, L1).goal(S4,\ Table).$$

Generated plan is:

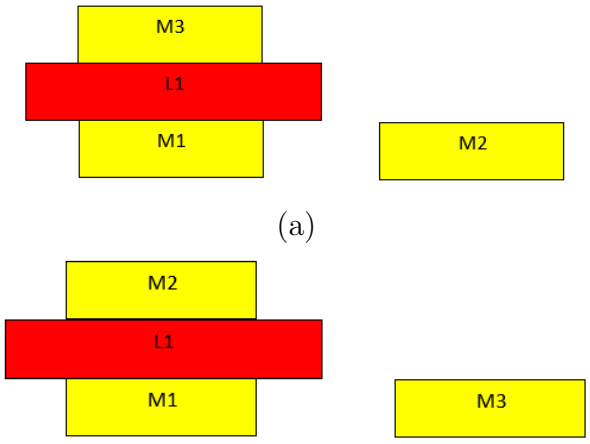
$$pick(Left, S3, 0).$$
$$placeOn(Left, S3, Table, 5, 1, 1), pick(Right, L1, 1).$$
$$pick(Left, S4, 2), placeOn(Right, L1, S3, 1, 3, 2).$$
$$placeOn(Left, S4, Table, 3, 1, 3), pick(Right, S1, 3).$$
$$pick(Left, S2, 4), placeOn(Right, S1, L1, 1, 1, 4).$$
$$pick(Right, S4, 5), placeOn(Left, S2, L1, 5, 1, 5).$$
$$placeOn(Right, S4, Table, 6, 1, 6).$$



(a) (b)

Figure 43: A construction problem: (a) initial state and (b) goal state.

9.6 True Concurrency



(a)

(b)

Figure 44: (a) initial state and (b) goal state.

Scenario 7 (Figure 44) Consider the following problem, where $M3$ and $M2$ are swapped by two concurrent actions. This problem shows the importance of true concurrency. Initial configuration is expressed by the following set of facts:

$init(M1, Table, 2, 1).init(M1, Table, 3, 2).init(M1, Table, 4, 3).init(L1, M1, 1, 2).$
$init(L1, M1, 2, 3).init(L1, M1, 3, 4).init(M3, L1, 2, 1).init(M3, L1, 3, 2).$
$init(M3, L1, 4, 3).init(M2, Table, 7, 1).init(M2, Table, 8, 2).init(M2, Table, 9, 3).$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(M1, Table).goal(L1, M1).$$
$$goal(M2, L1).goal(S3, Table).$$

Generated plan is:

$$pick(Left, M2, 0), pick(Right, M3, 0),$$
$$placeOn(Left, M2, L1, 2, 1, 1), placeOn(Right, M3, Table, 7, 1, 1).$$

When the box $M2$ is placed on $L1$, as a direct effect the first unit space of $M2$ is on the second unit space of $L1$; as its indirect effects, the second unit space of $M2$ is on the third unit space of $L1$, and the third unit space of $M2$ is on the fourth unit space of $L1$.

9.7 IndirectOn

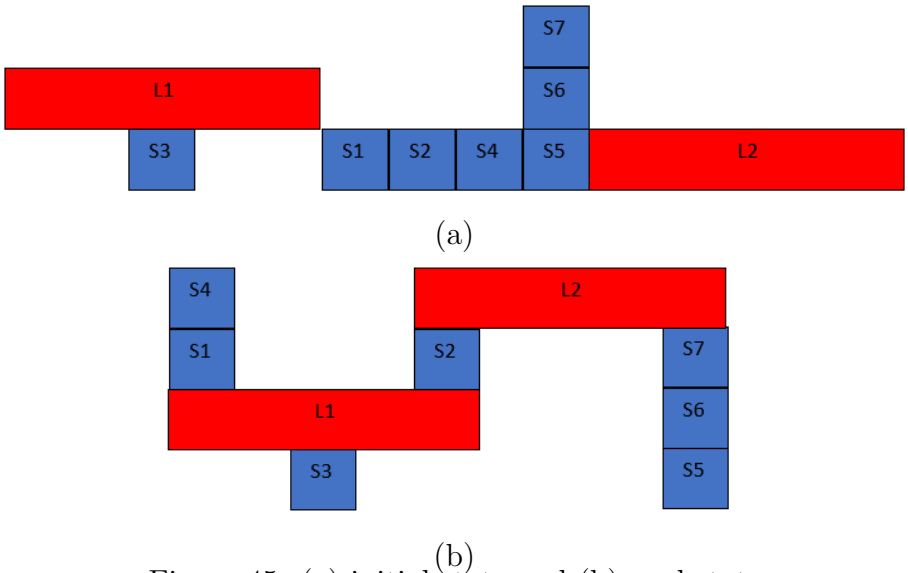


Figure 45: (a) initial state and (b) goal state.

Scenario 8 (Figure 45) This construction problem shows the importance of ramifications in our framework. Here when $L2$ is placed on $S2$ at the final

step. As an indirect effect of this action, $L2$ becomes on top of the block $S7$ as well. Initial configuration is expressed by the following set of facts:

$$init(S3,\, Table, 3, 1).init(S1,\, Table, 6, 1).init(S2,\, Table, 7, 1).init(S4,\, Table, 8, 1).$$
$$init(S5,\, Table, 9, 1).init(L2,\, Table, 10, 1).init(L2,\, Table, 11, 2).init(L2,\, Table, 12, 3).$$
$$init(L2,\, Table, 13, 4).init(L2,\, Table, 14, 5).init(L1, S3, 1, 3).init(S6, S5, 1, 1).init(S7, S6, 1, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3,\, Table).goal(L1, S3).goal(S1, L1).$$
$$goal(S2, L1).goal(S4, S1).goal(S5,\, Table).$$
$$goal(S6, S5).goal(S7, S6).goal(L2, S2).goal(L2, S7).$$

Generated plan is:

$$pick(Left, S1, 0), pick(Right, S2, 0),$$
$$placeOn(Left, S1, L1, 1, 1, 1), placeOn(Right, S2, L1, 5, 1, 1),$$
$$pick(Left, S4, 2), pick(Right, L2, 2),$$
$$placeOn(Left, S4, S1, 1, 1, 3), placeOn(Right, L2, S2, 1, 1, 3),$$

9.8 Stability

Scenario 9 (Figure 46) Consider a construction problem instance with four boxes on the table: two small boxes $S1$ and $S2$, a medium box $M1$, and a large box $L1$. All the boxes are initially on the table. Goal conditions for a final configuration are specified by a set of facts:

$$goal(S1, L1).goal(S2, L1).goal(M1, S1).goal(M1, S2).goal(L1,\, Table).$$

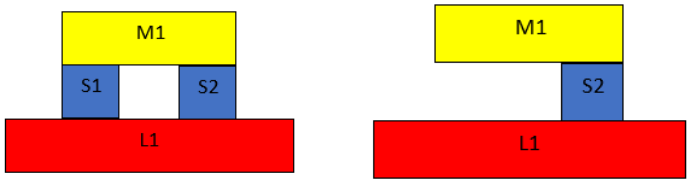
According to this description, $S1$ and $S2$ are on $L1$, $M1$ is on $S1$ and $S2$, $L1$ is on the table.

These goal conditions are ensured at a specified maximum step M by

constraints as follows:

$$\leftarrow goal(b, l), not\ onAux(b, l, M + 1).$$

Note that since $onAux$ atoms are projections of on, the final positions of objects are selected nondeterministically.



(a) (b)

Figure 46: (a) A final stable configuration for Scenario 9. (b) An intermediate configuration of boxes, obtained by a plan without feasibility checks.

A possible final stable configuration is shown in Figure 46(a). Such a configuration is achievable by the following hybrid plan of length 4 (with 6 actions), computed by DLVHEX:

$$pick(Left, S2, 0), pick(Right, S1, 0),$$
$$placeOn(Right, S1, L1, 2, 1, 1),$$
$$placeOn(Left, S2, L1, 4, 1, 2), pick(Right, M1, 2),$$
$$placeOn(Right, M1, S2, 1, 3, 3).$$

According to this plan, first the smaller boxes are picked and placed on the long box; afterwards, the medium box is picked and placed on top of the two small boxes.

Now let us consider the domain description without any feasibility checks. Then DLVHEX computes the following nonhybrid task plan of length 4 (with

6 actions):

$$pick(Right, M1, 0), pick(Left, S2, 0),$$
$$placeOn(Left, S2, L1, 4, 1, 1),$$
$$placeOn(Right, M1, S2, 1, 3, 2), pick(Left, S1, 2),$$
$$placeOn(left, S1, L1, 2, 1, 3).$$

This nonhybrid plan is computed in 60.5 seconds. According to this plan, first the medium box $M2$ and the small box $S2$ are picked, then the small box is placed on the longer box $L1$, then the medium box $M2$ is placed on $S2$. At this point, an unstable configuration is obtained, as shown in Figure 46(b). Therefore, integration of feasibility checks generate achievable plans, although such plans take longer to compute.

