PERFORMANCE IMPROVEMENTS FOR THE MULTIPLE GRAVITY ASSIST, LOW THRUST SPACECRAFT TRAJECTORY OPTIMIZATION SUITE (PI-MGALT-STOPS)

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Aerospace Engineering

by

John P. Caldwell

June 2023

© 2023
John P. Caldwell
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Performance Improvements for the Multiple Gravity Assist, Low Thrust Spacecraft Trajectory Optimization Suite (PI-MGALT-STOpS)

AUTHOR: John P. Caldwell

DATE SUBMITTED: June 2023

COMMITTEE CHAIR: Kira Abercromby, Ph.D.
Professor of Aerospace Engineering

COMMITTEE MEMBER: Eric Mehiel, Ph.D.
Professor of Aerospace Engineering

COMMITTEE MEMBER: Paul Choboter, Ph.D.
Professor of Mathematics

COMMITTEE MEMBER: Michael Griffith Malloy, M.S.
Aerospace Engineer

ABSTRACT

Performance Improvements for the Multiple Gravity Assist, Low Thrust Spacecraft Trajectory Optimization Suite (PI-MGALT-STOpS)

John P. Caldwell

The following work is a continuation of the Spacecraft Trajectory Optimization Suite (STOpS). It follows in the footsteps of previous authors such as Timothy Fitzgerald, Shane Sheehan, and most recently Griff Malloy who were all graduate students at California Polytechnic State University, San Luis Obispo. The previous works were created due to the need of a publicly available preliminary trajectory design suite. STOpS by Fitzgerald [1] was a high thrust trajectory optimizer that created the framework for the suite. It included four (4) metaheuristic algorithms and the Island Model Paradigm. Low Thrust STOpS (LT-STOpS) by Sheehan [2] then adapted the suite to allow for low thrust trajectory optimization. The next step for the suite was Multiple Gravity Assist LT-STOpS (MGALT-STOpS), which allowed for low thrust trajectories to perform gravity assists. The current state of MGALT-STOpS uses four (4) metaheuristic algorithms in the Genetic Algorithm, Differential Evolution Algorithm, Particle Swarm Optimization Algorithm, and Monotonic Basin Hopping Algorithm.

This work proposes four (4) performance improvements for implementation to the suite: porting MGALT-STOpS from MATLAB to Python 3 and adding components for Thruster Angle Pruning, Cluster Pruning, and Best Leg Tracking. The goal of porting the code into Python was to decrease run time and increase accessibility to the public, as MATLAB requires a paid license to run. Thruster Angle Pruning aimed to decrease the sub-optimal thruster profile that is very common in one of the variable transcription methods. Cluster Pruning was implemented in an attempt to

capture the relationship between the locations of the planets and the optimality of trajectories as seen in high thrust missions. Finally, Best Leg Tracking was created in the hopes of preventing the suite from removing valuable solutions from algorithm populations due to one (or more) poorly optimized transfers.

All proposed performance improvements were verified against the mission from MGALT-STOpS Verification Test Case 2. It was a real-life Earth to Mars to Jupiter trajectory planned to test the viability of using low thrust propulsion and gravity assists as a means to reach the outer planets. All performance improvement methods passed verification as they all offered unique benefits to the user. Furthermore, the performance improvement methods can also be used together at the same time due to their mutually exclusive approaches to finding better trajectories. The final form of this work is a Python 3 suite that optimizes low thrust trajectories with or without gravity assists, along with multiple options to the user regarding improving various aspects of their trajectory search.

ACKNOWLEDGMENTS

Thank you to my advisor, Dr. Kira Abercromby, for her continued support throughout my undergraduate and graduate degrees. Dr. A. inspired me to pursue orbital mechanics and this Master's Degree, and I could not be where I am now without her guidance and assistance. Her passion for Aerospace Engineering and teaching is truly her superpower, and I know that the Cal Poly Aerospace Department will continue to be great as long as she is there.

Thank you to my parents, who push me to do what I love. Their constant support throughout my life has led me to pursue Aerospace Engineering and now trajectory optimization. They even supported my decision to stay in higher education an extra two years, which I see now as a gesture of their unconditional love. I am lucky to be their son and appreciate all the sacrifices they have made along the way.

Thank you to my girlfriend Lauren Lozano for pushing me to always be my best. My biggest achievements in life thus far would not have happened if it were not for her encouraging me to put myself out there. I am grateful for the way she motivates me to continue my work, even when I find it very difficult. She is my best friend and my role model and I hope to be as strong-willed as her one day.

Last but not least, I would like to thank my Siberian Husky, Coach. He truly lives up to his name by keeping me active and constantly reminding me to take breaks and get some fresh air. While we may not speak the same language, we do both understand that a moment not enjoyed is a moment forever lost.

TABLE OF CONTENTS

Page

LIST OF TABLES xii

LIST OF FIGURES xv

CHAPTER

1 Introduction 1

1.1 Problem Statement 1

1.2 Purpose of Study 3

1.3 Current Optimization Programs 3

1.4 Structure of Paper 5

2 MGALT-STOpS Optimization Methodology 7

2.1 Variable Transcription Methods 7

2.1.1 Direct Method 7

2.1.2 Indirect Method 8

2.2 Multiple Forward/Backward Shooting Method 10

2.3 Cost Function 12

3 Optimization Algorithms 14

3.1 Genetic Algorithm 14

3.2 Differential Evolution 15

3.3 Particle Swarm Optimization 16

3.4 Monotonic Basin Hopping 16

3.5 Island Model Paradigm 17

4 Thruster Angle Pruning 19

4.1 Traditional Constraint Pruning 19

4.2 Direct Method 21
4.3 Indirect Method 24
5 Cluster Pruning 25
5.1 Traditional Cluster Pruning 25
5.1.1 Skinny-Dip Algorithm 27
5.1.2 DBSCAN Algorithm 29
6 Multi-Objective Optimization 31
6.1 Premature Pruning 31
6.2 Pareto Fronts 32
6.2.1 Pareto Front Modification 34
6.3 Algorithm Population Substitution 34
7 Verification Approach 36
7.1 Problem 36
7.1.1 Direct Method 37
7.1.2 Indirect Method 38
7.2 Settings 38
7.3 Verification Criteria 40
8 Python Verification 42
8.1 MBH Changes 42
8.2 Direct Method Results 43
8.3 Indirect Method Results 47
8.4 Final Remarks 49
9 Thruster Angle Pruning Verification 51
9.1 Direct Method Results 51
9.2 Indirect Method Results 55

9.3 Final Remarks 56

10 Cluster Pruning Verification 57

10.1 DBSCAN Algorithm Results 57

10.2 Direct Method Results 60

10.2.1 Cluster Pruning with Thruster Angle Pruning 62

10.3 Indirect Method Results 66

10.4 Final Remarks 68

11 Best Leg Tracking Verification 69

11.1 Direct Method Results 69

11.1.1 Best Leg Tracking with Thruster Angle Pruning 73

11.1.2 Best Leg Tracking with Cluster Pruning 76

11.1.3 Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning 79

11.2 Indirect Method Results 83

11.2.1 Best Leg Tracking with Cluster Pruning 86

11.3 Final Remarks 89

12 Conclusion 90

12.1 Closing Thoughts 90

12.2 Recommendations 93

12.2.1 Further Multi-Objective Optimization 93

12.2.2 Additional Cost Function 93

12.2.3 3D Orbit Optimization 94

12.2.4 Gravitational Perturbations 94

12.2.5 Non-Planetary Targets 95

12.2.6 Re-tuning Parameters 95

12.2.7 Coding Improvements 96
12.2.8 Deep Space Flyby 96
12.2.9 Powered Flyby 97
12.2.10 B-Plane Targeting 97
12.2.11 Re-Implementation of Forced Heliocentric Orbits 97
12.2.12 Additional Flyby Requirements 98
BIBLIOGRAPHY 99
APPENDICES
A User Guide 103
A.1 Creating a Mission 103
A.2 Planets, Departure Date, and Times of Flight 104
A.3 Solver Selection 106
A.4 Spacecraft Parameter Selection 107
A.5 Island Model Parameters 108
A.6 Cost Function Parameters 109
A.7 Algorithm Parameters 110
A.7.1 Differential Evolution 110
A.7.2 Genetic Algorithm 112
A.7.3 Particle Swarm Optimization 115
A.7.4 Monotonic Basin Hopping 116
A.8 Performance Improvement Control 118
A.9 Final Form 118
B Additional Verification Results 120
B.1 Python Verification 120
B.1.1 Direct Method 120

B.1.2 Indirect Method 129

B.2 Thruster Angle Pruning Verification 137

B.2.1 Direct Method 137

B.2.2 Indirect Method 146

B.3 Cluster Pruning Verification 146

B.3.1 Direct Method 146

B.3.1.1 Cluster Pruning with Thruster Angle Pruning 157

B.3.2 Indirect Method 168

B.4 Best Leg Tracking Verification 179

B.4.1 Direct Method Results 179

B.4.1.1 Best Leg Tracking with Thruster Angle Pruning . . . 187

B.4.1.2 Best Leg Tracking with Cluster Pruning 196

B.4.1.3 Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning 207

B.4.2 Indirect Method Results 218

B.4.2.1 Best Leg Tracking with Cluster Pruning 226

C MGALT-STOpS Demo 238

LIST OF TABLES

Table		Page
1.1	JPL Low Thrust Trajectory Tools	4
1.2	Availability of JPL Low Thrust Trajectory Tools	4
7.1	Island Model Parameters	39
7.2	Genetic Algorithm Parameters	39
7.3	Differential Evolution Parameters	39
7.4	Particle Swarm Optimization Parameters	40
7.5	Monotonic Basin Hopping Parameters	40
8.1	MGALT-STOpS Direct Method MATLAB/Python Comparison	44
8.2	MGALT-STOpS Indirect Method MATLAB/Python Comparison	47
12.1	Python MGALT-STOpS Direct Method Cost Performance Breakdown	90
12.2	Python MGALT-STOpS Indirect Method Cost Performance Breakdown	91
A.1	Differential Evolution Parameters Guide	112
A.2	Genetic Algorithm Parameters Guide	114
A.3	Particle Swarm Optimization Parameters Guide	116
A.4	Monotonic Basin Hopping Parameters Guide	117
B.1	MGALT-STOpS Direct Method Python Results	120
B.2	MGALT-STOpS Direct Method Python Cost Breakdown	121
B.3	MGALT-STOpS Indirect Method Python Results	129
B.4	MGALT-STOpS Indirect Method Python Cost Breakdown	129

B.5 MGALT-STOpS Direct Method Thruster Pruning Results 137

B.6 MGALT-STOpS Direct Method Thruster Pruning Cost Breakdown 138

B.7 MGALT-STOpS Direct Method Cluster Pruning Results 146

B.8 MGALT-STOpS Direct Method Cluster Pruning Cost Breakdown . 146

B.9 MGALT-STOpS Direct Method Cluster Pruning with Thruster Angle Pruning Results 157

B.10 MGALT-STOpS Direct Method Cluster Pruning with Thruster Angle Pruning Cost Breakdown 157

B.11 MGALT-STOpS Indirect Method Cluster Pruning Results 168

B.12 MGALT-STOpS Indirect Method Cluster Pruning Cost Breakdown 168

B.13 MGALT-STOpS Direct Method Best Leg Tracking Results 179

B.14 MGALT-STOpS Direct Method Best Leg Tracking Cost Breakdown 179

B.15 MGALT-STOpS Direct Method Best Leg Tracking with Thruster Angle Pruning Results 187

B.16 MGALT-STOpS Direct Method Best Leg Tracking with Thruster Angle Pruning Cost Breakdown 188

B.17 MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning Results 196

B.18 MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning Cost Breakdown 196

B.19 MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Results 207

B.20 MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Cost Breakdown 207

B.21 MGALT-STOpS Indirect Method Best Leg Tracking Results 218

B.22 MGALT-STOpS Indirect Method Best Leg Tracking Cost Breakdown 218

B.23 MGALT-STOpS Indirect Method Best Leg Tracking with Cluster Pruning Results 226

B.24 MGALT-STOpS Indirect Method Best Leg Tracking with Cluster Pruning Cost Breakdown 227

LIST OF FIGURES

Figure		Page
2.1	Forward/Backward Shooting Method Example [3]	11
4.1	GALLOP Earth-Mars-Jupiter Optimal Trajectory [4]	21
4.2	MGALT-STOpS Test Case 2 Earth-Mars-Jupiter Trajectory [3]	22
4.3	Direct Method Thruster Angles	23
5.1	2005 Earth-Mars Opportunities Porkchop Plot [5]	26
5.2	Skinny-Dip Algorithm Example Data Set [6]	28
5.3	Histogram and Cumulative Density Function Over X-Axis [6]	28
5.4	Finding Clusters Over Y-Axis [6]	29
6.1	Premature Pruning for Indirect Method on Final Migration	32
6.2	Arbitrary Min-Min Pareto Front [7]	33
8.1	MATLAB MGALT-STOpS Direct Method Final Solution	45
8.2	Python MGALT-STOpS Direct Method Final Solution	45
8.3	MATLAB MGALT-STOpS Indirect Method Final Solution	48
8.4	Python MGALT-STOpS Indirect Method Final Solution	48
9.1	Direct Method Trajectory without Thruster Angle Pruning	52
9.2	Direct Method Thruster Angles without Thruster Angle Pruning	52
9.3	Direct Method Trajectory with Thruster Angle Pruning	53
9.4	Direct Method Thruster Angles with Thruster Angle Pruning	53
10.1	Direct Method DE Cluster Found by DBSCAN	59

10.2 Indirect Method DE Cluster Found by DBSCAN 60

10.3 Direct Method Cluster Found with Cluster Pruning 61

10.4 Direct Method Trajectory with Cluster Pruning 62

10.5 Direct Method Trajectory with Cluster Pruning and Thruster Angle Pruning 64

10.6 Direct Method Cluster Pruning and Thruster Angle Pruning Cluster 64

10.7 Direct Method Cluster Pruning with Thruster Angle Pruning Thrust 65

10.8 Indirect Method Cluster Found with Cluster Pruning 67

10.9 Indirect Method Trajectory with Cluster Pruning 67

11.1 Best Leg Tracking Direct Method Trajectory 70

11.2 Best Leg Tracking Direct Method Thrust 71

11.3 Best Leg Tracking Direct Method Leg Tracking 71

11.4 Direct Method Best Leg Tracking with Thruster Angle Pruning Trajectory 74

11.5 Direct Method Best Leg Tracking with Thruster Angle Pruning Thrust 74

11.6 Direct Method Best Leg Tracking with Thruster Angle Pruning Leg Tracking 75

11.7 Direct Method Trajectory with Best Leg Tracking and Cluster Pruning 77

11.8 Direct Method Best Leg Tracking with Cluster Pruning Thrust . . 77

11.9 Direct Method Clustering with Best Leg Tracking and Cluster Pruning 78

11.10 Direct Method Final Migration Costs with Best Leg Tracking and Cluster Pruning 78

11.11 Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Trajectory 80

11.12 Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Thrust 81

11.13 Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Clustering 81

11.14 Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Final Migration Costs 82

11.15 Indirect Method Final Migration Costs without Best Leg Tracking . 83

11.16 Indirect Method Final Migration Costs with Best Leg Tracking . . 84

11.17 Indirect Method Trajectory with Best Leg Tracking 84

11.18 Indirect Method Trajectory with Best Leg Tracking and Cluster Pruning 87

11.19 Indirect Method Clustering with Best Leg Tracking and Cluster Pruning 87

11.20 Indirect Method Final Migration Costs with Best Leg Tracking and Cluster Pruning 88

A.1 Creating a Mission 103

A.2 Setting Bodies and Times of Flight 105

A.3 Setting the Solver 106

A.4 Setting the Spacecraft Parameters 107

A.5 Setting the Island Model Parameters 108

A.6 Setting the Cost Function Parameters 109

A.7 Setting the Algorithm Parameters 110

A.8 Setting the Performance Improvements 118

A.9 Custom Mission Final Form 119

B.1 Python MGALT-STOpS Direct Method Optimal Solution 1 Trajectory 121

B.2 Python MGALT-STOpS Direct Method Optimal Solution 1 Thrust 122

B.3 Python MGALT-STOpS Direct Method Optimal Solution 1 Leg Tracking 122

B.4 Python MGALT-STOpS Direct Method Optimal Solution 2 Trajectory 123

B.5 Python MGALT-STOpS Direct Method Optimal Solution 2 Thrust 123

B.6 Python MGALT-STOpS Direct Method Optimal Solution 2 Leg Tracking 124

B.7 Python MGALT-STOpS Direct Method Optimal Solution 3 Trajectory 124

B.8 Python MGALT-STOpS Direct Method Optimal Solution 3 Thrust 125

B.9 Python MGALT-STOpS Direct Method Optimal Solution 3 Leg Tracking 125

B.10 Python MGALT-STOpS Direct Method Optimal Solution 4 Trajectory 126

B.11 Python MGALT-STOpS Direct Method Optimal Solution 4 Thrust 126

B.12 Python MGALT-STOpS Direct Method Optimal Solution 4 Leg Tracking 127

B.13 Python MGALT-STOpS Direct Method Optimal Solution 5 Trajectory 127

B.14 Python MGALT-STOpS Direct Method Optimal Solution 5 Thrust 128

B.15 Python MGALT-STOpS Direct Method Optimal Solution 5 Leg Tracking 128

B.16 Python MGALT-STOpS Indirect Method Optimal Solution 1 Trajectory 130

B.17 Python MGALT-STOpS Indirect Method Optimal Solution 1 Thrust 130

B.18 Python MGALT-STOpS Indirect Method Optimal Solution 1 Leg Tracking 131

B.19 Python MGALT-STOpS Indirect Method Optimal Solution 2 Trajectory 131

B.20 Python MGALT-STOpS Indirect Method Optimal Solution 2 Thrust 132

B.21 Python MGALT-STOpS Indirect Method Optimal Solution 2 Leg Tracking 132

B.22 Python MGALT-STOpS Indirect Method Optimal Solution 3 Trajectory 133

B.23 Python MGALT-STOpS Indirect Method Optimal Solution 3 Thrust 133

B.24 Python MGALT-STOpS Indirect Method Optimal Solution 3 Leg Tracking 134

B.25 Python MGALT-STOpS Indirect Method Optimal Solution 4 Trajectory 134

B.26 Python MGALT-STOpS Indirect Method Optimal Solution 4 Thrust 135

B.27 Python MGALT-STOpS Indirect Method Optimal Solution 4 Leg Tracking 135

B.28 Python MGALT-STOpS Indirect Method Optimal Solution 5 Trajectory 136

B.29 Python MGALT-STOpS Indirect Method Optimal Solution 5 Thrust 136

B.30 Python MGALT-STOpS Indirect Method Optimal Solution 5 Leg Tracking 137

B.31 Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory 138

B.32 Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust 139

B.33 Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking 139

B.34 Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory 140

B.35 Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust 140

B.36 Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking 141

B.37 Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory 141

B.38 Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust 142

B.39 Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking 142

B.40 Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory 143

B.41 Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust 143

B.42 Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking 144

B.43 Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory 144

B.44 Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust 145

B.45 Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking 145

B.46 Cluster Pruning Direct Method Optimal Solution 1 Trajectory . . . 147

B.47 Cluster Pruning Direct Method Optimal Solution 1 Clustering . . . 147

B.48 Cluster Pruning Direct Method Optimal Solution 1 Thrust 148

B.49 Cluster Pruning Direct Method Optimal Solution 1 Leg Tracking . 148

B.50 Cluster Pruning Direct Method Optimal Solution 2 Trajectory . . . 149

B.51 Cluster Pruning Direct Method Optimal Solution 2 Clustering . . . 149

B.52 Cluster Pruning Direct Method Optimal Solution 2 Thrust 150

B.53 Cluster Pruning Direct Method Optimal Solution 2 Leg Tracking . 150

B.54 Cluster Pruning Direct Method Optimal Solution 3 Trajectory . . . 151

B.55 Cluster Pruning Direct Method Optimal Solution 3 Clustering . . . 151

B.56 Cluster Pruning Direct Method Optimal Solution 3 Thrust 152

B.57 Cluster Pruning Direct Method Optimal Solution 3 Leg Tracking . 152

B.58 Cluster Pruning Direct Method Optimal Solution 4 Trajectory . . . 153

B.59 Cluster Pruning Direct Method Optimal Solution 4 Clustering . . . 153

B.60 Cluster Pruning Direct Method Optimal Solution 4 Thrust 154

B.61 Cluster Pruning Direct Method Optimal Solution 4 Leg Tracking . 154

B.62 Cluster Pruning Direct Method Optimal Solution 5 Trajectory . . . 155

B.63 Cluster Pruning Direct Method Optimal Solution 5 Clustering . . . 155

B.64 Cluster Pruning Direct Method Optimal Solution 5 Thrust 156

B.65 Cluster Pruning Direct Method Optimal Solution 5 Leg Tracking . 156

B.66 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory 158

B.67 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Clustering 158

B.68 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust 159

B.69 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking 159

B.70 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory 160

B.71 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Clustering 160

B.72 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust 161

B.73 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking 161

B.74 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory 162

B.75 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Clustering 162

B.76 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust 163

B.77 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking 163

B.78 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory 164

B.79 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Clustering 164

B.80 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust 165

B.81 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking 165

B.82 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory 166

B.83 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Clustering 166

B.84 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust 167

B.85 Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking 167

B.86 Cluster Pruning Indirect Method Optimal Solution 1 Trajectory . . 169

B.87 Cluster Pruning Indirect Method Optimal Solution 1 Clustering . . 169

B.88 Cluster Pruning Indirect Method Optimal Solution 1 Thrust 170

B.89 Cluster Pruning Indirect Method Optimal Solution 1 Leg Tracking 170

B.90 Cluster Pruning Indirect Method Optimal Solution 2 Trajectory . . 171

B.91 Cluster Pruning Indirect Method Optimal Solution 2 Clustering . . 171

B.92 Cluster Pruning Indirect Method Optimal Solution 2 Thrust 172

B.93 Cluster Pruning Indirect Method Optimal Solution 2 Leg Tracking 172

B.94 Cluster Pruning Indirect Method Optimal Solution 3 Trajectory . . 173

B.95 Cluster Pruning Indirect Method Optimal Solution 3 Clustering . . 173

B.96 Cluster Pruning Indirect Method Optimal Solution 3 Thrust 174

B.97 Cluster Pruning Indirect Method Optimal Solution 3 Leg Tracking 174

B.98 Cluster Pruning Indirect Method Optimal Solution 4 Trajectory . . 175

B.99 Cluster Pruning Indirect Method Optimal Solution 4 Clustering . . 175

B.100 Cluster Pruning Indirect Method Optimal Solution 4 Thrust 176

B.101 Cluster Pruning Indirect Method Optimal Solution 4 Leg Tracking 176

B.102 Cluster Pruning Indirect Method Optimal Solution 5 Trajectory . . 177

B.103 Cluster Pruning Indirect Method Optimal Solution 5 Clustering . . 177

B.104 Cluster Pruning Indirect Method Optimal Solution 5 Thrust 178

B.105 Cluster Pruning Indirect Method Optimal Solution 5 Leg Tracking 178

B.106 Best Leg Tracking Direct Method Optimal Solution 1 Trajectory . . 180

B.107 Best Leg Tracking Direct Method Optimal Solution 1 Thrust 180

B.108 Best Leg Tracking Direct Method Optimal Solution 1 Leg Tracking 181

B.109 Best Leg Tracking Direct Method Optimal Solution 2 Trajectory . . 181

B.110 Best Leg Tracking Direct Method Optimal Solution 2 Thrust 182

B.111 Best Leg Tracking Direct Method Optimal Solution 2 Leg Tracking 182

B.112 Best Leg Tracking Direct Method Optimal Solution 3 Trajectory . . 183

B.113 Best Leg Tracking Direct Method Optimal Solution 3 Thrust 183

B.114 Best Leg Tracking Direct Method Optimal Solution 3 Leg Tracking 184

B.115 Best Leg Tracking Direct Method Optimal Solution 4 Trajectory . . 184

B.116 Best Leg Tracking Direct Method Optimal Solution 4 Thrust 185

B.117 Best Leg Tracking Direct Method Optimal Solution 4 Leg Tracking 185

B.118 Best Leg Tracking Direct Method Optimal Solution 5 Trajectory . . 186

B.119 Best Leg Tracking Direct Method Optimal Solution 5 Thrust 186

B.120 Best Leg Tracking Direct Method Optimal Solution 5 Leg Tracking 187

B.121 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory 188

B.122 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust 189

B.123 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking 189

B.124 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory 190

B.125 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust 190

B.126 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking 191

B.127 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory 191

B.128 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust 192

B.129 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking 192

B.130 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory 193

B.131 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust 193

B.132 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking 194

B.133 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory 194

B.134 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust 195

B.135 Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking 195

B.136 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Trajectory 197

B.137 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Clustering 197

B.138 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Thrust 198

B.139 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Leg Tracking 198

B.140 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Trajectory 199

B.141 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Clustering 199

B.142 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Thrust 200

B.143 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Leg Tracking 200

B.144 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Trajectory 201

B.145 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Clustering 201

B.146 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Thrust 202

B.147 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Leg Tracking 202

B.148 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Trajectory 203

B.149 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Clustering 203

B.150 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Thrust 204

B.151 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Leg Tracking 204

B.152 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Trajectory 205

B.153 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Clustering 205

B.154 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Thrust 206

B.155 Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Leg Tracking 206

B.156 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory 208

B.157 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Clustering 208

B.158 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust 209

B.159 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking 209

B.160 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory 210

B.161 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Clustering 210

B.162 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust 211

B.163 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking 211

B.164 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory 212

B.165 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Clustering 212

B.166 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust 213

B.167 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking 213

B.168 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory 214

B.169 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Clustering 214

B.170 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust 215

B.171 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking 215

B.172 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory 216

B.173 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Clustering 216

B.174 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust 217

B.175 Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking 217

B.176 Best Leg Tracking Indirect Method Optimal Solution 1 Trajectory . 219

B.177 Best Leg Tracking Indirect Method Optimal Solution 1 Thrust . . . 219

B.178 Best Leg Tracking Indirect Method Optimal Solution 1 Leg Tracking 220

B.179 Best Leg Tracking Indirect Method Optimal Solution 2 Trajectory . 220

B.180 Best Leg Tracking Indirect Method Optimal Solution 2 Thrust . . . 221

B.181 Best Leg Tracking Indirect Method Optimal Solution 2 Leg Tracking 221

B.182 Best Leg Tracking Indirect Method Optimal Solution 3 Trajectory . 222

B.183 Best Leg Tracking Indirect Method Optimal Solution 3 Thrust . . . 222

B.184 Best Leg Tracking Indirect Method Optimal Solution 3 Leg Tracking 223

B.185 Best Leg Tracking Indirect Method Optimal Solution 4 Trajectory . 223

B.186 Best Leg Tracking Indirect Method Optimal Solution 4 Thrust . . . 224

B.187 Best Leg Tracking Indirect Method Optimal Solution 4 Leg Tracking 224

B.188 Best Leg Tracking Indirect Method Optimal Solution 5 Trajectory . 225

B.189 Best Leg Tracking Indirect Method Optimal Solution 5 Thrust . . . 225

B.190 Best Leg Tracking Indirect Method Optimal Solution 5 Leg Tracking 226

B.191 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Trajectory 227

B.192 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Clustering 228

B.193 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Thrust 228

B.194 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Leg Tracking 229

B.195 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Trajectory 229

B.196 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Clustering 230

B.197 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Thrust 230

B.198 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Leg Tracking 231

B.199 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Trajectory 231

B.200 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Clustering 232

B.201 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Thrust 232

B.202 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Leg Tracking 233

B.203 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Trajectory 233

B.204 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Clustering 234

B.205 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Thrust 234

B.206 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Leg Tracking 235

B.207 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Trajectory 235

B.208 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Clustering 236

B.209 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Thrust 236

B.210 Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Leg Tracking 237

C.1 Demo: Sample Trajectory with Indirect Method 239

C.2 Demo: Sample Thruster Profile with Indirect Method 239

Chapter 1

INTRODUCTION

Spacecraft trajectory optimization has been a topic of major interest ever since humans began sending objects into space. