Assembly Planning in Constrained Environments: Building Structures with Multiple Mobile Robots

Frederik W. Heger
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/dissertations

Recommended Citation
Assembly Planning in Constrained Environments:
Building Structures with Multiple Mobile Robots

Frederik W. Heger
CMU-RI-TR-10-34

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

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

August 20, 2010

Thesis Committee:
Sanjiv Singh, Chair
Reid Simmons
James Kuffner
Randy Wilson (Google, Inc.)

© 2010 by Frederik W. Heger. All rights reserved.
ABSTRACT

Assembly is a task at which robots excel, but only as long as they operate in well-controlled environments such as factories or assembly lines. This thesis presents a comprehensive planning and execution framework that enables mobile manipulator robots to overcome this limitation and assemble large structures in physically challenging environments. In order to be useful in realistic scenarios, the developed system must be robust to inevitably occurring execution-time failures, generate plans efficiently and scale to large problems including complex structures being assembled by multiple robots.

Complex tasks today are approached in one of two fundamental ways: as a problem of sequencing abstract tasks, or as fine-grained motion planning problems. The prior has no problem reasoning about large problems, but lacks understanding of the physical context of the scenario, whereas the latter considers physical reality but lacks the scope for complex scenarios. This thesis develops a graph-based representation that exploits synergies between both methodologies to solve large, physically grounded problems.

Beyond planning complex assembly tasks, the fundamental challenge for a useful system lies in its ability to successfully execute the planned task even when unforeseen problems occur during operation. Comprehensive repair and re-planning strategies are essential to avoid terminal failures, and ultimate robustness is achieved by allowing an expert operator to assist with error recovery at the behavioral, executive and planning levels as appropriate.

Once assembly plans can be executed reliably, the efficiency of plan generation can be improved both when finding the initial plan, as well as when repair and re-planning become necessary. The techniques presented in this thesis reduce the time required to generate an assembly plan by 2.5 orders of magnitude and more compared to a baseline solution performing best-first search over possible assembly steps. These improvements make the system practical to be applied to larger problems. Hierarchical problem decomposition, as well as computational and physical parallelization, allow the framework to scale to large structures of hundreds or thousands of components being assembled by multiple robots.

The comprehensive assembly planning and execution framework developed in this thesis pushes the boundary of what mobile manipulator robots can accomplish in realistic environments. It provides the high-level intelligence necessary to apply low-level capabilities to very large problems, and it includes error recovery strategies to cope with inevitable exceptions while performing complex and challenging tasks.
ACKNOWLEDGMENTS

I would like to thank my advisor Sanjiv Singh for his guidance and support throughout my time at Carnegie Mellon. Thank you also to my committee members Reid Simmons, James Kuffner and Randy Wilson. Their alternative points of view helped me see aspects of my work I had not considered before and ultimately made it stronger and more complete. Special thanks goes to Randy - having someone who is not a professor as my external committee member gave me a unique perspective and the fastest draft edit and feedback turn-around time ever!

No robotics project - my work included - is ever accomplished by an individual person. It takes a team to cover all necessary areas of hardware, software and integration. During my time at Carnegie Mellon I was fortunate to work with many excellent colleagues. Without their (at times completely unrelated) efforts, this dissertation would not have been possible. In no particular order, many thanks to Brennan Sellner, Laura Hiatt, Nik Melchior, Brad Hamner, Seth Koterba, Kartik Babu, Joseph Lisee, Heather Justice, Greg Armstrong, and many more.

I would also like to acknowledge the support of the sponsors for the various projects I was involved in: NASA for their support of the DIRA, Trestle, IDSR and Cluster projects, as well as GM for their support of Ace and the opportunity to spend a summer with the Manufacturing Systems Research Lab at the R&D Center in Warren, MI. Throughout these projects I was able to interact with many excellent researchers, among them teams at TRACLabs, NASA’s Johnson Space Center, JPL, Ames and GM.

I made some great friends in Pittsburgh, both at CMU and outside, and I am sure our paths will cross many times in the future. There are too many of them for me to list, but they (and I) know who they are.

More so than for anyone mentioned above, this dissertation would not have happened without the continued support from my parents. They let me go on an adventure to study abroad many years ago. Nobody at the time thought that a one year exchange program would last many times longer, and I am grateful beyond words that they let me run with it all the way to this point, and beyond.

Going forward, I am very fortunate to share my life with my wife Megan. She has supported me throughout my graduate career, made sure I kept a reasonable schedule, rowed with me, and just been a steady and wonderful part of my life.
DEDICATION

Für Opa Willi, Oma Rosa, Opa Heiner & Oma Ottitlie
# Contents

1 Introduction ................................................................. 1
  1.1 Overview .................................................................. 1
  1.2 The Assembly Planning Problem ................................ 2
    1.2.1 Challenges ....................................................... 2
    1.2.2 Scope and Limitations ....................................... 3
  1.3 Applications and Significance ................................. 3
    1.3.1 Robotic Assembly in Space ................................. 4
    1.3.2 Military and Logistics Applications ..................... 5
    1.3.3 Domestic Applications ...................................... 6
  1.4 Thesis Statement ....................................................... 7
  1.5 Approach Sketch ...................................................... 8
    1.5.1 Assembly Graph Representation ......................... 8
    1.5.2 Robust Execution ............................................. 9
    1.5.3 Efficient Plan Generation .................................. 10
    1.5.4 Scaling to Large Structures and Multiple Assembly Robots ............................................... 11
  1.6 Document Road Map .................................................. 12

2 Background ................................................................. 13
  2.1 Overview .................................................................. 13
  2.2 Thesis Context ........................................................ 14
    2.2.1 State of the Art ............................................... 14
    2.2.2 Challenges ...................................................... 15
    2.2.3 Scope and Assumptions ..................................... 15
  2.3 Related Work .......................................................... 16
    2.3.1 Symbolic Planning ............................................. 16
    2.3.2 Assembly Planning ............................................ 18
    2.3.3 Motion Planning ............................................... 24
    2.3.4 Other Related Work ......................................... 27
    2.3.5 Shortcoming of Existing Approaches ................... 28
## CONTENTS

2.4 Summary ........................................... 31

3 Basic Assembly Planning Approach .......................... 34
  3.1 Overview ........................................... 34
  3.2 Terminology ........................................ 35
  3.3 Representation ...................................... 36
    3.3.1 Assembly Graph ................................. 37
    3.3.2 Task Templates ................................. 45
  3.4 Plan Generation ..................................... 46
    3.4.1 Search for Assembly Sequence .................. 46
    3.4.2 Task Parameterization ............................ 50
    3.4.3 Completeness and Correctness .................... 50
  3.5 Plan Execution ..................................... 53
    3.5.1 Nominal Operation ............................... 53
    3.5.2 Execution-Time Exceptions ....................... 54
  3.6 Summary ........................................... 56

4 Robust Robotic Assembly ................................ 58
  4.1 Overview ........................................... 58
  4.2 Motivation ......................................... 58
  4.3 Executing Assembly Plans Robustly .................... 59
    4.3.1 Typical Execution-Time Exceptions ............... 59
    4.3.2 Autonomous Exception Handling .................. 62
    4.3.3 Sliding Autonomy During Plan Execution ........ 69
    4.3.4 Results ........................................ 72
    4.3.5 Effects on Completeness and Correctness .......... 79
  4.4 Planning for Robustness ................................ 80
    4.4.1 Metrics for Robust Assembly Sequences .......... 80
    4.4.2 Results ........................................ 82
    4.4.3 Effects on Completeness and Correctness .......... 83
  4.5 Summary ........................................... 83

5 Efficient Plan Generation .................................. 85
  5.1 Overview ........................................... 85
  5.2 Motivation ......................................... 85
  5.3 Increasing Edge Evaluation Efficiency ................ 88
    5.3.1 Motion Planning with Approximations ............ 88
    5.3.2 Deferred Path Smoothing .......................... 92
    5.3.3 Clustering Similar Assembly Graph Edges .......... 95
    5.3.4 Effects on Completeness and Correctness .......... 99
  5.4 Reducing the Number of Graph Edges to Evaluate ....... 99
    5.4.1 Heuristic Search ................................. 99
    5.4.2 Effects on Completeness and Correctness .......... 102
## CONTENTS

5.5 Combined Efficiency Improvement Strategies ........................................ 103
5.6 Summary ......................................................................................... 103

6 Larger Structures and Multiple Robots ................................. 108
6.1 Overview ..................................................................................... 108
6.2 Motivation .................................................................................... 108
6.3 Scaling to Larger Structures ............................................................. 110
   6.3.1 Approach ................................................................................. 110
   6.3.2 Implementation ................................................................. 113
   6.3.3 Results .................................................................................. 117
   6.3.4 Effects on Completeness and Correctness ....................... 117
6.4 Scaling to Additional Assembly Robots .................................................. 118
   6.4.1 Approach ................................................................................. 118
   6.4.2 Implementation ................................................................. 122
   6.4.3 Results .................................................................................. 124
   6.4.4 Effects on Completeness and Correctness ....................... 126
6.5 Summary ......................................................................................... 126

7 Conclusions ..................................................................................... 127
7.1 Summary ..................................................................................... 127
   7.1.1 Robust Execution ................................................................. 127
   7.1.2 Efficient Planning ................................................................. 128
   7.1.3 Scaling to Larger Structures and Additional Robots ............ 129
7.2 Contributions ................................................................................ 130
7.3 Future Work .................................................................................. 131
   7.3.1 Comprehensive Operator Interaction ............................... 131
   7.3.2 Large Structure Decomposition .......................................... 132
   7.3.3 Implementation Improvements ........................................... 133
7.4 Closing Remarks .......................................................................... 134

Bibliography ................................................................................... 135
## List of Figures

1.1 Robotic assembly scenarios in space ................................................. 4  
1.2 Robotic materials handling scenarios .............................................. 6  
1.3 Domestic applications of robotic assembly ....................................... 6  
1.4 Pictorial overview of main thesis content ....................................... 12  

2.1 Space of existing approaches to assembly planning ............................. 17  
2.2 This thesis in the space of related work .......................................... 29  
2.3 Shortcomings of symbolic planning in the context of assembly planning  30  
2.4 Shortcomings of motion planning in the context of assembly planning .... 31  

3.1 Assembly graph vertices ................................................................. 37  
3.2 Structure that cannot be represented by assembly graph ......................... 38  
3.3 Complete assembly graph ............................................................... 40  
3.4 Representative assembly graph sizes ................................................. 41  
3.5 Assembly graph construction .......................................................... 43  
3.6 Generic assembly task template ....................................................... 45  
3.7 Planning steps involved in assembly graph edge evaluation .................... 49  
3.8 Assembly that requires intermediate structure repositioning ................... 51  
3.9 Assembly that requires assembly via sub-assemblies ........................... 52  
3.10 Assembly planning system overview ............................................... 54  

4.1 Execution exceptions due to sensing problems .................................... 61  
4.2 Execution exceptions due to manipulation problems ............................ 62  
4.3 Exception resolved via contingency response  ................................... 63  
4.4 Contingency exception handling flow chart ....................................... 64  
4.5 Assembly step repair ................................................................. 65  
4.6 Exception resolved via step repair .................................................. 66  
4.7 Step repair exception handling flow chart ....................................... 67  
4.8 Assembly sequence re-plan ......................................................... 68  
4.9 Exception resolved via sequence re-plan ....................................... 70
## LIST OF FIGURES

4.10 Sequence re-plan exception handling flow chart ........................................ 71  
4.11 Overview of robust execution results ......................................................... 73  

5.1 Representative structures (8, 13 and 21 components) ............................... 86  
5.2 Experimental setup for testing approximated motion planning .................... 90  
5.3 Clustering similar assembly graph edges ..................................................... 96  
5.4 Straight-line heuristic to guide search of assembly graph .......................... 100  
5.5 Summary of combined efficiency improvements for single square structure .... 104  
5.6 Summary of combined efficiency improvements for two-square structure ...... 105  
5.7 Summary of combined efficiency improvements for four-square structure ...... 106  

6.1 Hierarchical decomposition of large structures. .......................................... 109  
6.2 Thought experiment: Why simply extending the standard approach to assembly planning does not scale to larger structures or additional robots ............ 110  
6.3 Increasingly larger sub-assemblies used to assemble planetary outposts from individual nodes and beams ................................................................. 111  
6.4 Outpost assembly scenario ........................................................................... 112  
6.5 Components in outpost ................................................................................ 115  
6.6 Solution outpost plan .................................................................................. 116  
6.7 Separable and non-separable outposts ......................................................... 119  
6.8 Dependency diagram for fully parallelizable outposts ................................ 120  
6.9 Dependency diagram for outposts requiring serialization .......................... 121  
6.10 Dependency diagram for outposts scheduled as a whole ......................... 122  
6.11 Alternative schedules for assembling an outpost with the minimum number of robots (two robots) .............................................................. 123
## List of Tables

2.1 Comparison of related work capabilities as relevant to this thesis ............... 33
4.1 Execution robustness without exception handling .............................. 74
4.2 Execution robustness with autonomous exception handling only ............... 76
4.3 Execution robustness with Sliding Autonomy exception handling ............... 78
4.4 Benefits of explicitly planning for robustness ................................. 82
5.1 Assembly graph properties for representative structures ....................... 87
5.2 Baseline assembly planner performance ........................................... 88
5.3 Motion planning with approximations in less constrained environment ......... 91
5.4 Motion planning with approximations in more constrained environment ....... 91
5.5 Efficiency improvements due to deferred smoothing (single square structure) .. 93
5.6 Efficiency improvements due to deferred smoothing (two-square structure) .... 94
5.7 Efficiency improvements due to deferred smoothing (four-square structure) .... 94
5.8 Efficiency improvements due to clustering (single square structure) .......... 97
5.9 Efficiency improvements due to clustering (two-square structure) ............. 97
5.10 Efficiency improvements due to clustering (four-square structure) .......... 98
5.11 Efficiency improvements due to heuristic search (single-square structure) .... 101
5.12 Efficiency improvements due to heuristic search (two-square structure) ....... 102
5.13 Efficiency improvements due to heuristic search (four-square structure) ....... 102
6.1 Assembly graphs at different levels of the planning hierarchy .................. 114
6.2 Makespan and robot idle time results of alternative outpost schedules .......... 125
6.3 Planning and scheduling time results ............................................. 125
List of Algorithms

3.1 Assembly graph generation ............................................. 42
3.2 Assembly graph expansion ............................................. 44
3.3 Assembly graph search .................................................. 47
3.4 Graph edge evaluation ................................................. 48
Chapter 1

Introduction

1.1 Overview

Consider a construction site where several buildings are being erected among already existing structures. Large machines and numerous workers have to cooperate to accomplish such a project. Before they can start, someone (e.g., an architect) plans who has to do what when and where. During construction, the workers will likely discover problems (some minor, other more serious) that require them to reconsider parts of their tasks to ensure that they reach their goal and successfully complete the project as desired.

The above example is a common scenario today that does not involve any robots: humans solving complex problems characterized by task ordering, environmental and resource constraints. This thesis develops technology necessary to enable mobile manipulator robots to perform tasks of equal (and greater) complexity with little human assistance. Such applications are a far cry from typical uses of robots to assemble things – in factories and on assembly lines – but they are crucial in improving the usefulness and capabilities of robots toward performing physically meaningful tasks in challenging environments.

Robotics technology has reached a point where the machines can begin to address (both mechanically and computationally) challenging problems that require more than just sensing and navigating through the environment. Specifically, they are now becoming capable of actively manipulating their surroundings by moving obstacles or assembling structures. This thesis focuses on the latter scenario, where multiple mobile manipulator robots cooperate to construct large structures in challenging environments. It develops the first comprehensive assembly planning and execution framework that is capable of generating plans efficiently, executing them robustly and solving problems involving large structures and many robots. With numerous real-world applications in the space, military and domestic domains, robotic assembly of structures lies at the intersection of several active areas of research, including task- and motion planning, manipulation, etc. Yet no other system today is capable of comprehensively planning, coordinating and monitoring large structure assembly tasks for multiple robots robustly, efficiently and in a scalable manner.

To date, assembly planning has been approached in one of two fundamental ways: as a task
1.2 The Assembly Planning Problem

This thesis addresses the problem of planning, coordinating, monitoring and executing structural assembly tasks for multiple mobile robots in unstructured and potentially constrained environments. It presents an approach that enables robust execution and efficient planning and is scalable to large structures and many assembly robots.

Consider a scenario (a simplified robotic version of the opening example of this chapter) where a number of mobile robots capable of performing basic assembly tasks (e.g., pickup components, transport them, and connect them to other components) are in an environment with an arbitrary initial distribution of structural components that can be assembled into some structure. The robots are given the desired final configuration of the available components once they are joined and the structure is complete. The environment may contain obstacles and is space-constrained relative to the size of the structure to be assembled. Finally, some assembly tasks may require the coordination of multiple robots to be accomplished.

1.2.1 Challenges

Given the desired structure configuration and initial state of components and robots in the environment, an assembly planner needs to take into account the capabilities and limitations of the available robots, as well as constraints imposed by the environment, to determine a feasible and desirable plan for assembling the structure given the resources available. In addition to generating an initial assembly plan, the system also needs to be capable of handling exceptions that occur during execution by appropriately repairing the plan or re-planning a new sequence of tasks.

The three important constraints to reason about during planning such tasks are structural constraints, environmental constraints and resource constraints. Structural constraints solely are con-
cerned with ensuring a feasible assembly sequence that guarantees that all internal components are installed while they are still reachable by robots around the structure. Environmental constraints affect the quality or goodness of assembly steps, which depend on the ability of a robot nominally capable of performing a task to actually reach a place from where to do so. Finally, resource constraints become relevant for larger assemblies where additional robots could be assigned to perform a number of different tasks resulting in more or less effective use of parallelization.

The fundamental challenge of assembly planning problems as addressed by this work is that they attempt to find solutions to large-scale problems that inherently depend on constraints at a much finer level of detail. Separate techniques exist for both large abstract and smaller fine-grained problems. Assembly planning requires both to be combined into a single comprehensive framework.

1.2.2 Scope and Limitations

The assembly planning problem as considered in this thesis is characterized by the following three aspects: 1) The robots are tasked to perform highly challenging and difficult tasks, thus failures and exceptions during execution are expected (or at least not unexpected) to occur; 2) Speed of execution, while not irrelevant, is less important than robust and reliable operation; and 3) The assemblies considered are one-of structures of generic components instead of carefully designed products that, once planned, are produced in large quantities in very structured environments.

This thesis primarily focuses on the integration of and interaction between different planning modalities within the assembly planning framework. To that end, simplifying assumptions have been made to limit the complexity of the problems representation. Specifically, as presented throughout this document, the work trades off completeness in some scenarios for realistic tractability for real-world problems. As Section 2.3 in Chapter 2 will show, techniques exist to address and remove the need for these simplifying assumptions without having to fundamentally alter the approach developed in this thesis. In the interest of understanding and analyzing the higher-level aspects of assembly planning, the addition and implementation of those techniques is left to future work.

1.3 Applications and Significance

Manipulation in general and assembly in particular are prime applications that enable robots to become useful in the real world. Sensing and navigation are clearly prerequisites for meaningful mobile manipulation, but only with high-level reasoning as developed in this thesis will general-purpose robots in uncontrolled environments be able to perform tasks at which robots are usually expected to excel: assembling things. While robots have long been used in assembly line environments to assemble products from cars to circuits boards, there are few applications where robots are tasked to perform assembly tasks in less structured environments and with greater flexibility in the order in which to perform assembly steps. On an assembly line, steps are carefully planned and sequenced for high-speed and high-volume operation. Failures during execution are often catastrophic and require a stop and manual reset before operations can continue. Mobile robots’ capabilities are steadily maturing toward allowing them to perform similar assembly tasks outside of factory settings. Operating in an environment that is not specifically designed to accommodate the tasks being
performed, the robots will experience failures during execution, which they need to be able to handle in order to continue operations and achieve their goal. In the context of this thesis, the term “assembly” is broadly considered to apply to any task where several low-level manipulation operations must be combined in a structured manner and with intermediate motion through the robots’ environment in order to accomplish a high-level goal.

Below is a sampling of application domains where robotic assembly can very obviously be applied. Assembling structures in space, handling materials, etc. are what is usually thought of in the context of assembly. In addition, there are numerous tasks robots might perform that follow similar patterns of a high-level task being decomposed into generic sub-tasks that have to take place in a certain order due to environmental and task constraints. Domestic assistance robots loading a dishwasher or stocking shelves could be such applications, among many others.

1.3.1 Robotic Assembly in Space

The bulk of the work leading to this thesis was done under various NASA contracts. The agency has long had a strong interest in robotic assembly (see Figure 1.1). Any structure in space is limited in size to what fits into the largest launch vehicle. Anything larger than that must be assembled, be it on a planetary surface or in orbit. While astronauts are capable of performing complex assembly tasks during EVA, this work is very dangerous and expensive. In particular when considering the assembly of a planetary habitat or outpost, there is a chicken and egg problem: The astronauts to build the outpost have no place to live and shelter until their work is mostly complete. Sending robots ahead of human missions to establish the necessary infrastructure, however, avoids this dilemma.

Figure 1.1: In the future, robots are envisioned to construct and maintain planetary outposts (left) and assemble large structures (e.g., for solar power arrays) in orbit (right), as depicted in these NASA artists’ renditions. More generally, space is a stand-in for the larger class of environments where humans cannot or do not want to go to perform complex tasks.
Furthermore, once the outpost is constructed and humans arrive, the robots can switch roles to support the astronauts in their mission, and switch back to construction and site maintenance as their services are not required otherwise.

“Assembly” and “construction” in space can be defined quite loosely. While there are ongoing efforts to develop robots that can perform tasks similar to construction on Earth where a site needs to be prepared before a building can be erected using primarily materials found on site or mined/generated from local raw materials, “assembly” often looks more like arranging pre-fabricated modules into a usable configuration. Environmental constraints in this domain include natural obstacles such as craters and rocks, and often the environment is not fully known prior to execution. During their tasks, the robots detect discrepancies with the assumed environment when exceptions occur. In a collaboration with TRACLabs Inc., the Cluster project\textsuperscript{1} develops a task planning and monitoring system for two mobile robots that retrieve, transport and dock components in the environment to achieve a specified final configuration. While it is assumed that the general assembly sequence is known beforehand, specific instantiations based on initial conditions discovered during execution are handled autonomously. A key aspect of this project is the handling of execution-time failures via Sliding Autonomy operator interaction for tasks that involve teams of robots.

Space is really only a stand-in for environments where humans cannot or do not want to go. Other examples include disaster areas, underwater or otherwise inhospitable places. Robotic assembly (or disassembly, which fundamentally is not all that different) in such environments is a challenging problem. Its solutions benefit both the people who are kept out of harms way (because robots perform the most dangerous parts of their work), and those rescued after natural disasters such as earthquakes by robots that can intelligently clear (“disassemble”) collapsed structures and erect new temporary shelters.

### 1.3.2 Military and Logistics Applications

As robotic manipulation is becoming increasingly capable, many applications with relevance to the military are moving into the realm of the possible. Most prominently, a recent initiative launched by DARPA investigates fieldable manufacturing techniques for large-scale assemblies in quantities of one. Imagine a unit on a mission to traverse some area. At some point they reach a river or canyon too wide to cross. Before continuing their mission, the soldiers have to take a long detour. However, part of the unit may be a truck loaded with generic truss assembly components and some robots capable of putting those pieces together. Specifying the key parameters of what is required of a temporary bridge structure to enable the soldiers to cross the chasm and continue with their mission, the soldiers task the assembly robots to provide them with a path across the river.

Materials handling and logistics applications are another area where robotic assembly technology can be applied (see Figure 1.2). A high-level task (e.g., load those crates onto a truck) is complicated by the fact that the crates may not be neatly stacked and ready for pickup, they may be located in a constrained environment, and it may not even be known how many of what kinds or shapes of crates are present. This problem can be posed as an assembly problem, and the techniques developed for this thesis could be applied to solve it.

\textsuperscript{1} Cluster project web page: http://www.frc.ri.cmu.edu/projects/cluster
1.3. Applications and Significance

Figure 1.2: Teams of Bear™ robots (see close-up picture of the robot in the picture on the right) developed by Vecna Technologies (www.vecnarobotics.com) package loose items into crates and then load those onto a waiting truck. The reverse process of unloading a truck and distributing its cargo in the environment could also be considered.

1.3.3 Domestic Applications

In domestic application scenarios, robots are making progress toward being able to perform tasks around the kitchen such as retrieving items from cabinets or loading and emptying the dishwasher (see Figure 1.3). These tasks require a combination of task-oriented high-level reasoning to accomplish abstract tasks such as loading a dishwasher and physically grounded motion and manipulation planning. Many such tasks could be posed as “assembly” problems. Consider for example the loading of a dishwasher, where the fully loaded machine is the goal to achieve, and the dirty dishes all over the kitchen provide the initial condition, or a robot folding laundry, stacking folded items and placing them on shelves. Today’s solutions in these domain largely focus on the fine-manipulation aspects of the problem (e.g., carefully grasping an item and placing it in another location) rather than the higher-level task of performing many similar tasks until for example the dishwasher is empty.

A motivating scenario for this thesis in the domestic domain has been the robotic assembly of IKEA furniture. Imagine the following situation: After moving into a new apartment, you probably

Figure 1.3: Robots assisting in the kitchen have to solve problems similar to assembly planning when they have to retrieve items from cabinets or load and unload the dishwasher. The pictures show HERB, the Home Exploring Robotic Butler developed by Intel Research (left), ARMAR, a domestic assistance robot developed at the Forschungszentrum Informatik (FZI) at the University of Karlsruhe, Germany (center) and Willow Garage’s PR2 folding laundry (right).
have some ideas of how you would like it to be furnished (e.g., a desk against this wall, the bed in
this room, a bookshelf by the door, etc.). After a trip to IKEA, all the parts required for all furniture
pieces for the apartment are unpacked in the living room. While you go out to explore the new
neighborhood, domestic assembly robots go to work assembling the furniture and placing each item
in its place. This task includes both figuring out how individual components need to be assembled
such that in the end there will be a bookshelf/table/bed/etc., as well as reasoning about the order in
which to assemble pieces of furniture to avoid getting stuck with half a bed assembled but no way
to get the dresser to where it is supposed to be. This thesis lays the foundation for making such an
application possible in the future.

1.4 Thesis Statement

Today, problems such as the one addressed by this thesis are generally approached in one of two
ways. Either as a high-level abstract task sequencing problem (without reasoning sufficiently about
physical feasibility in a real-world application), or as a fine-grained motion or manipulation problem
of (re-)grasping objects, manipulating and transporting them (without sense of a high-level goal be-


tween the several step that might be necessary to accomplish the specific task at hand). What interac-
tion there exists between the two fundamental approaches is in general ad-hoc and information-poor,
rendering them insufficient to reason about all the relevant aspects and constraints of the assembly
planning problem. For an extensive review of related literature, see Section 2.3 in Chapter 2.

Both avenues of research clearly address important and complementary aspects of assembly
planning, but neither area alone provides a satisfying solution. This thesis aims to establish a com-
prehensive planning and execution framework that leverages key capabilities of both approaches
linked in an information-rich manner to create an overlap of both methods. To that end, it develops
a holistic approach that considers the entire span from planning assemblies to executing the planned
tasks, all in a robust, efficient and scalable manner. Given the shortcomings of the state of the art,
this thesis argues that neither abstract task-oriented planning nor fine-grained motion manipula-
tion-focused planning alone can effectively solve the problem of planning structural assemblies of large
structures with multiple robots. In order to solve assembly planning problems as considered here,
task planning requires extensions to reason about physical feasibility and reality of the tasks be-
ing considered, and the fine-grained approaches require higher-level guidance toward goals that are
beyond their problem representation. Recognizing the potential for powerful synergies, this thesis
makes the following statement:

A combination of symbolic and fine-grained algorithms for planning structure assembly
tasks for multiple mobile robots can yield robust, efficient and scalable performance.

In this hybrid approach, abstract task planning focuses fine-grained motion planning toward
providing real-world feasibility information that in turn guides the search for high-level assembly
plans toward those that are both structurally and physically feasible and desirable. In addition, as
execution-time exceptions occur (caused by failed interactions with the real world), they can be
recovered from at the appropriate level of abstraction, either with other fine-grained motion plans,
or increasingly more abstractly at the level of individual assembly steps within the plan, or globally
with a partial re-plan of the high-level task. As a result, not only are realistically large assembly problems planable, they can also be executed effectively and robustly in the presence of execution-time failures and exceptions.

1.5 Approach Sketch

This thesis presents a planning framework for general-purpose mobile manipulation robots to intelligently assemble very large structures (both in physical dimension and number of components) in less-structured (compared to factory or assembly line settings) and moderately constrained environments. While it is assumed that a desired structure can actually be assembled, the system detects when this assumption is violated before any robot starts moving. If the system finds an assembly plan, a nominally feasible assembly sequence is guaranteed to exist. Throughout plan execution the system monitors progress, reacts to and corrects any problems that arise in the real world.

This thesis considers three important aspects that extend basic robotic assembly (Chapter 3): Robust execution of assembly plans (Chapter 4), efficient generation of assembly plans (Chapter 5), and planning for large structures that require and/or benefit from multiple robots working together or in parallel (Chapter 6). The following brief sketch of the approach to robotic assembly planning highlights the main results and contributions of this thesis. For further details refer to the subsequent chapters that cover each area in depth.

1.5.1 Assembly Graph Representation

The underlying representation for assembly planning problems is a state graph with intermediate assembly states at its vertices connected by assembly steps on its edges. The assembly graph for a structure is constructed in its entirety at the beginning of the planning process and then searched for desirable assembly sequences. It includes all structural constraints (e.g., component ordering constraints, etc.) and is static for a given structure, allowing it to be pre-computed, stored and re-used. In the absence of environmental constraints (e.g., obstacles in the workspace, physical size of robots, etc.), any trace through the assembly graph is a possible assembly sequence. In order to find a feasible and desirable assembly plan in realistic environments, the planner evaluates individual assembly steps (i.e., graph edges) in the context of the environment as part of a best-first graph search strategy.

Planning assemblies by searching the assembly graph exploits the high-level step-by-step structure of the problems at hand. During assembly step evaluation, this high-level search is combined with fine-grained motion planning techniques to determine nominal feasibility of assembly steps in the context of the environment. This finer level of detail provides assembly step costs (as defined by some cost function that can include distance traveled by robots, difficulty of environment traversed, etc.) and, if the step is found to be feasible, a complete parameterization of the required robot behaviors to execute the task. In addition to relying on fine-grained planners to provide quality feedback about particular assembly steps, the planner also considers the abstract structure of the problem (as encoded in the assembly graph) to guide the search for a desirable assembly sequence.

The assembly planner sits at the top of a traditional tiered robot architecture consisting of planning, execution and behavior layers. If the planner finds an assembly sequence for a given structure,
the solution trace through the assembly graph contains all parameters necessary to task robots to perform the corresponding assembly tasks. These parameters instantiate a task tree for the executive layer that is used to sequence robot behaviors and monitor their progress throughout the operation. In the event of execution-time failures and exceptions, the executive contains error recovery mechanisms throughout that can address problems at different levels of abstraction, and with or without involving the rest of the planning framework.

1.5.2 Robust Execution

No matter how an assembly plan is generated (either by an automated system as described throughout this document, or scripted by human experts), the main challenge remaining between having a plan and obtaining the planned-for goal are countless execution-time exceptions. These exceptions range from simple temporary issues to serious problems and fatal errors. Since the developed system enables robots to perform useful work (e.g., assembly) in the real world, there is a strong interest in being able to actually accomplish the goal despite the execution-time exceptions that will undoubtedly occur. As it is impossible to anticipate, foresee and plan for all possible errors that can (and will!) occur, it is essential to have mechanisms in place to mitigate any problems as they arise.

Robustness is achieved in several ways. During planning, the robots’ capabilities and constraints are incorporated by the fine-grained motion planner when evaluating assembly steps and selecting good or better ones to be part of assembly sequences. The existence of a motion planning solution for an assembly step is a necessary but not always sufficient condition for the realistic feasibility of assembly steps. In a perfect world, finding a valid motion guarantees feasibility of the particular task. However, in more realistic situations, additional criteria are important when judging the desirability of robots’ actions. These include the approachability of assembly locations for the robots performing the work, the stability of sub-assemblies, or even the number of available alternatives should the selected action fail irrecoverably.

Even the best plans, once executed on real robots in real environments or in simulation, cannot guarantee nominal operations at all times. To deal with execution-time failures, the assembly planner developed for this thesis is closely integrated into a traditional multi-tier robot architecture (with planning, executive and behavioral layers) and allows for different levels of recovery strategies where they are most appropriate. It is assumed that the robot behaviors tasked with carrying out assembly operations are capable of detecting when something is wrong. They do not have to understand what the particular problem is, only that they are in an off-nominal state and should not continue with their current task to avoid damage to themselves or the structure.

As a first response, simple contingency operations often succeed in getting execution back on track. If they fail, local plan repair may solve some problems where different parameterizations of currently executing tasks allow the robots to proceed. If all strategies fail, high-level re-planning can exhaust the remaining alternative ways to accomplish the high-level task at hand. Recovery assistance at any level can also be rendered by a human operator via Sliding Autonomy. This

\[^2\] Under Sliding Autonomy operation, the autonomous system performs tasks as well as it can. If an anomaly is detected (either by the robots themselves, or by a human supervisor), control of the problematic aspects of a task can be transferred to an operator while the rest of the system continues to operate autonomously. When the exception is resolved, control passes back to the autonomous system. This mode of operation effectively combines autonomous
mode of exception handling effectively provides the required robustness for useful operation of the overall system in several simulated scenarios of varying complexity. With autonomous strategies in place to handle many less severe problems, the operator’s efforts can be best targeted toward where they are most necessary and effective.

Comprehensive error recovery strategies can also be used to compensate for problems artificially introduced when attempting to plan more efficiently or execute plans with additional resources in parallel. Instead of having to reason in detail about many issues, the system can find approximate solutions and rely on the built-in exception handling capabilities to compensate for any discrepancies between them and the real world.

1.5.3 Efficient Plan Generation

Assembly problems, even for simple structures of few components, are quite complex. As the number of components increases, the number of possible assembly sequences grows exponentially, quickly exhausting the capabilities of a straightforward planning approach. In addition, typical applications for assembly planning systems as considered in this thesis generally are low-volume, one-of-a-kind assemblies. As such, they will never be executed sufficiently often for long and deliberate planning to pay off. Instead, it is of interest to be able to generate plans (reasonably) quickly both when finding the initial solution as well as (and especially when) execution-time exceptions occur and re-planning becomes necessary.

The bulk of the computational expense in assembly planning lies in the number and difficulty of potential assembly steps being evaluated. Two fundamental strategies make planning more efficient: evaluating potential options more efficiently, or evaluating fewer options during the search for an assembly sequence. Chapter 5 evaluates methods of both kinds that exploit the structure of assembly problems to gain planning efficiency.

Evaluating options more efficiently is achieved by making strong assumptions that simplify the fine-grained planning queries required. Planning with simplified motion models (i.e., plan for a robot without non-holonomic motion constraints, even though the actual robot executing the plans is a skid-steered vehicle) or delaying motion smoothing until a complete assembly plan has been found provide such efficiency gains, but at the cost of artificially introducing execution-time problems where the assumptions are violated. Relying on the error recovery capabilities described above, the overall system can maintain the required robustness with a planning efficiency gain that greatly outweighs the additional error recovery effort required. Another strategy generalizes motion solutions to other similar queries without specifically evaluating them to reduce planning time. In some cases the structurally different motion queries are (wrongly) clustered and treated as similar, resulting in the planner generating a physically infeasible plan. This problem, however, is detected and can be corrected before the plan is sent to the robots for execution.

Evaluating fewer options yields the greater benefit. Guiding the search for an assembly plan by computing an admissible cost-to-go heuristic from each state in the assembly graph to the goal state ahead of time and then searching the graph using A* provides the most individual benefit. This robots’ efficiency with operator-based reliability. Sliding Autonomy is also referred to as Adjustable Autonomy [Dorais et al., 1999] or Mixed Initiative [Riley, 1989] operation.
1.5. Approach Sketch

strategy reduces the portion of the entire assembly graph being evaluated during planning from 66-75% to 1-20% of all graph vertices and edges and thus is able to reduce planning time significantly. Combining techniques from both categories yields the greatest efficiency improvement. A speedup of 2.5 orders of magnitude (or a reduction of planning time from 11.6 hours to 2.5 minutes) or more compared to the baseline solution where the assembly graph is searched for assembly plans using best-first search is possible. In addition, the realizable improvement increases with the complexity of the structure being planned. With these enhancements, planning structures of few tens of components can be accomplished in a reasonable amount of time.

1.5.4 Scaling to Large Structures and Multiple Assembly Robots

Computational resource constraints limit the assembly planning system to problems of moderate size (up to few tens of components). To be truly useful, an assembly planner has to be able to work with structures of hundreds to thousands of components. The assembly graph representation does not extend to the required exponential number of assembly states that come with either larger structures or additional robots, or both. Even if the representation supported the increase, the resulting computational burden to search the larger graphs would be significant, even with the efficiency improvements described above.

Building on the foundation of a system that allows robust execution and efficient planning, this thesis hierarchically decomposes large problems into several smaller ones that fit within the scope of the approach and representation. In this manner, individual components are first assembled into sub-assemblies, which become individual components in larger partial structures until finally large pre-assembled parts are combined into the desired goal structure. While computationally there are nice recursive effects to be taken advantage of, it is important to realize that there is a physical reality that governs what is possible when robots execute the plans. As assembly robots cannot simply scale in size hierarchically with larger and larger sub-assemblies, cooperation between multiple robots becomes essential to achieve the desired high-level goals. Using this hierarchical decomposition technique, the planner finds solutions for assembling planetary outposts comprised of several buildings, each of which is assembled of several segments, each of which in turn consists of individual components. In total, 1,250 individual components need to be assembled to construct the simulated outpost.

Just as it is unreasonable to assume that the robots performing the assembly will simply grow along with the larger and larger parts to be manipulated, it is also unlikely that large structures will be constructed by only one or two robots. Instead, additional robots will be available, and the system can use them to take advantage of opportunities for parallelization during execution. While additional robots can reduce the total time required to accomplish the goal, the nature of the scenario is such that adding too many robots again has detrimental effects as they will interfere with each other more than they are able to successfully work toward the assembly goal. The assembly planner generates serial assembly plans for the minimum number of robots required to assemble an outpost. If additional robots are available, the system relies on a heuristic scheduler to parallelize tasks according to the available resources. It differentiates between two extreme cases: outposts where all buildings are sufficiently separated that they do not impede each other’s construction, and ones where buildings are in close proximity. A simple test determines which type of outpost the
planner is dealing with. If inter-building constraints are not important, each building’s construction has to be planned only once, and all tasks can be maximally parallelized by the scheduler, resulting in the shortest overall makespan of the schedule. If inter-building constraints need to be considered, a more extensive search for a desirable assembly sequence is required. Once a full assembly plan has been found, the entire outpost is treated as a long series of segments-into-building assembly tasks. The scheduler’s task then is to assign the available resources to either pre-fabricate segments or perform the actual installation.

With the decomposition of large structures into recursively smaller ones given (as this work assumes), the planning approach can be distributed with minimal modifications to the algorithms that are used to plan smaller structures. Being able to automatically decompose problems and even dynamically adapt the decomposition in response to execution-time exceptions is an interesting open problem for future work. While the hierarchical decomposition makes the same robustness strategies as described above available for large structures, it is not obvious how to best provide operator assistance to teams of robots in parallel execution scenarios.

### 1.6 Document Road Map

Figure 1.4 provides a pictorial overview of the main themes of this thesis. After discussing background and related work in Chapter 2, Chapter 3 presents the fundamental approach to assembly planning, followed by three areas of refinement and improvement: robust execution (Chapter 4), efficient plan generation (Chapter 5) and scalability to large structures and multiple robots (Chapter 6). Chapter 7 concludes the thesis and points to directions of future work.

![Diagram](image_url)

**Figure 1.4:** A pictorial overview of all the aspects of assembly planning considered in this thesis. In order for mobile robots to construct large structures in challenging environments in a useful manner, their plans need to be executed robustly, they must be generated efficiently, and the problems must be large enough to be of physical relevance and importance.
Chapter 2

Background

2.1 Overview

This chapter aims to put this dissertation into context with respect to prior work, similar problems and the current state of the art. Of the many aspects of the assembly planning problem, it highlights the specific challenges that make the problem difficult and that are addressed in this work. Finally, it describes the limitations in scope that carve out particular aspects of the overall problem to be addressed in more detail.

The assembly planning problem addresses the question of how individual parts can be brought together to form a larger construct – an assembly or a structure. Traditionally, only the parts coming together have been considered, with the assumption that they can either move themselves or that some non-interfering mover can accomplish the required motion. This approach views the problem as a motion planning problem in a high-dimensional configuration space.

Another way of posing the problem assumes a human designer who specifies the order of assembly, and assembly planning is used usually in reverse (i.e., as disassembly planning) to ensure serviceability of certain components or sub-assemblies throughout the design process. Again, considering only the parts of the assembly, the question being asked is “Can this component be removed from the final product without having to do major disassembly?”, and the answer is used to make the necessary design changes.

The important aspect missing from the problem description thus far is any consideration for the machines or robots that are to perform the planned assembly tasks. Most parts cannot move themselves, and in relatively unconstrained environments (i.e., outside of well-structured assembly lines), the assembler has to operate in the same workspace in which it constructs a growing obstacle (the assembly itself) as the task progresses. This thesis aims to unify the two separate approaches into a comprehensive assembly planning framework. It focuses primarily on the integration aspects of different planning modalities and makes some simplifying assumptions to maintain a tractable scope. A task is considered to be an “assembly” task if it requires a number of low-level manipulation and motion tasks to be performed in a structured manner in order to achieve a complex high-level goal. In addition to the structural assembly scenarios considered in this thesis, tasks that
fit into this category include unstructured materials handling operations (loading and unloading of trucks, etc.) and domestic scenarios such as loading a dishwasher.

2.2 Thesis Context

This thesis builds on two large bodies of work: symbolic planning (capable of solving large structured problems) and fine-grained motion planning (considering interactions between robots and their environment). It has long been recognized throughout the assembly planning literature that a comprehensive solution to general assembly problems requires a combination of both approaches, but no integrated system exists to date. The major contribution of this work is the development and analysis of such a planner capable to solving problems of realistic relevance and interest.

2.2.1 State of the Art

Purely symbolic planning techniques employ abstractions that allow them to solve large problems efficiently and in a general manner [Fahlman, 1973]. Those same abstractions also severely limit the planner’s ability to reason about fine-grained and problem-specific constraints. The only way important characteristics such as physical reachability or quality of sensor information can be incorporated is with the use of an outside (oracle) process. This thesis exploits the abstract structure of assembly problems but establishes more information-rich interactions between symbolic and fine-grained planning techniques to find physically feasible solutions.

Traditional assembly planning is closely related to symbolic AI techniques, but they include some concept of a physical structure being assembled. The vast majority of the work in this area, however, focuses on applications in well-controlled environments, such as assembly lines and factories where the environment is assumed to be engineered to suit the task at hand [Homem de Mello and Lee, 1991]. Approaches are generally limited to design assistance tools to ensure serviceability where geometric constraints at a very low level (e.g., nuts and bolts, connectors, tolerances, etc.) are critical [Kaufman et al., 1996]. Some work considers access for tools locally around where an assembly step takes place [Wilson, 1998], but no solution has yet been developed that considers the assembler along with the assembly to ensure physical feasibility of the entire task. Using fine-grained planning techniques combined with abstract symbolic ones, this thesis specifically considers reachability of workspace locations (and thus realistic feasibility) for robots performing individual assembly steps.

Motion planning solutions inherently reason about bodies moving through and avoiding obstacles in their environment. Assemblies of self-mobile parts can be posed as high-dimensional motion planning problems [Lengyel et al., 1990], but as the number of components increases and robotic assemblers become necessary to perform the tasks, purely motion-based solutions are unable to solve interesting assembly problems. They primarily lack the ability to exploit the step-by-step structure of many problems. In this work, a graph-based problem representation is used to guide low-level motion planners toward solving complex multi-step problems.

There has been tremendous progress in enabling robots performing difficult tasks involving dextrous manipulation (e.g., in grasping and manipulating objects, in domestic assistance scenarios, etc. [Siméon et al., 2002, Marder-Eppstein et al., 2010, Maitin-Shepard et al., 2010, Srinivasa et
However, the scope and action horizon of these capabilities is limited and insufficient to apply to the kinds of large-scale assembly problems considered here. This thesis provides the necessary high-level reasoning and planning capabilities to apply existing and developing manipulation techniques to large and interesting problems beyond what is possible today.

2.2.2 Challenges

The fundamental challenges in planning large-scale realistic assemblies for mobile robots in challenging environments is the desire to find solutions that not only respect the abstract structure of the problem but that are also feasible to be executed reliably in real-world situations. Existing solutions to this type of scenario either abstract away important aspects of the problem or are limited to solving small problems in order to remain tractable. In addition, simply generating an assembly plan is not sufficient, as execution-time exceptions are inevitably going to occur, requiring some sort of recovery action before the task can continue. Ensuring robust operation in realistic environments is the primary concern in assembly planning.

Once robust execution can be assured, planning efficiency becomes of interest, both for finding initial solutions, and especially when reacting to execution-time exceptions by modifying plans. As the size of the structure to be assembled grows, there exists an exponential number of possible sequences in which the components can be brought together. Finding feasible and then desirable solutions without having to exhaustively consider all options presents interesting challenges. This aspect of the problem specifically focuses on the interactions between abstract high-level and fine-grained low-level planning strategies to generate comprehensive and physically realistic solutions.

Finally, efficiently generated assembly plans that can be executed robustly are the foundation of solving problems involving ever larger structures and additional assembly robots. Scaling the system in this manner presents complex challenges in representing such large problems and reasoning about parallelism during execution that becomes possible as additional robots become available. Straightforwardly applying the same methods to ever larger problems in intractable, but exploiting both the structure of the assembly problems and the availability of a human operator/ supervisor, the planning work can be distributed and solved in a hierarchical fashion.

2.2.3 Scope and Assumptions

This thesis develops a comprehensive assembly planning and execution framework capable of executing plans robustly, generating plans efficiently and solving complex problems involving large structures and multiple robots. Its input is a desired structure to be assembled (including the appropriate decomposition into sub-assemblies, as appropriate), a representation of the environment and the available resources. It generates the parameters necessary for low-level assembly behaviors to execute tasks and provide status feedback. In response to this feedback, the system implements exception-handling strategies to ensure robust operation.

The focus of this work is on the integration of different planning techniques, and on providing algorithms and techniques to enable robust execution, efficient planning and scalability to problems of interesting complexity. To that end, a number of simplifying assumptions are made. First, the desired structures are assumed to be assemblable monotonically by adding one component at a time,
and that they have no internal degrees of freedom. Second, the planner only considers assembly steps and no disassembly steps. Once a component is added to the structure, it is assumed to remain in that position relative to the structure for the remainder of the assembly task. Third, partial structures are considered “valid” if they are fully connected, and components can only be added to the convex hull (i.e., reachable from outside) of the structure. Fourth, robot behaviors are assumed to exist that can execute basic assembly tasks (e.g., pickup, transport, connect, etc.) and provide feedback about either successful task completion or some new information about the world that caused an exception to occur. Fifth, for handling exceptions that require operator assistance, the system assumes that an appropriate operator interface exists through which the necessary exception resolution can be provided. Finally, sixth, if a structure is too large to be planned as a whole, an appropriate decomposition into smaller sub-assemblies is assumed to be given to the planner.

Note that these assumptions were made primarily to limit the complexity of the problems representation and allow the thesis to be focused on higher-level interaction issues between the many components involved. No fundamental changes to the approach presented in Chapter 3 are necessary when removing them, but additional computational resources will be required in terms of memory and processor cycles. The remainder of this chapter will highlight existing work that could be included in this framework with additional implementation work.

2.3 Related Work

The assembly planning problem lies at the intersection of two major bodies of planning research: traditional (primarily symbolic) assembly planning approaches and motion planning. Each only addresses a subset of the three primarily relevant constraints (see Figure 2.1): ordering tasks to accomplish structurally valid plans, considering physical constraints to ensure robots can get to where they need to be to perform their tasks, and reasoning about available resources. This section provides an overview of existing approaches and shows how none of them individually are sufficient to solve the problem addressed by this thesis.

2.3.1 Symbolic Planning

Traditional AI planners generally are domain-independent planners that operate at a high level of abstraction. Operators with precondition and effects are chained to transform an initial state into a desired final state. Such abstraction is necessary to enable domain-independent planning, but for assembly planning applications, they are generally too strong: important constraints are either abstracted away, or they are expressed as pre-conditions for operators that require an outside oracle to evaluate. The main ideas from AI planning that will be useful to this thesis are least-commitment [Weld, 1994] and partial-order planning [Younes and Simmons, 2003], since there are many situations where environmental conditions will require slight changes in plan execution. Instead of fully specifying rigid plans, more flexible approaches will be more robust in the entire task performance cycle. Instead of planning a specific solution sequence, the planner will defer making decisions not directly related to the task at hand as long as possible. The resulting plans often represent a family of plans that all meet the set constraints but are not identical. In the context of assembly plan robustness, having such alternatives available is desirable in order to be able to react to execution-time
failures and problems in a forward-moving direction without having to backtrack. While backtracking may not be a problem for the abstract planner, many assembly scenarios, including the ones considered in this thesis, allow only assembly but not disassembly steps, making physical backtracking impossible. This assumption made throughout this work is based on assembly hardware in the Trestle project\(^1\) assembly test bed, where components are connected via latches that engage once the parts are in place and cannot be undone by the robots.

A classic AI planning example is the Blocksworld domain where blocks have to be rearranged by an external disembodied gripper to achieve a desired configuration [Fahlman, 1973]. While on the surface this may sound like an assembly problem, it is not. The planner works with symbolic predicates describing the state, and any realistic constraints (e.g., obstacles, etc.) are abstracted away. This abstraction is necessary since in the formulation, pre-conditions have to be evaluated for feasibility, and evaluating more involved conditions becomes in itself a larger planning problem in a less abstract domain. The restriction to simple yes/no queries about feasibility leads to another problem with symbolic planning approaches: they are primarily useful for generate-and-test approaches,

\(^1\) Trestle project web page: http://www.frc.ri.cmu.edu/projects/trestle
2.3. Related Work

but they cannot avoid repeatedly asking very similar questions until they find a solution. For example, instead of understanding that an entire component is unreachable by a robot (e.g., because the robot does not fit through a narrow gap between its current location and the component), each possible approach direction would have to be queried before the same result would be achieved.

Hierarchical task network (HTN) planning uses concepts similar to how humans think about task decomposition for planning. Methods decompose tasks into successively more detailed sub-tasks until atomic actions are reached that can be executed. SHOP/SHOP2 [Nau et al., 2004] and O-Plan/O-Plan2 [Tate et al., 1992] are two planners that incorporate some domain information and use the HTN formulation. The planner developed in this thesis assumes that the decomposition of assembly steps into atomic robot behaviors is provided (via task templates, see Chapter 3, Section 3.3.2). HTN planners would be suitable to generate such templates automatically.

2.3.2 Assembly Planning

There is a large body of work in assembly planning dating back to the 1980s and 1990s with influential work by Homem de Mello [Homem de Mello and Sanderson, 1986], Wilson [Wilson, 1992] and others [DeFazio and Whitney, 1987, Wolter, 1989]. The focus of this line of research was the generation of sequences that would allow robots in an industrial assembly line setting to assemble a product based on design data files (e.g., CAD data). An important aspect of this work was to enable concurrent design for manufacturing, and consequently the primary concerns were to satisfy low-level constraints such as mating constraints, tolerances, etc. In general, the methods do not consider the agents that will perform the assembly operations.

Assembly planning is concerned with the generation of assembly sequences, often optimal ones given certain cost criteria, for a particular product in an assembly line. All work in this area is focused on the product itself, robots that perform the assembly are treated only tangentially. In general, this is a valid assumption for this domain since the environment can usually be engineered for the task at hand. Common across the literature is the approach of planning the disassembly of products (mostly for efficiency considerations) [Homem de Mello and Sanderson, 1988a, DeFazio and Whitney, 1987, Lee and Moradi, 1999]. The key components of assembly planning are the generation of sequences of assembly steps, the consideration of (often problem specific) constraints, and finally using those constraints to guide the partitioning of assemblies into smaller subassemblies.

A number of books and survey papers provide an overview of the kinds of problems considered by traditional assembly planning approaches. Homem de Mello and Lee published a collection of papers about computer-aided mechanical assembly planning [Homem de Mello and Lee, 1991]. Gottschlich et al. presented a taxonomy of assembly and task planning when the IEEE Technical Committee on Assembly and Task Planning was created [Gottschlich et al., 1994]. This work presents a comprehensive overview of key issues concerning assembly planning, however, no trace of this committee can be found anymore. More recently, Lambert compiled a survey of work in disassembly sequencing [Lambert, 2003]. Disassembly as a whole is a much larger class of problems than simply assembly in reverse (and in fact, there are several cases where this assumption is not valid). Some key approaches and their relevance to the more general problem considered in this thesis are highlighted below. Note that nearly all work in assembly planning dates back to the 1990s and earlier. Since then this area of research has gone somewhat out of fashion with many problems
left unsolved – specifically that of how to integrate different planning modalities to obtain solutions for realistic problems in less-structured environments as addressed by this thesis.

2.3.2.1 Assembly Sequencing

The ultimate goal of assembly planning is to produce a sequence of assembly actions that achieve the assembly of a given product. If such a sequence exists, the problem has a solution, otherwise not. Of the (potentially many) possible solutions, the planner for an assembly line setting aims to find the optimal one, usually in terms of assembly time, in order to maximize throughput of the product being assembled. Since execution-failures are engineered out of the process, execution can be considered to be deterministic and the flexibility allowed by partial order plans as described above is not required. Thus, assembly planners find fully ordered sequences [Homem de Mello and Lee, 1991, Romney et al., 1995]. In contrast, assembly tasks as considered in this thesis are rarely ever executed successfully as initially planned, and finding the optimal solution is less important than considering possible alternatives that yield more robust assembly plans.

There are two principle ways of generating assembly sequences. Earlier work involved asking questions to a knowledgeable engineer to determine precedence relationships, later work generally employed a search over the space of all possible sequences. DeFazio and Whitney developed a system to aid an engineer in finding a good (or the best) assembly sequence by asking a series of questions to determine precedence relationships between operations [DeFazio and Whitney, 1987]. Their method greatly reduced the number of questions required (to two questions per connection between components: “What connections have to be established before this one can be considered?” and “What connections have to be established after this one is in place?”) from earlier work [Bourjault, 1984], but the remaining ones still require an expert to answer. In particular, as assembly complexity increases, many constraints may be non-obvious even to a well-trained engineer and thus might be missed. Similarly interactive is the Archimedes 2 assembly planning system developed at Sandia National Laboratories [Kaufman et al., 1996]. Starting with CAD data for a complete assembly and an initial set of constraints specified by an engineer, a geometry engine produced a candidate assembly sequence that respects all current constraints. The user was shown an animation of this sequence and could specify additional constraints that had become obvious. This process repeated until a satisfactory sequence was found. The interactive methods described here all focused on the low-level (clearance, fit, etc.) constraints to determine whether or not a particular product could be assembled, or if design changes were required (to be made by a human). In contrast, the hybrid planner developed in this thesis assumes that structures can be assembled, but that there are different ways to accomplish this task, and some are better than others.

Precedence relationships have the drawback that a single precedence graph cannot encode different classes of disassemblies. In an effort to overcome this issue, Homem de Mello and Sanderson introduced AND/OR graphs to the problem of planning assembly sequences for products [Homem de Mello and Sanderson, 1986]. This seminal work provided a compact representation of all possible assembly sequences for a given product. As is common throughout the literature [Homem de Mello and Sanderson, 1988a, DeFazio and Whitney, 1987, Lee and Moradi, 1999], this work considered assembly as a disassembly planning problem that was then executed in reverse. The fully assembled goal configuration of an assembly is maximally constrained and thus the branching factor during
2.3. Related Work

disassembly is smaller than it would be for directly planning the assembly of the product (several products can be assembled from the same parts in different orderings, so a forward planner would explore many unnecessary directions) [Homem de Mello and Sanderson, 1988b, Röhrdanz et al., 1996, Homem de Mello and Sanderson, 1988a, Homem de Mello and Sanderson, 1991a, Homem de Mello and Sanderson, 1991b]. The HighLAP system was a high-level system for generating and evaluating assembly sequences [Röhrdanz et al., 1996]. CAD data of the desired assembly was used to construct an undirected graph of component connections. The assembly sequence generation algorithm then checked all cutsets of that graph for stability of sub-assemblies and existence of depart motions (infinitesimal motions of parts relative to one another that do not result in collisions between components) in order to construct an AND/OR graph representing all valid assembly sequences. Homem de Mello and Sanderson presented a complete and correct algorithm for generating the AND/OR graph in a way that also incorporated some basic feasibility checking of the considered assembly operations [Homem de Mello and Sanderson, 1988a, Homem de Mello and Sanderson, 1991a]. Starting with a graph of connections between the parts of the assembly, a cutset method was used to generate all decompositions of the assembly into two subassemblies (binary monotone assemblies). Each decomposition then had to pass feasibility checks to determine its geometric feasibility (Is there a path?), mechanical feasibility (Can the necessary motions/forces be generated by the agents?) and stability. In reality these tests are difficult to perform since they require a level of reasoning about the task and the specific environment that cannot easily be incorporated into the assembly planning framework. This limitation was one of the primary motivations for this thesis. AND/OR graphs intuitively represent parallelism that is possible for assembly sequences. However, they consider only the (intermediate) states of the structure being assembled, not the assemblers performing the task. Instead of considering arbitrary sub-assemblies being assembled (as AND/OR graph solutions allow), this thesis limits its scope to monotonic assemblies where only a single component is added at each step. This simplification allows for a simpler state-graph representation that includes both the state of the structure and the assembler throughout the task (see Chapter 3). Sub-assemblies to be considered for this work have to be specified by an operator. AND/OR graph approaches might be useful to aid or assist the operator in selecting helpful decompositions in future extensions of this work.

Wolter presented an assembly planner that considered a larger scope of the assembly problem than other work [Wolter, 1989]: In addition to planning the sequence of operations, the planner also considered fixturing and workspace design. This level of reasoning can be useful in industrial settings where the workspace layout can be optimized for a given assembly task, but it is less applicable to assembly tasks in less-structured environments. STAAT was another assembly planning system for industrial assembly sequence generation [Romney et al., 1995]. This work extended Wilson’s thesis work [Wilson, 1992] on non-directional blocking graphs (NDBGs) and used them as the underlying data structure. NDBGs are efficient representations to determine how an assembly can be separated into subassemblies, but they only consider infinitesimal depart motions for components relative to each other or single-direction motions to infinity. In assembly line settings, this restriction is rarely a significant limitation as these motions correspond to the capabilities of work cell robots. In realistic scenarios such as those considered in this thesis, especially in constrained environments, the ability to remove components from a structure generally depends on the existence of a complicated motion planning solution for the robot performing the task.
2.3. Related Work

Fox and Kempf proposed an opportunistic scheduling system for robotic assembly lines that used the fact that many assembly sequences can assemble parts into the same product [Fox and Kempf, 1985]. Instead of relying on a single fixed assembly sequence, the system used parts as they became available if they were part of any sequence that would bring the assembly closer to its goal state. This method was the assembly planning equivalent of partial-order and least-commitment symbolic planning strategies. In the presence of execution-time exceptions, however, such an approach is only feasible if physical backtracking is possible. Xia and Bekey described another system that separates planning and scheduling into off-line and on-line processes, respectively [Xia and Bekey, 1988]. During planning, desirable assembly sequences were encoded in a hyper-graph. At execution-time, robots and resources were allocated to the desired actions. If such an allocation was not possible, the scheduler switched into opportunistic mode and tried to make progress with whatever resources were available until it could get back on track with the desired plan. Such an opportunistic approach could extend the work presented here to give the entire system additional options for exception handling and error recovery. However, as realistic assemblies are subject to many constraints, opportunistically choosing a locally best step might not be desirable in the interest of global execution robustness. Lazzerini and Marcelloni presented a genetic algorithm (GA) based approach to assembly planning [Lazzerini and Marcelloni, 2000]. Starting with randomly initialized (and potentially unfeasible) assembly sequences, purposefully designed crossover and mutation operators evolved the sequences first toward feasible ones and then toward optimal ones in terms of minimal repositioning required, minimal grasp changes, etc. This approach can take advantage of the inherent structure of assembly problems only very indirectly through properly designed mutation operators that may have to be modified for each new structure being planned. In contrast, this thesis uses geometric properties of the components and structure to generate and evaluate possible assembly actions and sequences.

2.3.2.2 Assembly Constraints

Assemblies are subject to numerous and often problem-specific constraints that make some assembly sequences better than others. Jones and Wilson presented a survey of many constraints required for traditional assembly planning in a factory setting [Jones and Wilson, 1998]. Wilson introduced a system to reason about tools used in assembly steps [Wilson, 1996]. The framework distinguished between pre-, in- and post-tools depending on when they were used in the assembly step. The applicability was a yes/no answer for a given assembly step and tool, without the option to constructively solve the problem if the answer was “no.” The underlying representation of the framework was an extended version of non-directional blocking graphs that encoded applicability of tools in certain subassemblies based on the use-volume and placement constraints required by a tool. For this thesis, tool usage extends to manipulator placement for robots bringing additional parts to the structure and to sensing agents monitoring the performance and state of the manipulators. In later work, Wilson described how tool constraints could be encoded in augmented NDBGs, which allow tool-level assembly planning to be performed in polynomial time [Wilson, 1998]. In that work, however, the kinds of tools considered were still fairly limited to vertical applications. In addition, the tool’s approach path and any space required for a robot or human to use the tool (a critical aspect in this thesis) were not included in the considerations.
2.3. Related Work

Shin et al. introduced a measure of disassemblability for selecting good assembly operations [Shin et al., 1995]. This criterion was a combination of the separability of components or sub-assemblies and the stability of both the moving and remaining components. Instead of pre-computing all possible sequences to then select the best of them, their approach greedily evaluated the selection criteria one step at a time and selected the locally best next component. Wolter described a number of important constraints on good assembly steps and how the weighting between those constraints is highly dependent on the assembly problem at hand and the agents available to complete the task [Wolter, 1989]. Due to this interconnection, this thesis argues that assembly planning cannot be treated as a standalone problem but instead has to be closely tied into the other task planning aspects such as motion planning and recovery/re-planning. Many of the goodness criteria presented in these approaches can be and are included when choosing desirable assembly steps.

2.3.2.3 Assembly Partitioning

Given a representation of the relationship between components in an assembly, assembly partitioning addresses the problem of finding (valid) decompositions of the assembly into sub-assemblies. Assembly planning systems represent assemblies as graphs of connections between the components. Introduced by Bourjault, and used by DeFazio and Whitney, graphs of liaisons are undirected graphs with vertices for all components and edges between components that share a connection [Bourjault, 1984, DeFazio and Whitney, 1987]. Other work has extended the idea of graphs of liaisons and included information about the type of connection between parts along the edges of the graph. Homem de Mello and Sanderson included connection information and constraints with the edges. Partitioning the graph of liaisons into subgraphs then becomes a graph cutting problem [Homem de Mello and Sanderson, 1991a]. All cutsets of the graph of liaisons representing two (or more if considering splits into more than two parts) are potential partitions. However, not all cutsets represent physically separable subassemblies. These methods then needed an additional step to confirm physical feasibility. Of all potential cutsets, for many assemblies, only few actually represented physically feasible partitionings. This thesis assumes that only a single component is added to the structure at each step, and that any decompositions into sub-assemblies are specified by an operator. Assembly partitioning techniques could be added to the planning framework in future extensions to enable the system to automatically find suitable or advantageous decompositions.

Motivated by the inherent generate-and-check inefficiency of cutset methods, more physical feasibility-based methods have been developed. Wilson introduced blocking graphs as a data structure that allowed computation of assembly partitionings in polynomial time based on geometric features of the components [Wilson, 1992]. A directional blocking graph encodes blocking relationships between components for a given direction as directed edges. Connected components of a blocking graph without outgoing edges represent removable subsets of components in the specific direction. Exploiting the fact that blocking relationships change only discretely and remain constant over ranges of potential directions, non-directional blocking graphs (NDBGs) are graphs of directional blocking graphs that compactly encode physical constraints and feasible separability of assemblies. In the computation of blocking relationships, generally only simple types of motions were considered (e.g., infinitesimal translations, infinitesimal translations and rotations, or translations to infinity) [Wilson and Latombe, 1994]. Lee and Moradi introduced Directional Force Graphs...
[Lee and Moradi, 1999] as an extension to Directional Blocking Graphs in an effort of encoding additional information about disassembly steps into the representation.

Motion space [Halperin et al., 1998] is a concept similar to configuration space in motion planning [Lozano-Perez, 1983]. Each point in the space marks one instance of a class of parametrized motions (e.g., single translations, infinitesimal rigid body motion, etc.) and has a directional blocking graph (DBG) associated with it. As a whole, the entire space represents the non-directional blocking graph (NDBG) of the assembly. This approach used the fact that of all possible assembly partitionings (an exponential number of them, as could be computed by cutset methods [Homem de Mello and Sanderson, 1991a]), usually only few are actually feasible. Instead of generating all potential candidates and checking them for feasibility, this method generated only feasible partitionings. Selecting good or best sequences of assembly steps was then left to other processes. The approach described was limited to very simple motions, and it only checked for binary feasibility. Incorporating costs indicating better or worse assembly steps remains an open problem.

Sundaram et al. used a high-dimensional motion planning approach based on probabilistic roadmaps (PRM) to do disassembly planning [Sundaram et al., 2001]. They treated each component of the assembly as an individual robot and operate in the composite configuration space of all parts. The naïve approach of applying standard PRM has the common problems with narrow passages in C-space. To overcome this issue, Sundaram et al. heuristically computed likely good motion directions for parts as they move apart by using face normals. This was a very simplistic heuristic that only worked for rectilinear parts, and it became much more complicated as more than one component moved at the same time. As the complexity of the environment increased, the heuristics were not strong enough and the planner suffered again from the narrow passage problem. This characteristic is problematic when considering large assemblies in challenging environments.

The assembly decomposition problem is a difficult problem (it is NP-complete [Kavraki and Kolountzakis, 1995]). Goldwasser et al. proved a number of hardness properties about decomposition problems and related problems (instead of decomposing the problem into individual parts, removing a key component, or separating a given pair of components) [Goldwasser et al., 1996]. They proved that even finding an approximate solution to the decomposition problem is a hard problem to solve by using approximation-preserving reduction techniques. Kavraki and Kolountzakis proved that partitioning a planar assembly into two connected sub-assemblies that can be separated is an NP-complete problem [Kavraki and Kolountzakis, 1995]. That work was an extension of an earlier proof that the assembly problem in space was also NP-complete. In its most general case, the assembly planning problem is PSPACE-hard [Natarajan, 1988]. However, when making a few assumptions (monotone, binary assemblies, infinite translations or infinitesimal rigid motions only), there exist algorithms to solve the problem in polynomial time. All assemblies considered in this thesis are of that class of problems. Natarajan also presented some handedness results that described the required number of hands for certain types of assemblies [Natarajan, 1988]. The analysis was restricted to very simple motions and very simple parts.

This thesis does not address automatic partitioning of assemblies. It assumes that an operator provides decompositions necessary to solve the problem. Future extension could use these and similar methods to include automatic decomposition (of the initial structure and dynamically in response to execution-time failures) in the assembly planning process. Including automatic decomposition prior to generating the initial assembly plan could be accomplished using one of the methods above.
2.3. Related Work

as a pre-planning step for the system as described here. Allowing the system to consider changes to
the decomposition in response to execution-time exceptions would require significant extensions to
the approach taken by this thesis.

2.3.3 Motion Planning

What makes this thesis unique in the context of assembly planning approaches is the fact that it
includes the motion of robots carrying out the assembly task in a constrained environment. Motion
planning expresses problems as robots (represented by certain degrees of freedom) moving through
a configuration space from some initial point to some final point. In the context of the assembly
planning problem, there are two major areas of motion planning that are relevant. Path planning
is the task of finding a trajectory of motion for one or more robots through an environment from
A to B without hitting obstacles. Manipulation planning considers robots moving with the goal of
manipulating movable but passive objects. The problems are posed in a robot-centric manner where
start and goal are expressed as configurations of the controllable degrees of freedom of the robots.

2.3.3.1 Path Planning

Path planning is concerned with finding a path for one or more robots through an environment with-
out colliding with obstacles or other robots. The canonical approach to this problem is to represent
the configuration space (or C-space) of the robot(s) where obstacles are expanded appropriately so
that the robots can be treated as points [Lozano-Perez, 1983]. Finding a path through the config-
uration space can be achieved in several ways. Grid-based methods (A*, dynamic programming,
etc.) can guarantee optimal solutions up to the resolution of the grid, but they quickly reach the
their limits of tractability as the number of degrees of freedom (and with it the dimensionality of the
grid) increases [Lengyel et al., 1990].

Methods have been developed to deal with such higher dimensional configuration spaces. In-
stead of explicitly representing the entire space and searching over it, they employ probabilistic sam-
pling techniques. Examples of such techniques are rapidly exploring random trees (RRTs, [LaValle
and Kuffner, 1999]) and probabilistic roadmaps (PRMs, [Kavraki et al., 1996]). These methods
give up optimality and completeness for efficiency and the ability to solve much larger problems.
However, they have a distinct drawback that makes their use tricky in constrained environments.
Since they rely on (random) sampling to explore or cover the free configuration space, they are
challenged by narrow passages. Either they require a high sampling density (and consequently lose
many of their speed-up advantages), or they require a very sophisticated sampling strategy (which is
difficult to develop and in general very domain specific). Motion planning for this thesis is accom-
plished using a single-query PRM-based planner with lazy collision checking (SBL) from Stanford
University’s Motion Planning Toolkit [Sanchez and Latombe, 2001, Schwarzer et al., 2005]. Since
the environment for assembly planning changes between most calls to the motion planner (due to
the growing structure), a single-query method is most appropriate in this domain.

Decomposition-based motion planning is especially suitable for configuration spaces with highly
constrained areas (narrow passages in configuration space). Brock and Kavraki developed algo-
rithms using decomposition-based motion planning that enable real-time motion planning for high-
DOF robots [Brock and Kavraki, 2001]. The approach first computes tunnels through the free configuration space to focus motion planning efforts. Yang and Brock developed an algorithm to focus roadmap planners toward tight passages along a robot’s path, and to plan out of those narrow passages into free space [Yang and Brock, 2005]. Once the robot is in free space, its motion can be planned with a much lower sampling density. The algorithm uses workspace information (which is of constant and low dimensionality) to restrict and guide configuration space exploration of much higher complexity. This thesis considers many metrics to evaluate the quality of assembly steps similar to the ones Yang and Brock’s approach used when pre-processing the environment through which the robots move (e.g., narrow passages along a robot’s path, constraintness of start and goal positions, etc.).

Alternatively, potential function approaches can be used where obstacles have repulsive potentials and the goal exerts an attractive force. The robots then follow the potential gradient to reach their goal [Barraquand et al., 1991]. Potential function methods can produce robot motion at any position within the configuration space, but special care has to be taken to avoid getting trapped in local minima. Navigation functions are potential functions with only a single, global minimum [Koditschek, 1992]. Lastly, the path planning problem can be approached geometrically. From the Voronoi decomposition of the configuration space, a roadmap can be constructed, which can then be used for navigation purposes. This method is primarily useful for planar problems, since the Voronoi diagram becomes disconnected in higher dimensions.

Most path planning applications consider the problem of finding paths through (complex, dynamic, etc.) environments where the robot is not allowed to collide with obstacles. A variant of this problem is the navigation among movable obstacles (NAMO) domain, where the robot moving through the environment is able to move certain obstacles out of its way. Chen and Hwang approached this problem using a simple heuristic to achieve fast, but not complete, performance in many practical settings [Chen and Hwang, 1991]. A global planner generated a sequence of sub-goals at which the cost of moving obstacles out of the way combined with a generalized cost for the robot’s path was optimized. These sub-goals are spaced closely enough for a local motion planner to be likely to succeed in connecting them, if that was possible. If the local planner failed, the global planner’s graph was updated by removing the failed edge, and the next best sequence was considered. Stilman and Kuffner developed algorithms to search a graph of disconnected areas of configuration space and simple paths in a planar workspace of the robot for obstacles to move in order to reach (sub-)goals [Stilman and Kuffner, 2004]. Stilman et al. presented a navigation planner for NAMO domains [Stilman et al., 2006]. The work focused primarily on a humanoid robot walking through an environment with movable obstacles. In order to accomplish the task, the state of the world needed to be estimated, motions of obstacles had to be evaluated, and then the actual walking motion of the robot had to be planned. The assembly planner developed in this thesis does not consider movable obstacles, but instead the construction of a growing obstacle (the structure being assembled) in such a way that earlier assembly steps do not prevent later steps from being possible.
2.3. Related Work

2.3.3.2 Manipulation Planning

Manipulation planning is another aspect of motion planning that is an important part of assembly operations at the finest level of detail where robots interact physically with the real world. The goal is to plan trajectories for robots so that an unactuated object is moved from one place to another. Grasp planning and similar low-level operations are key components of such systems. Siméon et al. presented a manipulation planner for a single robot and one movable object in a complex environment based on probabilistic road maps [Siméon et al., 2002]. Unlike previous work, they allowed continuous sets of grasp and placement positions for the robot and movable object. The problem was represented as a manipulation graph where vertices mark connected components of the PRM in the intersection of the grasp and placement C-spaces, and edges encode transitions between those connected components. Gravot et al. proposed an approach using multiple connected probabilistic roadmaps to solve the larger manipulation problem [Gravot et al., 2002]. The key idea was to break the problem up into several layers of resolution, with a roadmap associated with each. Additionally, specific tasks (such as motion, grasping, etc.) had their own roadmaps. The connectivity relationships between the roadmaps dictated how new nodes were propagated to higher or lower level maps, and at what level the maps needed to be extended. Finding a solution then became a coordinated search through all roadmaps, to connect a local grasping roadmap for a movable object to an approach motion roadmap for the robot, followed by a transfer motion of the robot carrying the object, etc. The coordination between different levels of resolution necessary to plan assemblies was similar, but at a much larger scale, than what Gravot et al. described. Instead of using the same techniques (in that case, PRMs) at different resolutions, this thesis leverages the distinct benefits of approaches specifically suited to large abstract or fine-grained motion problems.

Bozma et al. developed a game theory-based approach for controlling a disk-shaped robot to assemble disk-shaped components in a 2-D environment [Bozma et al., 1995, Bozma and Koditschek, 2001]. This work stands out among planners for assembly scenarios as it considers robots and components occupying the same workspace. While the robot worked toward moving all parts to their desired goal locations, its behavior was switched between three discrete controllers: next-part, mate-part and move-part. The next part was selected to be the part with the steepest gradient of the navigation function in the environment with all other parts considered as obstacles. If the robot was able to mate with that part, it moved it until the navigation function hit a (local) minimum. If the goal was reached, the local minimum was also the global minimum, otherwise the part was left there and another part was selected to be moved. Experimental results showed that on average, each part was picked up three times over the course of the assembly. An implementation of the switching control strategies developed was presented using a simple disk-shaped mobile robot [Karagöz et al., 2002]. Switching between different gradient directions of high-dimensional navigation functions implicitly assumes that the robot may leave a part being transported any location in the workspace that corresponds to a local minimum along the currently active dimensions of the potential function. This is an unreasonable assumption for this thesis where components can only be left at their storage location or connected to other components forming a structure.

Domestic assistance scenarios are beginning to play an increasingly important role in robotic applications. A number of platforms have been introduced that are capable of dextrous grasping and manipulation of common household items. Willow Garage’s PR2 [Marder-Eppstein et al., 2010]
has been used to demonstrate tasks from navigating through an apartment while opening and closing doors plugging itself in and even folding laundry [Maitin-Shepard et al., 2010]. Intel Research’s Home Exploring Robotic Butler (HERB) is able to retrieve items from cupboards [Srinivasa et al., 2009]. Closest to what could be thought of as an “assembly” problem, the domestic assistance robot ARMAR developed at the Forschungscentrum Informatik at the University of Karlsruhe, Germany can load and empty a dishwasher [Astour et al., 2006].

In the context of this thesis, manipulation planning is part of the robot behaviors being chained together into assembly sequences. The goal of the assembly planner is to make sure that when a low-level task (e.g., “Connect the component you are transporting to the structure via this other component”) is to be executed by a robot, that robot is in a position from where to accomplish the task given nominal operations. Therefore, manipulation planning is used to determine the required target locations from which a robot can accomplish (parts of) assembly steps. The assemblies considered in this thesis require only simple reachability considerations when determining goal locations for robots’ motions through the environment. Assembly tasks involving more complicated manipulators can include explicit manipulation planning within the presented approach when evaluating the quality of potential assembly steps.

2.3.4 Other Related Work

Klavins et al. took a graph-theoretic approach to the problem of self-assembly by large numbers of small robots [Klavins et al., 2006]. Given an initial state of the robots and their connections and a set of rules (a graph grammar), the robots communicated and applied rules as they were applicable, modifying the graph of connections. In assembly scenarios, the goal was generally to have a single stable configuration where no rules can be applied anymore, and have that configuration be the desired assembly. In closely related work, Klavins introduced characteristic automata of graph grammars that could be used to analyze the behavior of that grammar from a given initial labeling of states [Klavins, 2006]. The automata are similar to marked graphs or Petri Nets and encode all reachable labelings of a state given its initial labeling and the rules available in the grammar and can be used to analyze reachability, cyclic behavior and deadlocks. This approach is applicable to “assembly” scenarios where large numbers of actively or passively self-mobile parts connect themselves into a growing “structure” as their (possibly random) trajectories intersect, such as in micro- or nano-robot domains. In large-scale applications as considered in this thesis, self-mobile components are not realistic.

Belta et al. presented a survey of symbolic planning and control methods for robot motion [Belta et al., 2007]. The tasks considered were robot-centric tasks involving formation control, navigation, etc. Their goal was to specify tasks in a high-level human-like language that the system used to autonomously synthesize controllers. The main focus of the paper was on the control aspects, rather than the motion or even cooperative behavior of the robots to perform a desired task.

The Trestle project is one of very few large-scale autonomous robotic assembly projects [Heger et al., 2005]. A team of three heterogeneous robots worked together to assemble simple truss-like structures. During their task, the robots followed a hand-written script that exactly specified the order of operations. The work proposed here will replace the manual scripting with an autonomous system capable of initially tasking the robots given just a high-level description of the desired goal,
2.3. Related Work

as well as deal with recovery operations in case of execution-time failures. Another robot system
that performs assembly tasks was developed at NASA’s JPL [Stroupe et al., 2005]. Two hetero-
genous rovers carried long beams between them from a storage location over natural terrain to a
building site where they had to precisely align the beam in order to place it on the growing struc-
ture. This work also focused entirely on the execution and control aspects of the problem using
force-feedback; no planning was incorporated.

Sliding Autonomy [Sellner et al., 2006] is a concept of shared control between a human opera-
tor and an autonomous robot system. It was used in the Trestle project\footnote{Trestle project web page: http://www.frc.ri.cmu.edu/projects/trestle} to increase reliability of task
execution by having a skilled human operator in the loop to solve problems as they arose. Similar
concepts of sharing control authority over robotic systems are also referred to as Adjustable Au-
tonomy [Dorais et al., 1999] or Mixed-Initiative Control [Riley, 1989]. These methods used shared
control as a way to relieve human operators from having to perform tedious tasks and instead focus
their attention to situations where the autonomous system had problems. While this earlier work
showed the clear benefit of shared control, it was primarily applied at the lowest behavioral execu-
tion level. This thesis expands the scope of Sliding Autonomy and applies it during error recovery
at the behavioral, executive and planning levels of the system. In an effort to improve task per-
formance modeling, earlier work developed a method that used a Markov system representation to
estimate overall task performance based on individual sub-task data [Heger and Singh, 2006]. Since
data about the robots’ performance on particular low-level tasks (e.g., repositioning, docking, etc.)
is abundant, it can be used to estimate overall task performance and likelihood of success. This
kind of information can also be used as part of the assembly planning process to find good or best
assembly sequences from a plan robustness point of view.

Karagöz et al. implemented their controller [Bozma et al., 1995, Bozma and Koditschek, 2001]
for a mobile robot for moving disk-shaped parts from an arbitrary initial condition to a specified
goal configuration [Karagöz et al., 2002]. The problem was approached from a game-theoretic
point of view where the robot iteratively moved components along the negative gradient of a high-
dimensional navigation function until it reached a minimum. The global minimum of each part was
at its desired goal location, but obstacles and other (at the moment stationary) parts could lead to
local minima before the goal was reached. By switching between components being moved, all
parts were guaranteed to reach their goal locations. There is no guarantee that local minima of the
navigation function correspond to valid assembly states in which the structure may be left while
other components are assembled. In a later paper, Karagöz et al. presented extensive results for
disk assemblies with their EDAR (Endogenous Disk Assembly Robot) of up to six disks [Karagöz
et al., 2004].

2.3.5 Shortcoming of Existing Approaches

As shown in Figure 2.1 and throughout this section, related work to date has been constrained by any
two of the three integral constraints on the assembly problem at a time. Since this thesis considers
problems that depend on all three constraints (see Figure 2.2), this sections describes what is missing
from each area of related work that falls short of a comprehensive solution. An overview of how
This thesis considers problems that are constrained in ways that neither traditional assembly planning nor motion planning can fully represent. However, a combined approach is able to reason about the entire problem and all relevant constraints.

Existing approaches perform on criteria that are important for this thesis is presented in Table 2.1 at the end of this chapter.

2.3.5.1 Symbolic Planning

Nowhere in symbolic planning approaches is there room for describing the physical world in which the problem has to be solved. In fact, symbolic planning approaches are efficient because they abstract away many aspect of the problems into simple heuristics. As a result, in the context of real-world problems where reachability constraints etc. are important, symbolic planners are generally either conservative (at best) or even wrong.

Resource and ordering constraints are considered at plan time, but physical constraints imposed by the environment cannot be represented. Once the robots start executing the “optimal plan” found by the planner, they will likely find themselves in situations where the robot assigned to perform a particular task cannot reach the location from which to do so, because there happens to be an obstacle or part of the structure in the way. This problem is usually detected when the robot maneuvered itself into a corner and discovered that the original plan is infeasible. Now the task planner
should take this information into account when re-planning the task, but the workspace that caused the problem is not in its representation. So the planner is left with (more or less) randomly selecting a different instantiation for the robots to try to execute. Clearly, symbolic planning alone is insufficient to solve the problem addressed by this thesis.

2.3.5.2 Assembly Planning

Traditional assembly planning is closely related to symbolic planning approaches, with the addition that some physical constraints are considered. In particular, constraints imposed by part of the structure or product being assembled are satisfied so that components will be considered during an assembly step only if they do not collide with others during assembly. This approach is generally sufficient in industrial settings where the environment is specifically engineered for the task at hand and so as to avoid collisions between the robots performing the assembly and the product. In addition, assembly motions are generally limited to vertical or horizontal operations where the robot remains outside of the workspace of the product being assembled.

From the point of view of the structure, assembly planning methods can produce optimal plans that respect a variety of construction-relevant constraints according to some cost metrics. These approaches do not consider plan repair or re-planning as such events are not part of an industrial application. Just as in the case of symbolic planning approaches, the inability to reason about the assembler and the workspace in which the assembly is being performed severely limits the direct applicability of traditional assembly planning work to the problem considered here (see Figure 2.3). Nevertheless, the underlying data structures used to represent the assembly problem are very useful and are incorporated into the comprehensive assembly planning system developed in this thesis.

Figure 2.3: Assembly planning cannot express constraints due to the environment in which the task is carried out. Such queries have to be outsourced to an oracle.
2.3.5.3 Motion Planning

Motion planning approaches consider, by design, the environment in which a task is carried out. However, they are often plagued by tractability considerations when several moving components are present. Motion planning problems are typically posed as finding a path or trajectory from an initial point in some high-dimensional configuration space to another point representing the goal. Several methods are available for finding such a path, but what is missing from the assembly problem for such methods to be applicable is the goal configuration (see Figure 2.4). While the global start and goal are easily determined as the fully disassembled product and the completely assembled one, respectively, the problem requires several sub-goals that correspond to the individual parts of the structure being assembled, or even sub-assemblies being added to a growing structure. Since motion planning is posed in the environmental space of the problem, the algorithms do not have a notion of the structure to be assembled, other than as parts at certain locations. For large structures, this representation results in a very high-dimensional space beyond computational tractability. What components can and cannot be assembled at any point along the assembly plan is not available information (which would constrain the complexity). Clearly, motion planning is an integral component of real-world assembly planning, but by itself, motion planning approaches are insufficient to solve the assembly planning problem.

![Motion Planner](image)

**Figure 2.4:** Motion planning by itself cannot reason about decomposing the problem into sub-problems that can easily be solved by a motion planner. Such queries that have to do with partitioning the structure have to be outsourced.

2.4 Summary

No single area of related work alone can provide satisfactory solutions for the more general assembly problem with real robots in challenging environments as considered in this thesis. However, the sum of approaches and concepts from several areas combined spans the space of required capabilities. The realization that assembly planning will require a motion planning component in order to become really useful and more general is not new. Due to the complexity of the resulting prob-
lem, to date no practical system demonstrating such a combination has been developed. This thesis changes this situation and enables symbolic assembly planners and continuous motion planners to meaningfully interact in order to generate reliable and robust assembly plans for multiple mobile robots in constrained environments. The specific contribution of this work lies in the integration between many different aspects of the problem and the planning techniques most suitable to them. The resulting framework builds on much of the work discussed in this chapter, and many techniques not implemented in the current version of the assembly planner could be added with little change required to the approach described in the next chapter. Where relevant, later chapters will highlight areas for future improvement by including additional work discussed in this chapter.
### Table 2.1: Assembly planning approaches are primarily focused on the structure being assembled, but they do not represent and consider the workspace in which the task takes place. Motion planners, on the other hand, do not have a sense of structure, or tasks, unless an external process gives them sub-goals, but they inherently consider workspace constraints. This thesis is unique in that it considers both types of constraints important for assembly in constrained environments. In order to make the problem tractable, the planner gives up optimality and generality and instead focuses on generating plans efficiently and executing them robustly.

**Legend:** ++ = can do or does well, + = can do or does, o = not a focus of this work, - = cannot or does not do, -- = does particularly poorly

<table>
<thead>
<tr>
<th>Approach</th>
<th>Feature / Capability</th>
<th>Consider workspace constraints</th>
<th>Consider structure constraints</th>
<th>Reason about physical feasibility</th>
<th>Reason about reliability / robustness</th>
<th>Can parameterize tasks</th>
<th>Can plan for large assemblies</th>
<th>Plan optimality</th>
<th>Planner generality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocksworld</td>
<td></td>
<td>- - + - - - - ++ ++ ++</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive Methods</td>
<td></td>
<td>0 ++ 0 - - - 0 o o o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate-and-Test</td>
<td></td>
<td>- ++ + - - - + 0 o o o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometric Methods</td>
<td></td>
<td>- ++ + - - - o o o - o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Navigation Functions</td>
<td></td>
<td>++ - - o o o - ++ + + +</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grid-based Approaches</td>
<td></td>
<td>++ - - o o o + - + +</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Probabilistic Approaches</td>
<td></td>
<td>++ - - o o o o o - - - o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manipulation Planning</td>
<td></td>
<td>++ - - + o + - - - o o o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMO</td>
<td></td>
<td>++ - - + o o o o o - o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>this thesis</td>
<td></td>
<td>++ ++ + ++ + + + - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Defazio and Whitney, 1987] [Kaufman et al., 1996]
[Homem de Mello and Lee, 1991]
[Wilson and Latombe, 1994] [Halperin et al., 1998]
[Koditschek, 1992]
[Lengyel et al., 1990]
[Kavraki et al., 1996] [LaValle and Kuffner, 1999]
[Siméon et al., 2002] [Stilman et al., 2007]
[Stilman et al., 2006]
Chapter 3

Basic Assembly Planning Approach

3.1 Overview

This chapter presents the approach taken to plan assemblies of large structures throughout this thesis. It contains fundamentals of the underlying representation and describes how that is used to generate assembly plans. Later chapters will then provide specific details on how this approach is extended to achieve robust (Chapter 4), efficient (Chapter 5) and scalable (Chapter 6) solutions.

Assembly planning as considered in this dissertation is the process of taking a desired goal configuration of components (i.e., a structure) at a given position in the environment and producing a plan of tasks that can be executed by mobile assembly robots. It is assumed that the robots are capable of performing elementary tasks such as “follow these waypoints” or “connect this component to the structure,” and that there is a common understanding of the initial conditions under which these behaviors can be activated. For example, the follow-waypoints behavior provides information about a robot’s characteristic size such that the motion planner does not return waypoints that are impossible to achieve. Similarly, the connect-this-component behavior indicates that it requires the robot to be within its sensing and manipulation range of the connection to be established before the behavior may be activated. While satisfying all initial conditions is no guarantee for success, they provide a line of separation between the responsibilities of the assembly planner and those of the robot behavior developer. The behavior developer is in charge of assuring that, barring any unforeseen problems, the robot can successfully complete the corresponding basic task and report its success or failure. In return, the planner’s responsibility is to ensure that the robot is in a state where all initial conditions for a behavior are met before activating it with the appropriate parameters. This division of responsibilities allows component behaviors to be developed and tuned in isolation and then made available to the planner to use. While the work done for this thesis assumes a human behavior developer, work done in parallel on the OpenRAVE robot framework [Diankov, 2010] could be incorporated into this assembly planning system to automatically generate low-level behaviors given robots, their geometries and capabilities.

Beyond producing a nominally feasible (i.e., feasible barring any unforeseen and unexpected execution-time failures) assembly plan, the planner must meet the following characteristics in order
3.2. Terminology

The remainder of this chapter first defines some important terms and concepts. It then describes the planner’s underlying graph-based problem representation. Finally, it sketches the approach to the three key areas of robustness, efficiency, and scalability. Each area of enhancement is treated in more detail and with experimental evaluation in the following three chapters.

3.2 Terminology

The following terms will be used throughout this dissertation when describing the planner, its problem representation or approach, and implementation details.

Components Components are the units that are brought together into a structure by the assembly robots as they execute the plan. The initial position of components in the environment is given to the planner as an input along with the configuration (i.e., connections to be established between components) and position of the desired structure. The term “component” is relative to the level of detail in the planner hierarchy – they can be individual nodes and beams to be assembled into walls, or walls to be assembled into buildings, or building to be erected to form an outpost.

Structure The structure is the collection of assembly components that are already assembled at any point along the assembly plan. Again, depending on the planner hierarchy used, the (partial) structure can describe a partially assembled wall of nodes and beams, or a partially assembled building of wall segments, or a partially completed outpost of different types of buildings.

Robot Behaviors Robot behaviors encapsulate a particular robot’s capabilities into lowest-level tasks to be chained into assembly sequences (e.g., “follow waypoints” or “connect component to structure”). Their applicability is specified by initial conditions that need to be met. The planner is responsible for parameterizing the tasks and ensuring that initial conditions are satisfied before
the behaviors are activated. During execution, the behaviors signal success or failure to the executive and planner. Behaviors are at the interface between the planned robot actions and the real world. Physical constraints are encoded in their applicability specification, and any execution-time exception starts with a failure of a behavior.

**Assembly State** An assembly state describes a partial structure (at any hierarchy level) and the position of any robot(s) involved around that structure. Assembly states are always structurally valid configurations (what constitutes a valid configuration may depend on the particular structure to be assembled). The robots’ positions around the structure are chosen such that they allow assembly behaviors to be activated from there.

**Assembly Step** An assembly step is the transition from one assembly state to another. During an assembly step, one component is added to the structure in the previous assembly state to create the structure in the next assembly state.

**Assembly Graph** The assembly graph is the underlying representation of the assembly problem. There is one graph at each level of the planner hierarchy that contains valid assembly states (always including the fully assembled and completely disassembled states) connected by nominally feasible (i.e., feasible if no unexpected execution-time errors are encountered) assembly steps. The planner searches this graph to find an assembly sequence.

**Assembly Sequence or Plan** Each assembly graph yields an assembly plan at the corresponding level of the planning hierarchy. The ultimate plan for execution is a concatenation of lowest-level plans along the highest level sequence (and any intermediate levels).

**Task Template** Task templates describe the requirements and parameters of the tasks to be executed by the assembly robots. An *AssembleComponentTask* representing a step in a lowest-level plan for example may be described as consisting of sub-tasks (each with sub-tasks of their own) to *Retrieve* (with its own sub-tasks *Goto*, *Align*, *Pickup*) and *Install* (with sub-tasks *Goto*, *Align*, *Connect*) the new component. The template’s leaf tasks (e.g., *Goto*, *Align*, etc.) activate behaviors on the robots.

### 3.3 Representation

The planning problem is represented as a directed acyclic graph of valid assembly states (vertices) connected by feasible assembly steps (edges). Of all potential assembly sequences in the graph, the planner selects the one that is most desirable based on given criteria of desirability (e.g., cost of assembly steps, accessibility by robots, size and structural stability of the partial structures along the plan, etc.).

The selected assembly plan contains all parameters necessary to instantiate a generic task template for the specific assembly task at hand. The parameterized task template contains all necessary inter-task relationships (serialization, parallelization, etc.) and can be directly executed by robots.
3.3. Representation

3.3.1 Assembly Graph

The assembly graph (AG) is a directed acyclic graph that represents all structurally feasible ways in which a given structure can be assembled:

\[
AG = (V, E) \tag{3.1}
\]

where

\[
V = \{\text{assembly state (i.e., structure state + robot position)}\}
\]

\[
E = \{\text{component being assembled}\}
\]

Its vertices represent intermediate assembly states (i.e., structure states (including the start and end state of completely disassembled and fully assembled structures, respectively) along with the position of the assembling robot(s) relative to that structure state (see Figure 3.1)). Assembly graph edges denote assembly steps of a single component being added to the structure. By including robot state at each vertex, each directed graph edge represents a well-posed assembly or motion planning sub-problem where the robot(s) initially are in some location (as specified by the robot state at the edge’s source vertex), they install a component (as specified by the graph edge) into the structure and end up in some new location (as specified by the robot state at the edge’s target vertex). Without requiring knowledge of preceding assembly operations, each edge can be evaluated individually to determine a cost or goodness score for performing the assembly step it described at the particular state of the overall assembly.

![Figure 3.1](image)

**Figure 3.1:** Assembly graph vertices contain a structure state and a robot position relative to that structure state. There are two special states: the completely disassembled state (where the structure state is empty and the assembling robot is at its home position, right), and the fully assembled state (where the structure is complete and the assembling robot is back at its home position, left).

Note that the structure of the assembly graph restricts the problems it can represent to linear and monotonic assemblies, i.e., ones where single components are added to the structure at each assembly step, and no disassembly takes place along the way. At the cost of increasing the size of the assembly graph, directed edges from larger to smaller assembly states could be included without
fundamental changes to the algorithms presented here to allow the planner to consider disassembly steps. This thesis does not consider disassembly steps as the robots and hardware in the assembly test bed do not allow such operations. However, as will become clear in Chapter 4, the ability to (partially) disassemble already connected parts can greatly increase the options available to the planner when recovering from execution-time exceptions.

While the monotonicity restriction can be removed without fundamental changes to the approach as described above, the requirement of linearity cannot unless time is somehow included in the assembly graph. Consider the structure in Figure 3.2. The only way this structure can be built is by simultaneously bringing all three components together. In order to represent structures like this, either the concept of assembly graph edges has to be expanded to include simultaneous installation of multiple components, or temporal links/constraints between graph edges have to be established. These extensions are open problems and left to future work. Another limitation of the assembly graph representation is that each assembly step is considered a unit (including component retrieval and installation). As such, the planner cannot consider combined setup operations where multiple components are retrieved from the storage location at once and then installed one-by-one at the structure location. Such a scenario would be beneficial if a storage location is far from the assembly site and the travel time for retrieval is significant.

Figure 3.2: This structure cannot be represented by an assembly graph. The only way the three components can come together is by simultaneously combining them into a structure. The required sense of time is not included in the assembly graph representation.

The assembly graph contains all valid sub-structures of an assembly, where structure validity is defined for a particular application but generally includes some properties of the structure itself (e.g., sub-structures are valid if they are fully connected, or structurally stable, etc.), as well as some reachability consideration for the robot(s) involved (e.g., robots’ locations are not inside the structure and thus unreachable, etc.). The extreme case where any subset of components is considered a valid sub-assembly yields the largest assembly graph. In realistic situations, however, many possible subsets of components are either not feasible (e.g., because components cannot float in mid-air without being connected to anything) or extremely undesirable (e.g., because the resulting structure is very unstable) so that assembly graphs in general are significantly smaller then they theoretically could be. The valid states are connected with directed edges from the smaller sub-structure to the
larger one, where a single component is added. For ideal robots of negligible size performing the assembly in an unconstrained environment, any trace through the assembly graph from the fully disassembled state to the completely assembled state represents a candidate assembly sequence. The planner’s goal is to evaluate possible assembly steps in the context of the available robots, as well as an unstructured and constrained environment, to determine good or better assembly sequences.

The properties of the assembly graph described above are achieved by construction in the process of generating the graph. As is common in the literature (e.g., [Homem de Mello and Sanderson, 1986, Shin et al., 1995, Sundaram et al., 2001]), the graph is built by considering the disassembly of the goal structure into individual components (this approach avoids growing dead-end parts of the graph that eventually result in assembly states without valid assembly steps to continue). This disassembly strategy makes the implicit assumption that any disassembly step executed in reverse is a valid assembly operation. This assumption holds for all assembly operations considered throughout this thesis.

The graph generation process is described pictorially and algorithmically in Figure 3.5 and Algorithms 3.1 and 3.2, respectively. The graph is initialized with the fully assembled structure and all robots at their home positions:

\[ V_0 = \{ \text{fully assembled structure, robots at home positions} \} \]  \hspace{1cm} (3.2)

Next, penultimate vertices with the same structure state (fully assembled), but robot(s) at their positions around the structure are added to the graph. Each reachable robot position corresponding to an approach direction of a removable component is added as a new state. Each new state represents the end of a final assembly step for different possible assembly sequences. Whether or not a component is removable and thus is considered in the graph expansion step is largely application dependent. In order to be removable, a robot to remove the component needs to be able to reach a position from where to manipulate the component (this requirement ensures that during assembly, a robot can reach a position from where to install the component in question). The examples in this thesis consider mobile robots assembling a planar structure. Thus, only components “on the outside,” i.e., part of the convex hull of the structure, are reachable in any assembly state. Thus, only external components are considered as they are the only valid last components to be installed. Each penultimate vertex is the source of a final directed edge to \( V_0 \). 3-D structures or the availability of crane-like robots working overhead would allow for different removability criteria.

\[ \forall \text{ read } \in (\text{Removable Components Approach Directions}) \]

\[ V_{\text{penultimate, read}} = \{ \text{fully assembled structure, robots at position read} \} \]  \hspace{1cm} (3.3)

\[ E_{V_{\text{penultimate, read}} \rightarrow V_0} = \{ \text{no component} \} \]  \hspace{1cm} (3.4)

After this initialization is complete, the disassembly based graph generation begins. Starting with one of the penultimate vertices, the robot’s position determines which component is considered for removal (since in the assembly direction, the component where the robot is positioned would have been the last component to be assembled). If removing that component yields a new valid structure state (according to the same validity criteria as above, e.g., connectedness, or struc-
3.3. Representation

Structural stability, etc.), all removable (in this case: external) components in that state are determined. Vertices with the new structure state and the appropriate next-removable robot positions are added to the graph. In addition, a directed edge is added to the graph from the new state to the previous state that represents the assembly of the removed component into the structure. If an equivalent vertex already exists in the graph, only a new edge is added.

\[
\text{if removing } c \text{ from assembly state at } V_i \text{ yields a new valid assembly state} \\
\forall \text{ rem in } \text{(Removable Components Approach Directions at new state)} \\
V_{\text{new}} = \{\text{new structure state, robots at position rem}\} \\
E_{V_{\text{new}} \rightarrow V_i} = \{c\} \tag{3.5, 3.6}
\]

If a new vertex was added, this process continues recursively until all partial structures are maximally disassembled. At that point the assembly graph is complete (see Figure 3.3). The complete assembly graph captures all possible ways to assemble the given structure. Generating the graph by disassembly guarantees that it contains only valid states from which the goal is reachable. It also

Figure 3.3: The complete assembly graph contains all possible ways to assemble the given structure. It respects all structural constraints of the assembly (e.g., internal components, etc.) and contains only valid states and assembly steps by construction. For clarity purposes, only a fraction of all vertices in the graph are shown here. The entire graph for the two-square structure contains 2,216 vertices and 10,798 edges.
3.3. Representation

implicitly ensures that all structural constraints (i.e., internal components must be assembled before external ones, etc.) are properly taken into account. For any structure, the associated assembly graph is static for any environment and thus can be pre-computed, stored and re-used. Plan generation (described in detail below) then is a graph search over the assembly graph to find the best sequence of instantiated assembly steps for a given environment and available robots.

Each vertex in the complete assembly graph has an in-edge for each reachable robot position around the associated structure state. It has an out-edge for each way a component can be added to its structure state (leading to new structure states and one or more robot positions at that new state). As the size of the structure (in number of components) and its interconnectivity increases, the assembly graph grows exponentially (see Figure 3.4 for some representative example structures and their assembly graph statistics). Since physically realistic structures are limited in their interconnectivity (even in many-component structures, no more than a few of them are ever directly connected), the theoretically possible worst-case combinatorial explosion in assembly graph size is avoided. However, even in realistic scenarios, the rapidly growing number especially of graph edges (which is where the computational expense lies during planning) will require smart techniques to keep planning tractable.

![Figure 3.4: A comparison of assembly graph sizes for different representative example structures. As the number of components and their interconnectivity increases, the size of the assembly graph grows rapidly. Shown are the number of graph vertices (left), number of graph edges (center) and number of potential assembly sequences (right) for each structure.](image)
Algorithm 3.1: The assembly graph is generated by considering the disassembly of the desired structure into individual components. The graph is initialized with a vertex marking the fully assembled structure and a set of penultimate vertices that add robot state at the end of the assembly process. From there, Algorithm 3.2 is called recursively until the all sub-structures are disassembled and the graph is complete.

**INPUT:** structure to be assembled (structure)

// INITIALIZE GRAPH WITH FULLY ASSEMBLED STRUCTURE
1 AssemblyGraph graph;
2 AssemblyState initialState = AssemblyState(structure, robot(s) at home position(s));
3 GraphVertex initialVertex = insert(initialState, graph);

// ADD PENULTIMATE GRAPH VERTICES
4 VertexList penultimateVertices;
5 FOREACH removable component of initialState DO
6   FOREACH reachable robot(s) position(s) at that component DO
7     // CREATE NEW STATE AND ADD IT TO THE GRAPH
8     AssemblyState newState = AssemblyState(structure, robot(s) position(s));
9     GraphVertex newVertex = insert(newState, graph);
10    addEdge(newVertex, initialVertex, graph);
11   END
12 END

// EXPAND GRAPH FROM PENULTIMATE VERTICES
13 FOREACH vertex in penultimateVertices DO
14   expandGraph(newVertex, graph); ←— call to Algorithm 3.2
15 END

**OUTPUT:** complete assembly graph (graph)
3.3. Representation

Figure 3.5: Assembly graph constructions starts with the fully assembled state. One vertex is added to the graph for each reachable robot position around the structure (outside of the convex hull of the structure). Each reachable (in this example: part of the convex hull of the structure) component is considered for removal. If the resulting sub-structure is valid, states for all next-removable robot positions are added to the graph. This process continues until all sub-structures are fully disassembled and the graph is complete. Algorithms 3.1 and 3.2 provide a more formal description of the graph construction process.
Algorithm 3.2: During assembly graph generation, sub-assemblies are recursively disassembled further until all valid assembly states have been generated and added to the graph.

**INPUT:** vertex to expand (vertex)

**INPUT:** assembly graph to expand (graph)

// RETRIEVE ASSEMBLY STATE FROM VERTEX TO EXPAND
1 AssemblyState state = AssemblyState(graph[vertex]);

2 FOREACH removable component of state DO
   // CREATE POTENTIAL NEW STATE BY REMOVING COMPONENT
3   StructureState newState = removeComponent(component, state);
4   IF isValidState(newState) THEN
5      FOREACH removable component of newState DO
6         FOREACH reachable robot(s) position(s) at that component DO
7            // EACH REACHABLE ROBOT POSITION AT THE NEW STATE
8            // CORRESPONDS TO A GRAPH VERTEX
9            AssemblyState nextState = AssemblyState(newState, robot(s) position(s));
10           IF stateExists(newState, graph) THEN
11              // EQUIVALENT VERTEX ALREADY EXISTS IN GRAPH,
12              // ONLY ADD AN EDGE
13              GraphVertex existingVertex = findVertex(newState, graph);
14              addEdge(existingVertex, vertex, graph);
15           END
16      ELSE
17         // EQUIVALENT VERTEX DOES NOT YET EXIST IN GRAPH,
18         // ADD VERTEX AND EDGE
19         GraphVertex newVertex = insert(newState, graph)
20         addEdge(newVertex, vertex, graph)
21         // RECURSIVELY EXPAND GRAPH FROM NEW VERTEX
22         expandGraph(newVertex, graph); ← recursive call to Algorithm 3.2
23      END
24   END
25 END
26 END
3.3.2 Task Templates

Task templates establish the link between sequences of assembly steps (represented by traces through the assembly graph) and executable tasks for the robots. Those are generic task decomposition trees that are used by the planner during assembly step evaluation to determine what parameters need to be generated. For example, the template for an assembly step for a single robot installing a component into a growing structure (Figure 3.6) decomposes the assembly step into two primary sub-tasks RetrieveComponent and InstallComponent and two optional sub-tasks BraceStructure and RepositionStructure. Each sub-task, in turn, decomposes further into Goto, Align and Manipulate (which can be a Pickup, Connect or even Sensing operation) tasks. In order to obtain the complete assembly sequence, several AssembleComponent tasks are chained together.

![Diagram of task templates and assembly sequence](image)

Figure 3.6: The assembly task template is a generic model for all task instantiations the planner can produce. Depending on the specific structure to be assembled, different parameterizations yield different behaviors by the robots.

At this abstract level, each assembly step (edge in the assembly graph) is the same kind of a task. What distinguishes them is their individual parameterization (e.g., which components are involved, where robots need to travel to perform assembly steps, where and how new components are to be retrieved, etc.). The task template tells the planner what options it needs to consider when evaluating and attempting to instantiate assembly steps into executable task trees. During plan generation (described in the next section), the planner uses the given task template and attempts to generate a valid parameterization a robot could execute to accomplish the particular assembly step.
3.4 Plan Generation

If such a parameterization can be found, its cost is determined and used for selecting good or better sequences through the graph. If no parameterization is possible, constraints of the environment or robots make the corresponding assembly step infeasible.

3.4 Plan Generation

Once generated, the assembly graph contains all possible ways a particular structure can be assembled (in a linear and monotonic fashion) with a given number of robots. Very abstractly, any trace through the assembly graph represents a potential assembly sequence. However, given robot and environmental constraints, not all possible assembly sequences are actually valid, and of the valid ones, some may be more or less desirable than others.

During plan generation, robot and environmental constraints are applied to the assembly graph (which already encodes all structural constraints) to determine the actual goodness of assembly steps. These weights are obtained by applying applicable task templates to assembly graph edges and attempting to instantiate partial plans. They then guide a graph search that determines the best assembly sequence for a given structure in a given environment with the available robots.

3.4.1 Search for Assembly Sequence

Finding an assembly plan is a matter of searching the assembly graph for a desirable sequence of assembly steps. At a symbolic level, this is easy to do. However, in the context of planning assemblies for real robots in constrained unstructured environments, evaluating costs of potential assembly steps become a non-trivial problem. In order to ensure nominal feasibility (assuming execution takes place as planned), the evaluation of each assembly step requires the solution of a number of motion and/or manipulation planning queries.

A graph search algorithm\(^1\) determines the order in which edges are evaluated for feasibility (based on goodness of previous edges, etc.). Each edge is then evaluated in the context of the environment and initial conditions by applying an applicable task template and attempting to instantiate it. This process involves several motion planning queries (at least one per lowest-level sub-task) and can be computationally very expensive depending on the complexity of the environment and the number of robots involved (Chapter 5 addresses strategies to improve planning efficiency).

Motion planning solutions required to instantiate task templates again can come form any motion planner. This work uses a single-query probabilistic roadmap planner (SBL) from Stanford’s Motion Planning Toolkit [Sanchez and Latombe, 2001, Schwarzer et al., 2005]. This planner uses the Open Inventor\(^2\) format to describe the environment and robots. Given components and robots

---

\(^1\) This thesis uses A* to search the assembly graph. Other algorithms could be used interchangeably. In fact, an interesting question for future consideration is how different graph search algorithms can exploit structure in assembly planning problems to improve the system’s performance. For example, structures are increasingly constrained in their environment as they approach the completed state. While early in the plan many potential options may be equally good for all intents and purposes, at later stages there is a much steeper gradient in assembly step quality. Beam search or algorithms with some look-ahead might be able to probe far enough into the difficult parts of the search to effectively guide the process. The evaluation of such alternate graph search techniques is left to future work.

\(^2\) Open Inventor project web page: http://oss.sgi.com/projects/inventor
with geometric information in this format the motion planner can easily be initialized for each individual query. Similarly, if additional robots are involved in a particular assembly step, and their motion has to be planned in the joint configuration space, the sampling-based planner can easily accommodate this situation.

Algorithm 3.3 describes the entire process of searching for assembly sequences and instantiating assembly steps. It is in principle a standard graph search, but the evaluation of edges during the process (line 8 in Algorithm 3.3) is computationally very involved (each edge requires a call to Algorithm 3.4).

**Algorithm 3.3**: Assembly plan generation is a graph search over the assembly graph during which task templates are instantiated for graph edges, yielding a weight for the particular edge. Edge evaluation is the part that makes this search computationally challenging, as each edge requires several motion or manipulation planning queries to be solved.

```
INPUT: Vertex where search starts (fromVertex)
INPUT: Vertex where search ends (toVertex)
INPUT: Assembly graph for the goal structure (graph)

// INITIALIZE SEARCH WITH FROM_VERTEX
GraphVertex currentVertex = fromVertex;

WHILE goal vertex not reached DO
  IF vertex == toVertex THEN
    BREAK;
  END
  ELSE
    // SELECT NEXT EDGE ACCORDING TO GRAPH SEARCH ALGORITHM
    GraphEdge nextEdge = selectNextEdge(vertex, graph);
    // EVALUATE THE EDGE
    edgePlan = evaluateEdge(nextEdge); ← call to Algorithm 3.4
    IF edgePlan is invalid THEN
      nextEdge.cost = ∞;
    END
    ELSE
      nextEdge.cost = computeCost(edgePlan);
    END
  END

  // SELECT CURRENT BEST VERTEX ACCORDING TO GRAPH SEARCH ALGORITHM
  vertex = bestVertex(graph);
END

// REACHED GOAL STATE
AssemblyPlan plan = getSolutionTrace(fromVertex, toVertex, graph);

OUTPUT: Instantiated sequence of assembly steps (plan)
```
3.4. Plan Generation

Algorithm 3.4: During edge evaluation, the planner attempts to parameterize a task template for the given edge. When using the template shown in Figure 3.6, plans are required to retrieve, (possibly brace and reposition the structure), and install the new component.

**INPUT:** Edge to evaluate (edge)

1. EdgePlan plan;
2. WorldState state = getWorldState(edge.source());
3. TaskTemplate template = selectSuitableTemplate(edge, state);

// THE FOLLOWING ASSUMES USE OF THE ASSEMBLY TEMPLATE SHOWN IN FIGURE 3.6

// RETRIEVE PLAN CONTAINS GOTO, ALIGN AND PICKUP STEPS
4. RetrievePlan retrievePlan = generateRetrievePlan(state, edge, template);
5. IF retrievePlan is invalid THEN
6. EdgePlan invalidPlan(retrieve);
7. plan = invalidPlan;
8. END

ELSE

// PROPAGATE WORLD STATE TO END OF RETRIEVE PLAN

// NOTE: BRACE AND REPOSITION STEPS OMITTED FOR CLARITY

// ASSUMING RETRIEVE, BRACE, REPOSITION PLANS ARE ALL VALID,
// CONTINUE WITH TEMPLATE INSTANTIATION
// OTHERWISE RETURN INVALID PLAN

// INSTALL PLAN CONTAINS TRANSPORT, ALIGN AND CONNECT STEPS
9. InstallPlan installPlan = generateInstallPlan(state, edge, template);
10. IF installPlan is invalid THEN
11. EdgePlan invalidPlan(install);
12. plan = invalidPlan;
13. END

ELSE

// TEMPLATE SUCCESSFULLY INSTANTIATED FOR ALL REQUIRED STEPS
14. EdgePlan validPlan(retrievePlan, bracePlan, repositionPlan, installPlan);
15. plan = validPlan;
16. END
17. END

**OUTPUT:** Instantiated edge plan (plan), can be valid or invalid
3.4. Plan Generation

Problem. For instance, series of edges that pass through vertices with high out-degree (i.e., with many alternate actions toward the goal should the one that was initially selected fail during execution) can be given preference over lower-cost alternatives that may require all steps to be executed perfectly to avoid system failure. The exact selection criteria for where to expand the graph search is very application dependent. What is important is the realization that the search is grounded in the real world by evaluating edges using motion planners and guided by both their feedback and the structure of the graph toward desirable assembly sequences. Once the goal is reached, the backtrace through the graph to the initial state contains the assembly plan found by the planner.

In the context of the assembly task template shown in Figure 3.6, the following process occurs for each graph edge being evaluated (see also Algorithm 3.4 and Figure 3.7 for an illustration). Starting from the known world state at the source vertex of the edge, the robots have to travel through the environment to near the known storage location for the component they have to assemble next (RetrieveComponent::Goto), they have to align with that component (RetrieveComponent::Align) and pick it up (RetrieveComponent::Pickup). From there, they have to travel back into the workspace (now carrying the component they just picked up) to near the install location (InstallComponent::Goto), align with the structure where the new component needs to be connected (InstallComponent::Align) and finally perform the actual installation (InstallComponent::Connect). Both Goto tasks require solutions of motion planning queries in the environment, first for the robots

![Diagram](image)

**Figure 3.7:** Edge evaluation is where the computational complexity of assembly planning becomes apparent. For each edge being evaluated, the planner has to verify that (1) the robots are able to retrieve their next component, (2) the component can be picked up, (3) it can be transported through the workspace to near the install location, and (4) it can successfully be installed there. Each step requires a motion or manipulation planning query to be solved in an ever-changing environment.
themselves, and then for the robots carrying a component. Similarly, the Align tasks require short motion plans very close to obstacles as they approach the structure before planning the manipulation involved in the Pickup or Connect tasks.

If the graph search finds a solution, there is guaranteed to be a nominally feasible assembly plan, and the parameters for that plan are included in the solution trace returned by the planner. If the search fails, the desired structure cannot be assembled in the given environment with the available robots and provided task templates. In this case, an operator needs to step in and provide an alternate structure or goal location for the assembly. The current implementation contains evaluation functions (used in Algorithm 3.4) for the three primitive tasks in the template in Figure 3.6. Any task template consisting of combinations of those same primitives can be handled. New task primitives require corresponding evaluation functions currently to be developed by hand. Future efforts toward automating the robot task/behavior programming (e.g., using techniques similar to those part of the OpenRAVE framework [Diankov, 2010]) could remove the need for manual intervention and increase the generality of the system.

3.4.2 Task Parameterization

In the process of searching the assembly graph for the solution sequence, all assembly steps along the way are parameterized based on the supplied task templates. Goto tasks contain the necessary waypoints that guide robots through the environment toward their next task location. Align tasks specify reference bodies the robots have to align themselves with, and Pickup or Connect tasks include the assembly component involved. The parameters required for individual tasks to be executed by the robots depend primarily on the behaviors to be used. The task template contains information about ranges of applicable initial conditions from which certain behaviors are able to successfully accomplish their desired outcome (e.g., Align requires the robot to be within line of sight of the target it is to align with, no more than a set distance and angle away from that target, and not too close to allow for some maneuvering). It is assumed that a) the planner is able to guide the robot into the funnel of attraction of the behaviors chosen (or determine that it is not possible), and b) the robots can detect (during execution) when the initial conditions are not within the required range and throw an appropriate exception.

The instantiated task template is then passed on to an executive that is responsible for tasking actual robots by parameterizing robot behaviors using the values from the task template. If the robots behave the same as the planner thinks they do, and if no unforeseen (or unforeseeable) exceptions occur, the desired structure will be assembled. When errors occur, recovery mechanisms built into the system become active to resolve any problems and enable execution to continue.

3.4.3 Completeness and Correctness

As mentioned above in Section 3.3.1 and illustrated in Figure 3.2, this thesis considers only monotone assemblies where single components (individual components or pre-defined sub-assemblies, depending on the level of granularity of the planner) are added to the structure at each assembly step. This approach covers a large variety of realistic assembly problems. However, two classes of assemblies cannot be solved by the methods described throughout this document: structures that
require intermediate repositioning during assembly (see Figure 3.8); and structures that can only be assembled by assembling sub-structures and joining them into the desired goal assembly (see Figure 3.9). Note that structures that can be assembled with intermediate repositioning could also be assembled using the sub-structure decomposition method, but not vice versa. The planner described in this document will fail to find a solution for assembly problems that fall into either of these two classes. This failure will be determined when the planner realizes that the assembly graph does not contain a valid assembly sequence (this happens before a robot starts moving).

Figure 3.8: The square structure shown cannot be assembled in its final location as either beam A or beam B can be added, but not both, due to space constraints. However, if the structure can be moved by the robot during assembly, some or all of the structure can be assembled in a less constrained part of the workspace. Once assembled, the (partial) structure is then moved into position. Note that the shown assembly sequence is not a unique solution to this particular problem.

While not addressed in this dissertation, future work could extend the approach described in this chapter to handle both cases. Moving the structure to intermediate locations could be accommodated by adding additional “layers” to the assembly graph. Each layer would correspond to a particular position of the structure in the workspace. In addition to graph edges that change the assembly state of the structure, the extended graph would also contain graph edges “up and down” between layers where the structure is repositioned without changing its assembly state. Additional layers could either be added at evenly spaced intervals or at some (somehow determined as good) positions. The prior approach requires significantly larger assembly graphs to ensure that a solution can be guaranteed in the resolution of the layers. The latter requires significant automatic pre-processing or an operator’s expert knowledge to determine good or useful layers to include. This graph expansion would not be necessary for every structure considered. If assembly planning without repositioning fails for a particular structure, additional graph layers could be added in an effort to overcome the failure and find a solution (if one exists) where the structure is repositioned intermittently.
3.4. Plan Generation

Figure 3.9: The square structure shown cannot be assembled in its final location (or with intermediate structure repositioning as shown in Figure 3.8) due to an obstacle in the environment. However, assembling sub-structures and joining them into the desired goal assembly accomplishes the task. Note that structures that require intermediate repositioning can also be assembled with the appropriate decomposition into sub-assemblies. Automatically decomposing a larger structure into smaller ones is a difficult problem that is not addressed in this thesis.

Automatically deciding which layers to include in the assembly graph (as opposed to adding layers for evenly spaced discretized structure poses) is closely related to the problem of partitioning the structure. As was mentioned in Chapter 2, automatic structure partitioning is an NP complete problem [Kavraki and Kolountzakis, 1995]. Making simplifying assumptions, the problem can be solved in polynomial time in structured environments where infinitesimal depart motions or motions to infinity are valid and sufficient to determine separability. These assumptions hold true in traditional assembly line and robot work cell scenarios, but they are invalid for the types of assembly problems considered in this thesis. Whether or not an assembly can be decomposed into specific sub-structures depends as much on the environment as it does on the structure. This environment-dependence is what invalidates simple separability checks and requires motion planning solutions to be computed to determine whether or not a decomposition is valid. The presented approach does include pre-specified sub-assemblies (see Chapter 6) where an expert operator provides a feasible decomposition a priori. Under this assumption, and with the appropriate partial structures defined, the planner will find solutions if they exist.

Aside from these two classes of problems mentioned above, if there is a solution to an assembly problem using the task template and task primitives available to the planner, then the system will find that solution. Note that robot behaviors are tuned for autonomous operation and thus in general have conservative bounds on their allowable initial conditions. For example, an Install behavior may require the robot to be in a position where the manipulability of its manipulator is at least 0.3
3.5 Plan Execution

Given a parameterized task template, the executive is in charge of tasking robots according to the plan and monitoring their progress along the task. The task template specifies all inter-task constraints (serialization, etc.), and the executive coordinates with the robots as the task progresses. As exceptions occur, the executive interacts with the planner as necessary to resolve the problem. Figure 3.10 shows an overview of the entire system. The basic structure is a three-tiered architecture of behavioral, executive and planning layers. The executive and planning layers were used extensively as part of the Trestle system [Sellner et al., 2006] where three mobile robots cooperated (with human operator assistance is necessary) to assemble space-inspired structures. The focus of that work was on sequencing behaviors from the executive and providing Sliding Autonomy assistance at the lowest level (i.e., directly via joystick to the robots). This dissertation extends the architecture’s capabilities by adding a planning layer and enabling (autonomous and operator-assisted) repair and re-planning at all levels throughout the hierarchy.

3.5.1 Nominal Operation

During nominal operation the task-level executive receives acknowledgements of tasks successfully accomplished by the robot behaviors and responds by sending the parameters for the next task to be executed. The executive internally (via the sequence-level executive) monitors task progress and informs the operator of task status. As long as all tasks complete successfully as planned, the planning layer is not involved after the initial plan is generated and passed on to the executive.

Even under nominal conditions, the operator may pro-actively choose to set break points at certain points along the task to pause execution and verify task completion. In rare cases, the
3.5. Plan Execution

The assembly planning system receives a desired structure to assemble from an operator. It decomposes the structure into an assembly graph, searches the graph for the best sequence and parameterizes that using specified task templates for execution by the robots. As exceptions occur, they escalate up through the system until they are resolved.

Figure 3.10: The assembly planning system receives a desired structure to assemble from an operator. It decomposes the structure into an assembly graph, searches the graph for the best sequence and parameterizes that using specified task templates for execution by the robots. As exceptions occur, they escalate up through the system until they are resolved.

operator may detect anomalies in status updates the executive considers unproblematic. In such cases, the operator may actively request control of the executive to further inspect progress and request corrective actions if necessary. While operator-initiated failure detection is not a focus of the work in this thesis, earlier work using a subset of the architecture can be extended to enable both system- and mixed-initiative Sliding Autonomy interactions [Sellner et al., 2006].

3.5.2 Execution-Time Exceptions

The steps involved in assembling structures are difficult and challenging tasks for mobile robots. Even with the best plan, failures during execution are expected (or at least not unexpected) to happen. Thus, for an assembly planning system to be useful in realistic situations, it has to explicitly include strategies and mechanisms to invoke in response to those failures. In fact, this thesis argues that providing such recovery mechanisms has benefits beyond enabling robust and reliable operation (which are a necessity for a practical system). With the appropriate failure recovery ca-
capabilities in place, the system is able to exploit efficiency and scalability strategies that introduce artificial exceptions during run time (in exchange for tractable planning performance and the ability to consider very large structures and many robots) without significant detrimental effects on overall system performance. Consider the following scenario. If the robotic assembly system is capable of successfully handling exceptions caused by a robot not being exactly where it is supposed to be (e.g., due to drifting state or inaccurate sensing), then it will also be able to compensate for situations where the robot is not exactly where it is supposed to be due to approximations during planning, etc. Exploiting shortcuts and approximations during planning that introduce artificial runtime problems similar to those that will occur anyway (and thus can be dealt with) enables the system to plan more efficiently without sacrificing the required robustness.

In response to execution-time exceptions, this thesis considers three levels of failure recovery. At the lowest level, recovery takes place at the interface between a robot behavior and the task-level executive – the planner never is involved (contingencies). More involved exceptions may require a step in the current plan to be repaired before execution can continue. Finally, the most severe exceptions may require the current plan to be abandoned and a new plan to be generated that reaches the desired assembly goal via a different trace through the assembly graph. This section provides a preview of how exception handling capabilities are included throughout the system architecture. Chapter 4 covers the details of the methods used to achieve execution-time robustness and their effect on overall system performance.

### 3.5.2.1 Contingencies

At the lowest level, simple contingencies can often get execution back on track without requiring interaction with the planner. “Try again” often is a reasonable recovery strategy, particularly in response to exceptions due to sensing or manipulation problems. If a particular fiducial did not get detected immediately, a second look often picks it up (assuming it is in the camera’s field of view and simply a shadow or the like caused it to not be detected initially). Similarly, a connector may not engage on the first insertion attempt because the alignment was just slightly off. A second attempt may be more successful. While traveling along a series of waypoints, a robot may encounter a small obstacle in its path. If its local sensors are sufficient to circumnavigate the obstruction and continue on its path, the robot can perform such a contingency behavior.

### 3.5.2.2 Assembly Step Repair

After a set number of unsuccessful contingency recovery attempts (this is a tunable parameter that depends on the particular application), the exception escalates up through the executive. After contingencies, the next strategy is to attempt to repair the failed assembly step and continue on with the remainder of the plan as originally planned. This level of recovery is done at the interface between the task- and sequence-level executives. If the task encapsulating the failed behavior (in the task-level executive) is unable to continue execution, the planner is called to re-parameterize the failed task only. This re-parameterization is a reasonable strategy as with the notice of task failure, the task-level executive also provided some information about what went wrong. This new data is taken into account when attempting to re-parameterize the task.
A prime example of a repaired assembly step is a new motion through the environment (new Goto parameters) after the robot discovers a previously unknown obstacle in its path too large to circumnavigate with its onboard sensors. Sending information about the newly discovered obstacle to the executive along with the exception allows the planner to modify the robot’s task parameters (for its current task only) and provide it with a new set of waypoints to follow. The new waypoints will guide the robot to the same goal it was initially trying to reach, but this time the (now known) obstacle is avoided.

3.5.2.3 Assembly Sequence Re-Planning

If an exception is so severe that local plan repair is not successful, the final escalation is to abandon the current plan and request a new assembly sequence from the assembly planner, with the current state of the assembly as the start and the desired fully assembled structure state as the goal. While generating the initial plan can take a significant amount of time, re-planning can often reuse prior evaluations and exploit already-computed task parameters. If the re-plan is successful, the new task parameters are passed on to the robots and execution resumes. If there is no alternate assembly plan to recover from the exception, the system fails. System failure is only reported as the last resort after all possible contingency, repair and re-plan options have been exhausted (autonomously and with operator assistance at each level and at each step along the way). Irrecoverable failure indicates that either something went wrong that the robots cannot deal with (e.g., a dropped component), or that newly discovered information about the world (e.g., previously unknown obstacles) make the task impossible. Note that this condition would have been detected during the initial planning had everything been known then.

3.6 Summary

Given a desired structure to assemble, the assembly planner decomposes it into an assembly graph of partial structures connected by assembly steps. The assembly graph captures only structural constraints, but it is static for a given structure and thus can be pre-computed. Given the exponential growth of the assembly graph, this representation is appropriate for structures of few tens of components. Chapter 6 presents extensions using hierarchical problem decompositions to enable the assembly planner to work with structures of hundreds to thousands of components.

Searching the assembly graph and evaluating assembly steps enables the planner to verify feasibility given the environment and robots available. In the process of evaluating edges, task templates for each assembly step are instantiated. Once the search terminates, if successful, the resulting plan is nominally feasible, assuming the planner’s model of the robots’ capabilities is of sufficient fidelity and no unforeseen issues arise during execution. Again, the large size of the assembly graph presents challenges for straight-forward applications of graph search and motion planning techniques. Chapter 5 presents improvements to the basic approach that reduce planning time by over two orders of magnitude (from 11 hours to 2.5 minutes for a 21-component structure) compared to a best-first search approach by heuristically guiding the graph search and requiring the evaluation of many fewer graph edges.
Finally, and most importantly, as the assembly robots perform challenging tasks, execution-time failures are virtually guaranteed. In order to deal with those, the assembly planning system includes powerful failure recovery strategies that can address and attempt to resolve exceptions at the level that is most appropriate. Chapter 4 describes the error recovery strategies that are incorporated throughout the entire assembly planning system. As will be shown, those mechanisms are not only able to ensure reliable and robust execution of assembly plans, they also can enable the planner to artificially introduce additional exceptions in exchange for more efficient planning or the ability to consider larger structures, without suffering a significant performance loss.
Robust Robotic Assembly

4.1 Overview

The previous chapter described the fundamental approach to assembly planning: generating an assembly graph of feasible assembly steps for the goal structure, searching that graph for a desirable sequence, and parameterizing tasks to be executed by the mobile robots performing the assembly. The entire planning process assumes that tasks that are found to be feasible will complete successfully as planned. During execution, however, unexpected and unforeseen exceptions can and will occur that interrupt operations. Unless these exceptions are dealt with, they result in terminal failures of the robots’ tasks, leaving the desired assembly unfinished. This chapter focuses on how the basic approach (see Chapter 3 in general and Section 3.5.2 in particular) is augmented to ensure execution of complete assembly sequences is robust to exceptions. In addition to recovering from failures after they occur (by re-trying, repairing or re-planning tasks, Section 4.3), the assembly planner also favors choices that allow for easier recovery when and if exceptions occur (Section 4.4). Results show that recovery mechanisms (which include both autonomous actions and operator assistance) closely integrated throughout the assembly task hierarchy enable robust operation of the entire robotic assembly system even as unexpected and unforeseen errors occur.

4.2 Motivation

Assembly outside of well-structured assembly line settings is a challenging task for mobile robots. Robots are being tasked to perform precise and dextrous operations in close proximity to (not completely) known environmental obstacles, while actively contributing to the constrainedness of the workspace by purposefully constructing a growing obstacle (the structure). As a result, execution-time exceptions are not only possible but have to be expected even with the perfect plan. Actual conditions encountered during execution may differ from those assumed during planning, and the robots’ progress toward their goal affects the availability and feasibility of recovery options.

Turning an apparent problem into an opportunity, this thesis exploits the fact that large-scale assembly tasks rarely are time-critical (i.e., successful completion of tasks is much more important
than fast execution – better to complete the entire structure taking some additional time, than to very quickly reach a terminal failure point and not complete the task at all) and usually have a human operator or supervisor available who can monitor and assist the robots throughout their tasks. While in a perfect or at least well-known world this situation would argue for deliberate and lengthy planning using high-fidelity models, there is a sufficient degree of uncertainty in the environment and the performance of tasks that has unmodelled and unmodellable effects on available actions. Many deliberately planned actions may be invalidated (and thus the long planning time wasted) by very simple and uncontrollable execution-time problems. Many of those same problems can be resolved with very little effort by an operator, so that the additional burden resulting from planning at lower fidelity is negligible compared to the expense of expending more effort during planning. Having the benefit to perform tasks deliberatively and slowly if necessary for robustness considerations enables the system to include operator-guided exception-handling in the repertoire of recovery options. As will be shown, this kind of Sliding Autonomy interaction where autonomous robots perform most of the work while everything is within their capabilities and request help from an operator when they detect that they are in an off-nominal state is the essential component of truly robust operation.

4.3 Executing Assembly Plans Robustly

The first step toward error recovery during execution is detecting that something has gone or is about to go wrong (see Section 4.3.1 for some typical execution-time exceptions). The assumption is that either the robots themselves or a human operator monitoring their progress is able to make that determination and provide some information about the problem. The key is that new information is available as a result of the exception, enabling either the autonomous planner or the operator to attempt finding a resolution.

Any execution-time exception encountered by the robots during operation is caused either by some imperfect interaction with their environment (e.g., inaccurate sensing or manipulation), or by the environment being different than expected (e.g., a previously unknown obstacle obstructing a robot’s path). Once new information due to an exception is available, this work considers two recovery paradigms: Fully autonomous error recovery through plan repair and re-planning, and Sliding Autonomy interactions, where an operator can perform or assist in recovery actions. Depending on the specific application scenario, one mode may be more appropriate than the other.

4.3.1 Typical Execution-Time Exceptions

Even the best assembly plan is only as good as the knowledge and fidelity of information about the environment, robot capabilities, etc. In many realistic situations, in particular those where robots are sent to far away places to establish infrastructure in preparation of humans arriving later (such as in space applications, etc.), the available information is good at a high level and coarse resolution, but it is unreasonable to assume that perfect high-resolution data is available. This fundamental limitation can be anticipated during planning (see Section 4.4) to lessen its impact on execution robustness to some extent. But first and foremost it has to be compensated for as execution-time problems arise to ensure the desired degree of reliability and robustness.
4.3. Executing Assembly Plans Robustly

Typical problems arising during execution due to the use of imperfect information at plan time can depend on the particular application, but there are a number of standard issues highlighted below. Those generic types of problems will be the focus of autonomous recovery strategies detailed in Section 4.3.2. Any more involved issues that may require significant amounts of domain knowledge and insight to resolve are usually best left to an expert operator (the assembly planning system also provides opportunities for this mode of recovery), because the additional effort involved in developing autonomous recovery strategies for every possible problem is in general not worth the benefit. Even after thinking of “all” possible failure conditions, there will be at least one more once the robots start to execute their tasks. In practice it is more effective to follow an 80-20 rule where the autonomous system is tuned to perform approximately 80% of the work well, and the operator assists to resolve any problems caused by the remaining 20%.

4.3.1.1 Unexpected Obstacles

During execution, the robots may encounter unexpected obstacles blocking their paths. These can either be static obstacles that were not present in the map used for planning the robots’ paths, or moving obstacles that happen to be in the path of the robot. Alternatively, if the robot’s state drifts sufficiently through the course of an assembly sequence, its inaccurate global position may cause its path to intersect obstacles that in a perfect world it would avoid. Assembly operations themselves are defined relative to known landmarks or already assembled parts of the structure and thus are not affected by poor global localization. Traveling through the environment between assembly steps, however, depends on the robots’ ability to follow “global” paths.

Once an obstruction is detected (it is implicitly assumed that the robots executing assembly plans are equipped with suitable sensors to detect any obstacles in their immediate surroundings and stop in a safe state), new information is available that can help the robot (and/or an operator) determine an appropriate course of resolution. Small obstacles may be avoidable by the robot in the local space within its sensor horizon. Larger obstacles may require more involved re-routing in the combined map of previously known and newly discovered information. Or it may turn out that due to this obstacle, the current task is no longer feasible (either in its current parameterization, or at all), and the planner needs to be involved in the resolution attempt.

4.3.1.2 Sensing Problems

Sensing problems are among the major hurdles practical robot applications have to overcome. Often the difference between noise in the sensor measurement and required accuracy to perform a task is very small, and, in general, robots are lacking the scene understanding to creatively interpret noise and incomplete sensor information (humans, on the other hand, are good at this, which is another strong incentive to leverage their capabilities through Sliding Autonomy). Robots usually are able to detect loss of sensor information (e.g., due to occlusion as in Figure 4.1, left), but noisy data for high-precision tasks can be a challenge. Such situations can benefit from a human operator stepping in and confirming task completion etc.

In order for assemblies to exhibit reasonable structural properties (e.g., stability when moved or dragged around the workspace, etc.), the connections between components have to be fairly tight
4.3. Executing Assembly Plans Robustly

Figure 4.1: Key sensing problems include occlusions (left) and uneven lighting (right).

(in the assembly test bed, the required accuracy is to within a few millimeters and a few degrees in order to successfully perform a docking). Obtaining this level of accuracy from a stereo vision fiducial tracking system positioned some distance away from its targets is not easy. In the test bed the tasks are designed to be minimally affected by noisy sensor measurements and extensive filtering is employed to accomplish docking tasks (see also [Sellner et al., 2008]).

Once a sensing problem is detected (e.g., required objects are not in view, or the data about an observed object is too noisy for the task at hand) the robot has additional information that might help to overcome the problem. The robot itself may attempt to search for missing objects or reposition its sensors to reduce the noise in its measurements. If this level of recovery is insufficient, additional knowledge about the task and components involved may be used, for example, to utilize different fiducials attached to the objects in question. These changes may also require more substantial repositioning of the robots. If the sensing problem cannot be overcome to complete the current task, the planner has to get involved to find an alternate assembly plan.

4.3.1.3 Manipulation Problems

Establishing connections between components during assembly requires fine manipulation with small (millimeter) clearances (Figure 4.2). Coupled with imperfect sensing described above, this situation is very challenging for assembly robots. Slight misalignments (either due to inaccurate grasping or due to a slipping grasp) can be enough to cause component connections to get jammed. In order to avoid damage to the robots and other equipment, manipulation has to be stopped at that point and the problem needs to be addressed before execution can continue. Some manipulation problems result in fatal exceptions (e.g., dropping a component during assembly – since there is no possibility for recovery for those, they are not considered here), while others can be recovered from either autonomously or with the help of a human operator.

Autonomous strategies for resolving manipulation problems include the use of alternate sensing strategies (e.g., force sensing in addition to visual servoing) and changes in the robot’s internal configuration to increase manipulability of its arm, etc. If the problem persists, the task may be
4.3. Executing Assembly Plans Robustly

Manipulation problems are generally caused by poor alignment and the resulting jamming of the connector. If not detected properly, these problems can result in damage to the robot’s arm or the components being assembled.

changed to modify the order in which connections are established (e.g., the far end before the close end, or vice versa). Manipulation problems are the most difficult to deal with autonomously and also those with the greatest risk of damage to robots or components. They can often be resolved by the operator taking manual control and tele-operating the robot through the problematic task. It is particularly important that the operator be skilled in working with the robot as in many cases manual control starts in an already compromised position.

4.3.2 Autonomous Exception Handling

Autonomous exception handling attempts to react to and correct errors as they occur during execution. The system provides three levels of recovery – at the lowest (behavioral) level (Section 4.3.2.1), at the level of an individual assembly step (Section 4.3.2.2), or at the highest level of the entire assembly sequence (Section 4.3.2.3) – with the goal to address any issue at as low a level in the plan as possible. This strategy prefers local fixes and repairs to the robots’ plans (that are less intrusive and can often be performed quickly by an operator or even fully autonomously) over more involved global re-planning. The system resorts to the big hammer solution if the less-intrusive recovery options prove insufficient.

4.3.2.1 Contingency Response

The first level of recovery is a contingency response. The planner is not involved here, instead the resolution mechanism is built directly into the lowest-level tasks that interact with the robot’s behaviors. “Try again” is often a valid first response, as is simple obstacle avoidance etc. Examples of contingency responses include failure to detect a particular fiducial, or failure to latch and engage a docking connection. Since it is known that in many cases the sensor noise of the robot’s perception
4.3. Executing Assembly Plans Robustly

System is on the order of the accuracy required to perform the tasks, repeated attempts at doing the same thing are a reasonable initial recovery strategy. This form of error recovery does not technically involve the assembly planner, thus they are mentioned here only briefly.

Figure 4.3 illustrates an instance of an exception that can usually be resolved via contingencies at the lowest level. The robot encounters an obstacle (in this particular case, another robot on a collision trajectory) during execution and detects an imminent collision were it to continue on its path as planned. After throwing an exception, the robot’s on-board sensors are sufficient to realize that the obstruction was only temporary, and the robot continues with its task. In general, contingency responses are applied directly by the autonomous robot.

Figure 4.3: Paths unexpectedly blocked trigger an exception. If the blockage is small enough (or temporary as in this case), the robot can simply try again (after a short delay) and often be able to successfully continue execution. Here, the stopped robot’s path is clear after yielding to the other robot.

Depending on the scenario and situation, it may be desirable to have an operator sign off on any recovery strategy prior to it being executed (this can also apply to the initial plan). Figure 4.4 illustrates the event flow at the lowest level of the robot’s task hierarchy. There are two places where operator assistance is possible: To authorize execution (either of the initial plan or after an autonomous contingency resolution) by granting consent to proceed, and to take manual control if either no suitable contingency exists, or if all contingencies have been attempted unsuccessfully. If all attempts to resolve the exception at this low level are unsuccessful, and if the operator is either unable or unwilling (the operator may decline to assist the system at the lowest level if he thinks that higher-level autonomous or manual recovery strategies are better suited to address the problem) the exception escalates to the next higher recovery level. Whether or not operator assistance is allowed or required at any level is part of the planner’s parameterization and can be adjusted to fit the particular application. Some applications may benefit from operator assistance early on and at a low level (this interaction also requires a high bandwidth link between the robots and operator that may not always be available), while others need more operator involvement at higher levels where unmodelled factors such as an expert’s “gut feeling” may be helpful in selecting better mid- and high-level strategies.
Figure 4.4: Contingencies are handled at the lowest level of the task hierarchy. “Try again” often is a reasonable recovery strategy. If they can resolve the current problem, execution continues. Otherwise the exception escalates to the next higher level of recovery (step repair, see Figure 4.7). How soon and how often the operator is involved in the recovery process is a tunable parameter of the system that can be adapted to fit the particular application and scenario.
4.3. Executing Assembly Plans Robustly

4.3.2.2 Assembly Step Repair

If contingencies are insufficient in clearing an exception, it escalates up the task hierarchy, and the planner attempts to repair the failed assembly step only (Figure 4.5). As long as a different parameterization can be found that gets the system to the same goal state from where the next step expects to start, a local step repair is sufficient for execution to get back on track.

![Diagram](image)

**Figure 4.5:** When repairing an assembly step, the planner attempts to re-parameterize the failed assembly step only. Once the exception has been resolved, execution of the original plan continues.

The failed assembly step corresponds to an edge in the assembly graph, and, together with any new information (e.g., an unexpected obstacle, required components outside the field of view, etc.), the planner can locally re-parameterize the assembly step in order to let execution continue.

Re-parameterizing an assembly step is very similar to parameterizing it in the first place, i.e., applying a task template and instantiating it for the available robots in the given environment. The key here is that along with the exception new information about the environment and/or the task at hand became available for the new parameterization to take into account. As a result, it is not unreasonable to expect that a valid repair can be found.

Note, however, that the ability of the planner to repair an assembly step depends on the task at hand, the kind of new information available, and, most importantly, how far along an assembly step the exception occurred. Due to the requirement of continuing on with the original plan for all steps beyond the one being repaired, there are strong constraints on the final conditions of the repaired step. In general, the further along an assembly step a repair is requested, the fewer re-parameterizations are available that allow the original plan to continue without modification. Plan repair also becomes more difficult as the structure being assembled approaches its fully assembled...
4.3. Executing Assembly Plans Robustly

state. The fact that there are more components and a larger (and growing) obstacle present in the workspace reduces the available approach directions and alternate parameterizations of assembly steps the system can consider.

Figure 4.6 shows an example of (part of) an assembly task being re-parameterized in response to an execution-time exception. The robot detects that there would be an imminent collision were it to continue on with its path as planned (due to an obstacle that either was unknown during planning or was considered unproblematic then). The problem illustrated in the figure arises due to differences in motion models used during planning and the actual execution. The plan was generated for robot with a holonomic base (for planning efficiency reasons, see Chapter 5) but executed by a skid-steered vehicle, resulting in the shown corner cutting problem. This particular problem does not occur if the motion planner plans using the same motion model as the vehicle executing the plan, but drifts in state and odometry can still cause similar issues.

Figure 4.6: Paths unexpectedly blocked trigger an exception. The robot's position when the exception occurs, along with any new obstacles detected that triggered the exception, provide enough input to the planner to find a new parameterization of the Goto portion of the task that uses a different set of waypoints to guide the robot to the same goal it was trying to reach in the first place. Note that the particular exception shown is due to a mismatch between the planned and actual motion models (i.e., planned for a holonomic base, executed by skid-steered vehicle) for planning efficiency reasons (see Chapter 5). Without this simplifying approximation, similar issues would still arise due to drifting state and odometry.

Invoking a step repair from the robot’s current position in the workspace and any new obstacles detected that triggered the exception, the planner is able to re-parameterize the Goto portion of the task with a new set of waypoints that enables the robot to reach its original goal without colliding with the structure. Simple cases of this type of exception might be resolvable using contingency responses (e.g., if the robot’s on-board sensors are sufficient to determine a new path to the same goal), but more involved instances require invoking a proper assembly step repair.

Similar to the lowest task level, there are two ways an operator can assist with plan repair (Figure 4.7): Autonomously generated repairs can be approved before they are sent to the robots for execution, and assembly steps can be manually repaired if no autonomous solution was available or could be found. In order to repair an assembly step, the operator can modify the parameterization of
Figure 4.7: If contingencies are insufficient to recover from an execution-time problem, the exception escalates to the repair stage where the planner attempts to re-parameterize the failed task only (under consideration of any new information available due to the exception). If a repair can be found, it is sent back to the lower level of the executive for execution, otherwise the exception escalates again to the next higher level (sequence re-plan, see Figure 4.10). Subsequent plan repair attempts can be successful as the intermediate attempts to execute the task with new parameters (but still failing) provide additional information for the planner to use. How soon and how often the operator is involved in the recovery process is a tunable parameter of the system that can be adapted to fit the particular application and scenario.
the failed task (e.g., change waypoints of a Goto task, etc.). As at the lowest level of the hierarchy, the operator may decline assisting the system if he thinks or knows that higher-level autonomous or manual recovery strategies are better suited to resolving the problem. Future work could enable the operator to immediately chose the best level of recovery, but the functionality to bypass and then re-enter the task flow properly is not included in the current implementation (at this point, the operator would have to wait for a request at the appropriate level in the hierarchy before he is able to provide assistance there). If repairing the failed step proves to be impossible (using all available autonomous and manual options), the exceptions escalates up further to be addressed with a high-level re-plan.

### 4.3.2.3 Assembly Sequence Re-Plan

If a local repair is not possible, more extensive re-planning is necessary. In this case, a new trace through the assembly graph has to be found from the current state forward to the goal (Figure 4.8). As with the step repair discussed above, any exception requiring a sequence re-plan to be found contains a reference back into the assembly graph. The source vertex of the graph edge corresponding to the failed assembly step becomes the current state from which a new sequence has to be found.

![Figure 4.8: When re-planning the assembly sequence, the planner attempts to find a new trace through the assembly graph from the current state to the original goal state.](image)

As before, the availability of additional information that was not present during the initial planning (and may have caused the exception to happen in the first place) is critical to any reasonable expectation of success of a re-planning episode. In its current implementation, the system considers information about new obstacles (potentially requiring new paths through the environment) as well
as information about failed sensing (e.g., missing fiducials or too noisy data, requiring the selection of different markers) and failed manipulation attempts. This new knowledge after an exception has occurred is incorporated into the plan repair and re-planning attempts. As a result, the system will not fail immediately again for the same reason (the same problem may still exist, but the new failure will provide additional information again). The system attempts to find a solution until either too many (a tunable parameter for a particular application) attempts have failed, or the acquired information about the recurring exception makes it clear that it cannot be resolved without higher-level (autonomous or operator-assisted) input.

It is also important to note that most failures occur “somewhere along” the failed graph edge, such that the system is not really at the state described by the corresponding edge’s source vertex. Before the new assembly sequence can be executed, some physical backtracking is required to get back to the assembly state at the beginning of the failed task. If the exception occurs while the robot is holding a component, that component is returned to its storage location. It is implicitly assumed that the robot is able to reset to the pre-task state (e.g., no failures occur with a partial connection established that the robot cannot undo). If this assumption is violated (e.g., if the planner cannot find a parameterization for the \textit{Reset} task, the system is irrecoverably stuck. Note that the current implementation considers only backtracking to the pre-task state of the failed assembly step. Future work could make recovery more general by allowing reset operations to arbitrary points along the assembly plan.

During re-planning, the planner can make use of any edges previously evaluated, but only to the point they were not affected by the new information now available. The next chapter discusses efficiency improvements to the assembly plan generation and addresses trade-offs between spending planning time up front on the initial plan (and have more options to reuse work already done) or shift the burden to the re-planning (once it becomes necessary) at the benefit of a more efficiently generated initial plan.

Figure 4.9 shows an example of an exception requiring a sequence re-plan. During execution, the robot detects that it cannot reach the position from where it is supposed to install the component it is currently transporting. No contingency response or plan repair has been successful in resolving the exception, so it escalates to the re-planning stage. After invalidating the graph edge corresponding to the robot installing the beam it carries from left to right, it may be told to instead install the component right to left.

As with contingencies and step repair assistance, an operator can be part of the re-planning process (Figure 4.10). The planner is the highest level of possible error recovery. Any exception that has not been resolved at a lower level and cannot be resolved with a re-plan causes the system to fail irreparably. Note that before failure is declared, all contingency, repair and re-plan options (autonomous and with an operator, if available) have been exhausted.

### 4.3.3 Sliding Autonomy During Plan Execution

Depending on the application, (fully) autonomous plan repair or re-planning may not be desirable. In such scenarios, exceptions can be passed on to an operator in charge of deciding on the most appropriate resolution. While operator-guided exception handling is generally not very efficient, it is extremely reliable [Heger \textit{et al.}, 2005, Sellner \textit{et al.}, 2006]. Even though a human may be only
4.3. Executing Assembly Plans Robustly

Figure 4.9: If a robot encounters an assembly task impossible to complete (e.g., due to an unreachable area of the workspace) even after a number of contingency and plan repair events, the planner may be able to find an alternate trace through the assembly graph that represents another sequence of steps to take to arrive at the same desired structure.

of limited use to an autonomous planner that can consider thousands of alternate actions in a very short amount of time, as the exception escalates up the task hierarchy without being resolved successfully, the operator’s big-picture understanding of the task at hand becomes increasingly useful. The flow charts in Figures 4.4, 4.7 and 4.10 indicate all possible places where the options of operator assistance is implemented throughout the system. For any given application, any of those where operator involvement is not possible (e.g., due to bandwidth limitations or lack of a user interface) or not practical (e.g., due to the sheer amount of information to be considered) can be short-circuited or bypassed. And, as mentioned above, the operator always has the option to decline requests for assistance when he feels or knows that other mechanisms are better suited to resolving a problem than he is.

Previous work has examined Sliding Autonomy interaction between a remote operator and a team of robots at the lowest (behavioral) level only [Sellner et al., 2006]. Building on this prior work, the assembly planner allows for operator assistance and intervention at all levels of error recovery (Figures 4.4, 4.7 and 4.10). During plan repair, the operator suggests new parameterizations (e.g., new waypoints, different fiducials to track, etc.). Operator-assisted re-planning takes suggestions from the operator of either an entire new task sequence (which will be verified before execution), or preferred intermediate states through which the re-plan should go. This input allows the planner to incorporate any higher-level insight the operator may have, but that is not easily incorporated in the problem description.

The focus of this thesis is on the planning system that is able to incorporate operator input and assistance at all levels throughout its hierarchical structure. The implementation includes a bare-bones user interface that provides task status to the operator and accepts the operator’s input during recovery operations. Making this interface intuitive, effective, user friendly, etc. is beyond the scope of this dissertation and is an area where extensive work is necessary. The available interface includes the following operator input capabilities: a mouse-based “joystick” with which the operator can
Figure 4.10: If repairing the plan locally is insufficient to recover from an execution-time problem, the exception escalates to the re-plan stage where the planner attempts to find a new sequence form the current assembly state to the goal state. If a re-plan can be found, it is sent back to the lower level of the executive for execution, otherwise the task cannot be competed. How soon and how often the operator is involved in the recovery process is a tunable parameter of the system that can be adapted to fit the particular application and scenario.
4.3. Executing Assembly Plans Robustly

directly command robots to move left, right, forward or backward (during assistance at the lowest level), a top-down view of the currently known workspace indicating planned/achieved/incomplete waypoints and the ability to modify them (during repair assistance), a pop-up field where task parameters can be changed (e.g., fiducials to track), and a point-and-click assembly graph view where the operator can select (partial) assembly plans (during re-planning assistance). None of these interface components have been designed for or evaluated for usability or user-friendliness. They are simply ways to provide operator information to the system where Sliding Autonomy is possible. Work done as part of the Cluster project\(^1\) is using parts of this dissertation and develops improved user interfaces for some Sliding Autonomy interactions.

Combined strategies provide the most desirable system performance, where a human is able to attempt recovery after the autonomous system decided it cannot find a solution, or an operator can receive autonomously planned suggestions of possible solutions to choose from (or to ignore and select another option). As in earlier work, Sliding Autonomy during assembly planning and execution achieves the robustness required by the application (by having an operator to fall back to if necessary) without placing an unreasonable and unrealistic burden on the operator. At the same time, the system performs efficiently where autonomous solutions exist.

4.3.4 Results

The rate at which execution exceptions occur primarily depends on two aspects of a given scenario: The difficulty or complexity of the structure being assembled, and the complexity of the environment in which the assembly takes place (and, of course, the robots’ abilities to perform the required tasks but this work assumes that the ability is there, as long as everything remains nominal). This chapter focuses on what to do when execution departs from the planned nominal path.

Figure 4.11 shows an overview of the types of structures and environments considered in this section to evaluate the effectiveness and performance of the recovery strategies described above. It also provides an intuitive overview of the results described in more detail below. In a given environment, the problem complexity increases with the size and complexity of the structure to be assembled (left-to-right). As the structure grows, the likelihood of it causing problems to the robot as it traverses the environment increases. In addition, new components must be installed in close proximity to existing parts of the structure and with limited options of approach directions, increasing the difficulty further. For a given structure, as the environment complexity increases, the availability of feasible assembly locations decreases, thus making the assembly process more difficult (bottom-to-top). The overall difficulty of planning for and executing large-scale assemblies increases from the bottom left toward the top right in Figure 4.11.

The structures and environments shown in Figure 4.11 reflect the structures and environments available in the assembly test bed (in particular, the middle row is a to-scale representation of the obstacles present there). Experimental results with these scenarios show that allowing the operator to assist in error recovery is essential to achieving the required levels of execution robustness for practical applications. Without Sliding Autonomy, only simple structures in relatively unconstrained environments can be executed reliably.

\(^1\) Cluster project web page: http://www.frc.ri.cmu.edu/projects/cluster
Figure 4.11: An overview of the different structures and environments considered to evaluate the system’s robustness to execution-time errors. Error recovery becomes increasingly necessary as the structure and environment become more complex, and ultimate robustness to failures requires the presence of a human expert interacting with the system under Sliding Autonomy.

The results presented in this section were generated in a simulated environment with a single robot performing the assembly operations. Figure 4.11 shows the initial robot position and desired structure and position in each environment. Throughout the simulated assembly experiments, exceptions were generated to exercise the recovery strategies. Clearance exceptions were simulated by checking the environment for collisions between the robot (and any transported components) with an obstacle. If a collision was detected, the robot stopped at the last collision-free position along its path and an exception was thrown. For this experiment, no unexpected obstacles were included, thus all clearance exceptions were due to discrepancies between the planned motion model (holonomic base) and the one used during execution (skid-steered vehicle). In response to such an exception, the system autonomously attempted to find alternate paths and the operator could use the simple mouse-based “joystick” to move the robot. Sensing and manipulation exceptions were generated randomly at 20% and 40% of all attempted alignment and manipulation tasks, respectively. In response, the system attempted to reset to the beginning of the particular behavior.
4.3. Executing Assembly Plans Robustly

4.3.4.1 Contingency Response Only

As a baseline for comparison, Table 4.1 presents results of simulated assembly attempts of all three structures in all three environments from Figure 4.11 where the only error recovery attempts allowed were simple contingencies. Those included trying again (which often is a reasonable approach in response to exceptions that occur randomly) and hardcoded back-up-50-cm behaviors (attempting to get the robot away from an obstacle blocking its path). Here and throughout the remainder of the experiments described, after five unsuccessful recovery events using contingencies, the assembly run was counted as a failed attempt. This value is a tunable parameter of the system and was chosen

Table 4.1: Without exception handling during execution, the system’s overall robustness is insufficient for useful operation in realistic scenarios. As more complex structures are to be assembled in increasingly difficult environments, the system is unable to complete its tasks in this mode. The lightly shaded scenarios were successfully completed in many or most attempts, but the darker shaded scenarios failed every time. Shown in each cell are the plan’s cost, the average number of contingencies per attempt and the average percentage of task completion before failure was declared.
4.3. Executing Assembly Plans Robustly

empirically as one where enough options were given to overcome simple problems but not too much time was spent trying and re-trying unsuccessful recovery attempts. The more sophisticated the available contingency responses are, the higher this threshold can be set.

Not surprisingly, the limited recovery strategies allowed at this stage were not sufficient to reliably assemble large structures in complex environments. Very simple structures (a single square, first column of Table 4.1) were completed successfully on many or most attempts (light shading only). However, all other scenarios failed on every attempt (indicated by the dark shading). Even without complications from more complex environments (first row in Table 4.1), exceptions due to structural constraints in the two- and four-square structures (second and third columns in Table 4.1) proved too much than to be resolved by simple contingencies.

Plan cost, an indicator of the difficulty of the problem being solved, increased top-to-bottom and left-to-right. The costs for the first and second row environments are the same because the obstacles in the middle environment are sufficiently out of the way to not influence the nominal planning solution. All cells in the table were equally affected by the randomly generated sensing and manipulation exceptions. However, the lightly shaded cells in the first column showed hardly any clearance exceptions (because any obstacles were far enough away and the structure was small enough to be assembled primarily “from the outside.” The two-square and four-square structures, on the other hand, as well as the cluttered environment, were hampered by clearance exceptions. The two scenarios represented by the lightly shaded cells were completed successfully most of the time except where random chance caused the system to hit the five-contingency limit and fail. The larger structures mostly failed at the same point along their plan (i.e., they did not get beyond a partial structure of three components) because of collisions with the structure due to the use of the simplified motion model. The two-square structure in the cluttered environment was consistently assembled further because the more complicated environment forced the planner to generate a different plan than for the two less constrained environments. The number of contingencies encountered in an average assembly attempt is primarily a function of how far the attempt got on average. An assembly attempt that failed after three steps could have at most 15 contingencies, whereas a successful run of 8 steps could have as many as 40.

4.3.4.2 Autonomous Error Recovery

In order to mitigate some of the shortcomings of contingency-only execution mentioned above, this second experiment enabled autonomous repair and re-planning mechanisms as described in Sections 4.3.2.2 and 4.3.2.3. Autonomous repair events included finding new paths through the environment if the current path is blocked during \textit{Goto} tasks, backing up and restarting \textit{Align} tasks, etc. Re-planning events included returning the current component to its storage area and then finding a new sequence of assembly steps to reach the overall goal. All results are summarized in Table 4.2. In addition to the five allowed contingencies, each step allowed up to five repair events. Beyond that, the system attempted to re-plan the plan as long as an alternate plan could be found.

The two scenarios where contingencies alone were almost enough to ensure robust operations were handled successfully by the autonomous recovery strategies and succeeded in every attempt. Compared to the planned cost (total distance traveled by the robots), error recovery incurred a 23-25\% increase of the actual distance covered (including backing up, following different paths, etc.)
### Autonomous Error Recovery (Repair and Re-plan)

<table>
<thead>
<tr>
<th></th>
<th>Plan 1</th>
<th>Plan 2</th>
<th>Plan 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Planned Cost</strong></td>
<td>53</td>
<td>81</td>
<td>147</td>
</tr>
<tr>
<td><strong>Contingencies</strong></td>
<td>43.67</td>
<td>64.33</td>
<td>37.33</td>
</tr>
<tr>
<td><strong>Repair Events</strong></td>
<td>10.33</td>
<td>23.33</td>
<td>16.00</td>
</tr>
<tr>
<td><strong>Re-plan Events</strong></td>
<td>1.67</td>
<td>3.67</td>
<td>2.87</td>
</tr>
<tr>
<td><strong>Accomplished</strong></td>
<td>68.8%</td>
<td>91.3%</td>
<td>94.8%</td>
</tr>
<tr>
<td><strong>Actual Cost</strong></td>
<td>66.33</td>
<td>91.3%</td>
<td>66.33</td>
</tr>
<tr>
<td><strong>Planned Cost</strong></td>
<td>76</td>
<td>81</td>
<td>154</td>
</tr>
<tr>
<td><strong>Contingencies</strong></td>
<td>43.67</td>
<td>64.33</td>
<td>57.50</td>
</tr>
<tr>
<td><strong>Repair Events</strong></td>
<td>10.33</td>
<td>23.33</td>
<td>19.83</td>
</tr>
<tr>
<td><strong>Re-plan Events</strong></td>
<td>1.67</td>
<td>3.67</td>
<td>2.87</td>
</tr>
<tr>
<td><strong>Accomplished</strong></td>
<td>68.8%</td>
<td>91.3%</td>
<td>94.8%</td>
</tr>
<tr>
<td><strong>Actual Cost</strong></td>
<td>66.33</td>
<td>91.3%</td>
<td>66.33</td>
</tr>
<tr>
<td><strong>Planned Cost</strong></td>
<td>246</td>
<td>147</td>
<td>246</td>
</tr>
<tr>
<td><strong>Contingencies</strong></td>
<td>64.33</td>
<td>47.83</td>
<td>37.33</td>
</tr>
<tr>
<td><strong>Repair Events</strong></td>
<td>12.83</td>
<td>12.50</td>
<td>3.50</td>
</tr>
<tr>
<td><strong>Re-plan Events</strong></td>
<td>3.67</td>
<td>2.00</td>
<td>0.87</td>
</tr>
<tr>
<td><strong>Accomplished</strong></td>
<td>32.5%</td>
<td>19.0%</td>
<td>32.5%</td>
</tr>
<tr>
<td><strong>Actual Cost</strong></td>
<td>66.33</td>
<td>19.0%</td>
<td>66.33</td>
</tr>
</tbody>
</table>

**Table 4.2:** With autonomous exception handling, the capabilities of the system are greatly improved. Attempts where the system was unable to complete the assembly task failed too far along tasks to allow for effective repair or re-planning. Allowing disassembly operations during error recovery would enable the planner to solve many of those cases. Shown in each cell are the plan’s cost, the average number of contingencies per attempt, the average numbers of repair and re-plan events, the average percentage of task completion before success or failure was declared, and the actual cost of execution for the consistently successful assembly scenarios (not shaded).

...in exchange for robust execution without any operator assistance. In addition to completely solving the two simplest problems, autonomous recovery strategies also were able to solve a large fraction of more difficult scenarios (two-square structures in open and slightly constrained environments, lightly shaded cells in Table 4.2). When recovery failed, it was due to the system getting stuck in a loop where the planner repaired or re-planed in such a way that the recovery action lead right back into the same or a similar error. After a few iterations of this cycle, the allowed maximum number of repairs was exhausted, and if there were no more alternate re-planning paths, the attempt failed.
Autonomous recovery methods still failed in the dark-shaded difficult scenarios with large structures and/or complicated environments. All observed failures followed the same pattern where the robot got stuck in a loop with the repair or re-plan failing again (for the same or similar reasons) until the maximum number of allowed exceptions was reached and the attempt failed. This situation was often encountered when the robot was carrying a beam to be installed into a “C” shape to complete a square and randomly triggered exceptions forced it to reset to an earlier state. In the process of driving back to the earlier position, the robot’s beam got caught in the “C” where both left and right runs were blocked as well as the straight-back direction. While one possibility would be to develop additional contingencies and repair strategies for these failure conditions, they can be very environment and structure dependent and may not provide a general solution. In addition, experience has shown that even after every possible way a task can fail is thought of, once execution starts, there is at least one more possibility that was forgotten. Enabling robots to detect such off-nominal conditions, stop while they are still safe and request assistance from an operator is a sensible way around this problem.

The large jump in the average number of contingencies encountered in Table 4.2 compared to Table 4.1 is due to the fact that repair and re-planning operations were enabled here. Each time a step was repaired, the contingency count was reset and another five contingencies were allowed before the exception escalated to another repair. Thus, if the maximum number of contingencies and repairs was exhausted for an assembly step, it could have encountered as many as 25 contingencies. As with contingencies, the number of repair events largely depends on the number of steps along the plan an assembly attempt got before failing. Thus, there are fewer repair events in the last column scenarios than in the first two. As before, the cluttered environment was actually beneficial in the last column of scenarios (four-square structure) by forcing the planner to generate a plan that could be assembled further than those generated in the two less-constrained environments.

4.3.4.3 Exception Handling with Operator Assistance

With the exception of very simple scenarios (where autonomous recovery strategies were sufficient), operator-assisted error recovery achieved robust and reliable operation under all environment-structure combinations (see Table 4.3). In the event of an exception, after autonomous contingency recovery options (i.e., try again, back up, and/or plan new path from where the exception occurred) were exhausted, the operator received a request to control the robot’s position using a mouse-based “joy-stick” (e.g., step forward, left, right, back) and a simple user interface where information about the current problem was displayed. The display included the plan the robot was trying to execute, how much progress was made toward that plan, where the robot was at the time of throwing the exception, and any new information gathered as a result of the error. At each request for assistance, the operator could choose whether or not to provide assistance (the operator was asked to provide assistance whenever possible). If the request was declined, the system continued to autonomously attempt to repair and re-plan the failed tasks. If the operator chose to provide assistance, in a few clicks and generally 5-15 seconds, he was able to get the robot unstuck enough for the autonomous system to take over again. Note that the operator never had to remain in control until a task was completed. All he needed to do was position the robot such that the next autonomous contingency or repair event was likely to succeed (i.e., the loop the robot may have been stuck in was broken).
Table 4.3: In Sliding Autonomy mode, the system could always fall back and rely on a human expert to solve any problems that came up. Consequently, all scenarios could be completed successfully. Note, however, that the human at times performed tasks the autonomous system did not have available to it as they might be considered marginally safe (e.g., safety bounds to obstacles are smaller under manual control than in autonomous mode). Shown in each cell are the plan’s cost, the average number of contingencies per attempt, the average numbers of repair, re-plan and operator intervention events, and the actual cost of execution.

Typical exception conditions where operator assistance was beneficial fell into two main categories: a robot stuck between parts of the structure and an obstacle in the environment where a combination of backing up and turning was required to maneuver clear to where the autonomous system could take over again; and the planner generating a path through the environment that was feasible for a holonomic base but impossible to execute with a skid-steered vehicle. Moving the robot a short distance was always sufficient to either allow an autonomous motion to succeed or the planner to generate another solution that was feasible for the actual robot to follow. The operator declined requests if he felt he would have to manually control the robot a large distance (many robot lengths or more). At no point did a denied request for assistance result in a failed assembly attempt,
4.3. Executing Assembly Plans Robustly

although this would have been possible. A sufficiently sophisticated user interface to be developed in the future should include information for the operator to indicate the effect of declining a particular request for help on the likelihood of successful repair (e.g., “If you decline this request, there is only one alternate plan to successfully assemble this structure. Are you sure you want to decline?”). Techniques that could provide some of this information are presented in the next section.

Even though the number of failures handled increased with Sliding Autonomy (in most cases simply because the assembly progressed further and did not fail terminally after only a few steps), the operator was always able to accomplish successful completion of the entire task. In this particular experiment, the operator always fixed the problematic task, which is why there were no re-plan events in any of the scenarios. Since the operator was asked to provide assistance whenever possible, almost all problems could be addressed at the lowest (i.e., contingency response) level. As a result, fewer autonomous repair events were required than in the experiment above (Table 4.2). The average number of operator interventions increases with the complexity of the problem (with the exception of the middle scenario, which was caused by the randomness of sensing and manipulation exceptions) and generally was small compared to the number of assembly steps in a plan (8, 13 and 21, respectively, for the three sizes of structures considered).

4.3.5 Effects on Completeness and Correctness

The error recovery strategies described in this section are activated in response to execution-time exceptions and do not affect the completeness of (re-)planning. If an assembly plan exists in the assembly graph given the available task templates and robots, it will be found. Note, however, that selecting thresholds for the maximum number of contingencies and repair events allowed before an assembly step is declared impossible has the potential to – during execution – mark graph edges as invalid even though with enough re-trying the robot might yet be able to complete them. Re-planning and searching for an alternate assembly plan in a graph with such false edge costs can then fail even though physically a feasible plan exists. In practice, execution-time exceptions rarely ever make subsequent attempts more likely to succeed. In general the effect is the opposite. Thus, if an assembly state is labeled as impossible after a (properly selected) maximum number of allowed contingencies and repairs, in all likelihood, the step in question is not a desirable one to be part of the robot’s plan. In this situation, the system accepts the risk of missing a solution that is likely going to be a poor one anyway.

Contingencies, step repair and re-planning do not affect the correctness of the generated solutions. If a plan is found, it is a feasible one for the robots involved. The use of a simplified or approximate motion model during planning can result in plans that the actual robot cannot execute. These effects are discussed in detail in the next chapter about efficiency improvements. Note, however, that in practice, the error recovery strategies presented in this chapter were always able to recover from such artificially introduced problems and find a solution to the assembly planning problem (if one existed).
4.4 Planning for Robustness

As the previous section showed, error recovery strategies are very necessary to ensure robust execution of assembly plans. With an operator in the loop and available to assist and augment autonomous solutions, complex structures in complex environments can be successfully and reliably assembled. If a certain amount of domain knowledge is available (e.g., expected performance of certain tasks, robot sizes, sensor characteristics, etc.), this information can be taken into account during planning to bias the solution assembly sequence toward ones that are likely to require fewer repair events and less assistance from the operator.

Planning for robustness involves considering the goodness of particular assembly steps and assembly states to bias the planner to select sequences passing through those. What makes a step or state good or better than others is generally very application and structure dependent. This section introduces a sample of goodness metrics that fall into two categories: ones that consider the (local) geometry of the structure or the robot relative to it, and ones that consider the higher-level structure of the assembly problem as a whole.

4.4.1 Metrics for Robust Assembly Sequences

Both assembly graph vertices (assembly states) and edges (assembly steps) have qualities associated with them. While the graph by construction contains only valid states, some are still more desirable than others. The following metrics provide a means to quantify the relative goodness of assembly states. As mentioned above, these metrics are examples of possible ways to guide the planner. They are implemented in the current version of the assembly planner and have shown promise for the scenarios considered in this chapter.

4.4.1.1 Qualities of Assembly States

**Stability** This metric describes the structural stability of intermediate states along a plan sequence to construct a structure. Since partial structures have to remain intact (and may require repositioning in some scenarios beyond those considered in this thesis) while additional components are retrieved to be assembled, the more stable an intermediate structure, the more desirable it is as part of the plan. In practical terms, the more stable a partial structure is, the more force can be applied when installing additional components, the less likely it is to break if subsequent installation attempts encounter manipulation problems (e.g., jamming, etc.), and the better later components will fit.

This metric is implemented rather abstractly in the planner as a fraction of connections established in an assembly state of all connections to be established between the components involved in the assembly state in the final structure. Say a particular state contains five of the thirteen components of the final structure. If those five components are part of 9 connections in the final structure, and 6 of those connections are already established in the state in question, then the state has a stability score of 0.67. Long chains of components will have many yet-to-be-established connections whereas clustered components are missing fewer connections from the final configuration, resulting in lower stability scores for the prior compared to the latter types of structure.
4.4. Planning for Robustness

Plan Options Going Forward Exceptions during assembly plan execution are expected (or at least not unexpected) to occur. Each time a failure is reported, the system considers plan repair and re-planning options to be able to continue. A key factor determining the success of individual recovery actions and eventually the success of the overall plan is the availability of alternate actions to take when those initially planned are found invalid or impossible to complete.

The assembly planner considers two levels of available options from a state. Locally the assembly graph vertex’s out-degree indicates how many alternate next steps can be taken from a particular state. The planner prefers states in the graph that have more outgoing edges over those with fewer. The idea is that if an exception occurs following one edge away from a state, the plan can be repaired and re-planned along an alternate graph edge. This local view ignores the fact that while there may be many immediate alternate possibilities, they all may depend on many steps to complete successfully without alternatives later on. On a global scale, the assembly graph also contains information about how many traces through the graph reach the goal state from any assembly state.

4.4.1.2 Quality of Assembly Steps

Approachability Approachability is a measure of how much space there is around an assembly location. The more room there is (up to a limit, beyond which additional room does neither help nor hurt) for the robot to be able to perform its task even though it may not be perfectly aligned, the more approachable the location. While technically feasible, assembly steps with low approachability usually are likely to fail due to noise in the system. Instead of excluding any assembly step with approachability below some threshold (which could make the system miss viable plans), the planner is biased toward preferring plans leading through more approachable states.

Approachability is computed as the fraction of an Install behavior’s funnel of attraction (robot locations from where the behavior can be activated and successful operation can be expected barring unexpected failures) that lies in free space. The assembly planner evaluates approachability by finding the largest sphere centered at the robot’s desired position to perform an installation that does not intersect any obstacle. The fraction of the size of this sphere compared to a characteristic size of the robot is an assembly state’s approachability measure. If the largest sphere is greater than the characteristic volume describing the robot, approachability is capped at a value of 1.0. Once there is “enough” (a tunable application-dependent parameter) space, having more space does not improve the quality of an assembly step.

Note that approachability is entirely a local measure describing how much wiggle room a robot has to compensate for exceptions, poor sensing, drifts in state etc. when performing its tasks and attempting to recover from errors. A perfectly approachable assembly state may be useless because the robot cannot reach it in a global sense.

Expected Performance Each assembly step eventually decomposes into a sequence of atomic tasks for the robots to execute. Based on prior performance of the robot the planner estimates the likely outcome of such a sequence, or of particular steps along a given sequence. Earlier work has considered ways of making such expected performance determinations [Heger and Singh, 2006].
4.4.2 Results

The benefits of specifically planning for robust execution can best be seen in difficult scenarios. Table 4.4 shows the results of experiments similar to those discussed in Section 4.3.4.3, but with plans that take into account the metrics outlined in this section. In addition to the distance-traveled cost metric used in the earlier experiment, the planner here evaluates the metrics described above and computes a quality score that is used during the graph search.

Stability, approachability and expected performance already are scores ranging from 0 (bad) to 1 (good). The number of outgoing options is converted into that range by dividing the out-degree of a vertex by 5 (another tunable and application-dependent parameter that was chosen empirically for the structures and scenarios considered here), where 5 or more outgoing edges are considered to be as good as an assembly state can get. Finally, distance traveled is converted into the 0-1 range by comparing the planned length of the path to the straight line distance between start and goal. A value of 1.0 indicates that the robot can actually follow the straight line path to its goal, and as obstacles need to be circumnavigated, the score is reduced.

Table 4.4: Explicitly planning for robustness as described in this section reduces the number of operator interventions required to successfully complete tasks by increasing the ability of autonomous repair and re-planning to successfully recover from exceptions. Shown in each cell are the plan’s cost, the average number of contingencies per attempt, the average numbers of repair, re-plan and operator intervention events, and the actual cost of execution.
Empirical tests have shown that approachability and stability (which in the current implementation is closely related to “size” of the structure relative to the target structure) are the most useful when generating desirable assembly plans. The results shown in Table 4.4 were generated with a weighting of 30% for both approachability and stability, 20% for distance traveled, and 10% each for outgoing edges and expected performance. The main reason why outgoing edges are weighted so low is that the operator was specifically told to assist the system whenever possible (same as in the experiments above), which resulted in successful performance throughout without any re-planning events. The outgoing-edges score is specifically designed to provide options to be used during re-planning, and thus this particular metric was of limited use in this scenario.

The first thing to note is that the planned cost (to allow comparison with earlier experiments, the cost shown is again only distance traveled) of each assembly scenario is greater if the plan is biased toward robustness than previously, where distance traveled was the only contributor to cost. The actual plan cost is comparable and even slightly lower than it was before. This indicates that the plan that was eventually executed (along with all necessary recovery attempts) is closer to what was originally planned than in the earlier experiments. When planned for robustness, the assembly system is more predictable based on the initial solution returned by the planner.

The number of exceptions thrown (as indicated by the numbers of contingencies, repair and re-plan events) is lower than it was before. While the randomly triggered exceptions (sensing and manipulation) were the same as previously, this drop indicates that the plans generated with robustness measures incorporated into the graph search caused fewer clearance exceptions where the robot (due to the described mismatch in motion models) had to stop to avoid an impending collision. This benefit primarily came from the use of approachability and stability (i.e., “size” as implemented) metrics. Along with fewer exceptions, the operator had to intervene fewer times to provide assistance to the system.

These results show that by providing information to the planner as to what constitutes more desirable states to be in or actions to take (for mostly application- and scenario-specific reasons), the system was able to find solutions that are more robust to begin with (i.e., they did not require as many repair, re-plan and operator assistance events) and, once executed successfully, were no more expensive than the others.

### 4.4.3 Effects on Completeness and Correctness

As long as the metrics and weightings between them are chosen such that no little desirable assembly steps are marked as invalid, planning with a bias toward robust solutions still is as complete and correct as before. The planner may not select the shortest possible plan, but any solution generated will be feasible (for the effects due to the use of approximated motion models see the discussion in the next chapter on efficiency improvements), and if a plan exists, it will be found.

### 4.5 Summary

Execution-time exceptions will occur when mobile robots attempt to assemble large structures in constrained environments, no matter how good their plan is, due to inherent unpredictable and unforeseeable problems. Imperfect sensing, difficult manipulation tasks navigation around obstacles
and more all conspire to make the robots fail. This thesis makes full use of the three-tier architecture to include recovery mechanisms at each level of the hierarchy. Each exception is addressed at as low a level in the task structure as possible to maximally limit the impact and scope of the required recovery. Contingencies are the first response to failures at the lowest level. If those are insufficient to resolve the problem, the exception escalates to an assembly step repair strategy. There, alternate parameterizations for the failed task are considered in an attempt to complete the task despite the problem and then continue on as planned. If the task is found to be irreparable, the exception escalates further and triggers a global re-planning event where an alternate assembly plan is generated from the current state to the desired goal. Only if no alternate plan can be found (after all contingency and repair options are exhausted) does the system fail.

In addition to providing error recovery strategies throughout the planning framework, the system specifically enables human operators to be in the loop and provide assistance when requested to do so by the autonomous system. Experiments show that while fully autonomous operation can successfully complete simple problems, operator input via Sliding Autonomy is essential to ensure robust system performance. While more sophisticated autonomous strategies might reduce the need for operator intervention, they cannot eliminate it entirely, and the effort in developing them is generally not worth the gain. Exploiting synergies between autonomy and human assistance provides the desired performance. It is important to note that the demands of the operator are very light – only enough to overcome a stumbling block and get the autonomous system back on track.

In addition to handling exceptions after they occur, the assembly planner reasons about what makes for good or better assembly plans in the context of desired robustness. Including some simple (but largely application-dependent) metrics, the planner generates plans that are more likely to succeed and require less operator assistance.
Efficient Plan Generation

5.1 Overview

The first priority of a practical robotic assembly system is robust and reliable operation (Chapter 4) – speed of operation is secondary to ensuring that the desired structure is assembled successfully. Robots are envisioned to perform assembly tasks in domains where humans cannot or do not want to go, and where they will not have to compete with human work crews on time to task completion. Nevertheless, as robust execution is achieved, efficient plan generation becomes the next topic of interest, as it enables simulation of different scenarios and conditions in preparation of an actual mission in a reasonable amount of time.

In contrast to typical robotic assembly scenarios where factory robots produce thousands of copies of the same product, this thesis considers the construction of one-of structures. In these domains, where exceptions are likely to occur during execution (requiring modification of the initial plan) generating good plans (as opposed to optimal ones) is sufficient, especially since powerful error recovery strategies are in place. In addition, planning efficiency not only applies to finding the initial plan. Every time the system has to re-plan in response to irreparable execution-time exceptions, an alternate plan has to be found without too much delay.

This chapter focuses on modifications of the basic approach as described in Chapter 3 that enable the system to generate (and repair/re-plan) assembly sequences more efficiently. These improvements are achieved by performing faster search over the assembly graph by either evaluating individual edges faster, evaluating fewer edges, or combinations of both techniques. Results show reductions in planning time of two orders of magnitude and more compared to a straight-forward application of the basic planning framework, and that the benefits of the methods increase as the structures become larger and more complex.

5.2 Motivation

Without modifications, the approach presented in Chapter 3 takes minutes to hours to find assembly plans for structures of only up to a few tens of components (see Figure 5.1). In the process, the
5.2. Motivation

Figure 5.1: The single square structure (left top) is a structure that has been assembled using mobile robots in the past, the two-square structure (left middle) is the smallest structure with an internal component, and the four-square structure (left bottom) is the largest structure assemblable with physical hardware in the test bed (right), as well as about the limit in assembly graph size that can reasonably be represented.

planner does a lot of work that eventually turns out to be unnecessary. By being smarter about where computational effort is spent, the planner can reduce the amount of time required to find plans (and to repair and re-plan) into the seconds to few minutes range for the same structures.

The baseline planner searches the assembly graph in a best-first fashion (using A* with a zero heuristic, in practice similar to breadth-first search), evaluating a large fraction of all graph edges in the process of finding the desired assembly sequence. Later improvements adding an informed heuristic will realize significant speed-ups in planning time (without any other modifications to the system). Future work on the planning framework presented here should include an evaluation of different basic search techniques. In particular, the structure of assembly problems where they are relatively unconstrained at first and becoming more constraint as the task progresses can probably be exploited. Note, however, that the choice of a different simple step early on in the plan can have significant effects on the feasibility and likelihood of successful execution of the entire plan.

During plan generation, each graph edge is evaluated using a motion planner for the robots involved in the corresponding assembly step. Once evaluated for feasibility, the quality or cost score for each edge is cached, eliminating the need to re-evaluate the same edge multiple times. Prior to the experiments, the assembly graphs for each structure were generated and saved. Each trial loaded the appropriate assembly graph and searched it for a solution sequence. The key properties of the three assembly graphs are summarized in Table 5.1. The size of the assembly graph grows
5.2. Motivation

exponentially as the number of components increases. The most dramatic increase is in the number of possible ways a structure can be assembled (as indicated by the number of traces through the assembly graph). While a human expert may have some high-level understanding about a particular assembly problem, he will definitely not be able to consider all (or even a large fraction of all) possible sequences to select the best or a good one, especially in cases where desirable sequences may not be obvious.

The baseline planner’s performance was evaluated for three representative structures of increasing complexity (see Figure 5.1 for the 8-, 13- and 21-component structures used for comparison) in the moderately constrained environment similar to the assembly test bed layout. Table 5.2 compares the plan generation statistics of the baseline approach for each structure. The planner was able to find assembly sequences for all three structures. However, the performance shown in Table 5.2 is not practical – a 21-component structure is not a very interesting one in the context of large-scale assembly, and waiting over eleven hours for a solution is not acceptable. The fact that the planner examined over two-thirds of all the graph edges and vertices (to find a 21-step sequence) seems unnecessary. The computational expense in searching the assembly graph comes from the evaluation of graph edges using a motion planner. Each edge represents an abstract assembly step, each of which requires several motions to be planned and evaluated (e.g., retrieving and installing components, and approaching components from different directions, etc.). On average, between three and four motion plans are required for each edge being evaluated.

The efficiency improvement strategies presented below focus on two places obvious from Table 5.2: computing the same number of motions plans as the baseline, but doing so faster for each plan (Section 5.3), and reducing the number of motion plans (i.e., graph edges evaluated) during planning (Section 5.4). Combining both strategies provides maximum benefit (Section 5.5).

<table>
<thead>
<tr>
<th>components</th>
<th>8</th>
<th>13</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertices</td>
<td>246</td>
<td>2,216</td>
<td>86,474</td>
</tr>
<tr>
<td>unique sub-assemblies</td>
<td>58</td>
<td>424</td>
<td>11,924</td>
</tr>
<tr>
<td>edges</td>
<td>924</td>
<td>10,798</td>
<td>599,520</td>
</tr>
<tr>
<td>potential sequences</td>
<td>1.29E+05</td>
<td>3.58E+09</td>
<td>7.17E+17</td>
</tr>
<tr>
<td>memory</td>
<td>0.8 MB</td>
<td>4.3 MB</td>
<td>191.4 MB</td>
</tr>
</tbody>
</table>

Table 5.1: Properties of the assembly graphs for the three structures (see also Figure 5.1) used for comparison throughout this chapter
5.3 Increasing Edge Evaluation Efficiency

Without changing the underlying search strategy (i.e., evaluating the same assembly graph edges as the baseline planner), the system can realize efficiency gains by being more efficient in the evaluation of each edge being processed. This section considers three approaches: 1) motion planning with approximations, where the system deliberately accepts execution-time failures due to inaccurate plans and relies on the exception handling capabilities described in Chapter 4 to compensate for any problems that arise; 2) deferred smoothing of the motion plan returned by the probabilistic path planner until an entire assembly sequence has been found; and 3) clustering similar graph edges and assigning the cost of one evaluated edge to all similar ones without further evaluation.

The second strategy finds plans that are longer than the best possible plan (since the true cost of a graph edge once the path is smoothed is shorter than the unsmoothed cost used for planning), but all plans are still valid and can be found up to 40% faster. The third strategy maximally reduces the time required to “evaluate” some edges to a simple look-up of a cost assigned to all similar edges. This method incurs overhead to determine similarity and has the potential to wrongly include invalid edges that were assigned similar costs (marking them valid) without ever being explicitly checked. Note that the invalidity of a plan is always detected before any robot starts executing.

5.3.1 Motion Planning with Approximations

The strategy described in this section specifically exploits the fact that the assembly planning system developed in this thesis includes comprehensive error recovery strategies throughout the entire framework (as described in Chapter 4). Given all the attention placed on creating a system that can robustly execute complex tasks on multiple mobile robots, how can these capabilities be exploited to speed up planning? The results described here were also reported in [Heger and Singh, 2010].

Table 5.2: Summary of the performance of the baseline assembly planner when generating assembly sequences for the three example structures (Figure 5.1).

<table>
<thead>
<tr>
<th></th>
<th>Structure 1</th>
<th>Structure 2</th>
<th>Structure 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>679 (73%)</td>
<td>7,148 (66%)</td>
<td>407,085 (68%)</td>
</tr>
<tr>
<td>vertices visited</td>
<td>177 (72%)</td>
<td>1,480 (67%)</td>
<td>59,027 (68%)</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>3,307</td>
<td>34,739</td>
<td>1,988,910</td>
</tr>
<tr>
<td>solution time</td>
<td>50.5 sec</td>
<td>691.8 sec</td>
<td>41,793.2 sec</td>
</tr>
<tr>
<td></td>
<td>(11.5 min)</td>
<td>(11.6 hrs)</td>
<td></td>
</tr>
</tbody>
</table>
5.3.1.1 Approach

The correct way of planning motions for several robots operating in a constrained environment to build a structure would plan in the joint configuration space of all robots involved. Given the ever-changing geometry of the environment and the robots carrying components, this approach soon reaches the limits of tractability as the number of robots increases. In addition, unless time dependence and delays caused by runtime failures are specifically taken into account, any break in synchronization between the robots can cause problems requiring potentially expensive re-planning.

The key to the approach described here is to recognize that even though there can be multiple robots involved in the assembly, most of the time they are working on separate tasks in the same environment, and many of their transfer motions lead through mostly open space until they get close to the structure. The first simplifying approximation is to plan motions for individual robots, only considering the structure obstacle while ignoring other robots in the environment. Clearly, left at that, this is a recipe for disaster where robots will pile up in the center of the workspace as their paths intersect. The second simplification comes from using simple motion models for the robots during planning (e.g., plan for a holonomic base, even though the actual robot is skid-steered) instead of more accurate models that are more expensive to plan with. As before, robot motions between the storage location and the structure that mostly lead through free space are minimally affected by this discrepancy. As the robots get closer to the structure, or as the environment becomes increasingly complex, problems will start to arise that need to be addressed.

The solution to the problems created by the two simplifications mentioned above is to rely on the same failure recovery techniques already in place to ensure execution robustness (see Chapter 4). The robots are able to detect exceptions (e.g., blocked paths, etc.), stop safely and request help (either from an autonomous repair/re-plan system or from a human operator). Depending on the situation, the recovery response can be a simple “continue now” (if the cause of the exception has passed, e.g., another robot temporarily in the way), a new motion plan from the current location (again with the same simplifications), or a larger-scale re-plan.

5.3.1.2 Results

This section describes an experiment to determine the effects of trading off sophisticated planning techniques (in the interest of planning time) for a comprehensive exception handling system (in the interest of robustness) that patches any problems caused by an overly optimistic motion planner. While the number of exceptions increased (not unexpectedly, if the robot’s motion is more complicated than the planner’s assumptions), execution robustness was still achieved with only a small number of directed instances of operator assistance that get execution back on track.

Using the simulated assembly environment, the system planned and executed the assembly of a 13-component structure (the two-square structure used in Chapter 4) at two different goal locations (see Figure 5.2). During execution, two different robots were considered: a holonomic base with the same motion model as assumed during planning, and a skid-steer vehicle that followed the planned path by turning toward and then driving to the next waypoint in the plan generated assuming a robot capable of holonomic motion. The skid-steered robot started and ended in the same position and orientation as the holonomic base, but the orientation of intermediate waypoints was ignored.
Increasing Edge Evaluation Efficiency

The effects of planning robot motions with simplified motion models (holonomic base vs. skid-steered vehicle) were evaluated by planning and executing the assembly of a 13-component structure in constrained locations in the environment.

(once an intermediate waypoint position was reached, the robot turned toward the next waypoint and headed off toward it). The second condition (skid-steered robot executing a plan generated for a holonomic case) was the same as used in the experiments presented in Chapter 4. The experiment described here justifies the use of the simplified motion planning techniques to gain planning efficiency without sacrificing the system’s robustness performance.

For each experimental condition, the planner produced an assembly sequence and then commanded a simulated mobile manipulator to execute it. During execution, a number of exceptions were triggered. If the robot detected an imminent collision of its body or an element it carried with another object, it stopped and threw a “Clearance” exception. In addition, “Sensing” and “Manipulation” exceptions were generated randomly. For this experiment, 20% of all alignments suffered “Sensing” exceptions (i.e., in the real world, the robot would be unable to sense everything it needs to align itself with its target), and 40% of all manipulation attempts would fail (i.e., the robot would try to pick up or install a component, but something goes wrong).

As in the experiments in Chapter 4, “Clearance” exceptions triggered a contingency response, repair or re-plan of the robot’s motion from its position where the failure occurred to the current
goal position. “Sensing” and “Manipulation” exceptions were recovered from using a “Try Again” strategy. After five exceptions in a single assembly step, a plan repair was triggered where the current assembly step was re-parameterized from the current state to the step’s goal state. If, after five repair attempts there was still no solution, the exception escalated to the re-planning level where the assembly graph is searched for an alternative sequence to the assembly goal state. For each scenario and experimental condition, the number of each type of exception that occurred, as well as the number of required operator interventions and the outcome of the run were recorded (Tables 5.3 and 5.4). Each experimental condition was run five times and the results averaged.

### Table 5.3: Scenario I: structure centered in workspace. The table shows the average number of each type of exception and recovery events per attempted assembly. All attempted assemblies were successful.

<table>
<thead>
<tr>
<th>Scenario I</th>
<th>planned for and executed as holonomic base</th>
<th>planned for holonomic robot, executed by skid-steered base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clearance Exception</td>
<td>0.0</td>
<td>6.4</td>
</tr>
<tr>
<td>Sensing Exception</td>
<td>20.8</td>
<td>14.6</td>
</tr>
<tr>
<td>Manipulation Exception</td>
<td>16.6</td>
<td>17.4</td>
</tr>
<tr>
<td>Operator Assistance</td>
<td>0.0</td>
<td>1.6</td>
</tr>
<tr>
<td>Repair Events</td>
<td>3.0</td>
<td>2.4</td>
</tr>
<tr>
<td>Re-plan Events</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Table 5.4: Scenario II: structure close to obstacle at edge of workspace. The table shows the average number of each type of exception per attempted assembly. All attempted assemblies were successful.

<table>
<thead>
<tr>
<th>Scenario II</th>
<th>planned for and executed as holonomic base</th>
<th>planned for holonomic robot, executed by skid-steered base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clearance Exception</td>
<td>0.0</td>
<td>15.6</td>
</tr>
<tr>
<td>Sensing Exception</td>
<td>15.8</td>
<td>22.8</td>
</tr>
<tr>
<td>Manipulation Exception</td>
<td>11.4</td>
<td>17.8</td>
</tr>
<tr>
<td>Operator Assistance</td>
<td>0.0</td>
<td>4.2</td>
</tr>
<tr>
<td>Repair Events</td>
<td>2.2</td>
<td>5.4</td>
</tr>
<tr>
<td>Re-plan Events</td>
<td>0.8</td>
<td>1.0</td>
</tr>
</tbody>
</table>

All assembly attempts summarized in Tables 5.3 and 5.4 were completed successfully. The recovery strategies described in Chapter 4 compensated for any artificially introduced exceptions.
5.3. Increasing Edge Evaluation Efficiency

due to the use of the simple motion model during planning. As expected, the system encountered no clearance exceptions when the robot executing the plan was obeying the same motion model (of a holonomic base) as used during planning. When the robot followed a skid-steered path along the planned waypoints, the number of clearance exceptions corresponded to the complexity of the problem with significantly more exceptions occurring in Scenario II than Scenario I.

Assemblies planned and executed with a holonomic base had to recover only from randomly triggered sensing and manipulation exceptions. Autonomous repair and re-planning (only Scenario II required re-planning) were sufficient to successfully complete the desired structure. As clearance exceptions were encountered by the skid-steered robot, some exceptions required operator assistance to get execution back on track. As in the experiments described in Chapter 4, the operator was asked to help when possible and to move the robot (via a mouse-based left-right-forward-back “joystick”) far enough from the obstacle causing the exception for the autonomous system to re-take control, and for execution to continue.

The average number of contingencies encountered during assembly attempts (beyond fluctuations due to the stochastic triggering of sensing and manipulation exceptions) is primarily an indicator of how many steps were attempted (or re-attempted in the course of repair and re-planning events). As a result, Scenario II shows more contingencies when executed by a skid-steered vehicle. The large difference in the number of sensing exceptions in Scenario I was likely due to stochastic effects and a small number of experimental runs.

5.3.2 Deferred Path Smoothing

Sampling-based probabilistic motion planners are nice in that they can plan in high-dimensional spaces (and thus are suitable when multiple robots are involved in assembly steps – something that will become more relevant for large structures discussed in Chapter 6). However, by their very nature, the solution they find is (at least at first) a zig-zag path connecting samples from the start state to the goal. Instead of smoothing the solution path each time an assembly graph edge is evaluated, this efficiency improvement strategy defers smoothing until an assembly sequence has been found and tasks are being parameterized in preparation of execution by the robots. In the case of a single robot performing assembly tasks, grid-based motion planners could be used that do not require any smoothing at all. But as later extensions toward larger and more realistic assembly scenarios require multiple robots to manipulate large components, there is a benefit to using a sample-based planner and to defer path smoothing as described here.

5.3.2.1 Approach

In the process of searching the assembly graph for feasible steps and sequences, a large number of motion plans have to be computed (on average three to four per evaluated assembly graph edge). The baseline planner evaluates between 3,000 and 2,000,000 motion plans in the process of planning assemblies for the three sample structures. Once each motion plan is first found it is smoothed to obtain the shortest path. This operation iterates over the initial zig-zag solution path and attempts to short-circuit the path between non-consecutive waypoints. Each potential shortcut requires computationally intensive collision checking to determine its validity. The vast majority of these evaluated
5.3. Increasing Edge Evaluation Efficiency

(and smoothed) plans will not be part of the final assembly sequence. Thus, the goal of this strategy is to avoid incurring the cost of smoothing without significantly sacrificing solution quality.

Any time the motion planner finds a (zig-zag) solution, that there exists a feasible motion for the robot(s), and the planner has an upper bound on the cost (measured in distance traveled) of that motion. In most cases, after smoothing the path, the true cost will be lower than the initial cost, but the smoothing operation takes a non-trivial amount of time compared to the actual path planning effort (up to an additional 40% depending on the environment). Deferring path smoothing is most beneficial in scenarios where the robot(s) have to circumnavigate a number of obstacles on their way to the goal. The initially returned solution path there contains many waypoints, and few if any shortcuts between randomly selected pairs of waypoints will result in a shortening of the path. Nevertheless, the path smoother runs for a set number of iterations. The simpler the environment, the less of a benefit can be realized by deferring path smoothing. In the extreme case, if a straight-line path is returned by the planner, no smoothing is necessary and thus no savings can be realized.

By deferring path smoothing to the end of the assembly planning process, the system is able to reclaim the time spent smoothing each intermediate motion planning solution. In exchange for this efficiency gain, however, it risks to overestimate the cost of assembly steps (i.e., edge weights in the assembly graph search). As a result, there is no longer a guarantee to find the shortest (or otherwise best) assembly sequence. As shown below, the penalty incurred in solution cost is negligible compared to the search time saved compared to the baseline planner.

5.3.2.2 Results

Deferring path smoothing until an assembly sequence has been found does not change how many vertices are expanded or how many edges are evaluated during the search over the assembly graph. Any slight variations there are simply due to the random nature of the motion planner. Tables

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Without Smoothing</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>679 (73%)</td>
<td>677 (73%)</td>
<td>− 0.3%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>177 (72%)</td>
<td>175 (71%)</td>
<td>− 1.1%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>3,307</td>
<td>3,299</td>
<td>− 0.2%</td>
</tr>
<tr>
<td>solution time</td>
<td>50.5 sec</td>
<td>39.3 sec</td>
<td>− 22.2%</td>
</tr>
<tr>
<td>solution cost</td>
<td>85.9</td>
<td>86.2</td>
<td>+ 0.3%</td>
</tr>
</tbody>
</table>

Table 5.5: Deferring path smoothing until an assembly sequence has been found has no significant effect on how many plans have to be evaluated. However, the planning time required to find a solution is reduced significantly with only a negligible increase in solution cost. For simple structures such as this, the benefits from deferred smoothing are less than for more complex ones as many “motion plans” here may not require any smoothing in the first place.
5.3. Increasing Edge Evaluation Efficiency

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Without Smoothing</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>7,148 (66%)</td>
<td>7,121 (68%)</td>
<td>−0.4%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>1,480 (67%)</td>
<td>1,466 (66%)</td>
<td>−0.9%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>34,739</td>
<td>34,624</td>
<td>−0.3%</td>
</tr>
<tr>
<td>solution time</td>
<td>691.8 sec</td>
<td>427.5 sec</td>
<td>−38.2%</td>
</tr>
<tr>
<td>solution cost</td>
<td>134.1</td>
<td>134.1</td>
<td>±0.0%</td>
</tr>
</tbody>
</table>

Table 5.6: Deferring path smoothing until an assembly sequence has been found has no significant effect on how many plans have to be evaluated. However, the planning time required to find a solution is reduced significantly with only a negligible increase in solution cost. More complex structures get a bigger benefit from this method than simpler structures because the number of motion plans that would be smoothed at each iteration is larger.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Without Smoothing</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>407,085 (68%)</td>
<td>407,020 (68%)</td>
<td>±0.0%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>59,027 (68%)</td>
<td>58,995 (68%)</td>
<td>−0.1%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>1,988,910</td>
<td>1,988,633</td>
<td>±0.0%</td>
</tr>
<tr>
<td>solution time</td>
<td>41,793.2 sec</td>
<td>26,230.6 sec</td>
<td>−37.2%</td>
</tr>
<tr>
<td>solution cost</td>
<td>226.2</td>
<td>226.3</td>
<td>+ 0.1%</td>
</tr>
</tbody>
</table>

Table 5.7: Deferring path smoothing until an assembly sequence has been found has no significant effect on how many plans have to be evaluated. However, the planning time required to find a solution is reduced significantly with only a negligible increase in solution cost. More complex structures get a bigger benefit from this method than simpler structures because the number of motion plans that would be smoothed at each iteration is larger.

5.5, 5.6 and 5.7 summarize the benefits of adding only deferred path smoothing as an efficiency improvement method to the baseline planner.

The key improvement of the baseline solution due to deferred smoothing is an up to 38.2% reduction in the time it takes the planner to find a solution. This benefit comes at the cost of a negligible increase in the planned solution cost. Note, however, that by using this method the planner can no longer provide a guarantee that it will return the shortest (measured in distance
traveled by robots) assembly plan in the assembly graph. In most practical situations, a slightly-longer-than-absolutely-necessary assembly task can easily be tolerated, if in return the planning (and re-planning) time can be reduced as shown here.

A second convenient benefit of this method is that its impact grows as the structure to be assembled and the environment in which the assembly takes place get more complex. In simple scenarios, many motion plans consist simply of straight line segments connecting the start and goal configurations. For such paths, no smoothing is necessary, and consequently deferring smoothing until the end has no effect. However, as the scenarios become more complex, paths become more involved and not smoothing after every planning query has the observed effect.

5.3.3 Clustering Similar Assembly Graph Edges

This technique is based on the observation that even though the structure and environment are changing for each motion planning query, many motion planning queries are very similar in that they start in approximately the same area of the workspace, end in approximately the same location, and take a similar path to get from start to goal. Instead of planning each of these motions separately, the planner recognizes this similarity and infers costs of unplanned edges based on ones already evaluated. For assembly steps that are determined to be “similar” to ones already evaluated, this method effectively reduces the amount of time spent evaluating the graph edge to near zero (only a look-up of an already-computed cost is required). Some overhead is required to determine similarity of edges. Note also that this technique may falsely declare assembly steps feasible. This situation is detected before execution starts, but additional search effort is required in that case.

5.3.3.1 Approach

Most assemblies are comprised of relatively few different parts that all have to be brought to approximately similar locations in the workspace for assembly. Qualitatively different paths may be required for assembly steps for different parts of the structure, but adjacent components often share similar plans. Instead of planning all these minimally different queries, the system instead attempts to recognize which steps require similar enough motion to be grouped together, and then uses a single query result in the evaluation of all similar graph edges (see Figure 5.3).

In order to decide which edges are “similar” and thus can be clustered, the system needs a measure of similarity. The current implementation considers assembly graph edges to be similar if the robot’s position in the source states and target states respectively are close by as measured by Euclidian distance. This criterion works reasonably well for simple structures and not very cluttered environments. However, as is shown in Figure 5.3, topologically different robot positions (e.g., two positions in close proximity but on opposite sides of the structure or an obstacle) are falsely considered to be “similar” by this measure.

For every assembly graph edge evaluated, the planner considers all other edges leaving the source state and compares the robot’s position in their target states to that in the currently evaluated edge’s. If the robot travels from the same state (in the source state) to a similar state (in another edge’s target state) and picks up a component in a similar location, then the same edge cost is assigned to the “similar” edge as for the one that was just evaluated. In order to avoid invalidating
5.3. Increasing Edge Evaluation Efficiency

Figure 5.3: Realizing that many assembly graph evaluations are quite similar (despite the ever changing environment), the system determines which edges are close enough to the one just evaluated and uses the evaluation result from one edge to score similar ones as well.

too many edges as being blocked, there are two different thresholds for proximity, depending on whether the evaluated edge is valid or not. These are tunable parameters of the planner that can be adjusted for different applications.

Since the planning framework includes powerful exception handling mechanisms (see Chapter 4), the similarity detector is biased toward classifying assembly graph edges as similar. If a motion plan returned a feasible solution, all other graph edges that require motions starting and ending within a generous distance of the actual start and end points of the edge evaluated are considered similar. Some of these classifications will be incorrect, but the system will detect and be able to fix that before execution begins. Alternatively, if a planned edge does not have a feasible solution path, other edges that are quite close in start and end to the failed edge will also be classified as impassable. The similarity threshold for failed graph edges is much tighter than for successful ones to avoid cutting too many actually feasible graph edges, resulting in the overall plan to fail without finding an assembly plan even though one exists. Instead, the planner errs on the side of allowing potentially infeasible edges to be treated as valid until it is determined that they are not. If one of those edges is part of the solution trace, its infeasibility will be detected during task parameterization. In that event, the edge cost is corrected and the search for an assembly sequence continues until either a valid solution is found or the problem is declared unsolvable. In rare cases the clustering technique declared the same not explicitly evaluated edge similar to a valid edge and also similar to another invalid edge. In that situation, no edge cost was assigned and the edge in question was flagged for explicit evaluation if it was considered during the graph search.
5.3. Increasing Edge Evaluation Efficiency

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>With Clustering</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>679 (73%)</td>
<td>688 (74%)</td>
<td>+ 1.3%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>177 (72%)</td>
<td>179 (73%)</td>
<td>+ 1.1%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>3,307</td>
<td>2647</td>
<td>− 20.0%</td>
</tr>
<tr>
<td>solution time</td>
<td>50.5 sec</td>
<td>43.2 sec</td>
<td>− 14.5%</td>
</tr>
<tr>
<td>solution cost</td>
<td>85.9</td>
<td>86.2</td>
<td>+ 0.3%</td>
</tr>
</tbody>
</table>

Table 5.8: Recognizing similar edge evaluation queries during plan generation reduces the number of queries that need to be solved, thus speeding up the search for an assembly sequence. Note that this method may return infeasible assembly plans if edges are accepted as part of a cluster even though they do not represent realizable assembly steps.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>With Clustering</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>7,148 (66%)</td>
<td>7,202 (67%)</td>
<td>+ 0.8%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>1,480 (67%)</td>
<td>1,463 (66%)</td>
<td>− 1.1%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>34,739</td>
<td>27,752</td>
<td>− 20.1%</td>
</tr>
<tr>
<td>solution time</td>
<td>691.8 sec</td>
<td>521.1 sec</td>
<td>− 24.7%</td>
</tr>
<tr>
<td>solution cost</td>
<td>134.1</td>
<td>HUGE</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.9: Recognizing similar edge evaluation queries during plan generation reduces the number of queries that need to be solved, thus speeding up the search for an assembly sequence. Note that this method may return infeasible assembly plans if edges are accepted as part of a cluster even though they do not represent realizable assembly steps. The plan generated for the two-square structure using clustering is infeasible. This situation is detected before execution begins.

5.3.3.2 Results

Tables 5.8, 5.9 and 5.10 summarize the effects of applying the clustering technique described in this section during assembly planning. The tables show only the results of the first attempt at solving the assembly planning problem, i.e., until a complete sequence through the assembly graph was found. Table 5.9 shows an instance of the problem described above where the solution returned by the planner is actually invalid. This case was detected when the system attempted to instantiate the task template for execution and additional searching was done to find a valid solution.
5.3. Increasing Edge Evaluation Efficiency

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>With Clustering</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>407,085 (68%)</td>
<td>450,866 (75%)</td>
<td>+ 10.8%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>59,027 (68%)</td>
<td>64,675 (75%)</td>
<td>+ 9.6%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>1,988,910</td>
<td>1,774,040</td>
<td>− 10.8%</td>
</tr>
<tr>
<td>solution time</td>
<td>41,793.2 sec</td>
<td>38,222.8 sec</td>
<td>− 8.5%</td>
</tr>
<tr>
<td>solution cost</td>
<td>226.2</td>
<td>241.1</td>
<td>+ 6.6%</td>
</tr>
</tbody>
</table>

Table 5.10: Recognizing similar edge evaluation queries during plan generation reduces the number of queries that need to be solved, thus speeding up the search for an assembly sequence. Note that this method may return infeasible assembly plans if edges are accepted as part of a cluster even though they do not represent realizable assembly steps.

With clustering, the number of edges the planner had to evaluate increased for all scenarios. This increase was proportional to the increase in graph edges processed and due to the fact that edges considered “similar” increased the number of equally “good” vertices that needed to be expanded during best-first search. While the number of edges evaluated grew compared to the baseline planner, the number of motion plans that had to be computed dropped by up to 20% for the single square structure. The savings for the largest (four-square) structure were smaller because the number of edges evaluated increased by 10.8%. This larger increase was due to the higher branching factor in the associated assembly graph compared to the smaller structures.

Table 5.9 has to be considered separately from Tables 5.8 and 5.10 because the search terminated without finding a valid assembly plan after examining only a smaller fraction of the assembly graph compared to the other two structures. Thus it is not surprising that the “solution” time decreased in this scenario. Keep in mind, however, that once the planning failure was detected, the search was continued with the invalid edge’s cost corrected and a valid solution was found. The final valid solution was found in an average of 624.7 seconds (or 9.7% faster than the baseline).

While this efficiency improvement strategy was able to reduce the time required to find a plan, its benefit was disappointing compared to the much simpler method of deferred path smoothing described above. A likely cause for the observed performance of the clustering technique was a combination of the fairly simple environment and the very simple similarity criterion of Euclidian distance between robot positions in assembly states. Future work is required to develop more useful similarity metrics (probably involving topological measures of the environment). In the environment used here, the time savings due to clustering was reduced by the fact that the majority of motion queries were solvable very quickly from scratch.
5.4 Reducing the Number of Graph Edges to Evaluate

5.3.4 Effects on Completeness and Correctness

The deferred smoothing technique presented above has potentially detrimental effects on the optimality of returned plans, but it does not negatively impact correctness or completeness. Planning with approximations, on the other hand, is complete but might produce plans that are infeasible for robots with more complicated motion models than those used during planning. Finally, clustering has the potential to produce invalid plans (as shown in Table 5.9) and wrongly declare a problem unsolvable. If edges are wrongly classified as being similar and are subsequently chosen as part of the solution trace through the assembly graph, the planner will fail when attempting to parameterize the corresponding tasks in preparation for execution. At that point, the incorrect graph weights are corrected and the search restarted. This additional computation cuts into the effectiveness of this method both during the initial plan generation as well as during later re-planning events. In the event of false-negatives (valid edges classified as “similar” to invalid ones), the planner may miss a solution in the graph without being able to detect that this has happened.

5.4 Reducing the Number of Graph Edges to Evaluate

As Section 5.3 showed, by evaluating individual edges more efficiently the system can reduce assembly planning time by nearly 40%. However, these techniques do not address the fundamental problem that the baseline planner processes approximately two-thirds of the entire assembly graph (which grows exponentially as the structure size increases). This section addresses this problem by reducing the number of graph edges that need to be evaluated. This reduction is realized by pre-computing an admissible cost-to-go heuristic to guide the graph search toward promising regions of the assembly graph (Section 5.4.1).

5.4.1 Heuristic Search

While reducing the time spent evaluating individual edges during assembly planning reduces the solution time, reducing the number of edges processed has a much greater effect as it significantly reduces the branching factor of the search. This effect is by no means specific to assembly planning, but is common to graph search in general. This section examines the effects of an admissible heuristic on the developed system.

5.4.1.1 Approach

A qualified operator’s intuition often is a good heuristic to guide plan selection toward desirable solutions. However, while some planning queries might take longer than others, even at a slow 0.1 seconds per query they are significantly faster than the best expert operator. Heuristics thus are required to allow the autonomous system to perform all the tedious work and only request help where it is most beneficial to the desired solution and feasible for an operator to help in ways the planner cannot easily reason about.
5.4. Reducing the Number of Graph Edges to Evaluate

![Diagram showing straight-line heuristic and cost-to-go heuristic](image)

**Figure 5.4:** Pre-computing a cost-to-go heuristic for states in the assembly graph reduces the search time significantly during planning. Assuming straight-line motions, the planner is guaranteed to obtain an admissible heuristic. If the workspace between the start, pickup and goal locations is fairly clear, the heuristic cost is close to the true cost. If instead the robot has to circumnavigate the structure while traveling through the workspace, the heuristic is less well-informed.

Of several heuristics considered\(^1\), using an A* cost-to-go heuristic proved the most useful for increasing planning efficiency (see Figure 5.4). This approach requires some pre-computation where idealized costs (assuming straight-line start-to-goal motion for the robot) are pre-computed into a cost-to-go for each state in the assembly graph. This pre-computation step uses Dijkstra’s search from the goal state toward the initial state in the assembly graph. During the assembly sequence generation, the A* graph search uses the pre-computed heuristic to guide its search.

For motions through wide-open areas of the workspace, this heuristic provides the true cost of the associated plan or one very close to it. As the environment becomes more cluttered, the accuracy of the heuristic decreases. Only when computing the estimated cost-to-go for assembly states that have the robot on the opposite side of the structure from where components are retrieved does the heuristic underestimate the true cost significantly. Once such assembly steps are evaluated, their true cost quickly indicates that in most cases they are not desirable actions to take.

Pre-computation of the cost-to-go heuristic starts with the fully-assembled goal state and works toward the completely-disassembled state. In the process, each intermediate state’s cost-to-go is

---

\(^1\)Other considered heuristics included use of size and stability metrics described in Section 4.4.1, and a measure of “externalness” of components to be chose for assembly steps.
computed. The method does, however, require the entire assembly graph to be generated first. As lazy combined graph generation and search is considered in future work, an alternate heuristic also has to be developed.

### 5.4.1.2 Results

Applying the straight-line cost-to-go technique described here is by far the single most effective strategy to reduce assembly planning time. For different structures, experiments show planning time reductions by 80-98% compared to the baseline planner (see Tables 5.11, 5.13 and 5.14). Compared to the realized planning time savings, the time required to pre-compute the heuristic values was negligible (under one second for the single-square structure, three to four seconds for the two-square structure and approximately 150 seconds for the largest four-square structure).

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>With Heuristic</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>679 (73%)</td>
<td>141 (15%)</td>
<td>− 79.2%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>177 (72%)</td>
<td>46 (19%)</td>
<td>− 74.0%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>3,307</td>
<td>644</td>
<td>− 80.5%</td>
</tr>
<tr>
<td>solution time</td>
<td>50.5 sec</td>
<td>9.4 sec</td>
<td>− 81.4%</td>
</tr>
<tr>
<td>solution cost</td>
<td>85.9</td>
<td>85.9 ± 0.0%</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.11:** Using heuristically guided search to find assembly sequences yields the largest improvement of any method considered. The savings are realized because only a small fraction of the entire assembly graph has to be evaluated. For this structure, pre-computing the heuristic values took well under one second.

Not only did using this heuristic reduce the planning time (as did the other techniques discussed above), it also effectively limited the portion of the assembly graph that is expanded while searching for an assembly sequence. In addition, the benefit of the method toward planning efficiency grows with the complexity of the problem. While the single square structure can realize only an 81.4% reduction in planning time, the largest structure considered reduces planning time to as little as 2% of the time taken by the baseline.

Note that the Tables 5.12 and 5.13 show the average solution cost to increase slightly despite the use of an admissible heuristic and graph search using A*. This effect is due to a combination of a small sample size (the baseline and with-heuristic experiments were run ten times each) and the fact that the solution cost is reported as the sum of robot paths lengths, which are generated by a probabilistic sampling-based planner. The standard deviations across the ten experimental runs was negligible with respect to the percent change in solution time and very small yet large enough to explain the shown cost increase for the reported solution costs.
5.4 Reducing the Number of Graph Edges to Evaluate

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>With Heuristic</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>7,148 (66%)</td>
<td>1,606 (15%)</td>
<td>-77.5%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>1,480 (67%)</td>
<td>458 (21%)</td>
<td>-69.1%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>34,739</td>
<td>4,053</td>
<td>-88.3%</td>
</tr>
<tr>
<td>solution time</td>
<td>691.8 sec</td>
<td>69.3 sec</td>
<td>-90.0%</td>
</tr>
<tr>
<td>solution cost</td>
<td>134.1</td>
<td>134.3</td>
<td>+ 0.1%</td>
</tr>
</tbody>
</table>

Table 5.12: Using heuristically guided search to find assembly sequences yields the largest improvement of any method considered. The savings are realized because only a small fraction of the entire assembly graph has to be evaluated. For this structure, pre-computing the heuristic values took between three and four seconds on average.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>With Heuristic</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges evaluated</td>
<td>407,085 (68%)</td>
<td>6,919 (1%)</td>
<td>-98.3%</td>
</tr>
<tr>
<td>vertices visited</td>
<td>59,027 (68%)</td>
<td>1,607 (2%)</td>
<td>-97.3%</td>
</tr>
<tr>
<td>motion plans computed</td>
<td>1,988,910</td>
<td>28,372</td>
<td>-98.6%</td>
</tr>
<tr>
<td>solution time</td>
<td>41,793.2 sec</td>
<td>696.4 sec</td>
<td>-98.3%</td>
</tr>
<tr>
<td>solution cost</td>
<td>226.1</td>
<td>226.8</td>
<td>+ 0.3%</td>
</tr>
</tbody>
</table>

Table 5.13: Using heuristically guided search to find assembly sequences yields the largest improvement of any method considered. The savings are realized because only a small fraction of the entire assembly graph has to be evaluated. For this structure, pre-computing the heuristic values took approximately 150 seconds.

5.4.2 Effects on Completeness and Correctness

While other techniques may impact the quality of the assembly plan returned by the planner (including returning plans that are not actually physically feasible, as with the clustering technique), the only adverse effect of using heuristically guided search is the time it takes to pre-compute the heuristic at the beginning of the planning process. For small structures (i.e., small assembly graphs), this process is virtually instantaneous. Large structures require a few seconds up to a few minutes of pre-computation time, but when reducing planning time from over eleven hours to approximately eleven minutes, another few minutes is well worth it.
5.5 Combined Efficiency Improvement Strategies

As shown above, individual strategies to improve assembly planning efficiency reduced planning time anywhere from under 10% (clustering for four-square structures) to over 98% (heuristic search for four-square structures). Figures 5.5, 5.6 and 5.7 summarize the benefits realized by combining deferred smoothing, clustering and heuristic search to achieve maximum planning efficiency.

The only individual efficiency improvement technique that can be problematic (in terms of allowing the planner to find solutions that are not actually valid and feasible ones) is clustering of similar graph edges, as was described above. All instances where combinations of techniques failed to find a solution included this method (C+H and C+DS+H for the single-square structure in Figure 5.5, as well as C and C+DS for the two-square structure in Figure 5.5). All combinations of efficiency improvement methods found valid solutions for the largest structure. This was likely due to the fact that that structure is represented by a significantly larger assembly graph that contains many more alternatives than the two smaller structures.

With the exception of the failed plans involving clustering just described, adding another method always was beneficial in terms of total planning time required. As the individual results above suggest, including the heuristic search technique yields the most benefit, no matter to which other method it is added. A surprising effect observed in Figures 5.5, 5.6 and 5.7 was the benefit of adding the clustering technique to the heuristic search technique. While clustering had little effect individually or when added to the deferred-smoothing technique, the combination C+H for the two- and particularly four-square structures showed a significant (and growing with the size of the structure) efficiency gain. This effect can be explained as follows. With the size of the structure, the out-degree of assembly graph vertices increases. When a graph vertex is expanded during best-first search, all its outgoing edges are evaluated and their target vertices added to the appropriate queues. Adding clustering during heuristic search, the expansion of graph vertices was sped up (since only some edges required actual motion planning). The same benefit was not realized for clustering combined with techniques that still considered large portions of the assembly graph (i.e., deferred smoothing), as clustering caused an increase in edges that required evaluation there. The heuristic search avoided this effect by having a more informed heuristic than other methods (which used a zero heuristic for the cost-to-go).

The cost comparison plots (red bars in the bottom half of Figures 5.5, 5.6 and 5.7 confirm that none of the efficiency improvements (up to 2.5 orders of magnitude with C+DS+H for the four-square structure) incur a noticeable penalty in solution quality.

5.6 Summary

At its core, assembly planning is posed as a graph search problem. As is typical in this domain, evaluating the same number of edges in less time and evaluating fewer edges both increase planning efficiency, with the latter being the much more effective way of reducing planning time. This chapter presents techniques with relevance to assembly planning of both categories. Planning with approximations, deferred path smoothing and clustering of similar planning queries all fall into the faster-per-edge category. Pre-computing a cost-to-go heuristic to guide the search reduces the
5.6. Summary

Figure 5.5: The top plot shows absolute planning time for a single-square structure (8 components). The bottom plot shows relative (100% marks the baseline performance) planning time (in blue) and solution cost (in red). Without a significant decrease in solution quality, the system accomplished a speed-up of almost one order of magnitude compared to the basic approach by combining the straight-line cost-to-go heuristic and deferred-smoothing techniques. The clustering technique caused the C+H and C+DS+H combinations to return invalid plans.

**DS** – Deferring path smoothing until an assembly sequence has been found (Section 5.3.2)

**C** – Clustering similar planning queries (and planning only one of them, Section 5.3.3)

**H** – Pre-computing straight-line heuristic to guide graph search (Section 5.4.1)
DS – Deferring path smoothing until an assembly sequence has been found (Section 5.3.2)
C – Clustering similar planning queries (and planning only one of them, Section 5.3.3)
H – Pre-computing straight-line heuristic to guide graph search (Section 5.4.1)

Figure 5.6: The top plot shows absolute planning time for a two-square structure (13 components). The bottom plot shows relative (100% marks the baseline performance) planning time (in blue) and solution cost (in red). Without a significant decrease in solution quality, the system accomplished a speed-up of approximately 1.5 order of magnitude compared to the basic approach by combining clustering, straight-line cost-to-go heuristic and deferred-smoothing techniques. The clustering technique caused the C and C+DS combinations to return invalid plans.
5.6. Summary

Figure 5.7: The top plot shows absolute planning time for a single-square structure (8 components). The bottom plot shows relative (100% marks the baseline performance) planning time (in blue) and solution cost (in red). Without a significant decrease in solution quality, the system accomplished a speed-up of nearly 2.5 orders of magnitude compared to the basic approach by combining clustering, straight-line cost-to-go heuristic and deferred-smoothing techniques. The clustering technique did not cause any problems for this structure.
number of edges to consider in search for a plan. Not surprisingly, the latter achieved the greatest individual gain and was the key contributor to combined approaches.

Combining efficiency strategies enabled the system to speed up assembly plan generation by 2.5 orders of magnitude and more. Assembly plans that used to take over 11 hours to generate were found in as little as 2.5 minutes. Beyond the initial plan generation, the techniques presented in this chapter are key to making efficient assembly operation a reality, as they also enable the system to re-plan efficiently in the event of execution-time exceptions.

The techniques described in this chapter made the basic assembly planning approach useful and usable in realistic applications. Together, robust and efficient assembly planning form the foundation from which it makes sense to think beyond individual structures toward larger assemblies with multiple robots. This is the topic of the next chapter.
Chapter 6

Larger Structures and Multiple Robots

6.1 Overview

Chapters 4 and 5 extend the basic approach to assembly planning to be practical in realistic applications where robustness to execution-time failures and planning efficiency are important. With available computational resources, the approaches presented thus far enable the planner to solve assembly problems of approximately 20 components in a planning time of a few minutes. Pushing computational resource boundaries, the same approach could be used to solve slightly larger problems, maybe with a few tens of components, but the increasing complexity fundamentally restricts the applicability to truly large structures (of hundreds or thousands of components).

In realistic scenarios, 20-component structures are of limited interest. To be truly useful, a planner has to be able to plan assemblies of 1000+-component structures. Straightforwardly extending the basic approach will be insufficient to get there. Instead, this work takes advantage of the hierarchical structure of many assembly problems where sub-assemblies of increasing size are combined into larger and larger sub-structures until the desired large structure is complete (see Figure 6.1). Specifically, this thesis considers the assembly of large planetary outposts that consist of several buildings, each of which consists of several wall segments, which, in turn are assembled from individual nodes and beams (the individual components).

This chapter presents an approach that extends the capabilities of the planner to large structures of hundreds or thousands of components without sacrificing planning efficiency and execution robustness as developed earlier (Section 6.3). It also considers another aspect of scalability where the system exploits the availability of additional assembly robots for tasks that could be completed by fewer robots, but that can be performed more efficiently and effectively with additional resources (Section 6.4).

6.2 Motivation

To motivate the approaches to scalability presented in this chapter, consider the following thought experiment (see Figure 6.2). Extending the assembly graph representation (see Section 3.3) to
6.2. Motivation

Figure 6.1: In order to maintain computational tractability, this thesis considers outposts consisting of buildings consisting of wall segments assembled from individual node and beam components. Each larger structure contains up to approximately 20 of the smaller structures – a problem that the basic approach is able to handle efficiently. The numbers along the left side of the figure indicate counts of each type of “component” assuming a minimum branching factor between 2 and 20.

This thesis takes a two-fold approach to handling both larger structures and additional assembly agents. Large structures are hierarchically decomposed into smaller ones to scale to larger and larger structures (Section 6.3). When there are additional robots available, the output of the planner (for the minimum number of robots) that by construction is guaranteed to be nominally feasible is post-processed using a heuristic scheduler to attempt to parallelize tasks for execution (Section 6.4).
6.3. Scaling to Larger Structures

Larger Structures
with basic representation

- requires exponentially more graph vertices and edges (see Figure 3.4) – not feasible for hundreds or thousands of components

Additional Robots
with basic representation

- requires more out-edges per vertex (one for each possible robot-component combination) – not feasible for tens or hundreds of robots

Figure 6.2: Thought experiment: Why the basic representation introduced in Chapter 3 does not scale to structures of interesting complexity or being assembled with multiple robots.

The techniques presented ensure that no structural constraints are violated, but they risk environmental constraint violations. However, as the system needs to be robust to those kinds of problems regardless of how many robots are available (i.e., even in the minimal robot case), it has powerful recovery strategies already in place that can easily handle the additional problems introduced by overly confident parallelization.

6.3 Scaling to Larger Structures

When scaling to larger structures, the basic approach breaks down after a few tens of components. However, given the structure of typical assembly scenarios, they naturally break down into sub-assemblies. By exploiting this decomposition into (if necessary, recursively smaller and) smaller sub-assemblies where each transition remains within the feasible range of the representation (i.e., a branching factor or up to 20), the system can plan for structures of hundreds or thousands of individual components.

6.3.1 Approach

The system plans for large (hundreds and thousands of components) structures by assembling them of hierarchically larger and larger sub-structures. Starting with individual nodes and beam as have been assembled with real robots in the test bed in the past (see Figure 6.3) this section defines different types of “wall segments” or subsets of the 21-component grid of nine nodes and 12 beams. Several wall segments are then combined into “buildings” according to a designer’s plan. While an individual robot is capable of assembling nodes and beams into wall segments, physical realism requires that two robots cooperate to install a wall segment as part of a building. In addition, once assembled, buildings are no longer movable. Thus, any building construction has to occur in-situ in the location as specified by the desired outpost layout.

A typical outpost scenario (see Figure 6.4) contains a number of large obstacles (e.g., rocks, craters, etc.) among which the desired outpost is to be constructed. There are also storage piles of individual components (nodes and beams) and pre-fabrication sites for different types of wall segments (where nodes and beams are assembled into segments). A robotic planetary outpost con-
6.3. Scaling to Larger Structures

Figure 6.3: Nodes and beams are assembled into wall segments (top row). Those are combined into buildings (middle row), which are part of an outposts (bottom row). The complete outpost consists of 1100+ nodes and beams (see also Figure 6.4).
6.3. Scaling to Larger Structures

Figure 6.4: The outpost is a collection of buildings, each of which is assembled from several segments, which are made of individual nodes and beams. Before installation into a building, segments are pre-fabricated by individual robots in designated pre-fabrication areas. After that, teams of two robots transport and install segments into buildings to construct the outpost. During construction, buildings have to be erected in-situ, as they cannot be relocated once they are built.

A construction site would have individual robots moving between storage sites and pre-fab areas assembling wall segments that would then be picked up by teams of two robots and transported into the workspace for installation into buildings to create the desired outpost.

Similar to the assembly plans for individual segments, building plans consist of a sequence of segments to be brought into the workspace and installed into a growing building. Outpost plans, on the other hand, have to be treated differently. Since buildings cannot be moved once erected and thus have to be built in-situ, the outpost planner is in charge of coordinating lower-level (building) planners by requesting a building plan in a specified environment given by the outpost state corresponding to an (outpost) assembly graph vertex. If a particular building can be assembled in that environment, the building plan cost becomes the edge weight in the outpost assembly graph, otherwise the corresponding edge in the outpost graph is impassable. In the process of searching the outpost assembly graph, the outpost planner requests building plan solutions from subsidiary building planners for each edge being evaluated.
As individual installation tasks (at both the pre-fabrication or outpost construction sites) encounter exceptions during execution, the robustness measures described in Chapter 4 become active and attempt to recover from any problems as close to the source of the issue as possible. Attempting low-level recovery before escalating the problem to higher levels of the hierarchy is even more important in outpost scenarios than in smaller problems since the complexity of the task the (re-) planner has to consider is so much greater (entire building plans – which include many individual component assemblies – instead of only a different order of components to install).

6.3.2 Implementation

The hierarchical decomposition of large structures into smaller sub-structures is a prime candidate for computational parallelization where each step represents a well-posed assembly planning problem that can be treated individually. For planning the assembly of an outpost, the system instantiates several planners (which are all the same binary, only with different configuration parameters), one at the top-level in charge of planning the outpost assembly sequence (of buildings) and coordinating auxiliary planners, one for each type of building (assembled by pairs of robots from wall segments) and one planner for each type of wall segment (assembled by individual robots from nodes and beams). This is the minimum number of planners necessary to plan an outpost, and the configuration used for the experiments described in this chapter. As additional computing resources are available, additional planners can be initialized to speed up the hierarchical planning. For example, as the outpost planner expands a graph vertex, the evaluation of each outgoing graph edge requires the generation of a building plan in a particular environment. All these plans are independent and could be evaluated in parallel with only small changes to the system as described here to handle the necessary book-keeping. In general, future work could apply many advances in graph search using parallel computation to the algorithms described throughout this thesis.

On start-up, each planner initializes any auxiliary planners it requires (e.g., the outpost planner initializes all building planners, which in turn initialize wall segment planners). Each planner generates or loads its own assembly graph (at the appropriate level of the sub-assembly hierarchy) that it searches for valid and desirable assembly sequences. Wall segments are planned in the same manner as the structures in previous chapters. Buildings are planned the same way, with the exception that two robots are required for each wall segment being installed (see Table 6.1). The recursive structure is broken for outpost plans, as buildings cannot be moved once assembled. At the outpost graph level, each graph edge represents a building plan in a particular environment defined by other buildings already assembled.

Outpost plan generation is a graph search of the assembly graph at the highest level of the hierarchy (i.e., the outpost level). As edges of the graph are evaluated, auxiliary planners are called to provide edge costs to guide the search. An edge evaluation at the outpost level instantiates a building planner with the environment described by the outpost state at the source of the graph edge. In the process of planning the requested building, the lower-level (building) planner instantiates segment planners to verify that all required wall segments can be assembled. As lower-level solutions are found, they translate into edge costs for the next higher planner, guiding the search there. If a solution is found in the top-level assembly graph, it is ensured that there is a (nominally) feasible assembly sequence for the entire outpost. As for smaller structures that do not require hierarchical
6.3. Scaling to Larger Structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Graph Vertices</th>
<th>Graph Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Segment</td>
<td>intermediate valid sub-assemblies</td>
<td>assembly steps adding one node or beam at a time</td>
</tr>
<tr>
<td>Building</td>
<td>intermediate valid sub-assemblies</td>
<td>assembly steps adding one segment at a time</td>
</tr>
<tr>
<td>Outpost</td>
<td>subset of all buildings in outpost</td>
<td>in-situ building assembly plans with already present buildings as obstacles</td>
</tr>
</tbody>
</table>

Table 6.1: Fundamentally, the assembly graph representation is the same at each level of the hierarchy. The specific graphs differ in what their vertices and edges represent. The outpost graph is more of a meta-graph coordinating the search for an assembly sequence at lower (building) levels.

decomposition, the plan contains all task parameters necessary to instantiate a task template at the executive level and command robots to perform the planned tasks.

In the current implementation of the assembly planner, segments and buildings follow the same assembly state validity rules (i.e., a state is valid if all components involved are connected). Requiring connectedness of assembly states is one of many possible criteria of validity. In the system here it is used as a simplifying assumption that limits the branching factor of the assembly graph (see Chapter 3 for more details). Outposts, in contrast, do not require connectivity – individual buildings may be connected, or they may be separate building in (close) proximity.

As structures become larger and more complex, the likelihood of execution-time failures only grows. Since fundamentally the hierarchical approach works just like the general assembly planning approach, the same robustness techniques described in Chapter 4 directly apply to ensure principled error recovery. As failures occur, the exception is either handled as close to where it occurred (and by the planner in charge of the failed task) if possible, or it escalates up through the assembly graph until it can be resolved at some intermediate level. If an exception reaches the highest-level planner and cannot be resolved there, there is no solution for assembling the desired outpost with the available task templates.

Note that the system considers buildings closed units in the planning hierarchy. As such, the planner attempts to find a solution for the entire building at the particular point in the plan being considered. Unless the building can be completed there, the corresponding edge is marked as impassable for the higher-level planner. Solutions where partial buildings are assembled at some point along the plan and then completed later are not generated. However, if an operator notices that the planner is unable to find a solution, he can modify the decomposition of buildings into smaller “sub-buildings” to be considered during planning. With the proper decomposition, the planner will then find a solution. At this time, decomposing large structures into smaller ones is the responsibility of the operator (see also the discussion in Chapter 3, Section 3.4.3). It is an open problem for future investigation how to automatically decompose structures into sub-structures to be most beneficial for planning.
The hierarchical decomposition approach lends itself to computational parallelization. Once the necessary planning graphs are generated (and they are static for a given structure), plan generation is essentially a matter of solving a large number of independent smaller assembly planning problems. With the sufficient number of processing cores available, any or all of them can be processed in parallel, such that, aside from the required overhead to coordinate the parallelization, the larger structure can be planned as efficiently as smaller ones in terms of time to find an assembly solution.

\[\text{Figure 6.5: The (still fairly simple) outpost of 5 buildings contains 63 wall segments and 1250 individual components. Using hierarchical structure decomposition as described in this chapter, along with efficiency improvements from Chapter 5, the planner found a feasible outpost plan in 8.6 hours (without any computational parallelization).}\]
Figure 6.6: The planner finds an assembly plan that ensures that each building is buildable with all already-present buildings treated as obstacles. Different wall segment colors indicate different types: red are wall segments, green are doors, blue are windows and yellow are gates.
6.3.3 Results

The assembly graph for the 5-building outpost used for experiments throughout this chapter contains 32 unique states connected by 80 edges. The outpost was decomposed into five buildings of three different types, each of which was assembled from four different types of wall segments (see Figure 6.5). In total, 1,250 individual nodes and beams were required to complete the entire outpost. The planner found a valid and feasible outpost plan (see Figure 6.6) after expanding 10 vertices and computing 25 building plans. Planning took approximately 31,000 seconds (8.6 hours), or on average 20 minutes for each building plan considered (building planning was done with deferred smoothing (DS) and heuristic (H) efficiency improvement methods). Compared to the (purely theoretic) performance of the basic planning approach for a structure with the number of components in the outpost (extrapolate in Figure 3.4), this is a dramatic improvement.

The outpost workspace is much larger than what was required to plan the assembly of individual wall segments (approximately 40x40 meters versus 10x10 meters), but the required clearances between wall segments was comparable to that between components in smaller assemblies. This clearance threshold determined the required sampling resolution of the motion planner used to evaluate building plans. Consequently, evaluation of building steps took longer than the same process for wall segment steps as the motion plans required were longer (the number of samples required to find a solution grew significantly).

The planner used for this experiment did not make use of the parallelizability of the planning process (with a sufficient number of available processors, all of the 25 building plans – or even all 80 building plans in the outpost assembly graph – could be computed in parallel). This modification would further reduce the outpost planning time to approximately the time required for one building plan (plus some overhead required to coordinate the parallelization).

6.3.4 Effects on Completeness and Correctness

For a given decomposition of an outpost (as specified by an operator) and using the deferred smoothing and heuristic search efficiency enhancement, the assembly planner finds a solution to the given problem if one exists (as before, a solution using the available task templates). If a solution is found, it is ensured that there is a (nominally feasible, barring unexpected failures) solution the robots can execute. If the robot executing the plan follows a more complicated motion model than that of a holonomic base used by the planner (as is the case here with skid-steered robots), correctness is no longer guaranteed. However, in practice the combination of autonomous and operator-assisted recovery mechanisms were able to resolve any problem that occurred.

With the operator in charge of specifying the decomposition of large structures into hierarchically smaller ones, the planner’s completeness is only as good as the specific decomposition. Specifically, structures at one level are considered units at the level above in the planning hierarchy. Unless the entire lower-level problem can be solved, the higher-level planner counts the corresponding edge as impassable. As a result, the assembly planner may report that an outpost cannot be assembled even though a different decomposition would allow the same robots using the same task templates to successfully complete the task.
6.4 Scaling to Additional Assembly Robots

For a system that is capable of planning assemblies of large structures, it is unreasonable to assume that in a realistic setting those structures will be assembled in the same linear fashion by individual robots or pairs of robots as smaller structures. However, as described above, straight-forwardly extending the basic assembly planning approach to consider the availability of large numbers of robots is not feasible.

6.4.1 Approach

Instead of explicitly planning for the parallelism possible with multiple robots, the system plans for the minimum number of robots only, as described in the previous section. Once a plan has been found, a guaranteed-to-be-nominally-feasible sequence of buildings exists that results in the outpost being constructed. In addition, for each building, there exists a sequence of segments that need to be (and are found feasible to be) connected to assemble the structure. The planner then post-processes the generated plan to take advantage of any parallelizability. This post-processing step makes use of a heuristic repair-based scheduler (ASPEN, developed at JPL [Fukunaga et al., 1997]) to enforce any resource constraints and take advantage of any parallelization opportunities.

ASPEN (Automated Scheduling/Planning ENvironment) is a modular planning/scheduling system that has been successfully applied to a number of advanced scheduling problems from spacecraft missions to telescope scheduling. In the context of this dissertation, the framework is used to determine resource allocations in scenarios where more robots are available than are minimally required to perform an assembly task. The planning approach cannot directly reason about resources (see Chapter 3). Instead, ASPEN is used to post-process the planner’s output to parallelize task execution. ASPEN iteratively repairs an initial (and in general flawed) plan until all specified ordering and resource constraints are satisfied. As part of the assembly planning system described here, (partial) order is already provided with the assembly planner’s output, thus primarily resource constraints are relevant here. The set of heuristics provided with ASPEN to determine which unsatisfied constraint(s) to attempt to repair (and how to repair them) during each repair iteration were sufficient for this application, allowing for easy integration.

The linear assembly plan generated for the outpost is generated assuming the minimum number of robots required to perform the work are available, and all assembly steps are executed in series. The planner outputs a domain specification for an ASPEN problem with all necessary resource and ordering constraints. The scheduler then starts with a maximally compressed schedule, inflates it until all constraints are satisfied, and finally optimizes it by compacting tasks on the time line until the shortest and most parallel schedule is found (in the number of repair/optimization iterations allowed). The remainder of this section describes the system’s implementation in detail.

The parallelization approach outlined above is likely to introduce execution-time exceptions that would otherwise not occur (e.g., robots’ paths crossing, or anything else the planner did not consider, as it did not reason about multiple robots). However, these artificially caused exceptions are of the same or similar types as those that are expected to occur even for perfect plans. Thus, the

\[1\] ASPEN web page: http://aspen.jpl.nasa.gov
Figure 6.7: Depending on the layout of an outpost, different amounts of parallelism can be applied to the linear plan initially generated. Separable outposts (left) can be maximally parallelized as the buildings do not impose constraints on each other. Non-separable outposts (right) can make use of parallelization of pre-fab actions only to ensure the task ordering necessary for ensuring nominal feasibility of the plan.

already-provided exception handling capabilities needed to ensure robustness for any structure are able to compensate for most if not all problems that arise during execution.

This section differentiates between two types of scenarios (see Figure 6.7): Outposts where individual buildings are sufficiently far apart to not cause conflicts for the construction of another building, and outposts where buildings are in close proximity, and thus the presence of buildings does constrain work on other buildings.

6.4.1.1 Outpost with Far-Apart Buildings

If buildings are far apart and can be treated individually, the planner can take a shortcut that avoids much of the planning work required otherwise. Whether or not this situation exists for an outpost can be determined by considering $N_B$ graph edges, where $N_B$ is the number of buildings in the outpost. Consider the $N_B$ in-edges leading to the fully-assembled outpost state in the assembly graph for an outpost. Each edge represents the assembly of the corresponding building as the last one in the plan (i.e., in the most constrained environment, where all other buildings are already erected and treated as obstacles). If the planner finds a valid plan for one of these edges, it implies that the corresponding building could be the last one assembled, which means that the presence of any of the other buildings does not affect the construction of this building. Not having other buildings in place might only make it easier to assemble the building in question. If all $N_B$ building plans corresponding to the in-edges of the fully assembled state have valid solutions, then any building could be assembled last without being affected by other structures, and thus the outpost in question is one with separate buildings.

For separable outposts, the system needs to ensure only that within each building the planned order of segments is maintained to guarantee nominal feasibility of the plan. Once those initial $N_B$
building plans are found, they are handed off to ASPEN for post-processing to determine how the available robots are best to be allocated to minimize the time it takes to execute the plan. Since all buildings are independent, there is no order that needs to be maintained between buildings or between segments of different buildings (see Figure 6.8).

Figure 6.8: If all outpost buildings are sufficiently far apart, there are no constraints on their ordering. Only within each building plan, the planned order of segments needs to be maintained. The space marked by horizontal arrows is available for scheduling pre-fabrication tasks.

For separable outposts, the scheduler has complete freedom over how to allocate available robots to buildings and segment pre-fabrication as possible. Since none of the buildings constrain others to the point they make assembly impossible, buildings can even be assembled partially and then be continued later on as resources become available again.

6.4.1.2 Outpost with Buildings in Close Proximity

Many outpost scenarios will not meet the criterion of having all buildings sufficiently separated so that the presence of one does not hinder the assembly of another. Instead, there will be a (partial, at least) ordering on how buildings need to be assembled in order to avoid getting stuck with a partially constructed outpost. If the test for separability of outpost buildings fails, the outpost planner falls back on a graph search over the outpost assembly graph, tasking the appropriate building planners with intermediate planning problems as was described in the first part of this chapter.
Note that in its current implementation, the planner considers any outpost that is not completely separable as being not separable at all, i.e., it does not produce plans that say “This building needs to be first, after that, the rest of the outpost can be assembled in any order.” Most realistic outposts fall into this intermediate category where some planning is necessary, but much of the time required for a full-blown graph search could be saved by recognizing partial separability. With the combinatorial number of possible sub-sets of buildings to consider, a principled approach to this problem is non-obvious and is left to future work.

If the planner finds a solution, there exists a nominally feasible assembly plan to be executed by the minimum number of robots required (i.e., two robots to install segments into buildings). In order to guarantee nominal feasibility, the planned order of buildings and segments within each building needs to be maintained (see Figure 6.9). With such a schedule, all available robots will first be assigned to (prefabrication of segments and then installation of segments into) the first building in the plan. Once that building is complete, they move on to the second one, and so on. If there are more robots available than can be used and allocated for a particular building, they have to remain idle once their assigned tasks are completed until the other robots complete their tasks, before the entire group of robots continues with the next building. While not particularly desirable, this mode of operation is guaranteed to be feasible.

![Figure 6.9: If buildings in an outpost impose constraints on one another, a fully serial (as planned) outpost schedule is guaranteed to be nominally feasible. This scheduling scheme has the most robot idle time as robots cannot start assembling the next building until the current one is done, which may require only fewer robots than are available, forcing the remainder to sit idle. The space marked by horizontal arrows is available for scheduling pre-fabrication tasks.](image-url)
of scheduling outpost tasks maintains the order of all tasks to be as planned. The only options for parallelization available to the scheduler in this case are ones internal to a building (e.g., where subsequent tasks’ Install sub-tasks have to be serialized to maintain installation order, but while one component is being installed, the next can be retrieved and brought into the workspace).

Having available robots sit idle and wait for other robots to complete their assembly tasks while there is still work to be done is clearly not a desirable behavior. In order to avoid excessive idle time (and to have a chance to obtain a shorter schedule makespan in the process), the system considers scheduling the entire outpost (once planned) as one single structure with many segments (see Figure 6.10). The segment order across all buildings is maintained, thus the schedule remains nominally feasible, but the strict division into buildings is removed. As a result, robots that cannot currently be tasked to install components (e.g., due to workspace constraints) can continue with the prefabrication of segments to be installed later on. While the entire outpost cannot be planned as one large structure (due to the prohibitive size of the assembly graph), once planned, the solution can be posed as a single (large) scheduling problem as described here.

![Figure 6.10](image)

**Figure 6.10:** As an alternative to full serialization (Figure 6.9), the entire outpost can be scheduled as if it were a single large structure where only the installation order of all components needs to be maintained. This approach allows the scheduler to avoid robot idle times. The space marked by horizontal arrows is available for scheduling pre-fabrication tasks.

### 6.4.2 Implementation

Considering possible scheduling alternatives for two robots (the minimum number required to erect buildings), there are two extreme schedules (Figure 6.11): they can either interleave segment prefabrication and installation, or they can first prefabricate all segments and then install them all. Any combination thereof is also valid. As more robots are available, similar patterns emerge. If there is
Figure 6.11: At a minimum, two robots are required to assemble an outpost: individual robots can pre-fabricate wall segments, but two robots are required to install them into buildings. Possible schedules for assembling the outpost with only two robots fall somewhere between alternating pre-fabrication and assembly (top) and completing pre-fabrication of all required components followed by their assembly (bottom).

An odd number of robots, there will always be at least one robot who cannot start an installation task until another robot becomes available. That robot could continue pre-fabricating segments while pairs of robots install complete segments, etc. As the number of robots available increases, the benefits of pre-fabricating-while-other-robots-assemble grows. In the ideal case (from an idle time perspective), the assembling robots would just barely never run out of pre-fabricated components to assembly into larger structures (i.e., pre-fabrication of a sub-structure completes just in time before the robots are ready to start installing it). Note, however, that with the described likelihood of execution-time failures, such tight temporal dependence is unrealistic to be useful in practice.

In order to make a scheduler generate meaningful schedules that exploit parallelizability of assembly tasks, the planner’s output is re-written as an ASPEN model. Along the lines of the task templates described in Section 3.3.2, an InstallSegment activity decomposes into Goto, Pickup, Transport and Connect sub-activities. Each sub-activity requires the use of 2 robots. In fact, the same two robots have to be used for all sub-activities of a segment. Goto and Transport activities have no further resource constraints. They correspond to the robot traveling through the environment (by itself and while carrying a component, respectively). If problems arise due to congestion in the workspace, execution-time error recovery strategies attempt to resolve them (see Chapter 4). Pickup and Connect sub-activities have additional resource constraints. Pickup requires that an appropriate segment be available (i.e., has been pre-fabricated prior to it being picked up), and only one Pickup activity for a type of segment can happen at the same time. Similarly, Connect activities require exclusive use of the installation workspace (of the corresponding building, if they are separable, or of the entire outpost, if they are not).

These constraints are modeled for ASPEN as resource reservations. Robots and locations in the workspace that can accommodate only single robots (or single pairs of robots) at a time are set up as non-depletable resources. An activity to be scheduled places a reservation against the
6.4. Scaling to Additional Assembly Robots

appropriate resource timeline that is released at the completion of the activity (i.e., after one Pickup activity is complete, the pickup location resource held by it is released and becomes available again). Components to be assembled are modeled as depletable resources that are replenished at the end of pre-fabrication activities and reduced at the beginning of Pickup activities. This mechanism ensures that the appropriate pre-fab activities are scheduled before the components are used for assembly operations by other robots.

In addition to altering the number of robots available, the number of pre-fabrication sites and the storage capacity of pre-fabricated segments can be changed (e.g., if this capacity is limited, pre-fabricating all segments prior to starting to install them may not be a feasible option). With these resource constraints, the model resembles closely what needs to happen either in simulation or with real assembly robots that are being tasked to assemble buildings and outposts.

While the work described in this section used ASPEN to perform task scheduling, the requirements for (re-)ordering tasks and assigning resources according to some model are fairly standard and could also be accomplished by other schedulers. ASPEN was chosen because its supplied heuristics were sufficient for the task at hand. The problem domain for ASPEN is defined according to the model described above. For fully parallelizable outposts, all building activities are placed on the timeline, and the scheduler spreads out the individual segment activities according to the resources available (using the MOVE repair method for conflict resolution). Randomly selected activities found to be causing a resource conflict are moved toward the future until the activities are spread out sufficiently for all resource conflicts to be resolved. Then the PACK and MOVE repair methods are applied during the optimization phase to reduce the makespan and robot idle time of the schedule while maintaining its validity. Completely serialized schedules consider each building separately, and outposts treated as single large structures contain a single activity that decomposes into sub-activities for all segments in all buildings in the outpost (ordered as planned in order to guarantee nominal feasibility). The output of ASPEN was then used to appropriately instantiate assembly task templates for execution on real or simulated robots.

6.4.3 Results

Table 6.2 summarizes the plan makespan and robot idle time results of scheduling experiments for the planned outpost as described above. The given outpost is clearly one with inter-building constraints. For comparison, the table contains a column for a fully parallelizable outpost where each of the planned buildings was treated as independent for scheduling purposes only. Both the second and third columns apply to the planned outpost and could have been executed in simulation.

In all three scenarios, the benefit of additional robots was sub-linear. As the number of available robots increased, the realized benefit in terms of schedule makespan grew only slightly. Note that this benefit consideration does not include any real-world complication during execution where too many robots would clog the workspace, causing many more exceptions than would otherwise occur. How much robot idle time should be accepted depends on the application at hand and the cost of robots being present but not performing work. For the outpost scenario considered here, the reduction in makespan when increasing the number of robots from two to five was deemed worth the increase in robot idle time, whereas adding another five robots resulted in an unreasonable increase in robot idle time for a marginal win in terms of makespan.
6.4. Scaling to Additional Assembly Robots

Table 6.2: If all buildings in an outpost are sufficiently separated to not interfere with each other, full parallelization of the assembly work is possible. If the buildings are closer together, their order needs to be maintained in order to guarantee feasibility. Treating the entire outpost (once planned) as a single scheduling problem yields the shortest outpost schedule that guarantees nominal feasibility.

The main difference between separable and non-separable outposts is the amount of planning time required (see Table 6.3). While the scheduling post-processing step also was faster for fully parallelizable (i.e., separable) outposts, the difference there was negligible compared to the planning time difference. As a result, whenever separability can be exploited, it should be – including in cases not considered in this thesis where parts of outposts are separable while others are not.

After spending the time planning for a non-separable outpost, treating the entire structure as one large problem to be scheduled required more time to find a solution than maintaining a strict serialization by building. This difference was due to scheduling conflicts in the latter case being more compartmentalized such that resolving one conflict had an effect on fewer related conflicts than when the entire outpost was a single activity with many conflict dependencies. However, the additional time spent scheduling the outpost as a whole compared to completely serialized was small in relationship to the savings in both plan makespan and robot idle time (see Table 6.2).

Table 6.3: Planning time depends primarily on the number of building plans required to generate the outpost plan. Scheduling time depends on the number of constraint violations that need to be resolved and the number of constraints between activities on the timeline.
6.4.4 Effects on Completeness and Correctness

The scheduling techniques described in this section are used after the system has generated a nominally feasible assembly plan as described throughout this document. Since parallelization of building plans is only applied when the outpost is found to be completely separable, and the order in all building plans is maintained as planned in all cases, the output of the scheduling step is still guaranteed to be nominally feasible for execution. Even if exceptions occur during execution, the progress of the entire plan will simply be delayed. Future work that considers partial separability will also need to include mechanisms for ensuring that execution-time problems in one part of the plan do not introduce fatal problems at sections of the plan that need to remain ordered because they correspond to non-separable parts of the outpost.

6.5 Summary

Scaling to larger structures and being able to generate plans for additional robots makes the assembly planner truly useful in practical applications. Larger structures are handled by hierarchically decomposing them according to some designer’s or operator’s decisions. Structures too large to be solved directly call auxiliary planners at lower levels of the decomposition hierarchy. Problems small enough (i.e., no larger than approximately 20 components) are solved directly, and the solution is passed to the next higher-level planner to be used in its graph search. Fundamentally the same approach as for planning individual structures is used for outposts, with different meanings of what graph edges and vertices represent at different level (e.g., individual components, wall segments, entire buildings, etc.). This recursive extension also lends itself well to computational parallelization. While not addressed in this thesis, parallel computing techniques could reduce planning times for large structures even further.

Since the assembly graph representation cannot easily be extended to natively reason about additional agents, the system performs a two-step process where the assembly planner generates a plan (if one exists) for the minimum number of robots required for the assembly (i.e., two in this case). This plan is then written as a scheduling problem where dependencies between tasks or sub-tasks are enforced (depending on the type of outpost). The scheduler outputs a parallelized solution to the outpost planning problem that still meets the nominal feasibility criterion and thus can be used to parameterize a task tree for execution by assembly robots. All error recovery mechanisms of the system still function and apply to outpost plans just as they did to segment or building plans.
Chapter 7

Conclusions

7.1 Summary

This thesis addresses the problem of robotically assembling large structures in unconstrained environments (i.e., not in a factory setting). Given a desired (large) structure to be assembled, the initial state of robots and components in the environment and available resources (robots, workspaces, etc.), a planner efficiently generates an assembly sequence that is nominally feasible. An executive system takes the planner’s output and tasks available robots to carry out their assigned tasks. It also monitors progress and, when necessary, enables failure recovery steps (autonomous actions, or ones that rely on operator intervention using Sliding Autonomy techniques) to ensure robust and reliable execution. Assembly planning is made relevant for realistic applications by enabling the system to plan for and execute large structures of hundreds or thousands of components, as well as coordinate the work of any available assembly robots to complete the task more quickly.

The work presented in this document addresses three essential aspects of assembly planning beyond the basic approach of representing structural assembly sequences in a graph and searching that graph for the best sequence (Chapter 3): Executing assembly plans robustly (unless reliability is given, robotic assembly is of little use in real-world scenarios, Chapter 4), planning assemblies efficiently (both to find initial solutions and to enable effective plan-repair and re-planning in response to execution-time failures, Chapter 5), and scaling to larger structures and additional agents (to be able to plan for realistic and interesting structures of hundreds of components and more with potentially multiple robots, Chapter 6). The remainder of this section summarizes the key points of each extension to the basic approach.

7.1.1 Robust Execution

Robust and reliable operation is critical for practical usability of a robotic assembly system. In general, reliability is paramount to execution speed, i.e., it is more important that a structure is completed as desired (even if it takes some extra time), rather than the robots failing irrecoverably part-way through their task due to uncaught exceptions. At a minimum, robots need to have contin-
7.1. Summary

Emergency behaviors in place that allow them to compensate for slight discrepancies between the plan and the real world. This level of error recovery is sufficient only for very simple problems (involving small structures in open environments, see Table 4.1). As assembly problems become increasingly difficult (either larger structures or more constrained environments, or both), additional exception handling capabilities are required.

Autonomous (local) plan repair and (global) re-planning enable the system to successfully accomplish more difficult assembly scenarios, but they are still insufficient for problems of high complexity (Table 4.2). Autonomous recovery mechanisms rely on the fact that each execution-time exception is linked to a specific edge in the underlying assembly graph where plan repair or re-planning has to start. In addition, whatever caused the exception (e.g., an unexpected obstacle, a sensing or manipulation problem, etc.) provides new information for the planner to use when generating an alternate assembly plan.

The ultimate solution to providing robust execution of difficult robotic tasks is to have an expert operator available to provide assistance at the most appropriate level (Section 4.3.3). At the lowest level of recovery, the operator may take manual control and teleoperate robots until an exception is resolved. During repair and re-planning, the operator’s high-level understanding of the task at hand may be useful to modify task parameters or select alternate assembly states to pass through on the way to the desired structure. This Sliding Autonomy interaction allows an operator to apply his assistance where it is both most needed and most appropriate and useful. At the same time, the operator does not need to pay constant attention to the robots (which in many applications is not feasible, e.g., in space scenarios where there is a significant lag in communication time, as well as limited bandwidth available), since they are capable to perform most of their tasks autonomously and stop in a safe state (and request help) if they detect an off-nominal condition. With an expert operator in the loop the system achieves robust and reliable performance in all scenarios (Table 4.3).

In addition to catching exceptions as they occur and resolving them, the assembly planner also incorporates reasoning in the planning process about what will likely make assembly steps more successful (Section 4.4). Many of these aspects are application and robot dependent, but they include metrics such as more stable/compact intermediate structures, sufficient room for robots to maneuver, etc. These methods were shown to accomplish the overall assembly task with the same cost on the part of the robots while requiring fewer instances of operator assistance (Table 4.4). Taking into account the properties and capabilities of autonomous error recovery strategies during planning, they were more likely to be able to resolve problems during execution without having to escalate them to an operator.

7.1.2 Efficient Planning

While execution efficiency and speed are secondary to reliability and robustness in many assembly scenarios, planning efficiency is of interest, especially when plan repair and re-planning are required. Waiting many minutes to several hours for an alternative plan once the current task has been determined to be impossible is not a practical solution. The complexity of many problems along with naïve search strategies make the basic planning approach very slow for interesting assembly problems (over 11 hours to plan the assembly of 21-component structures, Table 5.2). More efficient planning is accomplished by either evaluating individual assembly actions more efficiently
(faster per-edge processing of the assembly graph), evaluating fewer options (fewer graph edges), or combinations of both.

While evaluating individual edges more efficiently carries some benefits in terms of overall planning efficiency, these techniques cannot overcome the fundamental limitation of the basic approach that requires it to process a much larger portion of the assembly graph than is eventually required to find a solution. The most individual benefit is achieved by pre-computing a cost-to-go heuristic for each state in the assembly graph and use that to guide the graph search for an assembly sequence (Section 5.4.1). At the cost of some pre-computation time, the overall planning time can be reduced by approximately two orders of magnitude using this technique alone.

Combining different strategies to both evaluate fewer graph edges and process the ones chosen more efficiently results in the most efficiency gain. Speed-ups in planning time of 2.5 orders of magnitude and more are possible, and the amount of improvement increases with the complexity of the problem (Figures 5.5, 5.6 and 5.7).

### 7.1.3 Scaling to Larger Structures and Additional Robots

Even after extending the basic approach to be robust and efficient, the assembly graph representation limits its applicability to structures of few tens of components. In the context of large-scale assembly, those are not very interesting problems. The basic representation, however, cannot easily be extended to scale to larger structures or additional assembly robots (which will be a practical consequence of larger structures, see Figure 6.2).

Instead, a (potentially recursive) hierarchical decomposition of large assembly problems into small enough sub-problems to fit into the standard representation is used to effectively plan for large structures (Section 6.3). Specifically, individual components are assembled into wall segments for buildings, which are part of a large planetary outpost. In total, the outpost considered consists of over 1,200 individual components. Planning its assembly sequence took 8.6 hours (or 20 minutes per building plan required to find the outpost plan). Additional improvements can be made here to exploit the computational parallelizability of the problem and find a solution even faster. Compared to the basic approach (which took 11 hours for a 21-component structure, Table 5.2), this already was a dramatic improvement.

When reasoning about additional assembly robots (Section 6.4), the planner differentiates between two kinds of large structures (i.e., outposts): those where all buildings are sufficiently separated from one another as to not create conflicts during execution, and those where buildings constrain the assemblability of others. The first kind of outpost requires only a small number of plans to be evaluated, while the latter involves a more extensive graph search to find an assembly sequence. The planner’s output (which is guaranteed to be nominally feasible for the minimum number of assembly robots required) is then posed as a scheduling problem and fed to a heuristic scheduler in charge of producing a parallelized schedule that obeys all resource constraints.

Outposts with separable buildings can both be planned fastest and result in assembly schedules with the shortest makespan (all buildings can be worked on in parallel without inter-building constraints). Connected outposts, on the other hand, require longer planning time and are more complicated to schedule (since there are inter-building constraints as well as intra-building constraints between segments). However, in terms of resulting schedule makespan and the time available robots
7.2 Contributions

Robots’ capabilities to perform physically useful tasks in challenging environments are continuously maturing and improving. This thesis focuses on the higher-level planning and execution aspects that combine individual robots’ low-level capabilities and enable them to accomplish large and complex tasks well beyond the scope of those capabilities. Throughout this work, the following contributions were made:

**A computationally parallelizable representation for complex, decomposable tasks.** Solving large-scale problems (e.g., assembly or materials handling) with both abstract symbolic and fine-grained constraints cannot be accomplished by simply applying existing planning techniques to those problems, as they are fundamentally limited to operating within their respective representation granularity (e.g., abstract operators for task planners or schedulers, and continuous spaces for motion and manipulation planners). Assembly graphs combine both planning modalities: abstract assembly states are linked by assembly steps which are grounded in the real-world workspace of the task. During planning, abstract graph search selects motion planning problems that lead toward a high-level assembly goal, and the solutions of those problems determine which abstract steps to consider next.

**A comprehensive planning and execution framework for assembly tasks performed by mobile manipulators in physically challenging environments.** Using the graph-based representation described above, this thesis developed a comprehensive planning and execution framework that enables robust execution and efficient planning, and that scales to problems large enough to be of interest in realistic situations. Specifically the failure recovery strategies implemented throughout the entire system made assembly planning feasible and useful for real-world applications. This framework enables robots to effectively apply their capabilities specifically designed and tuned for easily manageable smaller-scale manipulation problems toward large problems that require reasoning beyond their local action horizon.

This thesis introduced a new graph-based representation that leverages synergies between abstract symbolic and fine-grained planning strategies toward solving problems involving physical manipulation on a large scale. It used this representation to plan and execute assembly task for multiple mobile robots in constrained environments – tasks of growing relevance as robots’ capabilities mature, but that have not been thoroughly addressed in the literature to date. This work, for the first time, presented a planning system for assembly tasks that combined motion and task planning to find physically realistic and feasible plans for problems with significant abstract structure to them.
7.3 Future Work

This thesis made significant progress toward planning complex high-level assembly tasks (such as the assembly of large structures) for mobile manipulation robots. The techniques presented provide essential tools that enable robots to apply their ever-increasing capabilities for physical manipulation toward tasks that go beyond simply navigating through and sensing their surroundings. In well-controlled settings (e.g., in factories, etc.) robots already excel at assembly tasks. This thesis lays a foundation for expanding similar capabilities into less structured environments. While the work presented throughout this document made assembly planning tractable and useful in real-world situations, there remains much work to be done. This section highlights three key areas where open problems remain.

7.3.1 Comprehensive Operator Interaction

This thesis specifically did not address issues related to user interface design and usability. Instead, the presented system included only the mechanism to process input from the operator and apply it in a useful manner to make assembly planning robust enough to be useful in realistic scenarios.

7.3.1.1 Sliding Autonomy During Planning (and Repair and Re-planning)

How operator assistance can work for robots during task execution is pretty obvious – the operator gets control over parts or all of the tasks via teleoperation. While an autonomously operating robot may be faster or more efficient, the operator’s reliability and resilience is a benefit particularly for error recovery. In general, at the behavioral level, the speed of operation of the robots and what the operator can accomplish are more or less comparable.

During planning, however, the approaches taken by humans and robots/computers differ dramatically. An operator reasons about high-level aspects of the problem when trying to find a solution. The computer, on the other hand, uses a more or less sophisticated brute-force approach that scores and searches through many possibilities to find the one that comes up on top according to some cost function. Combining both approaches in a way that does not artificially hamper the system overall is not a trivial problem. If it takes longer for the operator to understand what the computer is doing and provide high-level input than the time the system would need to search through enough alternatives to come to the same conclusion, the interaction adds little value. To a large extent, this topic ties in with the next one of providing the user with a meaningful interface to facilitate any interaction that can then make the system more capable.

7.3.1.2 Useful and Functional Operator Interface

Section 4.3.4.3 showed that operator assistance is essential to achieving the robustness and reliability required for useful operation. The planning and execution system presented abstractly includes the ability for an operator to assist the system at any level of the hierarchy. In order to best make use of this resource, further studies are required to determine how operators can best provide the required assistance. The ultimate solution is probably a hybrid between operator control and autonomous recovery, where the operator can provide high-level suggestions (which benefit from his big-picture
understanding of the task) that is evaluated and checked by the robots (which know their specific capabilities and thus can determine the goodness of certain steps).

As important as the facilitation of operator input is the topic of what information to provide to the operator to enable him to be most effective to the system overall. A key aspect there will be to create a link between fast low-level information flows (generated during automated search) and slow high-level information (which the operator can process and to which he is able to react).

7.3.2 Large Structure Decomposition

The system as it is presented throughout this document relies on the operator to provide a decomposition of large structures to make them tractable.

7.3.2.1 Automated Decomposition

Automated structure decomposition could use metrics similar to those presented in Section 4.4 combined with graph cutting techniques to create decompositions given a large structure to be assembled. Different environments or robots available to perform tasks will likely benefit from different decompositions. Given examples of operator-specified decompositions combined with the resulting plans and the performance of the system when executing those, it may be possible to learn what is a good or better decomposition for particular scenarios.

7.3.2.2 Partial Outpost Separability

When scaling to larger structures and additional robots, this thesis considered the two extreme cases of outposts where buildings do not constrain each other’s assembly and ones that do not fit into this class (where the ordering of buildings must be maintained). In general, large structures being assembled will fit into both categories at different times along the assembly process. While there may be constraints between subsets of buildings, others may not be affected by the presence of already-erected structures. Efficiently finding the most beneficial path that makes use of both modalities as necessary or possible is an open problem.

7.3.2.3 Dynamic Outpost Decomposition

All structure decompositions considered throughout this thesis are static. An automated system as described above could track the goodness of a particular decomposition and reevaluate it as the environment changes (e.g., due to execution-time exceptions, etc.). With a static partitioning scheme, execution-time problems may force the system to report failure even though the desired structure could still be assembled with a slight change in decomposition. Adding this kind of functionality, however, requires fundamental changes to the assembly graph representation, because the higher-level assembly graph is constructed for a particular set of sub-assemblies. As the structure partitioning changes, so does the assembly graph at the corresponding level of the hierarchy.
7.3. Future Work

7.3.3 Implementation Improvements

7.3.3.1 General Language for Task Templates

In order to be as general as possible, the planner and executive (and any operator interface, etc.) should be specified and set up by a single input. One choice would be to augment task templates with additional information such as known robot performance characteristics, alternate decompositions, etc. The desired goal structure to be assembled along with these generalized task templates would be the problem description given to the planner to solve, removing the need for implicit assumptions made by one component of the system of how other components are expected to perform.

7.3.3.2 Lazy Assembly Graph Generation

The size of the assembly graph is one limiting factor to the size of a problem that can be represented. The graph size grows exponentially with the number of components in a structure. With the most efficient graph search strategy (using a cost-to-go heuristic, Section 5.4.1) only a small fraction of the entire graph is actually needed during search. Instead of pre-computing the entire graph before searching it, lazy methods could be used to only generate as much assembly graph as is required to find a solution. The current implementation would have to be modified, as it requires the entire graph to compute cost-to-go values to speed up search, but the memory savings associated with smaller graphs may well be worth it.

7.3.3.3 More General Error Recovery Option

In its current form, the system allows only error recovery within tasks (during contingencies and repair events), or complete re-plans from the current state of the assembly. Depending on the cause of an exception, the operator providing assistance may understand that the problem (e.g., due to a newly discovered obstacle) also impacts later tasks in the plan. Instead of waiting for the system to continue execution until that future problem is encountered, the operator should have the ability to preemptively change tasks. While for many recovery events it may be sufficient to simply go back to the beginning of the failed task, it may be more desirable to go back several steps before making progress toward the goal again.

7.3.3.4 Graph Search that Exploits the Structure of Assembly Problems

Assembly problems have a distinct structure to them. Early along the plan, the structure is still small, and many approach directions for individual components are reachable. At that level, many alternate assembly steps are of very similar goodness. However, as the system progresses, the tasks are becoming increasingly difficult. The current best-first search over the assembly graph does not take into account this problem structure. Alternate techniques such as beam search or graph search with look-ahead may be able to focus their computation better toward the more difficult parts of the planning problem (i.e., later steps in the assembly).
7.4. Closing Remarks

7.3.3.5 Parallel Computation

The underlying assembly graph structure of the planner is a prime candidate for computational parallelization. Each assembly graph edge describes a separate well-posed planning problem. With sufficient computational resources available, each edge could be evaluated in parallel, followed by a graph search using the computed costs to find a solution. More elegant solutions would only evaluate (in parallel) a small portion of all possible assembly steps, until a solution is found.

7.4 Closing Remarks

Assembly planning is a fascinating area of research at the intersection of task and motion planning. It enables robots to be truly useful in their environments by performing complex tasks involving physical interaction with their surroundings. The work presented in this thesis spans a large spectrum of granularities appropriate to different aspects of the task at hand. While the actual execution of assembly tasks requires fine-grained planning for precise dextrous manipulation, the robots’ ultimate goal is to construct a planetary outpost on a much larger scale where buildings constitute “units” that have to be constructed in a particular order. This order at the abstract level, however, is dictated by constraints of the robots moving through and interacting and the real world at the lowest level. Only with the proper links and interactions between many different planning modalities can difficult tasks such as assembly planning be solved, and only with such comprehensive solutions can robots become truly useful in realistic assembly scenarios.
Bibliography