Scenario 10 (Figure 47) Consider a robot construction problem instance with six boxes on the table, $S1$, $S2$, $S3$, $M1$, $M2$, and $L1$, with the following goal conditions:

$$goal(M2, S1).goal(S2, M1).$$
$$goal(L1, S2).goal(L1, M2).$$
$$goal(S1, Table).goal(S3, Table).goal(M1, Table).$$

A possible final stable configuration is shown in Figure 47. Such a configuration is achievable by the following hybrid plan of length 4 (with 6 actions), computed by DLVHEX:

$$pick(Left, M2, 0), pick(Right, S2, 0),$$
$$placeOn(Left, M2, S3, 1, 3, 1), placeOn(Right, S2, M1, 1, 2, 1),$$
$$pick(Right, L1, 2),$$
$$placeOn(Right, L1, S2, 1, 5, 3).$$

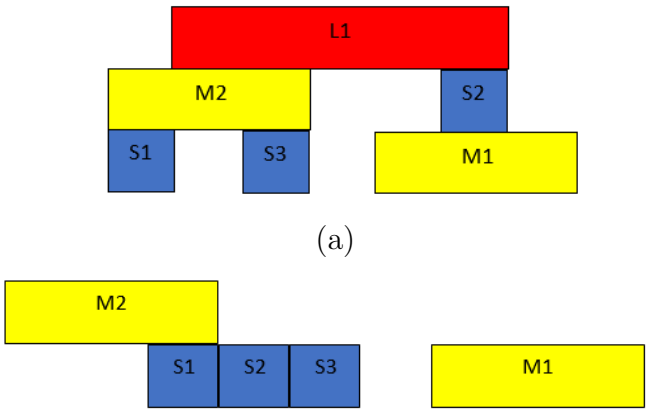


Figure 47: (a) A final stable configuration for Scenario 10. (b) An intermediate configuration of boxes, obtained by a plan without feasibility checks.

Without feasibility checks, a nonfeasible plan of length 4 (with 6 actions) is computed as follows:

$$pick(Right, M2, 0)$$
$$pick(Left, L1, 1), placeOn(Right, M2, S1, 1, 3, 1),$$
$$pick(Right, S2, 2), placeOn(Left, L1, M1, 3, 1, 2),$$
$$placeOn(Right, S2, M2, 1, 1, 3).$$

This nonhybrid task plan leads to an unstable intermediate configuration, where the medium box $M2$ is placed on the small box $S1$ at one end, leaving the center of mass outside the small box.

Scenario 11 (Figure 48) Consider a problem instance with six boxes on the table: $S1$, $S2$, $S3$, $M1$, $M2$, and $L1$. Goal conditions for a final configuration are specified as follows:

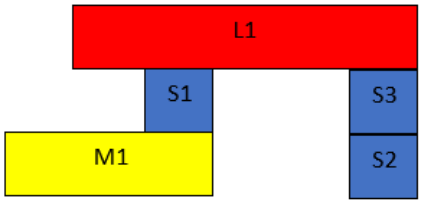
$$goal(S1, M1).goal(S3, S2).goal(L1, S3).$$
$$goal(S2, Table).goal(M1, Table).$$

A possible final stable configuration is shown in Figure 48. Such a configuration is achievable by the following hybrid plan of length 4 (with 6 actions), as computed by DLVHEX in 4905.2 seconds:

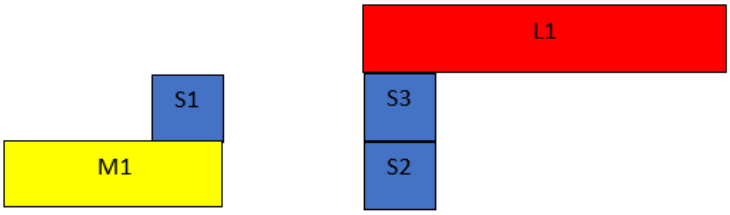
$$
\begin{aligned}
&pick(Left, S1, 0), pick(Right, S3, 0),\\
&placeOn(Left, S1, M1, 3, 1, 1), placeOn(Right, S3, S2, 1, 1, 1),\\
&pick(Right, L1, 2),\\
&placeOn(Right, L1, S1, 1, 2, 3).
\end{aligned}
$$

Without feasibility checks, a plan of length 4 (with 6 actions) can be computed in 1064.7 seconds. However, this nonhybrid task plan leads to an unstable configuration (where the long box is placed on a small box in an unstable way).

$$
\begin{aligned}
&pick(Left, S1, 0), pick(Right, S3, 0),\\
&placeOn(Left, S1, M1, 3, 1, 1), placeOn(Right, S3, S2, 1, 1, 1),\\
&pick(Right, L1, 2),\\
&placeOn(Right, L1, S3, 1, 2, 3).
\end{aligned}
$$



(a)



(b)

Figure 48: (a) A final stable configuration for Scenario 10. (b) The final configuration of boxes, obtained by a plan without feasibility checks.

9.9 Overhang Scenarios

In these sample scenarios, the yellow blocks are used for construction purposes, while the green and purple blocks are used as counterweights. The yellow and purple blocks occupy 3 unit spaces and green blocks occupy 1 unit space. The purple and green blocks are denoted by alphabet 'C' and 'S' respectively. The purple blocks are 10 times heavier than the yellow blocks, while the green blocks are are 3 times heavier than the yellow blocks. In all the overhang scenarios, goal conditions are defined according to the rules in 5.

Scenario 12 Consider a construction problem involving 8 blocks with 5 of them as counter weights in Figure 49. The goal here is to achieve a maximum overhang of 3 units. This requires careful balancing of weights to stabilize the structure. Initial configuration of blocks is given as:

$init(S1, Table, 1, 1).init(S2, S1, 1, 1).init(S3, Table, 2, 1).init(S4, S3, 1, 1).$
$init(S5, S4, 1, 1).init(M1, Table, 3, 1).init(M1, Table, 4, 2).init(M1, Table, 5, 3).$
$init(M2, M1, 1, 1).init(M2, M1, 2, 2).init(M2, M1, 3, 3).init(M3, M2, 2, 1).init(M3, M2, 3, 2).$

The plan is as follows:

$0 : pick(Left, S3), pick(Right, S1).$
$1 : place(Left, S3, M2, 1, 1), place(Right, S1, M3, 1, 1).$
$2 : pick(Left, M2).$
$3 : place(Left, M2, M1, 2, 1), pick(Right, M3).$
$4 : place(Right, M3, M2, 3, 1).$

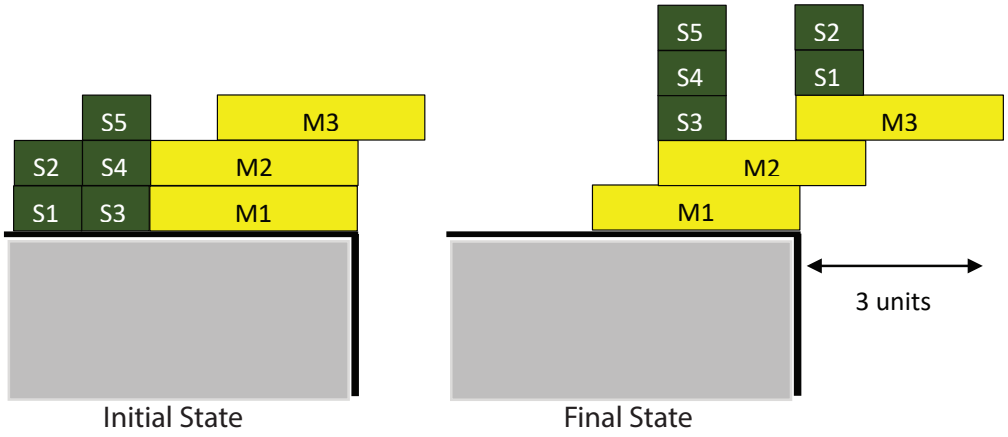


Figure 49: Stable construction of a 3 unit overhang

Scenario 13 This construction problem involves 7 blocks with 4 of them as counter weights Figure 50. In this scenario the counter weights are heavier and occupy more space. Here the goal is to achieve a maximum overhang of 4 units. Initial configuration of blocks is given as:

$init(C1,\ Table, 1, 1).init(C1,\ Table, 2, 2).init(C1,\ Table, 3, 3).init(C2, C1, 1, 2).$
$init(C2, C1, 2, 3).init(C3, C2, 1, 1).init(C3, C2, 2, 2).init(C3, C2, 3, 3).$
$init(M1,\ Table, 4, 1).init(M1,\ Table, 5, 2).init(M2, M1, 1, 1).init(M2, M1, 2, 2).$
$init(M2, M1, 3, 3).init(M3, M2, 1, 1).init(M3, M2, 2, 2).init(M3, M2, 3, 3).$
$init(C4, M3, 1, 2).init(C4, M3, 2, 3).$

The plan is as follows:

$0 : pick(Left, C3), pick(Right, M3).$
$1 : place(Left, C3, M2, 1, 2), place(Right, M3, C2, 1, 1).$
$2 : pick(Left, M2), pick(Right, C4).$
$3 : place(Left, M2, M1, 2, 1).$
$4 : pick(Left, M3), place(Right, C4, C2, 1, 1).$
$5 : pick(Right, C1).$
$6 : place(Left, M3, M2, 3, 1), place(Right, C1, C3, 2, 1).$

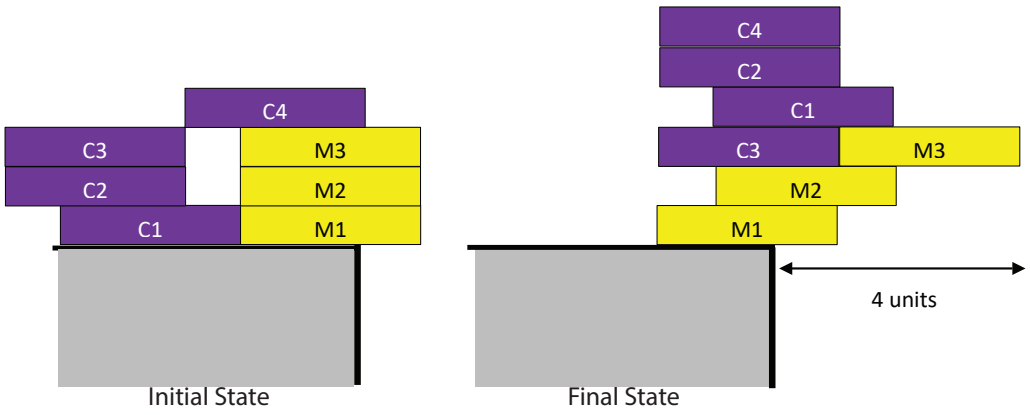


Figure 50: Stable construction of a 4 unit overhang

Scenario 14 In this construction problem, maximum overhang of 5 unit is achieved Figure 51. Initial configuration of blocks is as following:

$init(C1, Table, 1, 1).init(C1, Table, 2, 2).init(C1, Table, 3, 3).init(C2, C1, 1, 2).$
$init(C2, C1, 2, 3).init(C3, C2, 1, 1).init(C3, C2, 2, 2).init(C3, C2, 3, 3).$
$init(C4, C3, 1, 1).init(C4, C3, 2, 2).init(C4, C3, 3, 3).init(C5, C4, 1, 1).$
$init(C5, C4, 2, 2).init(C5, C4, 3, 3).init(M1, Table, 4, 1).init(M1, Table, 5, 2).$
$init(M2, M1, 1, 1).init(M2, M1, 2, 2).init(M2, M1, 3, 3).init(M3, M2, 1, 2).$
$init(M3, M2, 2, 3).init(M4, M3, 2, 1).init(M4, M3, 3, 2).init(C6, M4, 1, 2).$
$init(C6, M4, 2, 3).$

The plan is as follows:

$0 : pick(Left, M2).$
$1 : place(Left, M2, M1, 2, 1), pick(Right, C5).$
$2 : pick(Left, M3), place(Right, C5, M2, 1, 2).$
$3 : pick(Right, C1), place(Left, M3, Table, 1, 3).$
$4 : pick(Left, C6), place(Right, C1, C5, 2, 1).$
$5 : place(Left, C6, C4, 1, 2).pick(Right, M3).$
$6 : place(Right, M3, M2, 3, 1).$

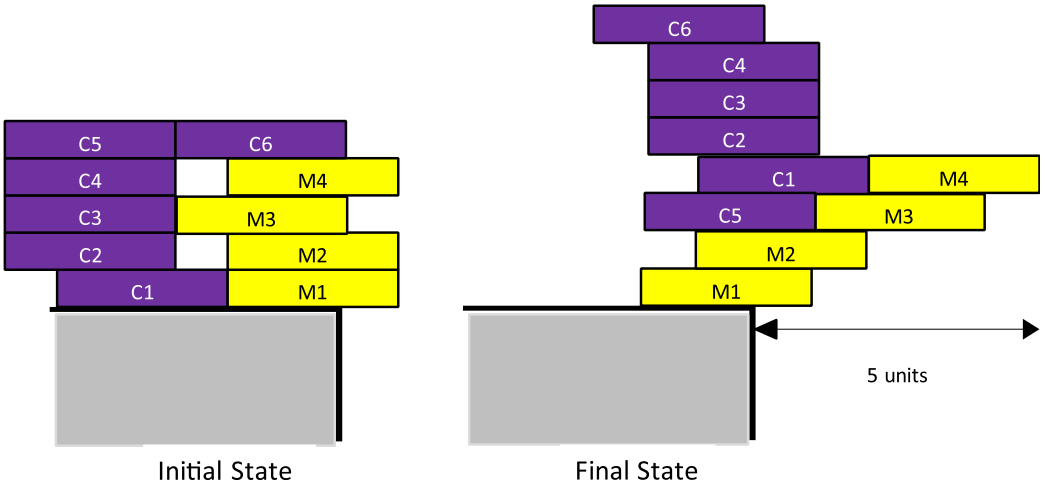


Figure 51: Stable construction of a 5 unit overhang

9.10 Bridge Scenarios

In bridge construction scenarios the goal is to join both sides of the river by a stable construction of blocks. These scenarios requires connectedness and also the stability. Connectedness is achieved by rules expressed in 5. For stability we rely on stability check. The goal in all these scenarios are expressed by the rules in 5.

Scenario 15 This bridge construction problem involves a distance of 9 units between the two sides. There total 15 blocks with 6 of them being

counter weights. Initial configuration is expressed by following set of facts:

$init(M1, Table, 1, 1).init(M1, Table, 2, 2).init(S6, M1, 1, 1).init(M4, M1, 2, 1).$
$init(M4, M1, 3, 2).init(S4, S6, 1, 1).init(M6, M4, 1, 1).init(M6, M4, 2, 2).$
$init(M6, M4, 3, 3).init(S2, S4, 1, 1).init(M7, M6, 1, 1).init(M7, M6, 2, 2).$
$init(M7, M6, 3, 3).init(M2, Table, 10, 2).init(M2, Table, 11, 3).init(M3, M2, 1, 2).$
$init(M3, M2, 2, 3).init(S5, M2, 3, 1).init(M5, M3, 1, 1).init(M5, M3, 2, 2).$
$init(M5, M3, 3, 3).init(S3, S5, 1, 1).init(M8, M5, 1, 1).init(M8, M5, 2, 2).$
$init(M8, M5, 3, 3).init(S1, S3, 1, 1).init(M9, M8, 1, 2).init(M9, M8, 2, 3).$

The plan for the example in Figure 52 is as follows:

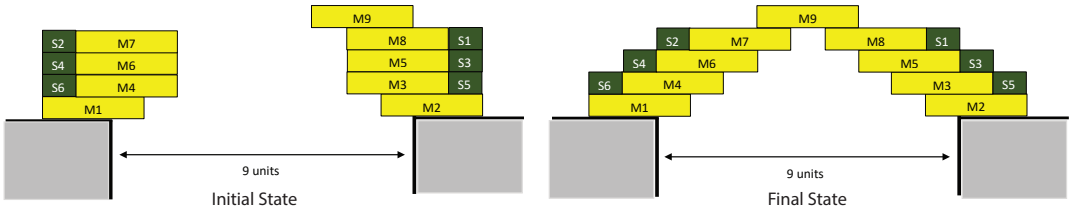


Figure 52: Stable construction of a 9 unit bridge.

$0 : pick(Left, M6), pick(Right, S4).$
$1 : place(Left, M6, M4, 2, 1).place(Right, S4, M4, 1, 1).$
$2 : pick(Left, M5), pick(Right, S3).$
$3 : place(Left, M5, M3, 2, 3).place(Right, S3, M3, 3, 1).$
$4 : pick(Left, M7), pick(Right, S2).$
$5 : place(Left, M7, M6, 2, 1).place(Right, S2, M6, 1, 1).$
$6 : pick(Left, M8), pick(Right, S1).$
$7 : place(Left, M8, M5, 2, 3).place(Right, S1, M5, 3, 1).$
$8 : pick(Left, M9).$
$9 : place(Left, M9, M7, 3, 1).$

Scenario 16 This bridge construction problem has 7 units distance between the sides Figure 53. There are 13 blocks with 6 of them being counter weights. Here the counter weights are heavier and occupy more space. Initial configuration is given by the following set of facts:

$init(M1, Table, 1, 1).init(M1, Table, 2, 2).init(M1, Table, 3, 3).init(M2, M1, 1, 1).$
$init(M2, M1, 2, 2).init(M2, M1, 3, 3).init(M3, M2, 1, 1).init(M3, M2, 2, 2).$
$init(M3, M2, 3, 3).init(M4, Table, 11, 1).init(M4, Table, 12, 2).init(M4, Table, 13, 3).$
$init(M5, M4, 1, 1).init(M5, M4, 2, 2).init(M5, M4, 3, 3).init(M6, M5, 1, 1).$
$init(M6, M5, 2, 2).init(M6, M5, 3, 3).init(M7, M6, 1, 2).init(M7, M6, 2, 3).$
$init(C6, Table, 15, 1).init(C6, Table, 16, 2).init(C6, Table, 17, 3).init(C5, C6, 1, 2).$
$init(C5, C6, 2, 3).init(C4, C5, 2, 1).init(C4, C5, 3, 2).init(C3, C4, 1, 2).$
$init(C3, C4, 2, 3).init(C2, C3, 2, 1).init(C2, C3, 3, 2).init(C1, C2, 1, 2).$
$init(C1, C2, 2, 3).$

