View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Aquila Digital Community

The University of Southern Mississippi

The Aquila Digital Community

Honors Theses

Honors College

Spring 5-2018

Self-Reconfiguration Planning in Modular Reconfigurable Robots

Keaton Griffith

University of Southern Mississippi

Follow this and additional works at: https://aquila.usm.edu/honors_theses

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation

Griffith, Keaton, "Self-Reconfiguration Planning in Modular Reconfigurable Robots" (2018). *Honors Theses*. 616.

https://aquila.usm.edu/honors_theses/616

This Honors College Thesis is brought to you for free and open access by the Honors College at The Aquila Digital Community. It has been accepted for inclusion in Honors Theses by an authorized administrator of The Aquila Digital Community. For more information, please contact Joshua.Cromwell@usm.edu.

The University of Southern Mississippi

Self-Reconfiguration Planning in Modular Reconfigurable Robots

by

Keaton Griffith

A Thesis
Submitted to the Honors College of
The University of Southern Mississippi
in Partial Fulfillment
of the Requirement for the Degree of
Bachelor of Science
in the School of Computing Sciences and Computer Engineering

August 2018

Approved by:

____
Bikramjit Banerjee, Ph.D., Thesis Adviser
Associate Professor,
School of Computing Sciences and Computer Engineering

____
Andrew H. Sung, Ph.D., Director
School of Computing Sciences and Computer Engineering

____
Ellen Weinauer, Ph.D., Dean
Honors College

Abstract

MSRs are highly versatile robots that work together to form into different configurations. However, to take advantage of this ability to transform, the MSR must utilize an SRP algorithm to determine what actions to perform to shape itself to reach its goal configuration. An SRP algorithm can be boiled down to a search method through an unexplored graph which we approach with four basic search algorithms to see which algorithm is best when designing an SRP algorithm.

To do this we create a general MSR model known as stickbots and use different search algorithms on a variety of SRP problems to test the model. With these tests we can observe how different algorithms are affected by different scenarios. With the data collected using this model we hope to show that certain algorithms are better suited for creating SRP algorithms and our model can be used to test more complex algorithms.

Keywords: modular self-reconfigurable robot (MSR), self-reconfiguration planning (SRP), chain-type modules, artificial intelligence (AI), modular robots, stickbots, search algorithm

Dedication

I dedicate this thesis to my parents for supporting me through this venture as well as looking out for my wellbeing and to a good buddy of mine, Jon Phebus, who is being deployed soon and helped motivate me to finish up this thesis.

Acknowledgements

I would like to thank my adviser, Dr. Bikramjit Banerjee, for helping me choose my thesis topic and for his patience and guidance throughout the thesis process. I would also like to thank the faculty and staff of the Honors College, School of Computing, and Office of Disabilities and Accommodations for all the guidance, support, and encouragement given to me during my time at the University of Southern Mississippi.

Table of Contents

List of Tables……………………………………………………………………….....ix

List of Figures…...……………………………………………………………………...x

List of Abbreviations…………………………………………………………………..xi

Chapter 1: Introduction…………………………………………………………………1

Section 1.1 Introducing the stickbot model……………………………………….1

Section 1.2 Introducing the 3D stickbot model…………………………………...2

Section 1.3 Summary of an SRP…………………………………………………..3

Section 1.4 Differences in Stickbots compared to real world MSRs……………...3

Chapter 2: Literature Review…………………………………………………………...5

Section 2.1 Real world MSRs……………………………………………………..5

Section 2.2 Abilities of MSRs……………………………………………………..6

Section 2.3 Previous work done with MSRs……………………………………...6

Section 2.4 Our Contributions……………………………………………………..8

Chapter 3:Methodology………………………………………………………………...9

Section 3.1 Data Collected………………………………………………………...9

Section 3.2 Algorithms used on experiments……………………………………12

Section 3.3 Implementation of the stickbot model………………………………12

Chapter 4: Results……………………………………………………………….……16

Chapter 5: Discussion…………………………………………………………………20

Section 5.1 Reason for rule 3 being broken by DFS and DLS…………………..20

Section 5.2 Successor Generation for Stickbot Model…………………………..20

Section 5.3 DFS observation……………………………………………………..23

Section 5.4 DLS observation……………………………………………………….25

Section 5.5 IDS vs BFS discussion…………………………………………….....26

Chapter 6: Conclusion………………………………………………………...…….30

Works Cited……………………………………………………………………...…31

List of Tables

Table 4.1. 2D Problems 2D stickbots search ……………………………………..……..16

Table 4.2. 2D Problems 3D stickbots search ……………….………………….……..17

Table 4.3. 3D Problems 2D stickbots search…………………………...……………..18

Table 4.4. 3D Problems 3D stickbots search ………………………………………..19

List of Figures

Figure 1.1. Basic SRP problem for stickbots …………………………………………..1

Figure 1.2. SRP problems that cannot be solved using 2D stickbot model ……………....2

Figure 1.3. M-TRAN module …………………………………………………….……4

Figure 2.1. A 3D representation of a chain-type M-TRAN module ………………...……5

Figure 2.2. M-TRAN SRP problem ……………………………………………………6

Figure 3.1. Handwritten procedure for 2D and 3D stickbots ..……………………………9

Figure 3.2. SRP problems solvable by 2D and 3D stickbot models ……………………10

Figure 3.3. SRP problems only solvable by 3D stickbot model ………………………...11

Figure 5.1. Successor generation for stickbot model variants ……………………..........22

Figure 5.2. DFS expanded nodes comparison …..……………………………………23

Figure 5.3. Rough image of explored graph that a SRP algorithm must navigate …......24

Figure 5.4. 3D stickbot experiment 1, DLS solution …………………………...……25

Figure 5.5. 2D stickbot experiment 1, DLS solution …………………………….…...25

Figure 5.6. 3D stickbot experiment 2, DLS solution ……………………………….....26

Figure 5.7. Time difference between BFS and IDS for 2D stickbots 2D problems……..27

Figure 5.8. Time difference between BFS and IDS for 3D stickbots 2D problems……..27

Figure 5.9. Time difference between BFS and IDS for 3D stickbots 3D problems……..28

List of Abbreviations

AI	Artificial Intelligence
BFS	Breadth First Search
CW	Clockwise
CCW	Counter Clockwise
DFS	Depth First Search
DLS	Depth Limited Search
IDS	Iterative Deepening Search
M-TRAN	Modular Transformer
MSR	Modular Self-Reconfigurable Robot
SRP	Self-Reconfiguration Planning

Chapter 1: Introduction

Section 1.1 Introducing the Stickbot model

This thesis will explore the utilization and effects of different Artificial Intelligence (AI) search algorithms on the modular self-reconfigurable robot (MSR) process of self-reconfiguration planning (SRP). This means that we will be creating a model for real-world MSRs and have this model undergo SRP utilizing different AI search algorithms such as depth-first search, breadth-first search, et cetera.

For this project, the MSR that we use as a model will mimic the characteristics of an MSR using data values to represent its configurations. We call this model a 2D stickbot; a stickbot configuration problem is shown in figure 1.1 where its graphical and string configurations are shown. A stickbot module is composed of two arms which connect to a central pivot. The arms face in one of four cardinal directions: North, South, East, or West, with the direction an arm is facing represented by the first letter of its directions. Modules are represented by two listed directions separated by a comma within parentheses. For example, (S,E) is a module with its first arm facing south and its second arm facing east. The arms rotate clockwise or counterclockwise in 90-degree increments around a pivot, which is the definition of an action for the stickbot SRP. For further example, if module (S,E) needs to match configuration (W,E), then module (S,E) would rotate arm one clockwise, resulting in the configuration (W,E) which matches its target configuration. An individual module's arms cannot point in the same direction at any point and only the

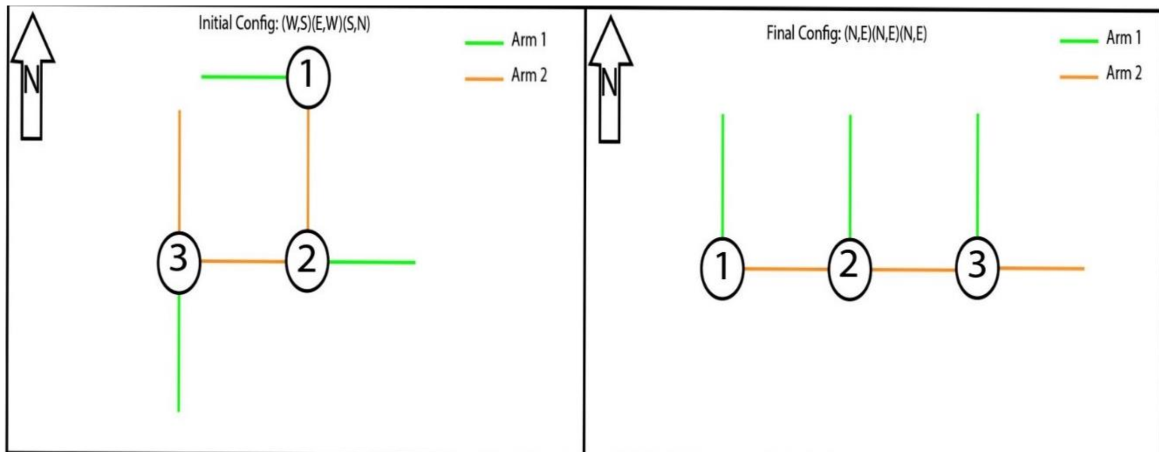

Figure 1.1. Basic SRP problem for stickbots

second arm can connect to other modules. When the second arm of any module overlaps with either arm of another module or points in the direction of a module's pivot an invalid configuration called a cycle is created, however, the first arm of a module may overlap with the first arm of other modules.

Section 1.2 Introducing the 3D stickbot model

The 2D stickbot serves as an elementary model for representing MSRs of similar types, but it is not capable of solving all problems. Consider figure 1.2, for our MSR to solve this reconfiguration problem one of its modules would have to overlap its arms. The 2D stickbot is stuck rotating its arms on a 2D plane which is more than sufficient to solve reconfiguration problems such as the one shown in figure 1.1, but it cannot solve problems such as the one shown in figure 1.2. To solve the problem posed by figure 1.2 the modules need to be able to rotate their arms in other directions; stickbots with the ability to do so are called 3D stickbots. These 3D modules can rotate their arms on three different axis, as opposed to the 2D stickbots which could only rotate on the X-Y plane. The 3D stickbot can rotate on the X, Y, and Z axis which means that not only can the arms point north, south, east, and west, but they can also point up and down. Where the 2D stickbots mimic the mechanisms of real-world MSRs, 3D stickbots mimic the abilities of MSRs to reconfigure into any configuration requested of them regardless of their initial configuration. Due to the verticality of these stickbots cycles do not exist. Also, even though the 3D stickbot can point its arms up and down, these directions will not be featured in our starting or target configurations due to difficulties in illustration. To simplify

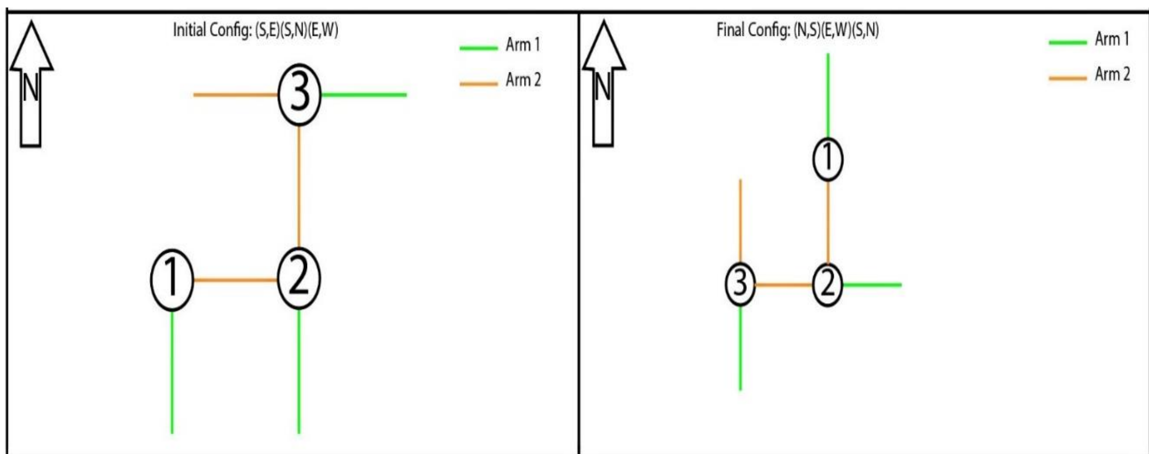

Figure 1.2. SRP problem that cannot be solved using 2D stickbots

computation for the purposes of this thesis, any action that features a rotation on the X and Y axis is performed twice so that the arms of the modules are always facing the four cardinal directions, but this feature does not limit the range of problems the 3D stickbot can solve so long as the target configuration does not feature the directions up and down. Note that this feature to act twice if the rotation is done on the X or Y axis can be removed so that target configurations, for example, the configuration (U,E), are accessible.

Section 1.3 Summary of an SRP

Our MSR model must transition from one configuration into a target configuration. To accomplish this, the stickbot will need to plan the sequence of actions needed to get to the target configuration; this is done using the SRP function. The purpose of the SRP function is to find the sequence of actions that will transition the initial configuration to its target configuration. The function does this by comparing successor configurations to the target configuration. If a successor configuration is equal to the target, then the list of actions leading up to that successor is saved and returned. The successors are generated using a separate function which checks the validity of each successor generated by using an action which is also saved, then a list of successor configurations and actions to get to those successors is saved and returned to the SRP function.

Section 1.4 Differences in Stickbots compared to real world MSRs

There are significant differences between actual MSRs and our model. First, MSRs can perform actions such as attach and detach whereas stickbots modules are not allowed to attach or detach. Clockwise and counter clockwise rotations count as actions for our model even though actual MSRs do not consider rotations to count as actions. However, consider the similarities that exist between the stickbot and an MSR, the modular transformer (M-TRAN), illustrated in figure 1.3. The M-TRAN is made up of two, semicircle-shaped blocks connected by a central link on which the blocks rotate about a pivot. If the pivot of the M-TRAN is the same as the pivot of the stickbot and the

block that rotates around the pivot is arm two of the stickbot, then the link that extends from the pivot of the M-TRAN would be arm one of the stickbot. Hopefully using the stickbots, we can determine the AI search algorithm that is the best to use when designing SRP functions. This will be done using both the 3D and 2D stickbots as testing models to

address a wider variety of problems. Figure 1.3 is of a standard module of an M-TRAN, an MSR that the stickbot attempts to mimic.

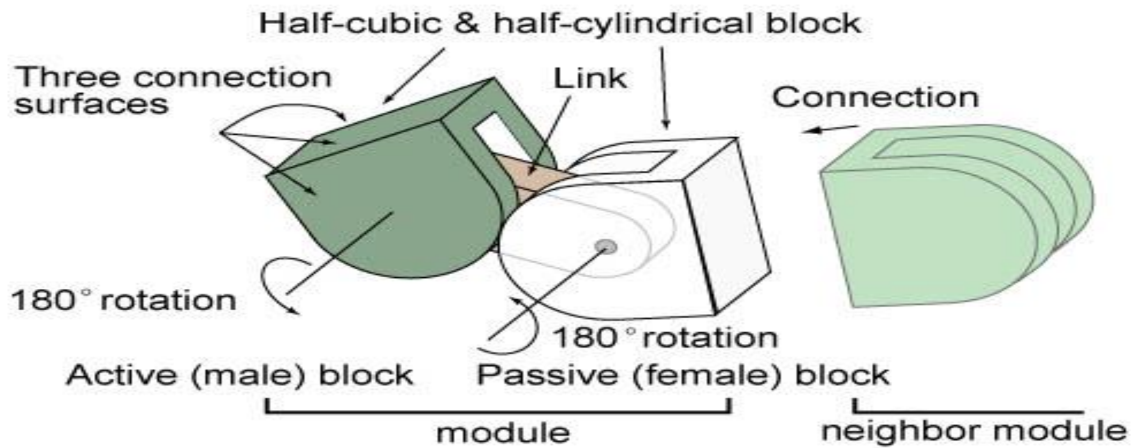

Figure 1.3. M-TRAN module, figure from Kokaji et al., "Self-reconfigurable M-TRAN structures and walker generation", Robotics and Autonomous Systems, 2006

Chapter 2: Literature Review

Section 2.1 Real world MSRs

Modular self-reconfigurable robots (MSR) are robots that can reconfigure themselves into numerous shapes so that they can perform different tasks. This ability to reconfigure into different shapes is possible through MSR's design. Instead of one robot, an MSR is formed using many smaller robots called modules. As discussed by Golestan et al., modules classify as either lattice-type or chain-type (2013). Modules classified as lattice-type can move and reconfigure by attaching and detaching themselves to and from each other respectively utilizing a method known as cluster-flow locomotion (Golestan et al., 2013). Chain-type can sometimes move or change direction without changing their configuration (Golestan et al., 2013). Chain-type modules cannot reconfigure while moving but must remain stationary and only continue traversing their environment once the reconfiguration process is complete. A better way to describe chain-type MSRs is to think of each module as a link in a chain. To see an example of a chain-type module, refer to figure 2.1 below. This literature review will focus on the research of chain-type modules. As it may be clear by now, MSRs possess the traits of being versatile, robust and cheap to produce, but they also possess the ability to navigate terrain that may not be safely accessible for human exploration.

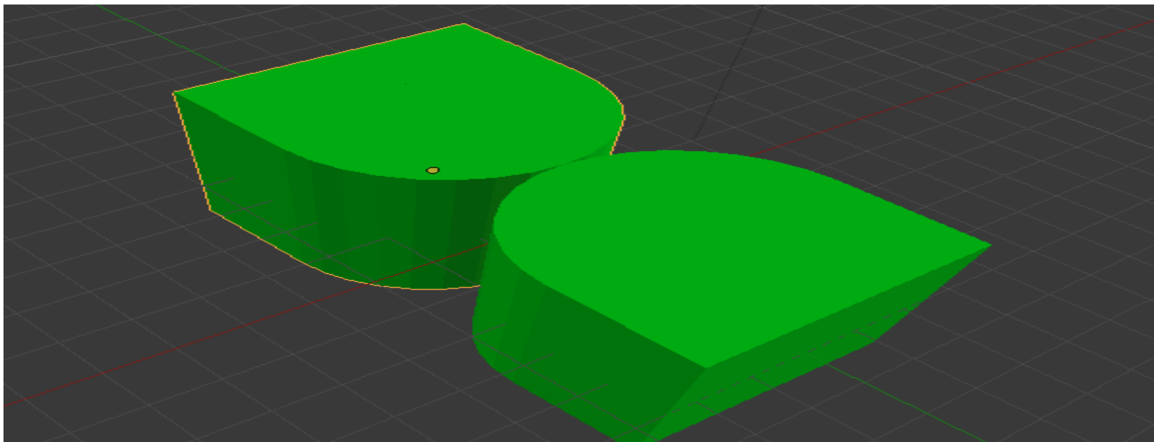

Figure 2.1. A 3D representation of a chain-type M-TRAN module

Section 2.2 Abilities of MSRs

The original appeal of MSRs is their ability to reconfigure when necessary. To take advantage of this ability, MSRs must engage in a process referred to in the literature as Self-Reconfiguration Planning (SRP). Figure 2.2 shows two different MSR configurations. The quadruped configuration is the current position while the snake configuration is the predetermined configuration into which the MSR should form. How to get from one configuration to the next is the purpose of an SRP. SRP involves determining the best, most time efficient and terrain applicable solution to reshape a modular robot from one configuration to the next (Golestan et al., 2013). SRP puts restraints on solutions for reconfiguration such as preventing the disconnection of a module from the body of the MSR unless the disconnection is necessary. This restraint is crucial when planning configurations for chain-type MSRs. The SRP solution must also keep in mind the logistics of changing into other configurations, so efficiency becomes a concern in SRP. The traditional method of constructing an SRP algorithm is by using a graph-based approach (Taheri et al., 2016).

Section 2.3 Previous work done with MSRs

Golestan et al. provided a comprehensive way of representing the connection between modules themselves as well as their configurations in a graph form referred to as

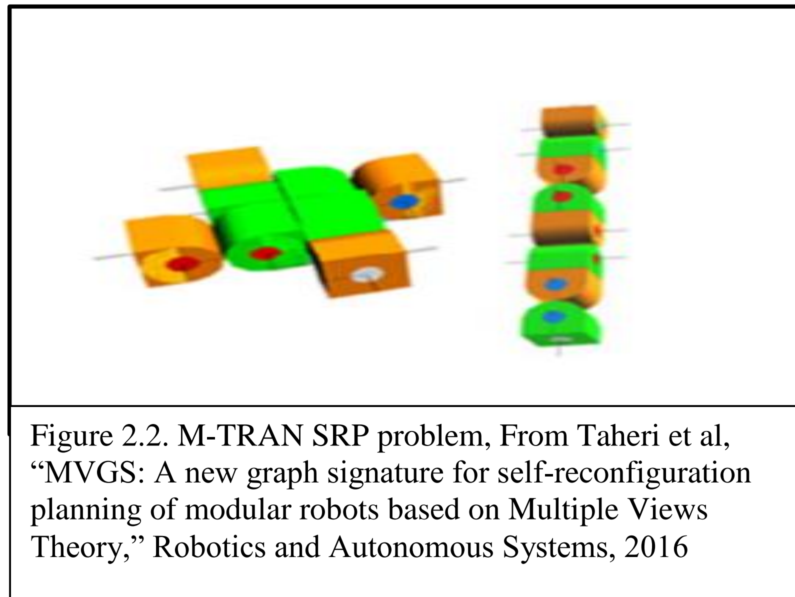

Figure 2.2. M-TRAN SRP problem, From Taheri et al, "MVGS: A new graph signature for self-reconfiguration planning of modular robots based on Multiple Views Theory," Robotics and Autonomous Systems, 2016

a configuration graph (2013). Every node of the graph represents a module with the

connection to other modules shown as an edge. Edges indicate male to female connections that are directed and can be shown with an arrow which points toward the female connector, whereas the undirected edge is genderless (Golestan et al., 2013). Each edge is also labeled based on factors such as the indices of the connectors, the direction of the connectors, and their relative orientation to each other (Taheri et al., 2016). Each connection face of the modular robot is indexed with no importance given to the order so long as every modules' connection faces are labeled in the same manner (Golestan et al., 2013). The orientation of a module is about the surfaces and modules to which it connects. MSRs can only rotate in 90-degree increments (Golestan et al., 2013).

Taheri et al. uses multiple depth-limited searches to search for SRP solutions based on Multiple Views Theory (2016). Multiple Views Theory is based on the statement that humans use many different viewpoints of objects to relate them to similar objects (Taheri et al., 2016). The search method provided by Taheri et al. is a heuristic search (2016). This search method takes the initial configuration and final configuration as parameters (Taheri et al., 2016). It then performs the necessary actions (attaching or detaching) to iterate to the next feasible configuration graph (Taheri et al., 2016). The process continues until one of the generated configuration graphs is equivalent to the configuration graph of the final configuration; the search process is stopped, and the sequence of actions necessary is returned as the solution (Taheri et al., 2016).

Developing solutions for helping MSRs to process or think faster with SRP solutions is required to ensure the robots can perform tasks at an optimal rate because there comes a point when the robots must be applied or put in unpredictable scenarios. Such applications are when the efficiency of the MSR's SRP becomes imperative because every action costs some amount of energy that consumes the robot's batteries. Dasgupta et al. partially address this issue by using a coalition structure graph (2012). This type of graph method searches through a graph of all possible connections between modules and then uses a function to access the cost of performing actions such as connecting to or detaching from other modules.

Section 2.4 Our Contributions

The stickbot model serves as a testing model for the algorithms used for self-reconfiguration planning. Even though the stickbots are only slightly related to the MSRs discussed above, we can use them to test the basic algorithms discussed by Taheri et al.'s (2016). By applying the basic algorithms to our models, we hope to determine their effectiveness when compared to other algorithms. Also, using stickbots we can explore and better understand the features that makes MSRs such useful and versatile modular robots while we learn how to recognize and possibly avoid the pitfalls encountered in the increasingly complex SRP algorithms used with MSRs.

Chapter 3: Methodology

Section 3.1 Data Collected

For this thesis, the data collected was numeric in form, unique to each experiment's SRP problem, and provided results for the time it took to solve the problem, the number of steps needed to reach the final solution, and the nodes expanded. A node or state is defined as a configuration. There is the starting state and goal state, and the goal state is reached by performing actions on a copy of the starting state. The global variable expanded_nodes allows us to keep track of the nodes that are expanded and increments every time a node is visited. There are two constants we keep track of during the experiments. The first constant is the number of modules in each of the ten experiments and the second constant is the predicted number of steps needed, which is determined by working out the SRP problem by hand. This handwritten procedure for SRP problems in experiments worked out using 2D stickbots is shown in figure 3.1a; figure 3.1b shows the handwritten procedure for SRP problems in experiments which require the abilities afforded by the 3D stickbots. Five of the ten experiments seen in figure 3.2 can be used to test the algorithms on both the 2D and 3D stickbots; the remaining five experiments, seen in figure 3.3, can only be solved using the 3D stickbot model.

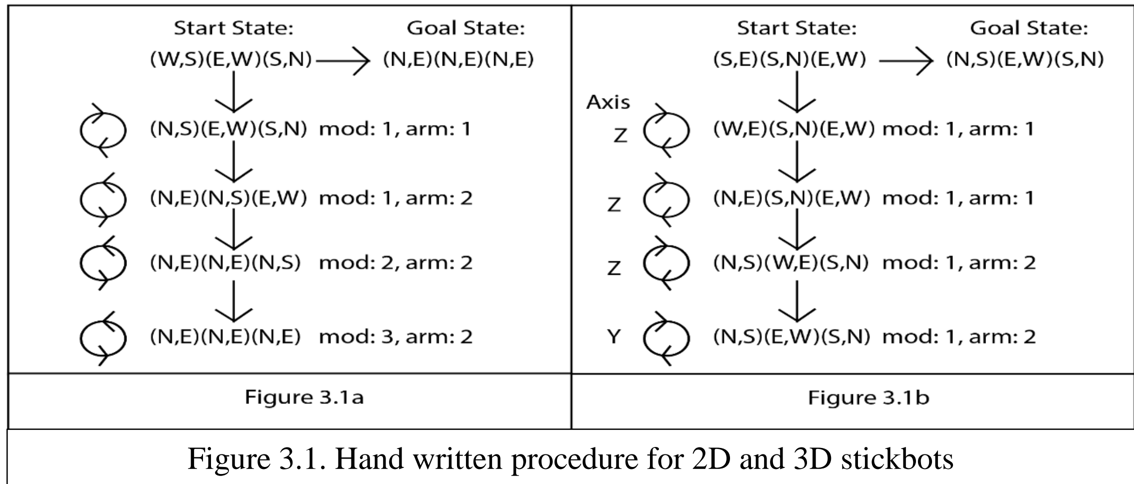

Figure 3.1. Hand written procedure for 2D and 3D stickbots

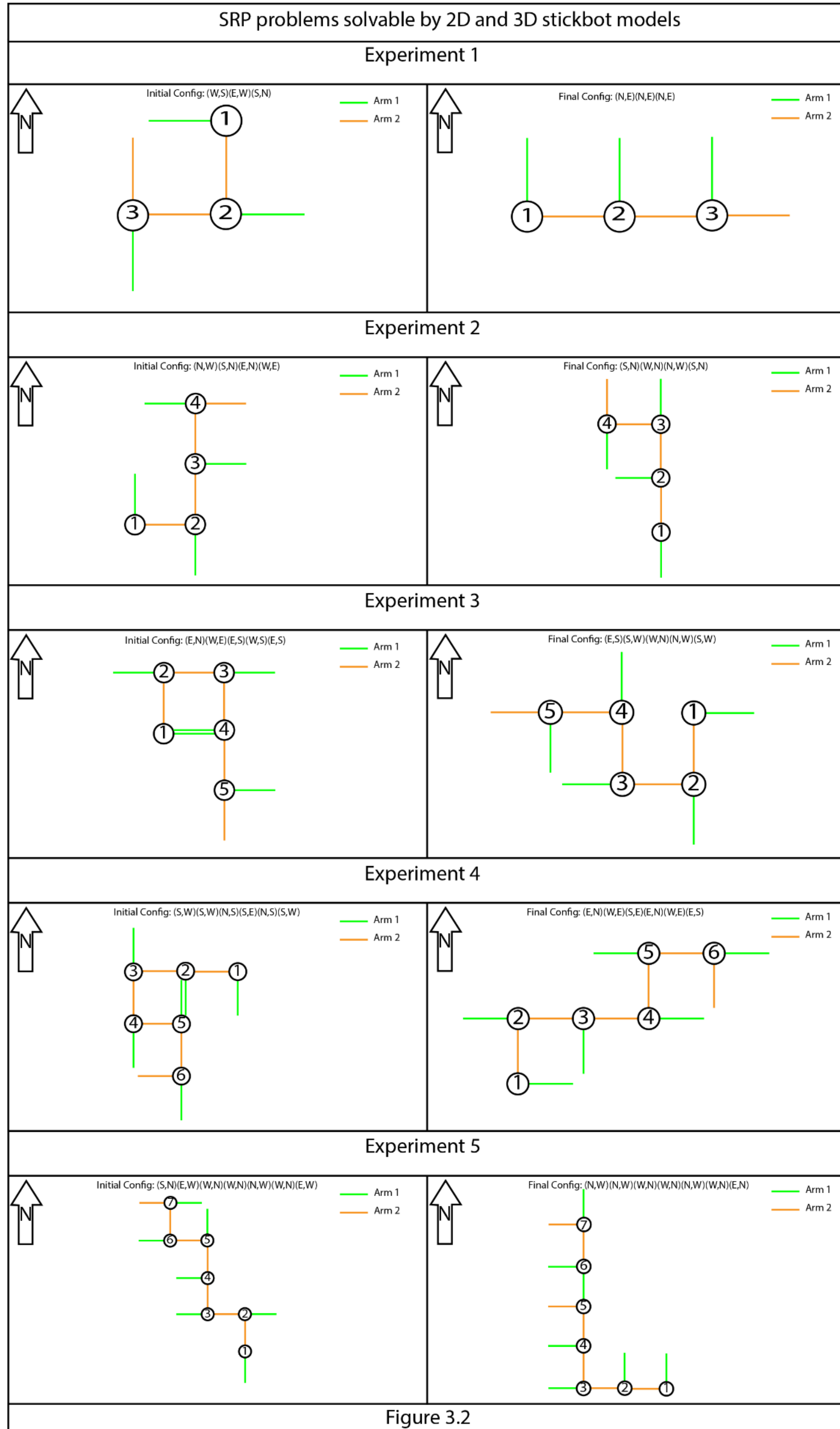

Figure 3.2

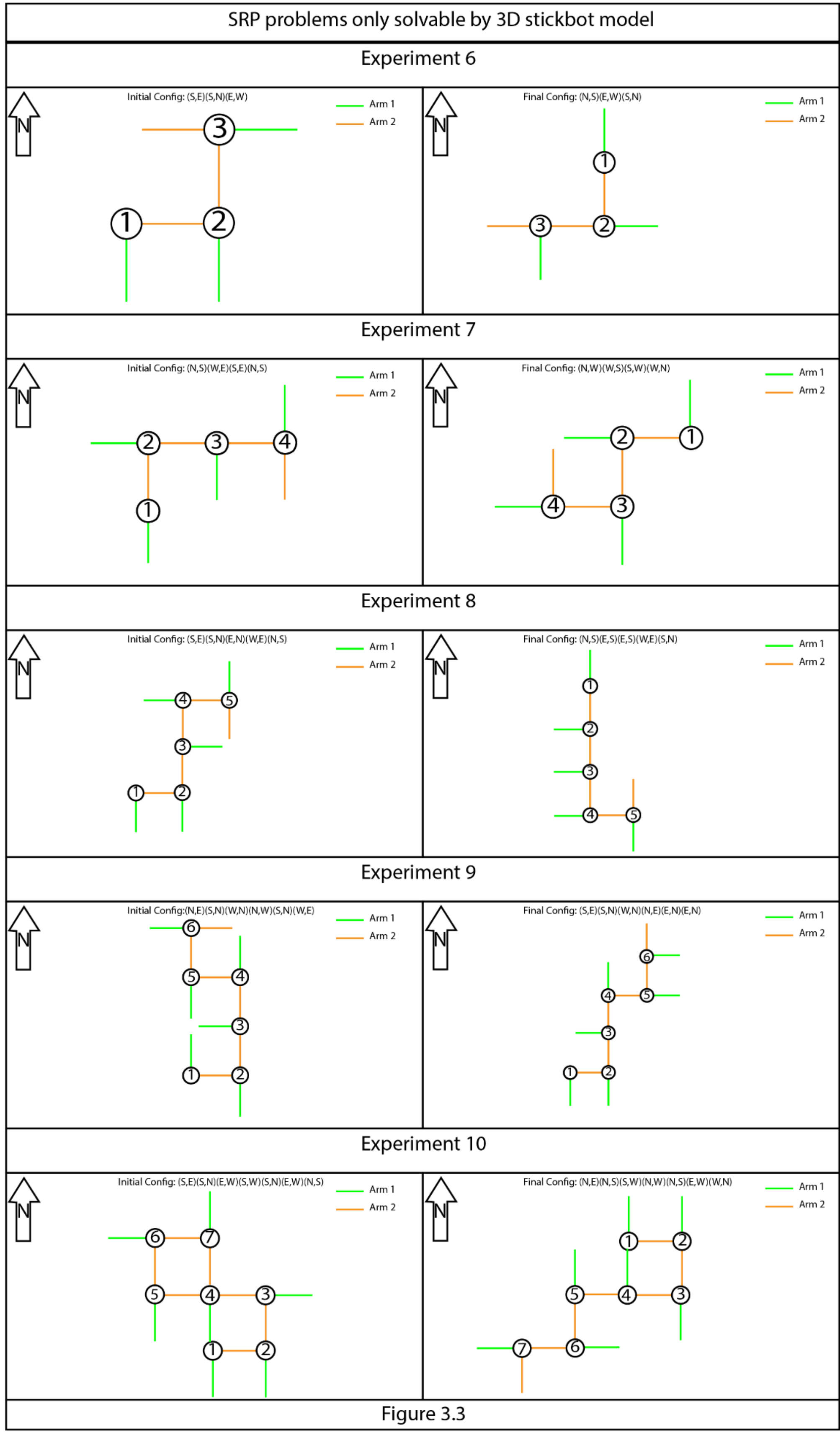

Figure 3.3

Section 3.2 Algorithms used on experiments

Each experiment is run using four common AI search algorithms: Depth First Search (DFS), Breadth First Search (BFS), Depth-Limited Search (DLS), and Iterative Depth-Limited Search (IDS), all of which are implemented in their standard formats. It was challenging to implement the algorithms correctly because some alterations gave the algorithms the ability to solve SRP problems quickly, but had unintended and dramatic effects on how well the problem was solved.

1) DFS: Utilizes a stack to keep track of all unexplored nodes. This implementation performs a goal check on the current node before attempting to expand. The implementation is iterative.

2) BFS: Utilizes a queue to keep track of all unexplored nodes. This implementation performs a goal check on the current node before attempting to expand. The implementation is also iterative.

3) DLS: Like DFS but cannot produce a solution that requires more than a set number of actions which is often referred to as the depth limit. DLS is more of a component of IDS as the range of problems it can solve is limited by depth limit but is included to reassure that IDS is appropriately implemented. The DLS implementation is iterative.

4) IDS: IDS calls DLS with a starting depth limit and then increments the depth limit whenever DLS returns no solution. IDS also has a max depth limit that is determined with the equation: max_depth_limit = 3^num_of_mods. We get the value three because the goal position of each module's arm has a chance of being in three different places if it is not already in its goal position. The num_of_mods refer to the number of modules in any configuration. The max_depth_limit is a loose upper bound and not always obtainable, but theoretically, IDS could show specific problems are not solvable by the 2D model if working the SRP problem by hand is not adequate proof.

Section 3.3 Implementation of the stickbot model

The implementation of the stickbot and the search algorithms are written in python 2.7.13, and the stickbot itself is modeled using six user-defined classes. For the original stickbot model, the foundation of the stickbot is the Directions class which is used to house

the string values of each cardinal direction and utilizes two dictionaries to specify what each cardinal direction rotates too, the names of which are CCW for counter clockwise and CW for clockwise. After which there is the Module class which houses the blueprints of forming a stickbot module. Module is initialized by receiving a list or a Module instance to copy the Module object being passed in the parameter. Module has a list of three arms that face unique directions, the third arm is a dummy arm and faces the opposite direction of previous module's second arm. It is important to note that the dummy arm of the very first module in a configuration is given the directions value "NODIR" whose character value is '.'. The Module class possesses an overloaded string operator to assist with creating the configuration string and three functions used to rotate the arms. The function set_arm is called to perform the rotation action on a module's arm and is passed the index number of the desired arm and the rotation direction in the form of a string, either "CW" or "CCW," as parameters. It is within the set_arm function that the Direction class's dictionaries are called. The functions CWrot and CCWrot are static methods that call Module's set_arm function but perform the procedure of rotating every arm in the module in the respective direction of the functions. These functions are useful whenever the second arm of any module is rotated and assist with the domino effect that occurs in such scenarios. This class is also critical in determining whether the entire configuration is valid. The function isValid is a static method within the Module class that checks if every arm points in a unique direction and returns false otherwise.

The Config class holds a list of module objects and is initialized using a string with the direction data that must be parsed to initialize the Module objects, but this class can also be initialized using a Config object for deep copying a Config onto a new instance. The Config class possesses two overloaded operators: the string and the equivalence operators. The string operator calls the Module's string operator for every Module and combines the Module strings into one string which is returned. The equivalence operator checks if two Config objects have the same number of modules and then continues to compare the string versions of the Config objects, which, through testing, have shown to be adequate comparisons for this thesis. The Config class also utilizes four static method functions. The functions CWcon and CCWcon perform their designated rotation actions on the arm of a specific module, all this information is passed in the functions' parameters,

by calling the set_arm function of the module object in question. If the arm being rotated is the module's second arm, then all Module objects after that module are put through its CWrot or CCWrot methods to handle the domino effect that occurs when a module rotates its second arm. The isValid method checks if the configuration forms a cycle which is checked using a function named cycle, and then checks every module using the Module class's isValid function. The function cycle utilizes a class called Pivpoint which keeps track of the x and y coordinates of module pivots. Cycle utilizes this class to navigate the pivots of the configuration, traversing to each pivot by incrementing its value based on the direction of the module's second arm.

The Action class is used to do and undo actions upon the current state to generate successors. The Action class is initialized by being given a rotation direction, arm index, and module index, all of which are stored in a tuple. Action has two static methods, doAction, and undoAction, both parse the instructions inside an Action object and pass these instructions onto the CWcon and CCWcon functions respectively, and on the 3D stickbot model the CWcon and CCWcon functions will be called twice if the axis of rotation is X or Y. Also note, the undoAction and doAction functions take a Config object as their second argument. The undoAction function performs the rotation action opposite of what the Action object's instructions say.

The searchProblem class manages the experiment data which is composed of the starting configuration and goal configuration which are used to initialize the search problem. There is also a copy of the start state which can be manipulated and have actions performed on it. This class has a getSuccessors static method which returns a list of successors for the search algorithms to sift through. The function getSuccessors utilizes multiple loops to perform every possible action on the current state, checks if the action performed results in a valid configuration and if it does the successor and the Action object that resulted in that successor are appended to a list of successors. Then the action is undone so that the current node remains the same while generating successors. Once finished the getSuccessor function will return the entire list of successors.

The 3D model of stickbot only requires that one parameter be added to every function that plays a role in arm rotation. This parameter specifies the axis that the arm is

meant to rotate upon. The Directions class should also have four more dictionaries added to map the rotation directions depended upon the axis specified, also the directions up and down need to be added to the Directions class. The axis dictates which dictionary is used during the rotation procedure in Module's set_arm function. There is no point to having the Pivpoint class for the 3D model of stickbots as the model does not have to check for cycles.

Chapter 4: Results

The following three rules characterize known relationships among the algorithms:

1. The plan length from IDS cannot be worse than that from BFS. In fact, they must be the same.
2. The number of nodes expanded by IDS cannot be smaller than that of BFS. However, it should not be order of magnitude worse either.
3. The plan length found by 3D search cannot be worse than that found by 2D search, since 2D search is a special case of 3D search.

Table 4.1. 2D Problems 2D stickbots search

Experiment	Time (s)	# of Steps	Nodes Expanded To	Number of Modules	Predicted steps
Depth First Search					
Experiment 1	0.011	24	45	3	4
Experiment 2	0.031	34	35	4	6
Experiment 3	0.317	194	211	5	5
Experiment 4	3.08	278	1511	6	4
Experiment 5	23.009	75	7921	7	5
Breadth First Search					
Experiment 1	0.019	4	67	3	4
Experiment 2	0.279	6	291	4	6
Experiment 3	0.621	5	413	5	5
Experiment 4	0.472	4	236	6	4
Experiment 5	4.58	5	1691	7	5
Iterative Deepening Search with depth limit set at 1					
Experiment 1	0.028	4	400	3	4
Experiment 2	12.4	6	88187	4	6
Experiment 3	2.11	5	9724	5	5
Experiment 4	0.606	4	2880	6	4
Experiment 5	33.4	3	134882	7	5
Depth Limited Search with Depth limit set at 5					
Experiment 1	0.02	4	419	3	4
Experiment 2	2.99	DNF	21524	4	6
Experiment 3	3.26	5	16959	5	5
Experiment 4	1.57	4	7048	6	4
Experiment 5	3.73	5	14623	7	5

The data collected from experiments 1-5 for both the 3D and 2D stickbot models, (see figure 3.2), is shown above in tables 4.1 and 4.2 below respectively. Note that DNF signifies that no solution path was found for an experiment.

Table 4.2. 2D Problems 3D stickbots search					
Experiment	Time (s)	# of Steps	Node Expanded To	Number of Modules	Predicted steps
Depth First Search					
Experiment 1	0.208	67	187	3	4
Experiment 2	3.66	85	2087	4	6
Experiment 3	11.5	2566	2996	5	5
Experiment 4	358.9	4374	58259	6	4
Experiment 5	Memory Overflow	N/A	N/A	7	5
Breadth First Search					
Experiment 1	0.217	4	192	3	4
Experiment 2	3.07	5	1409	4	6
Experiment 3	8.36	5	2188	5	5
Experiment 4	3.04	4	834	6	4
Experiment 5	16.8	4	3344	7	5
Iterative Deepening Search with depth limit set at 1					
Experiment 1	2.63	4	29882	3	4
Experiment 2	52.8	5	527041	4	6
Experiment 3	293	5	2210137	5	5
Experiment 4	83.5	4	652777	6	4
Experiment 5	116.4	4	758417	7	5
Depth Limited Search with depth limit set at 5					
Experiment 1	25.6	5	281396	3	4
Experiment 2	39.2	5	382241	4	6
Experiment 3	223	5	1801985	5	5
Experiment 4	960	5	6395163	6	4
Experiment 5	3739	4	23632762	7	5

3D stickbot SRP problems take longer to solve which can be seen by comparing Table 4.1 and 4.2. However, the 2D stickbot model is limited in what problems it can solve such as the ones shown in figure 3.3. The 3D stickbot model was developed to solve the problems in figure 3.3. The 2D stickbot model focuses on mimicking the physical mechanisms of a chain-type MSR such as M-TRAN. However, 3D stickbots mimic the M-TRANS ability to reshape itself into any configuration given any initial configuration.

To prove that specific problems could not be solved using the 2D stickbot models we developed experiment 6 through 10 in figure 3.3. We had 2D stickbots attempt to solve these problems using the DFS algorithm. We could use the IDS method which we mentioned was theoretically possible, but for IDS to confirm there being no solution, it would have to visit increasingly deep depths that would be determined by the number of modules in the configuration being tested which showed to be problematic as discussed later. In theory, DFS will end up visiting every single vertex in the configuration graph it explores, and table 4.3 shows us that the goal state does not exist for certain configuration graphs given an initial configuration. Thanks to this method of checking we can see how many configurations are available to a 2D stickbot's initial configuration by looking at the expanded nodes. We ran these tests again using the 3D stickbot model with the results shown in Table 4.4.

Table 4.3. 3D Problems 2D stickbots search

Experiment	Time (s)	# of Steps	Nodes Expanded	Number of Modules	Predicted steps
Depth First Search					
Experiment 6	0.023	DNF	109	3	5
Experiment 7	0.323	DNF	325	4	6
Experiment 8	1.49	DNF	973	5	4
Experiment 9	3.899	DNF	2917	6	4
Experiment 10	15.85	DNF	8749	7	4

Table 4.4. 3D Problems 3D stickbots search					
Experiment	Time (s)	# of Steps	Nodes Expanded	Number of Modules	Predicted steps
Depth First Search					
Experiment 6	0.137	85	143	3	5
Experiment 7	1.051	464	672	4	6
Experiment 8	29.382	381	12834	5	4
Experiment 9	384.7	2001	76272	6	4
Experiment 10	Memory Overflow	N/A	N/A	7	4
Breadth First Search					
Experiment 6	0.381	5	413	3	5
Experiment 7	3.097	6	1983	4	6
Experiment 8	3.804	4	1688	5	4
Experiment 9	11.025	4	3360	6	4
Experiment 10	24.81	4	5925	7	4
Iterative Deepening Search					
Experiment 6	3.653	5	52597	3	5
Experiment 7	910	6	7131420	4	6
Experiment 8	4.245	4	41722	5	4
Experiment 9	15.008	4	121860	6	4
Experiment 10	38.99	4	293081	7	4
Depth Limited Search with Depth Limit set at 5					
Experiment 6	0.478	5	6941	3	5
Experiment 7	203.6	DNF	1989588	4	6
Experiment 8	68.709	4	657944	5	4
Experiment 9	334.1	4	2758687	6	4
Experiment 10	1142.64	4	8215614	7	4

Chapter 5: Discussion

Section 5.1 Reason for rule 3 being Broken by DFS and DLS

Initial observation shows that rule three does not hold when comparing the solution paths of DFS between 3D and 2D stickbots. There is reason to believe that the way in which the stickbots generate successors combined with the characteristics of the DFS, and in turn DLS, algorithms' search strategy is responsible for this failure to satisfy rule 3.

Section 5.2 Successor Generation for Stickbot Model

Stickbots generate successors by running through every combination of actions that can be performed on every arm of every module in the configuration rotating clockwise and counter clockwise. If the action performed produces a valid configuration the action and configuration are added to the list of successors that are returned for the current configuration. Calculating how many successors this will generate while excluding all invalid configurations or possibly and previously visited nodes is very scenario driven so we will calculate a loose upper bound for how many successors are generated. For the 2D stickbot model consider the configuration A = (N,S), which can generate the maximum number of successors with no invalid configurations.

With the configuration A we first perform a clockwise rotation on one arm and then undo the action so that the configuration remains unchanged for generating the next successor, then perform counter clockwise rotation on the same arm, undo the action, and then repeat the process for the next arm. We produce four possible successors for one module. We could stop here and say that the loose upper bound for successors generated by the successor function of stickbots produces 4n, n being the number of modules in the configuration. However, only the first module of a configuration can produce a maximum of four configurations. Consider the configuration B = (N,S)(E,W). The first module's second arm points south which means no arm on the second module can point north as it would cause two arms to overlap. Thus, the first arm of module two of configuration B will produce an invalid configuration when it performs a counter clockwise rotation which will not be added to the list of successors. The same is true for the second arm of the second module on configuration B when it performs a clockwise rotation. This means the second module only produces two successors during successor generation. If we add a

module (S,N) as the third module to configuration B such that B = (N,S)(E,W)(S,N), we observe that module three is under the same conditions as module two and so only produces two successors. If we continue adding modules to configuration B such that every module can produce its maximum number of successors, careful of also avoiding cycles if possible, then the maximum number of successors generated by any configuration is equivalent to the formula below:

$$S \leq 4 + 2(n-1) \text{ for } n > 1$$

Where n represents the number of modules in the configuration and S represents the number of successors generated by the configuration if the successors returned by the function are the maximum number of successors possible.

The 3D stickbot has a wider array of actions it can perform on each arm of a configuration. However, 3D stickbots can perform actions where there is no change to the configuration. For example, if configuration C = (S,N) on the 3D stickbot model tries to produce a clockwise or counter clockwise action on the Y-axis it will not produce any successor that are different from the current state and as a result, produces a duplicate. To put it simply the successor function is not allowed to add duplicates to the list of successors. Note that if we were to add module (W,E) such that the configuration C = (S,N)(W,E), arm 2 of module 1 could rotate on the Y-axis in order to add the configuration C = (S,N)(E,W), to the successor list. Configuration D = (E,W) can produce four successors by performing actions solely on the Z axis. Rotations on any other axis would produce either an invalid configuration or duplicates. If we allowed successors such as the successor of configuration D = (U,W) to exist by having actions on the X and Y-axis only occur once, then the total number of successors for one module increases to eight, and in that case the upper bound of successors generated by the successor generation function would be as shown below:

$$S \leq 8 + 4(n-1) \text{ for } n > 1$$

S representing the maximum number of successors returned, and n being the number of modules in the configuration. However, this thesis does not feature target configurations utilizing the up and down directions, so rotations on the X and Y axes can only produce

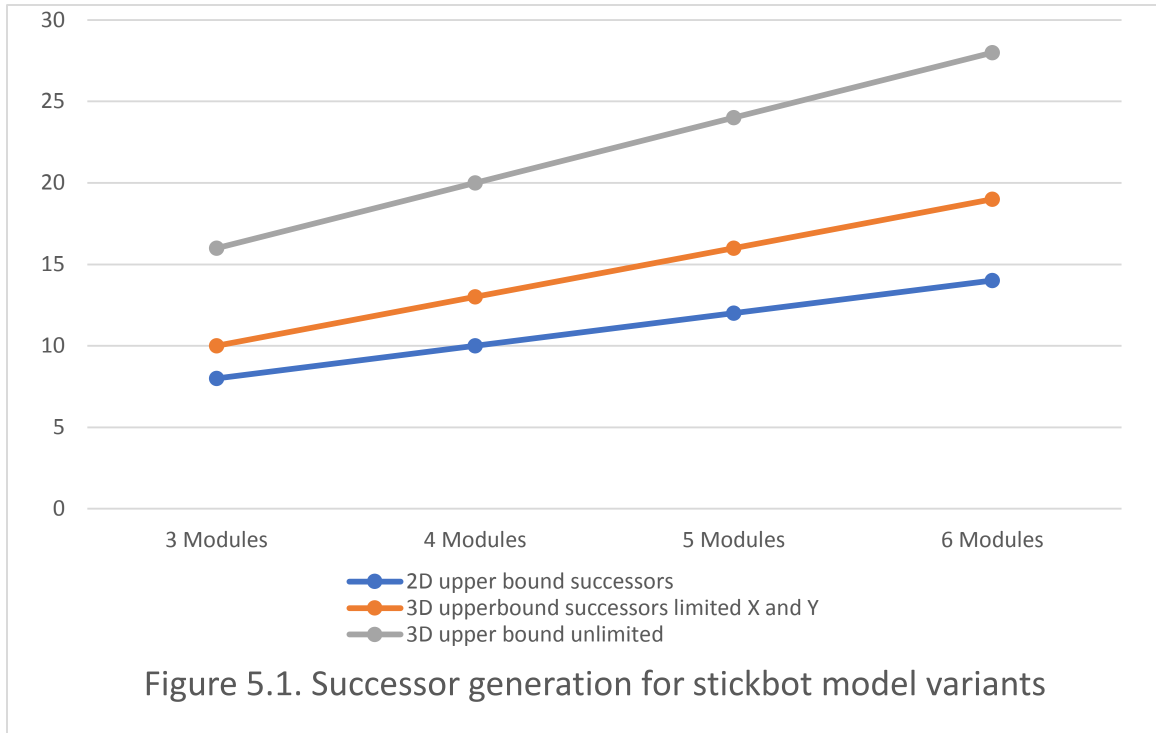

Figure 5.1. Successor generation for stickbot model variants

one successor. Thus, the upper bound of successors generated by successor generation function resembles that shown below:

$$S \leq 5 + 3(n-2) + 2 \text{ for } n > 2$$

Regardless of the limitations placed on the 3D stickbot model's capabilities, we can still see in figure 5.1 that the number of successors generated by the 3D stickbots with limited actions on the X and Y axes grows faster than the number of successors generated by the 2D stickbots' upper bound. Figures 5.2 below shows a similar trend when we measure the nodes expanded before finding the solution using DFS. We will show this trend through experiment 4 since experiment 5 for the 3D stickbot model resulted in memory overflow. Multiple attempts were made to prevent this outcome, but nothing short of replacing the DFS, which was not an option in the framework of this experiment, would stop the fringe from growing out of control. Therefore, the memory error appears to be an unfortunate side effect of a 3D module's rapid successor list growth.

The trend seen in Graph 5.2 from experiment 1 through 4 is similar to that seen in Graph 5.1. To understand why 3D stickbots are creating worse solution paths than that of the 2D stickbot model we should first address the fact that the DFS solution paths are worse

than BFS, DLS, and IDS. Thus, the reason rule three does not hold for the DFS algorithm tested on 2D, and 3D stickbot models has to do with DFS's search strategy.

Section 5.3. DFS observation

The DFS algorithm must navigate a sprawling unexplored undirected graph G = (V, E) where V is a vertex symbolic of a configuration which can be reached using E which is an edge symbolic of an action. To get an idea of how large and interconnected the graph can become, an explored graph, is shown in figure 5.3 on the following page, was created by mapping all the nodes and edge connections while the 3D stickbot model tried to solve the problem presented in Experiment 5 DLS. Every red dot in figure 5.3 represents a unique configuration and each configuration is reachable within five actions of the initial configuration.

For DFS to find the solution, it follows a linear path of actions down an infinite depth until it eventually finds the solution. Every configuration visited during this process is marked as visited and never visited again. The interconnected nature of the graph that can be formed while it is being explored might attribute to why MSRs are so interesting and theoretically the most versatile of any robot, but it presents a problem for SRP algorithms created using DFS. DFS only backtracks if it encounters a configuration that it has visited before. Because of this DFS can manage to find the solution, but at times

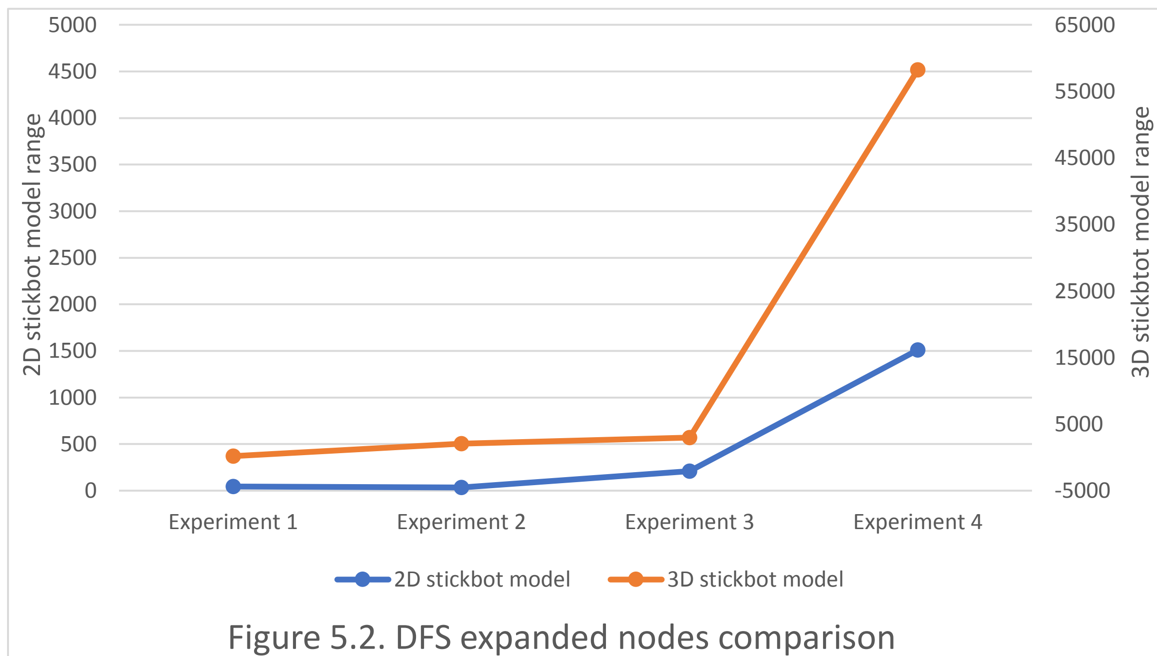

Figure 5.2. DFS expanded nodes comparison

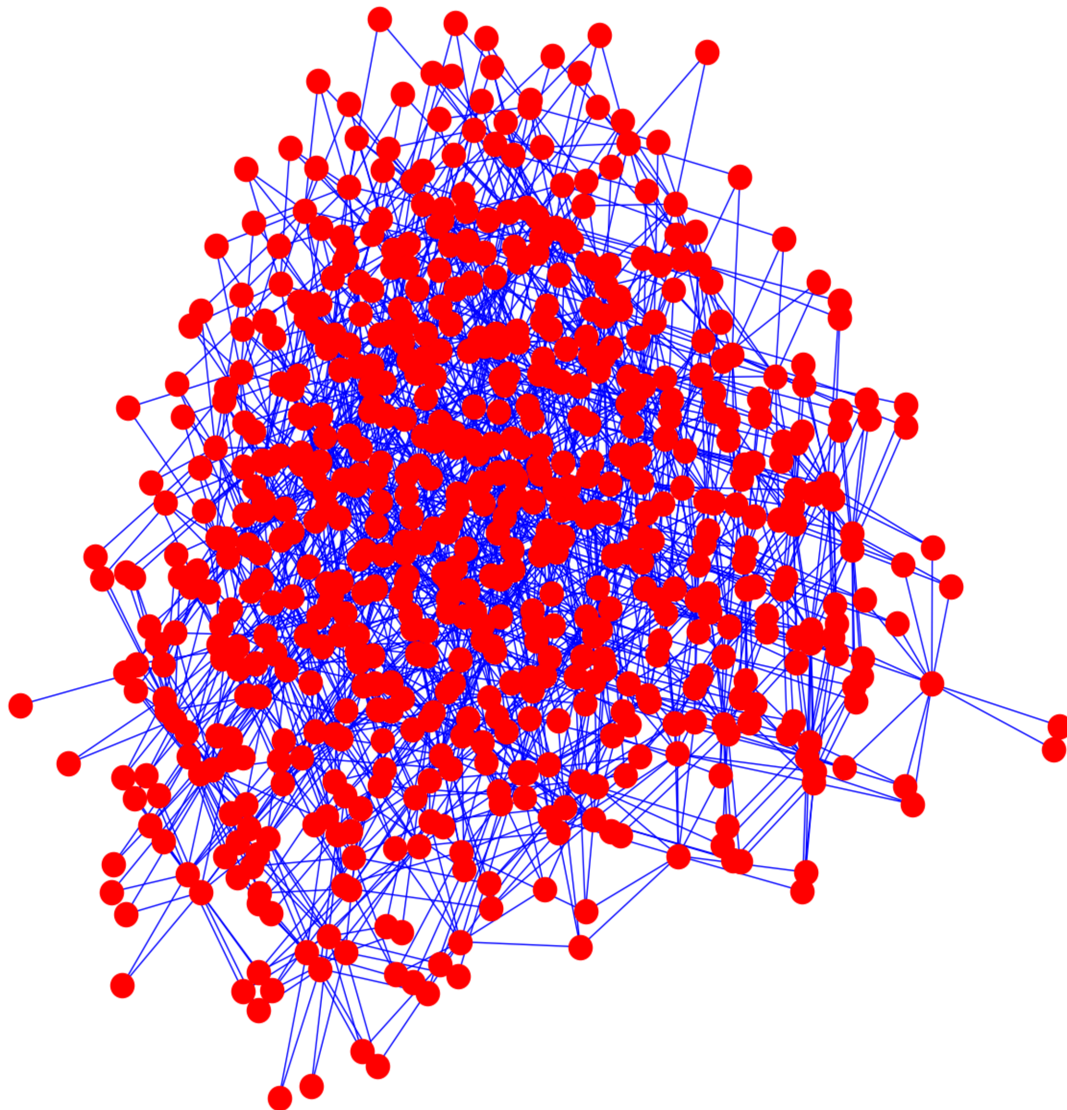

Figure 5.3. Rough image of explored graph that a SRP algorithm must navigate

through the most convoluted path. The reason the 3D stickbot model produces a worst path when tested with DFS has to do with how many successors the algorithm can visit when compared to its 2D stickbot counterpart. Based on the initial configuration there are sets of configurations that cannot be reached using the 2D stickbot model. Thus, the 2D stickbot model is limited on which configurations it can reach based on how many modules are in the configuration and limited based on the initial configuration. However, the 3D stickbot is only limited on how many modules are in the configuration. Since the 3D stickbot can reach more configurations, it exponentially increases the solution path found

by DFS. This leads to the conclusion that DFS is not the most practical algorithm for SRP planning since it rarely concerns itself with finding an optimal path problem.

Section 5.4 DLS observation

DLS, on experiments 1 and 4 for 3D stickbots, produces worst solution paths than those found for the 2D stickbot model. DLS utilizes the same search strategy as DFS but cannot perform more actions than the value specified by the depth limit. However, if the limit exceeds that of the depth required to solve the problem, DLS has a chance of finding the solution with a more extended procedure then necessary. This often happens because DLS utilizes actions that are redundant and unnecessary. Figure 5.4 shows the solution path found by DLS when tested on the 3D stickbot model. DLS found a solution by first

Start (W,S)(E,W)(S,N)
1. Rotate arm 1 of module 1: Counter Clock Wise on Y axis.
(E,S)(E,W)(S,N)
2. Rotate arm 2 of module 3: Counter Clock Wise on Z axis.
(E,S)(E,W)(S,W)
3. Rotate arm 2 of module 2: Counter Clock Wise on Z axis.
(E,S)(E,S)(E,S)
4. Rotate arm 1 of module 1: Counter Clock Wise on Z axis.
(N,S)(E,S)(E,S)
5. Rotate arm 2 of module 1: Counter Clock Wise on Z axis.
(N,E)(N,E)(N,E)
Final (N,E)(N,E)(N,E)

Figure 5.4. 3D stickbot experiment 1, DLS solution

performing an action where the first arm of the first module rotated on the Y axis even though such actions are not necessary. Compared to how DLS found the solution for the

Start (W,S)(E,W)(S,N)
1. Rotate arm 2 of module 1: Counter Clock Wise.
(W,E)(N,S)(E,W)
2. Rotate arm 1 of module 1: Clock Wise.
(N,E)(N,S)(E,W)
3. Rotate arm 2 of module 3: Counter Clock Wise.
(N,E)(N,S)(E,S)
4. Rotate arm 2 of module 2: Counter Clock Wise.
(N,E)(N,E)(N,E)
Final (N,E)(N,E)(N,E)

Figure 5.5. 2D stickbot experiment 1, DLS solution

2D stickbot model as shown in figure 5.5. However, giving DLS the freedom to do this has allowed Experiment 2 to find a solution at a depth limit of five as shown in figure 5.6. This is brought up to make a point. It is possible to limit what kind of successors are generate by 3D stickbots to improve the paths found by DLS; however, this would prevent DLS from finding the path found in figure 5.6. Thus, we conclude that DLS is behaving normally and the deviance from the rules we set earlier is the result of how DLS searches for its solution. In reality, DLS was not meant to be unpaired from IDS or algorithms like IDS.

Start (N,W)(S,N)(E,N)(W,E)
1. Rotate arm 1 of module 1: Counter Clock Wise on X axis.
(S,W)(S,N)(E,N)(W,E)
2. Rotate arm 2 of module 3: Counter Clock Wise on Z axis.
(S,W)(S,N)(E,W)(S,N)
3. Rotate arm 1 of module 3: Counter Clock Wise on Z axis.
(S,W)(S,N)(N,W)(S,N)
4. Rotate arm 2 of module 2: Counter Clock Wise on Z axis.
(S,W)(S,W)(W,S)(E,W)
5. Rotate arm 2 of module 1: Clock Wise on Z axis.
(S,N)(W,N)(N,W)(S,N)
Final (S,N)(W,N)(N,W)(S,N)

Figure 5.6. 3D stickbot experiment 2, DLS solution

Section 5.5 IDS vs BFS discussion

IDS searches by starting from a depth limit of one and then iteratively increases the depth limit every time DLS returns with no solution. BFS searches by checking all the vertex of the current depth before advancing to the next depth. Like DFS, BFS will never visit a vertex twice in the graph being explored, but this comes at the cost of memory. IDS, on the other hand, has no procedure to ensure that it is not visiting the same vertex in a cycle which in theory causes IDS to take more time. Below in Figure 5.7 and 5.8 we compare the time taken between each algorithm to produce their solutions for each experiment on both models of stickbots. Figures 5.7 and 5.8 show what would be expected

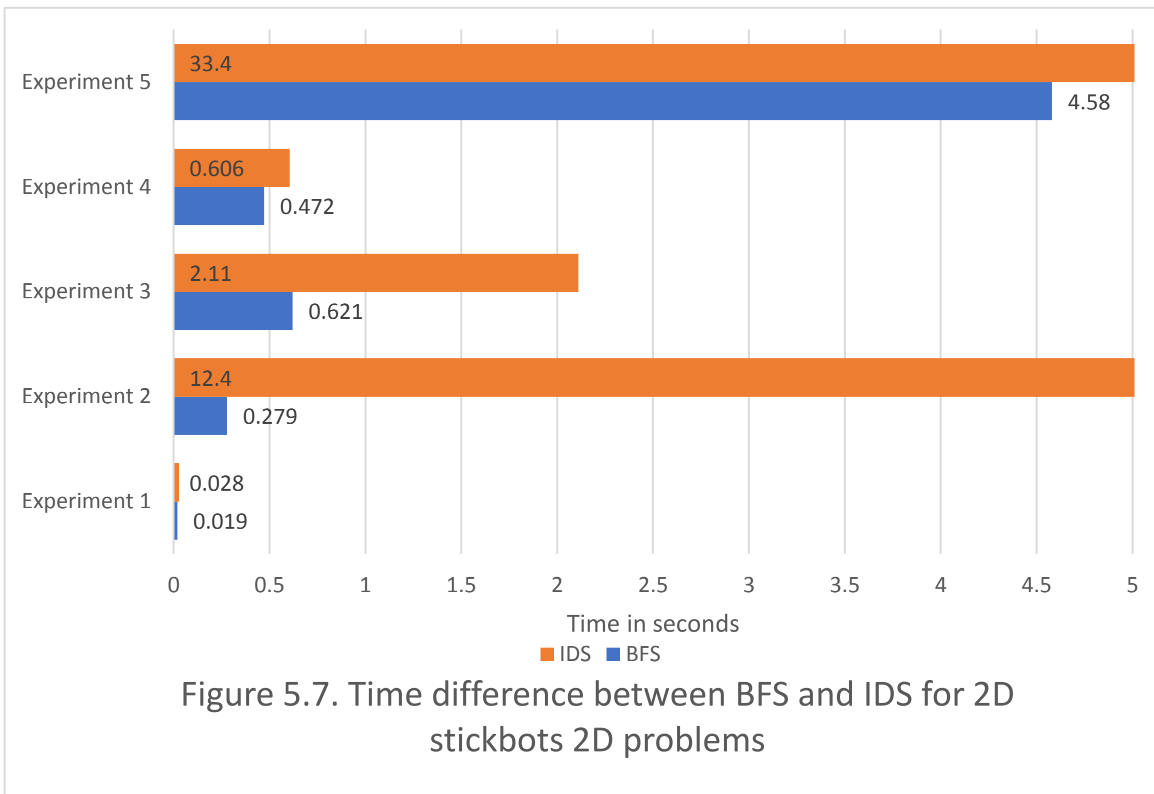

Figure 5.7. Time difference between BFS and IDS for 2D stickbots 2D problems

when comparing the time difference between BFS and IDS. BFS beats IDS in every experiment. A random spike in the amount of time taken for IDS to solve the problem presented in experiments 2 and 5 is most likely due to the complexity for the SRP problems in experiment 2's case, or the number of modules present in the case of experiment 5. The

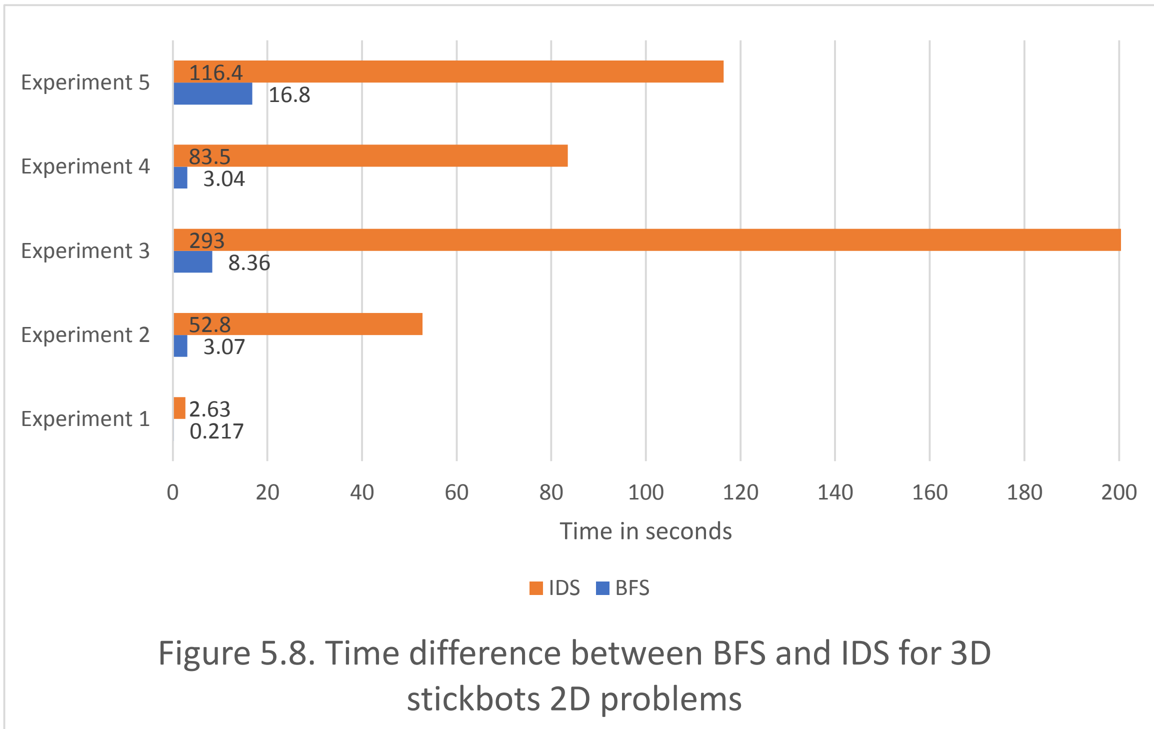

Figure 5.8. Time difference between BFS and IDS for 3D stickbots 2D problems

reason we do not see the same spike occur in Figure 5.8 is in one part due to the 3D stickbot's ability to lower the solution depth for experiments 2 and 5. Even though the solution depths were the same for experiments 2 and 3, experiment 3 took longer to solve for IDS. The only thing different between the results of both experiments apart from the time and nodes expanded would be the number of modules in the configuration. Since the 3D stickbot was able to simplify the solution for experiment 5 it shares a similar solution depth with experiment four, and again the only difference between them happens to be the number of modules. Though experiment 5 and 4 have a lower solution depth than three. Even BFS show similar patterns but not the same. BFS's times suggest that BFS is slightly more influenced by how many modules are in the configuration when compared to how such information affects IDS. Though if this were the case, then experiment 2 would have taken less time than experiment 4.

Looking at figure 5.9 we see that IDS takes incrementally more time to solve experiments 8, 9, and 10. IDS also has the longest time attempting to solve the SRP problem presented by experiment 7. Experiment 7 also has the deepest solution depth while the solution depth for 8, 9, and 10 remains the same. This further suggest that IDS

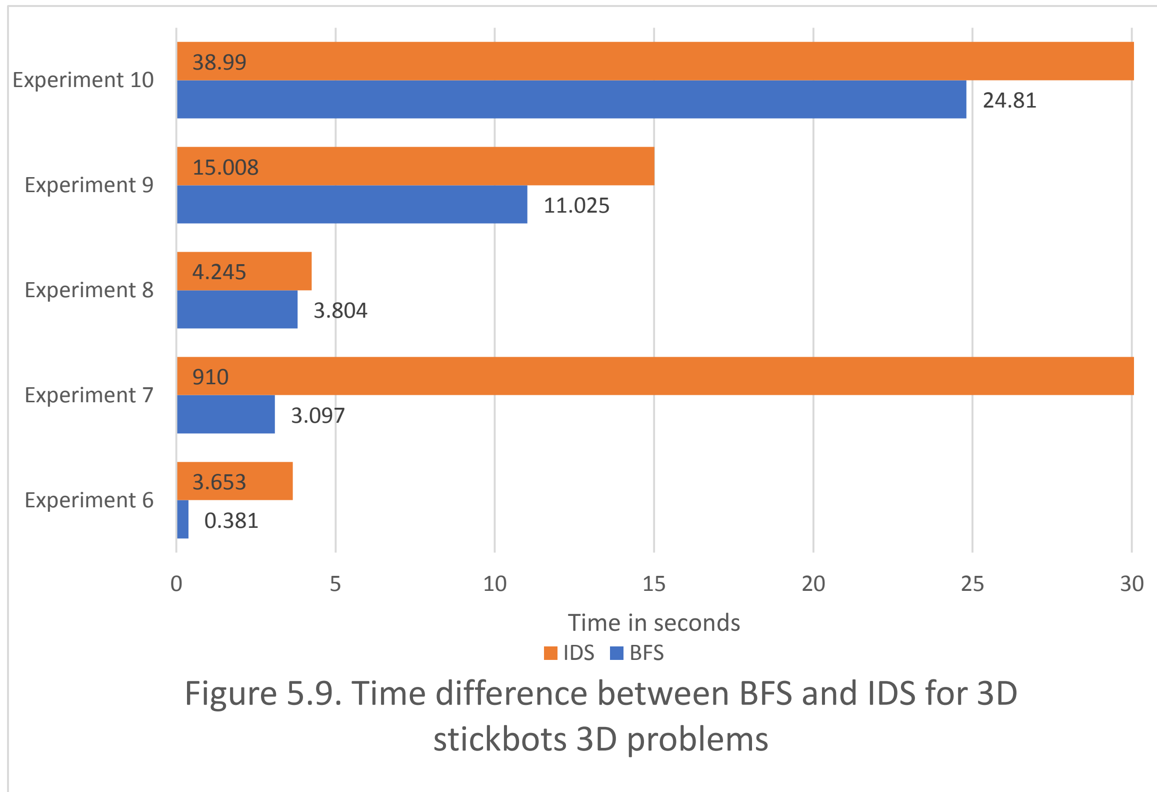

Figure 5.9. Time difference between BFS and IDS for 3D stickbots 3D problems

is affected by the number of modules in the configuration as well as the depth of the solution; however, this is not the same for BFS. Experiment 7 takes less time to solve then experiment 8 even though the solution depth for 7 is deeper than for 8 which helps support the idea that BFS is affected more by how many modules are included in the configuration.

Even though IDS take longer to solve SRP problems than BFS, IDS is not without its benefits. It is important to remember that IDS's memory usage is linear which cannot be said for BFS. This is attributed to the fact that IDS do not keep track of configurations that it has already visited which partly attributes to how much time it takes since it can go back and forth between two configurations up to the depth limit. For a modular robot that possibly relies on less memory than a laptop, IDS and DLS will never cause memory overflow whereas BFS has the potential to and DFS will cause memory overflow (see table 4.2 experiment 5 and table 4.4 experiment 10). Stickbots do not have any data structure that implements any form of memory constraints and what those memory constraints should be based on is still to be determined. Though if said memory constraints were placed onto stickbots then the algorithms we run experiments through would narrow to algorithms that utilize linear memory space.

Chapter 6: Conclusion

In conclusion, DFS is best used to find if a path exists for the 2D stickbot but is redundant to use on the 3D stickbot model since its abilities guarantee that the solution path exists. BFS will beat IDS and is greatly affected by how many modules are in the configuration, whereas IDS is affected more by a combination of solution depth limit and number of modules in a configuration. Future iterations of tests using the stickbot model needs to feature changes that allow the stickbot to more closely mimic its real-world chain-type MSR counterparts. New logic could be implemented to make it so that actions do not have to be performed to check if a configuration is valid. Also, more complex algorithms should be tested using this model and memory constraints should be implemented on the stickbot based on the memory constraints of real world chain-type MSRs. The stickbot model should serve to better introduce the concept of chain-type MSRs. The data collected during this thesis project can help serve as a guide for anyone seeking a new approach when developing their SRP algorithms.

Works Cited

Dasgupta P, V. Ufimtsev, C. Nelson, and S. M. G. Mamur. "Dynamic Reconfiguration in Modular Robots Using Graph Partitioning-Based Coalitions." In AAMAS, pages 121-128. Valencia, Spain, 2012. http://dl.acm.org/citation.cfm?id=2343593 Accessed 30 July 2018.

Golestan K, M. Asadpour, Hadi Moradi. "A New Graph Signature Calculation Method Based on Power Centrality for Modular Robots." Distributed Autonomous Robotic Systems, Vol 83, pages 505-516. 2013. http://www.pami.uwaterloo.ca/pub/kgolesta/Keyvan_Golestan_DARS2010.pdf Accessed 30 July 2018.

Russel S. J, P. Norvig. Artificial Intelligence A Modern Approach: Third Edition. New Jersey: Pearson Education Inc. 2010. Print

Taheri K, H. Moradi, M. Asadpour, P. Parhami. "MVGS: A new graph signature for self-reconfiguration planning of modular robots based on Multiple Views Theory." Robotics and Autonomous Systems, Vol 79, pages 72-86. 2016. http://www.sciencedirect.com/science/article/pii/S0921889016000191 Accessed 30 July 2018.