The plan is as follows:

$0 : pick(Left, M2), pick(Right, M5).$
$1 : place(Left, M2, M1, 2, 1), place(Right, M5, M4, 1, 2)$
$2 : pick(Left, M3), pick(Right, C3).$
$3 : place(Left, M3, M2, 3, 1), place(Right, C3, M2, 1, 2).$
$4 : pick(Left, M6), pick(Right, C6).$
$5 : place(Left, M6, M5, 1, 3), place(Right, C6, M5, 3, 2).$
$6 : pick(Left, M7).$
$7 : place(Left, M7, M6, 1, 3).$

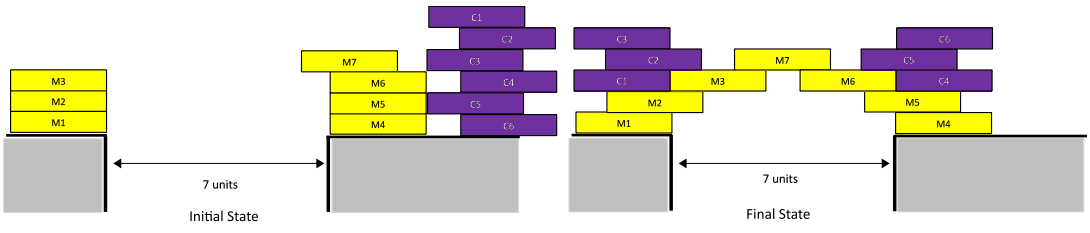


Figure 53: Stable construction of a 7 unit bridge.

9.11 Asymmetric Bridge Scenarios

These asymmetric bridge scenarios are similar to the bridge scenarios in terms of goal conditions. The only difference in these scenarios is that both sides are at different heights. The height difference between the sides is expressed by the rules in 5.

Scenario 17 This bridge construction scenario has 9 blocks with 4 being the counter weights Figure 54. Here right side is 2 units higher than left side and 4 units apart. Initial configuration is given by the following set of facts:

$init(S1, Table, 1, 1).init(M1, Table, 2, 1).init(M1, Table, 3, 2).init(S2, S1, 1, 1).$
$init(M2, M1, 1, 1).init(M2, M1, 2, 2).init(M2, M1, 3, 3).init(S3, S2, 1, 1).$
$init(M3, M2, 1, 1).init(M3, M2, 2, 2).init(M3, M2, 3, 3).init(S4, S3, 1, 1).$
$init(M4, Table, 8, 2).init(M4, Table, 9, 3).init(M5, M4, 1, 1).init(M5, M4, 2, 2).$
$init(M5, M4, 3, 3).$

The plan for the example in Figure 54 is as follows:

$0 : pick(Left, S1), pick(Right, M2).$
$1 : place(Left, S1, M1, 1, 1).place(Right, M2, M1, 2, 1).$
$2 : pick(Left, S2), pick(Right, M3).$
$3 : place(Left, S2, M2, 1, 1).place(Right, M3, M2, 2, 1).$
$4 : pick(Left, S3), pick(Right, M5).$
$5 : place(Left, S3, M3, 1, 1).place(Right, M5, M3, 3, 1).$

Scenario 18 This bridge construction scenario has 9 blocks with 4 being the counter weights Figure 55. Here right side is 4 units higher than left side

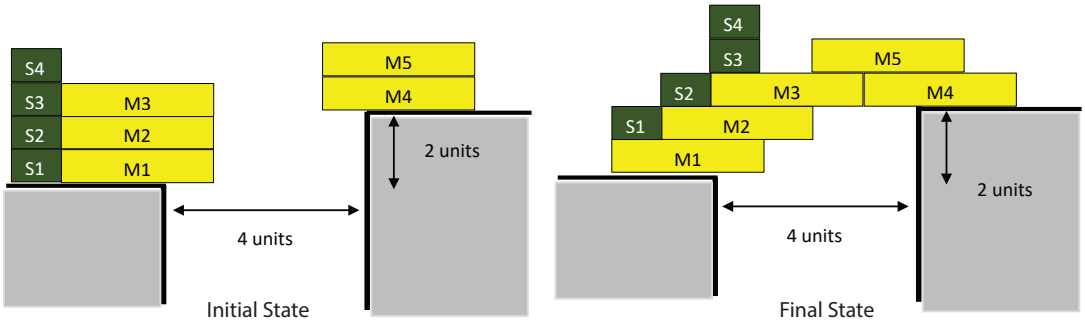


Figure 54: Stable construction of a 4 unit unleveled bridge.

and 5 units apart. Initial configuration is given by the following set of facts:

$init(C4,\ Table, 1, 1).init(C4,\ Table, 2, 2).init(C4,\ Table, 3, 3).init(S1,\ Table, 4, 1).$
$init(S2, S1, 1, 1).init(S3, S2, 1, 1).init(S4, S3, 1, 1).init(S5, S4, 1, 1).$
$init(S6, S5, 1, 1).init(S7, S6, 1, 1).init(M1,\ Table, 5, 1).init(M1,\ Table, 6, 2).$
$init(M1,\ Table, 7, 3).init(M2, M1, 1, 1).init(M2, M1, 2, 2).init(M2, M1, 3, 3).$
$init(M3, M2, 1, 1).init(M3, M2, 2, 2).init(M3, M2, 3, 3).init(M4, M3, 1, 1).$
$init(M4, M3, 2, 2).init(M4, M3, 3, 3).init(M5, M4, 1, 1).init(M5, M4, 2, 2).$
$init(M5, M4, 3, 3).$

The plan is as follows:

$0 : pick(Left, S6), pick(Right, M2).$
$1 : place(Left, S6, M1, 1, 1), place(Right, M2, M1, 2, 1).$
$2 : pick(Left, S5), pick(Right, M3).$
$3 : place(Left, S5, M2, 1, 1), place(Right, M3, M2, 2, 1).$
$4 : pick(Left, C4), pick(Right, S3).$
$5 : place(Left, C4, S7, 1, 2).$
$6 : pick(Left, M4), place(Right, S3, M3, 1, 1).$
$7 : pick(Right, S1), place(Left, M4, M3, 2, 1).$
$8 : pick(Left, M5), place(Right, S1, M4, 1, 1).$
$9 : place(Left, M5, M4, 3, 1).$

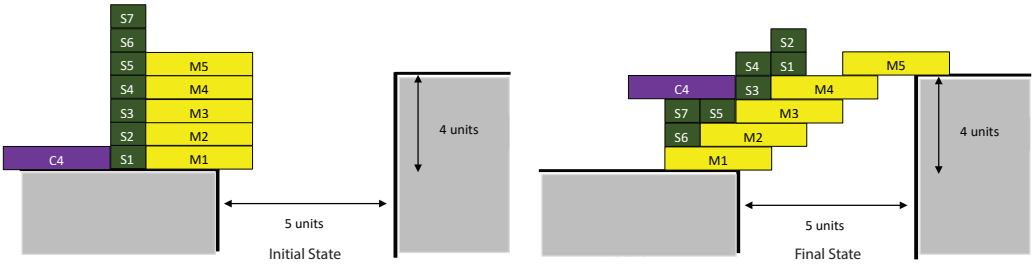


Figure 55: Stable construction of a 5 unit unleveled bridge.

9.12 Tower Stacking Benchmarks

In tower staking benchmarks the goal is to maximize or minimize the height of a particular box in a stack of given number of boxes. Initially all the boxes are on the table. Goal conditions are again given according to the rules in 5.

Scenario 19 This construction problem asks to maximize the height of box $S3$ in the stack. There are total 5 boxes in the stack. The plan for Figure 56 is as follows:

$0 : pick(Left, S1), pick(Right, M2).$
$1 : place(Left, S1, M1, 2, 1).$
$2 : pick(Left, S2), place(Right, M2, S1, 1, 2).$
$3 : place(Left, S2, M2, 2, 1), pick(Right, S3).$
$4 : place(Right, S3, S2, 1, 1).$

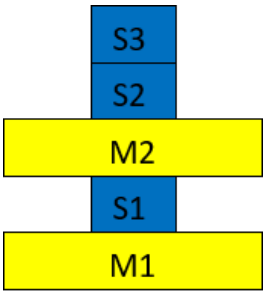


Figure 56: Maximizing the height of box s3 in a stack of 5 boxes

Scenario 20 This construction problem asks to maximize the height of box $S3$ in the stack. There are total 8 boxes in the stack. The plan for Figure 57 is as follows:

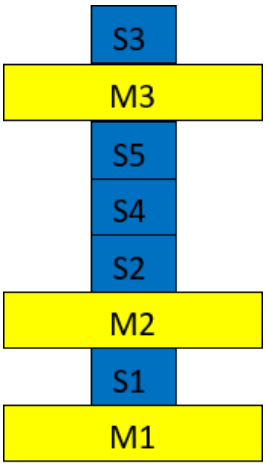


Figure 57: Maximizing the height of box s3 in a stack of 8 boxes

$0 : pick(Left, S1), pick(Right, M2).$
$1 : place(Left, S1, M1, 2, 1).$
$2 : pick(Left, S2), place(Right, M2, S1, 1, 2).$
$3 : place(Left, S2, M2, 2, 1), pick(Right, S4).$
$4 : place(Right, S4, S2, 1, 1), pick(Left, S5).$
$5 : place(Left, S5, S4, 1, 1), pick(Right, M3).$
$6 : place(Right, M3, S5, 1, 2), pick(Left, S3).$
$7 : place(Left, S3, M3, 2, 1).$

Scenario 21 This construction problem asks to minimize the height of box $S3$ in the stack. There are total 5 boxes in the stack. Please note $S3$ needs to be part of the stack that's why it cannot be placed directly on the ground/table. The plan for Figure 58 with two manipulators is as follows:

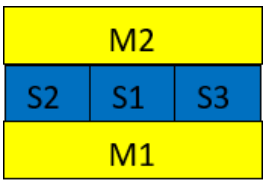


Figure 58: Minimizing the height of box s3 in a stack of 5 boxes

$0 : pick(Left, S1), pick(Right, S3).$
$1 : place(Left, S1, M1, 2, 1), place(Right, S3, M1, 3, 1).$
$2 : pick(Left, S2), place(Right, M2).$
$3 : place(Left, S2, M1, 1, 1), place(Right, M2, S1, 1, 2).$

The plan for Figure 58 with three manipulators is as follows:

$0 : pick(A1, S1), pick(A2, S3), pick(A3, S2).$
$1 : place(A1, S1, M1, 2, 1), place(A2, S3, M1, 3, 1),$
$place(A3, S2, M1, 1, 1).$
$2 : pick(A2, M2).$
$3 : place(A2, M2, S3, 1, 3).$

Scenario 22 This construction problem asks to minimize the height of box $S3$ in the stack. There are total 8 boxes in the stack. Please note $S3$ needs to be part of the stack that's why it cannot be placed directly on the ground/table. The plan for Figure 59 with two manipulators is as follows:

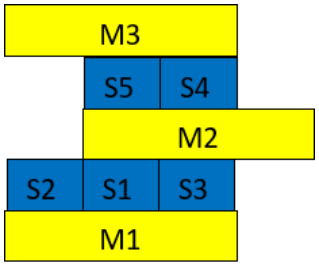


Figure 59: Minimizing the height of box s3 in a stack of 8 boxes

$0 : pick(Left, S1), pick(Right, S2).$
$1 : place(Left, S1, M1, 2, 1), place(Right, S2, M1, 1, 1).$
$2 : pick(Left, S3), pick(Right, M2).$
$3 : place(Left, S3, M2, 3, 1), place(Right, M2, S1, 1, 1).$
$4 : pick(Left, S5), pick(Right, S4).$
$5 : place(Left, S5, M2, 1, 1), place(Right, S4, M2, 2, 1).$
$6 : pick(Left, M3).$
$7 : place(Left, M3, S5, 1, 2).$

The plan for Figure 59 with three manipulators is as follows:

$0 : pick(A1, S1), pick(A2, S2), pick(A3, S3).$
$1 : place(A1, S1, M1, 2, 1), place(A2, S2, M1, 1, 1),$
$place(A3, S3, M1, 3, 1).$
$2 : pick(A1, S5), pick(A2, S4), pick(A3, M2).$
$3 : place(A3, M2, S3, 1, 2).$
$4 : place(A1, S5, M2, 1, 1), place(A2, S4, M2, 2, 1), pick(A3, M3).$
$5 : place(A3, M3, S4, 1, 3).$

9.13 Cylinder Scenarios

In cylinder scenarios, additional goal conditions with respect to the cylinders need to be specified.

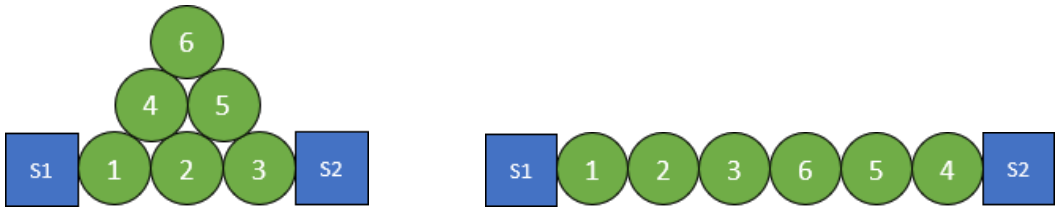
Scenario 23 The initial configuration in Figure 60 can be given by the following set of facts:

$init(S1, Table, 1, 1).init(S2, Table, 5, 1).$
$init\_cyl(1, Table, 2).init\_cyl(2, Table, 3).init\_cyl(3, Table, 4).$
$init\_top(4, 2, Topleft).init\_top(5, 2, TopRight).init\_top(6, 5, TopLeft).$

Here *on* predicate is initialized by *init*, *on_cyl* state is initialized by *init_cyl* and *top* state is initialized by *init_top*. Goal conditions for the above states are denoted by *goal*, *goal_cyl* and *goal_top* respectively. For this particular scenario it is defined as:

$goal(S1, Table).goal(S2, Table).$
$goal\_cyl(1, Table).goal\_cyl(2, Table).goal\_cyl(3, Table).$
$goal\_cyl(4, Table).goal\_cyl(5, Table).goal\_cyl(6, Table).$

The plan for Figure 60 is as follows:



(a) (b)

Figure 60: (a) Initial State (b) Goal State

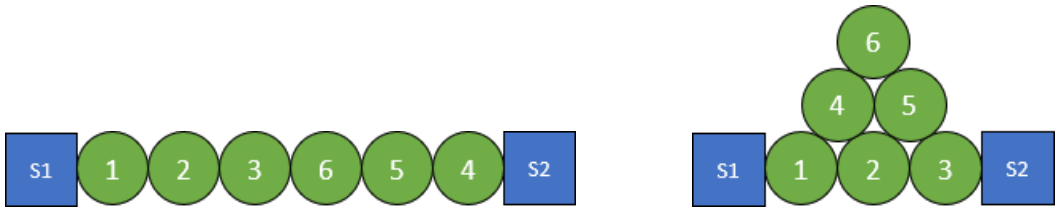
$0 : pick(Left, 6).$
$1 : place\_cyl\_on(Left, 6, Table, 5), push(Right, S2, Right).$
$2 : pick(Left, 5).$
$3 : place\_cyl\_on(Left, 5, Table, 6), push(Right, S2, Right).$
$4 : pick(Left, 4).$
$5 : place\_cyl\_on(Left, 4, Table, 7), push(Right, S2, Right).$

Scenario 24 The construction problem in Figure 61 is the opposite of Figure 60. The initial configuration can be given by the following set of facts:

$init(S1, Table, 1, 1).init(S2, Table, 8, 1).$
$init\_cyl(1, Table, 2).init\_cyl(2, Table, 3).init\_cyl(3, Table, 4).$
$init\_cyl(6, Table, 5).init\_cyl(5, Table, 6).init\_cyl(4, Table, 7).$

Goal conditions are defined by the following set of facts:

$goal(S1, Table).goal(S2, Table).$
$goal\_cyl(1, Table).goal\_cyl(2, Table).goal\_cyl(3, Table).$
$goal\_top(4, 2).goal\_top(4, 1).goal\_top(5, 2).$
$goal\_top(5, 3).goal\_top(6, 4).goal\_top(6, 5).$



(a) (b)

Figure 61: (a) Initial State (b) Goal State

The plan is as follows:

$0 : pick(Left, 4), push(Right, S2, Left).$
$1 : place\_cyl\_top(Left, 4, 1, TopRight).$
$2 : pick(Left, 5), push(Right, S2, Left).$
$3 : place\_cyl\_top(Left, 5, 2, TopRight).$
$4 : pick(Left, 6), push(Right, S2, Left).$
$5 : place\_cyl\_top(Left, 6, 5, TopLeft).$

9.14 Discussion

In this is subsection we discuss the results corresponding to the various instances discussed in the previous subsections.

Table 1 shows the results for scenarios 1-11. It lists the number of blocks, number of ASP rules, plan length and the timing information for the scenarios. By carefully observing the results, it is quite obvious that as the number of ASP rules increase, solving time also increases. Another observation is related to the type of boxes considered. Scenarios with medium and large boxes generally take more time as seen in scernario 8.

Table 1: Experimental evaluation of Scenario 1-11

Sc. No	**No. of Blocks T(S+M+L)**	**No. of Rules**	**Plan Length**	**Time (sec)**			
				Grounding	**Stability Check**	**Solving**	**Total**
1	10(7+2+1)	308735	2	3.875	3.254	20.128	27.257
2	4(3+0+1)	72728	6	0.259	12.08	0.907	13.246
3	4(3+1+0)	164817	5	1.427	19.806	7.841	29.074
4	6(5+0+1)	94375	4	0.905	8.557	3.968	13.430
5	5(3+1+1)	181241	7	1.606	20.022	11.041	32.669
6	5(4+0+1)	288686	7	2.719	14.184	34.105	51.008
7	4(0+3+1)	142163	2	1.043	2.573	2.725	6.341
8	9(7+0+2)	816740	4	7.378	48.735	1822.07	1878.183
9	4(2+1+1)	136035	4	1.259	12.197	4.866	18.322
10	6(3+2+1)	55316	4	4.818	15.819	400.736	421.373
11	5(3+1+1)	204566	4	1.774	39.028	125.759	166.561

Table 2: Experimental evaluation of overhang scenarios

Max Overhang (units)	**Memory (mb)**	**No. of Rules**	**Time(sec)**				**Plan Length**	**No. of Blocks T(S+M)**
			Grounding	**Solving**	**Stability Check**	**Total + Overhead**		
3	2214	276389	32.87	785.07	16.078	872.666	4	8(5+3)
4	6314	828995	480.026	14081.236	115.32	14951.132	6	7(0+7)
5	10462	1919177	1034.654	38951.03	303.47	42663.154	6	10(0+10)

Table 3: Experimental evaluation of bridge construction scenarios

Bridge Gap (units)	**Memory (mb)**	**No. of Rules**	**Time(sec)**				**Plan Length**	**No. of Blocks T(S+M)**
			Grounding	**Solving**	**Stability Check**	**Total + Overhead**		
3	1306	23357	0.209	0.377	0.361	0.997	1	3(0+3)
5	2798	376623	128.234	1175.298	44.213	1442.991	3	7(2+5)
7	29178	10329946	5932.126	185231.91	989.23	196934.266	7	13(0+13)
9	37256	11153744	7104.148	230146.18	1504.35	245171.678	9	15(6+9)

Table 4: Experimental evaluation of asymmetric bridge construction scenarios

Bridge Height (units)	**Memory (mb)**	**No. of Rules**	**Time(sec)**				**Plan Length**	**No. of Blocks T(S+M)**
			Grounding	**Solving**	**Stability Check**	**Total + Overhead**		
2	4326	592210	17.428	176.764	4.536	206.143	5	9(4+5)
3	8421	2633963	63.621	946.12	11.234	1066.103	7	11(5+6)
4	10964	3510906	94.3778	1387.98	18.923	1512.4228	9	13(7+6)

Table 2 shows the results for overhang scenarios. Results for overhang of 3,4 and 5 has been shown. Number of blocks used, memory consumption and the total time increases as the maximum overhang increases. Table 3 and Table 4 shows the results for symmetric and asymmetric bridge construction scenarios respectively. Here again the time required to compute the plan increases as the gap between bridges or the height difference between the sides increases.

Chapter 10

10 Execution

After the plan for a particular program has been computed using ASP and continuous domain motion trajectories for robot grippers are computed using OMPL library actual execution can be performed. We use a bimanual Baxter robot for the execution purposes. In this chapter, we will discuss aspects of executing the computed plans. Video link to the physical and dynamic simulation is as follows: `http://cogrobo.sabanciuniv.edu/?p=1111`

10.1 Robot

We are using a bimanual Baxter for execution of the computed planes. Baxter is a humanoid with two seven degrees of freedom arms. It also has force, position and torque sensors. Camera support is also available for vision purposes. Baxter has control at every joint of arm. This robot can only be used for manipulation purposes.

10.2 Blocks

We are using three kinds of blocks: small, medium and large; for testing purposes. The size of smallest block is $30 * 30 * 30$, the size of medium block is $90 * 30 * 30$ and the size of the largest block is $150 * 30 * 30$. All the blocks are aluminum made are spray painted with different colors for identification.

10.3 Dynamic Simulation

For dynamic simulation, we are using Gazebo version 7. The positions of all the boxes are known initially. After computing the plan, OMPL library is used to generate motion plan for every action in the plan and Moveit commander is used to actually follow that motion plan. Dynamic simulation of scenario 5 (see Figure 42). Initially all the boxes are on table and the

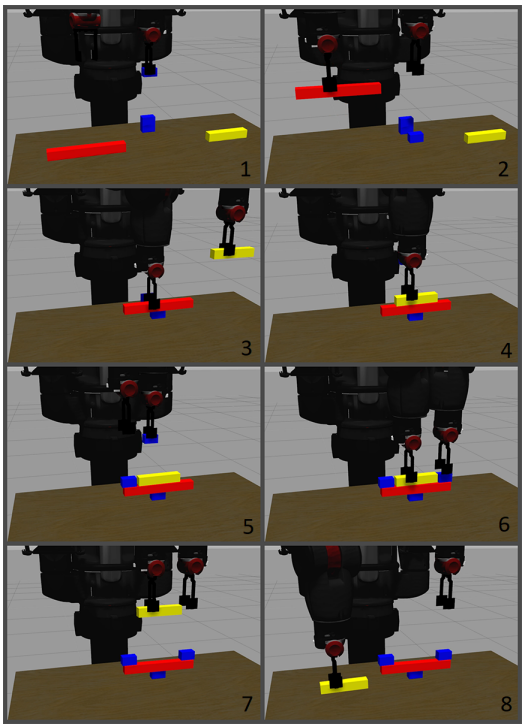


Figure 62: Snapshots present dynamic simulation of Scenario 5(benchmark: counterweight) with two grippers of a Baxter robot

robot picks up a small box (snapshot 1). The robot places the small box on table and picks the large box (snapshot 2). Robot places the large box on the small box and picks the medium box (snapshot 3). The robot places the medium box on the large box (snapshot 4). The robot places the a small box on the large box and picks up another small box (snapshot 5). The robot places the small box on the large box (snapshot 6). The robot picks up the

medium box (snapshot 7) and places it on the table (snapshot 8). In this scenario, medium box is used as a counter weight to prevent the structure from falling.

10.4 Physical Simulation

For physical simulation we use an actual baxter robot. We have also implemented execution monitoring algorithm to make sure that if failures occur during execution, the robot can recover from them. The details are mentioned in the next sub section.

10.5 Execution Monitoring

After plan for a particular program has been computed using ASP and continuous domain motion trajectories for robot grippers are computed using OMPL library, execution can be performed. We use a bimanual Baxter robot for the execution purposes. During execution there may be unexpected interventions, such as a human picking up a box and placing it somewhere else. To handle these interventions we have implemented an execution monitoring algorithm as in [27]. For the sake of simplicity we only consider only human intervention as the source of mismatch between actual state and planner history. It is also assumed that the robot is always successful in performing an action.

10.5.1 State Recognition

We use RGB camera of a Microsoft Kinect sensor to obtain visual information. Small boxes are *yellow* colored, medium boxes are *blue* colored and large boxes are *red* colored. Instead of monitoring at all times, we check for discrepancies after every time step. An important aspect of execution monitoring is to recognize the current state. In order to do so, we capture the current state using RGB camera and preprocess to convert it into *on* and

holding states. First the boxes are recognized in the image using color information. After this, the sizes of boxes are determined using pixel lengths. In order to determine if a box is on top of another box, we find out the contact area of all boxes. These contact areas are stored in pixel ranges along x and y directions. In order to check if a box is on top of another box, we see if the contact area of two boxes matches in the y direction, i.e. they share the same range of pixel values. Next in order to specify the exact location, we discretize the whole world by forming grid with smallest cell equal to the size of the smallest box. A large box is five times the size of small box and a medium box is three times the size of smallest box. Based on this difference we can easily find the exact units occupied by a box on top of another box. For *holding*, we simply check if the grippers have a box or subassembly in hand or not. Here we use color information again to differentiate among the boxes.

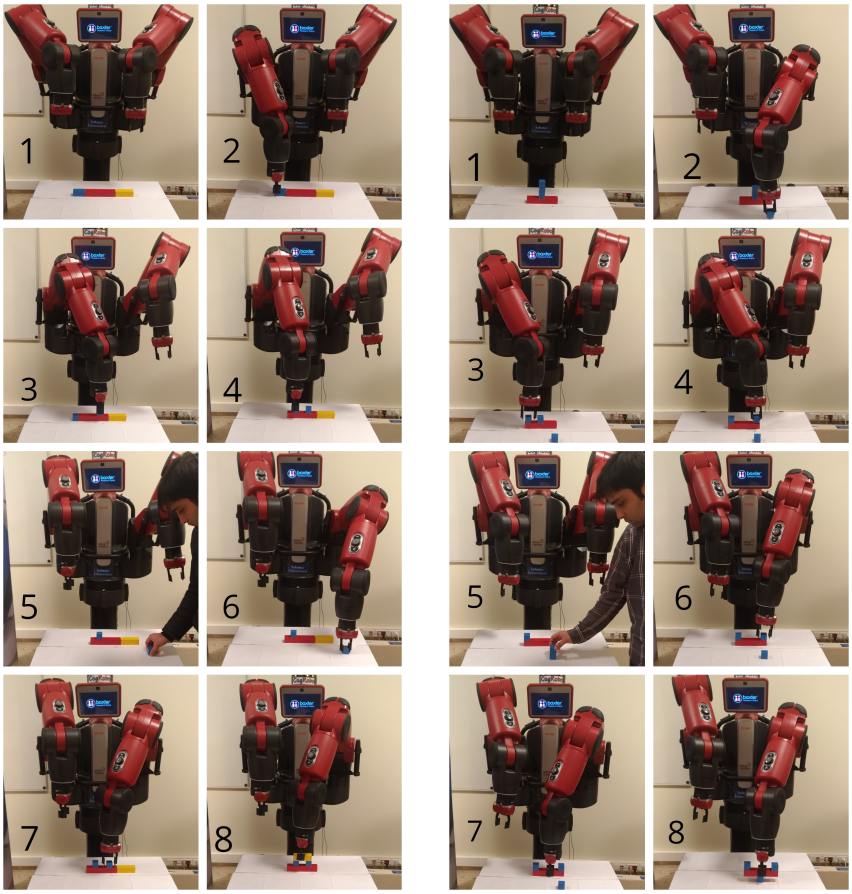
10.5.2 Discrepancy Check

After getting the current state information, the next step is to check if the current state information matches with the planner history. If it matches, the robot continues the execution of plan, but if it does not match then robot stops the execution because there exists some discrepancy between planner history and current state information. The algorithm sends the current state as an instance to the planner and computes a new plan from the current state and starts the execution again. Note that we only perform re-planning when a discrepancy occurs.

10.5.3 Alternative Approach

In the above mentioned approach we check for discrepancy at the beginning of each time step, this can be quite time consuming especially in cases when plan makespan is long and the human interferes with the state during the execution. In order to speed up the process, we have implemented an alter-

native approach in which the camera monitors for human intervention at all times. When a human arrives, the robot stops the execution immediately after saving the current trajectories and states. The robot waits for the human to leave the area and then checks for discrepancy. If there is no discrepancy, then the robot starts the executing the plan from the same saved state, but if there is some discrepancy the robot goes to home location which is some known location. New state information is gathered using the camera. Note if the robot is holding some boxes, they would be accommodated in *holding* state information and the rest of the boxes are accommodated in *on* predicate. This information is passed to planner for re-planning from these states. This approach saves time by checking for discrepancy only after intervention. Snapshots of physical simulations of scenario 2 (benchmark: subassembly) and scenario 9 (benchmark: stability) are shown in Figure 63:



(a) (b)

Figure 63: (a) Scenario 2 (b) Scenario 9.

In Figure 63(a): Initially the all the boxes are on the table (snapshot 1). Robot picks up a small box (snapshot 2) and the places it on the large box (snapshot 3). After that it places another small box on the large box (snapshot 4). The human intervenes and picks the small box and places it on the table (snapshot 5). The robot realizes that there is a discrepancy asks for another plan by passing the current state as initial state to the planner. After replanning, it picks up small box on the table (snapshot 6). It places small box on the large box (snapshot 7) and then places medium box on the small box (snapshot 8) to achieve the goal state.

In Figure 63(b): Initially boxes are stacked on the table (snapshot 1). The robot places a small box on the table (snapshot 2). After that it places another small box on the large box (snapshot 3). It adjusts the position of the last small box to make a balanced sub assembly (snapshot 4). At that point the human intervenes and changes the position of the small box to table (snapshot 5). The robot senses the discrepancy and asks for another plan from the current state. After replanning, the robot places the small box again on the large box again (snapshot 6). The robot picks the large box along with the small boxes on top of it (snapshot 7) and places it on the small box (snapshot 8) to achieve the goal state.

10.5.4 Discussion

The above mentioned approaches provide effective means to implement execution monitoring algorithm. The implementation of the algorithm can still be improved. Some failures may still take place while using the above mentioned approaches:

- Failure in detecting discrepancy. This occurs when the algorithm can not detect the mismatch between actual state and planner history.
- Incorrect discrepancy detection. This occurs when the algorithm detects there is a mismatch between actual state and planner history, although no human intervention actually took place. The primary cause of this failure is due to discretization as the positions of boxes are decided based on their pixel ranges. An incorrect position leads to this failure.

Chapter 11

11 Conclusion and Future Work

11.1 Conclusion

We have introduced a formal framework to address multi-robot construction problems that are challenging for both AI and Robotics not only due to modeling challenges (e.g., due to ramifications of manipulation actions, true concurrency of actions, supportedness of blocks by other blocks), but also due to necessity of stability checks of constructions as they are being built. We address these challenges by a general hybrid planning framework developed over the logic-based formalism and automated reasoners of Answer Set Programming (ASP): ASP allows true concurrency, embedding outcomes of stability checks into state constraints by external atoms, recursive definitions of sophisticated concepts, like supportedness and connectedness, and nested recursive definitions of global positions of blocks from their relative positions.

We introduce a set of challenging robot construction benchmark instances that include bridges and overhangs constructed with counterweights, scaffolding and true concurrency of manipulations. We have shown the applications of our approach over various instances of these benchmarks.

We have also introduced the extension of our framework to cylindrical objects and demonstrated the applicability of our approach using dynamics and physical simulations on bi-manual Baxter robot.

11.2 Future Work

The applicability of the framework presented in this thesis has been shown by solving several instances of the proposed benchmarks. In all the instances, the framework has been successful in solving the problem. The framework can still be extended as follows:

In this work, we have considered only regularly shaped objects such as rectangles and cylinders. In real world, many objects are irregularly shaped. Although irregular shaped object can be converted into regular shaped object by carefully dividing the structure, this kind of division is not always possible or efficient. Generalization to irregular shaped objects poses lots of challenges because of the inherent unknown geometry of the object under consideration.

Another future direction could be improving the stability checks. Nowadays, the state of the art construction methods rely on shake tables to check the stability of a structure subject to simulated earthquakes. One extension to the stability check may be designing a standard stability check to test the robustness of the structure by simulating it under real recorded earthquakes.

Another extension to the framework can be in terms of the computation times. We have used DLVHEX to model the robot construction problems due to its expressiveness when external atoms are concerned. However, this expressiveness comes with cost of long construction times, as DLVHEX takes a lot of time to compute plans as seen in the results section. One extension can be to try out the framework with other state of the art grounders and answer set solvers.

References

[1] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Manipulation with multiple action types," in *Experimental Robotics*. Springer, 2013, pp. 531–545.

[2] L. Beyeler, J.-C. Bazin, and E. Whiting, "A graph-based approach for discovery of stable deconstruction sequences," in *Advances in Architectural Geometry*. Springer, 2015, pp. 145–157.

[3] M. Blum, A. Griffith, and B. Neumann, "A stability test for configurations of blocks," *AIM*, 1970.

[4] T. Bock, "Construction automation and robotics," in *Robotics and Automation in Construction*. InTech, 2008.

[5] N. Boneschanscher, H. van der Drift, S. J. Buckley, and R. H. Taylor, "Subassembly stability," in *AAAI*, vol. 88, 1988, pp. 780–785.

[6] G. Brewka, T. Eiter, and M. Truszczynski, "Answer set programming at a glance," *Commun. ACM*, vol. 54, no. 12, pp. 92–103, 2011.

[7] F. Calimeri, M. Fink, S. Germano, A. Humenberger, G. Ianni, C. Redl, D. Stepanova, A. Tucci, and A. Wimmer, "Angry-hex: An artificial player for angry birds based on declarative knowledge bases," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 8, no. 2, pp. 128–139, 2016.

[8] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4627–4632.

[9] E. D. Demaine, M. L. Demaine, M. Hoffmann, and J. O'Rourke, "Pushing blocks is hard," *Comput. Geom.*, vol. 26, no. 1, pp. 21–36, 2003.

[10] S. Derhami, J. S. Smith, and K. R. Gue, "Optimising space utilisation in block stacking warehouses," *International Journal of Production Research*, pp. 1–17, 2016.

[11] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, vol. 33, no. 3, pp. 217–236, 2012.

[12] T. Eiter, M. Mehuljic, C. Redl, and P. Schüller, "User guide: dlvhex 2.x," Vienna University of Technology, Institute for Information Systems, Tech. Rep. INFSYS RR-1843-15-05, September 2015.

[13] E. Erdem, M. Gelfond, and N. Leone, "Applications of answer set programming," *AI Magazine*, vol. 37, no. 3, pp. 53–68, 2016.

[14] E. Erdem and V. Patoglu, "Applications of asp in robotics," *KI-Künstliche Intelligenz*, pp. 1–7, 2018.

[15] C. Erdogan and M. Stilman, "Planning in constraint space: Automated design of functional structures," in *IEEE International Conference on Robotics and Automation*. IEEE, 2013.

[16] R. Fagin, "Monadic generalized spectra," *Mathematical Logic Quarterly*, vol. 21, no. 1, pp. 89–96.

[17] S. E. Fahlman, "A planning system for robot construction tasks," *Artificial intelligence*, vol. 5, no. 1, pp. 1–49, 1974.

[18] ——, "A planning system for robot construction tasks," *Artificial intelligence*, vol. 5, no. 1, pp. 1–49, 1974.

[19] L. Ferreira and C. Toledo, "Generating levels for physics-based puzzle games with estimation of distribution algorithms," in *ACM Conference on Advances in Computer Entertainment Technology*, 2014, p. 25.

[20] ——, "A search-based approach for generating angry birds levels," in *Proc. of CIG*, 2014, pp. 1–8.

[21] F. Furrer, M. Wermelinger, H. Yoshida, F. Gramazio, M. Kohler, R. Siegwart, and M. Hutter, "Autonomous robotic stone stacking with online next best object target pose planning," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 2350–2356.

[22] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, vol. 9, pp. 365–385, 1991.

[23] N. Gupta and D. S. Nau, "On the complexity of blocks-world planning," *Artificial Intelligence*, vol. 56, no. 2-3, pp. 223–254, 1992.

[24] J. F. Hall, "Fun with stacking blocks," *American journal of physics*, vol. 73, no. 12, pp. 1107–1116, 2005.

[25] S. D. Han, N. M. Stiffler, K. E. Bekris, and J. Yu, "Autonomous design of functional structures," *Advanced Robotics*, vol. 29, no. 9, pp. 625–638, 2015.

[26] ——, "Efficient, high-quality stack rearrangement," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1608–1615, 2018.

[27] G. Havur, K. Haspalamutgil, C. Palaz, E. Erdem, and V. Patoglu, "A case study on the tower of hanoi challenge: Representation, reasoning and execution," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4552–4559.

[28] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu, "Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach," in *IEEE International Conference on Robotics and Automation*. IEEE, 2014, pp. 445–452.

[29] Z. Jia, A. Gallagher, A. Saxena, and T. Chen, "3d-based reasoning with blocks, support, and stability," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1–8.

[30] Z. Jia, A. C. Gallagher, A. Saxena, and T. Chen, "3d reasoning from blocks to stability," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 5, pp. 905–918, 2015.

[31] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[32] A. Krontiris, R. Shome, A. Dobson, A. Kimmel, and K. Bekris, "Rearranging similar objects with a manipulator using pebble graphs," in *IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 1081–1087.

[33] A. Krontiris and K. E. Bekris, "Dealing with difficult instances of object rearrangement," in *Robitics: Science and Systems*, 2015.

[34] ——, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner," in *IEEE International Conference on Robotics and Automation*, 2016, pp. 3924–3931.

[35] S. Lee and Y. G. Shin, "Assembly planning based on geometric reasoning," *Computers & graphics*, vol. 14, no. 2, pp. 237–250, 1990.

[36] V. Lifschitz, "What is answer set programming?" in *Proc. of AAAI*. MIT Press, 2008, pp. 1594–1597.

[37] ——, "Answer set programming and plan generation," *Artif. Intell.*, vol. 138, no. 1-2, pp. 39–54, 2002.

[38] R. K. Livesley, "Limit analysis of structures formed from rigid blocks," *International Journal for Numerical Methods in Engineering*, vol. 12, no. 12, pp. 1853–1871, 1978.

[39] ——, "A computational model for the limit analysis of three-dimensional masonry structures," *Meccanica*, vol. 27, no. 3, pp. 161–172, 1992.

[40] P. Lourenço, P. Roca, C. Modena *et al.*, "Limit analysis of three-dimensional masonry structures," *Structural Analysis of Historical Construction, Vol 2 (Set of 3 Volumes): Possibilities of Numerical and Experimental Techniques*, vol. 2, p. 1107, 2006.

[41] R. Mattikalli, D. Baraff, and P. Khosla, "Finding all stable orientations of assemblies with friction," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 2, pp. 290–301, 1996.

[42] R. Mattikalli, D. Baraff, P. Khosla, and B. Repetto, "Gravitational stability of frictionless assemblies," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 374–388, 1995.

[43] R. Mojtahedzadeh, A. Bouguerra, E. Schaffernicht, and A. J. Lilienthal, "Support relation analysis and decision making for safe robotic manipulation tasks," *Robotics and Autonomous Systems*, vol. 71, pp. 99–117, 2015.

[44] H. Mosemann, F. Röhrdanz, and F. Wahl, *Stability of assemblies as a criterion for cost evaluation in robot assembly.* Springer, 1998, pp. 157–168.

[45] H. Mosemann, F. Rohrdanz, and F. M. Wahl, "Stability analysis of assemblies considering friction," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 6, pp. 805–813, 1997.

[46] N. Napp and R. Nagpal, "Distributed amorphous ramp construction in unstructured environments," *Robotica*, vol. 32, no. 2, pp. 279–290, 2014.

[47] K. Okada, A. Haneda, H. Nakai, M. Inaba, and H. Inoue, "Environment manipulation planner for humanoid robots using task graph that generates action sequence," in *IEEE IROS*, vol. 2, 2004, pp. 1174–1179.

[48] R. S. Palmer, "Computational complexity of motion and stability of polygons," Cornell University, Tech. Rep., 1989.

[49] J.-S. Pang and J. Trinkle, "Stability characterizations of rigid body contact problems with coulomb friction," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 80, no. 10, pp. 643–663, 2000.

[50] M. Paterson, Y. Peres, M. Thorup, P. Winkler, and U. Zwick, "Maximum overhang," *The American Mathematical Monthly*, vol. 116, no. 9, pp. 763–787, 2009.

[51] M. Paterson and U. Zwick, "Overhang," in *ACM-SIAM symposium on Discrete algorithm*, 2006, pp. 231–240.

[52] ——, "Overhang," *The American Mathematical Monthly*, vol. 116, no. 1, pp. 19–44, 2009.

[53] S. Rakshit and S. Akella, "The influence of motion paths and assembly sequences on the stability of assemblies," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 615–627, 2015.

[54] F. Röhrdanz, H. Mosemann, and F. Wahl, "Generating and evaluating stable assembly sequences," *Advanced robotics*, vol. 11, no. 2, pp. 97–126, 1996.

[55] J. M. Schimmels and M. A. Peshkin, "Force-assembly with friction," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 4, pp. 465–479, 1994.

[56] B. Schmult, "Autonomous robotic disassembly in the blocks world," *International Journal of Robotics Research*, vol. 11, no. 5, pp. 437–459, 1992.

[57] T. Shao, A. Monszpart, Y. Zheng, B. Koo, W. Xu, K. Zhou, and N. J. Mitra, "Imagining the unseen: Stability-based cuboid arrangements for scene understanding," *ACM Transactions on Graphics*, vol. 33, no. 6, 2014.

[58] P. D. Spanos, P. C. Roussis, and N. P. Politis, "Dynamic analysis of stacked rigid blocks," *Soil Dynamics and Earthquake Engineering*, vol. 21, no. 7, pp. 559–578, 2001.

[59] M. Stephenson and J. Renz, "Procedural generation of complex stable structures for angry birds levels," in *IEEE Conference on Computational Intelligence and Games*, 2016, pp. 1–8.

[60] ——, "Procedural generation of complex stable structures for angry birds levels," in *Proc. of CIG*, 2016, pp. 1–8.

[61] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[62] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics*, vol. 2, no. 04, pp. 479–503, 2005.

[63] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proceedings of ICRA 2007*. IEEE, 2007, pp. 3327–3332.

[64] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[65] V. Thangavelu, Y. Liu, M. Saboia, and N. Napp, "Dry stacking for automated construction with irregular objects," *Artificial intelligence*, 2018.

[66] S. Thiébaux, J. Hoffmann, and B. Nebel, "In defense of pddl axioms," *Artifiical Intelligence*, vol. 168, no. 1–2, pp. 38–69, 2005.

[67] K. K. Thomsen and M. Kraus, "Simulating small-scale object stacking using stack stability," in *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2015, pp. 5–8.

[68] M. Toussaint and M. Lopes, "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 4044–4051.

[69] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning." in *IJCAI*, 2015, pp. 1930–1936.

[70] P. A. Wałega, M. Zawidzki, and T. Lechowski, "Qualitative physics in angry birds," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 2, pp. 152–165, 2016.

[71] P. A. Wałega, M. Zawidzki, and J. Mozaryn, "Qualitative evaluation of stability in disassembling block structures with robot manipulator," *Association for the Advancement of Artificial Intelligence*, 2015.

[72] W. Wan, K. Harada, and K. Nagata, "Assembly sequence planning for motion planning," *Assembly Automation*, vol. 38, no. 2, pp. 195–206, 2018.

[73] J. Wang, P. Rogers, L. Parker, D. Brooks, and M. Stilman, "Robot jenga: Autonomous and strategic block extraction," in *IEEE/RSJ Inter-*

national Conference on Intelligent Robots and Systems, 2009, pp. 5248–5253.

[74] E. Whiting, J. Ochsendorf, and F. Durand, "Procedural modeling of structurally-sound masonry buildings," in *ACM Transactions on Graphics*, vol. 28, no. 5. ACM, 2009, p. 112.

[75] E. J. W. Whiting, "Design of structurally-sound masonry buildings using 3d static analysis," Ph.D. dissertation, Massachusetts Institute of Technology, 2012.

[76] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proceedings of the Fourth Annual Symposium on Computational Geometry*, ser. SCG '88, 1988, pp. 279–288.

[77] R. H. Wilson and J.-C. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence*, vol. 71, no. 2, pp. 371–396, 1994.

[78] T. Winograd, "Understanding natural language," *Cognitive Psychology*, vol. 3, no. 1, pp. 1 – 191, 1972.

[79] U. Zwick, "Jenga," in *ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 243–246.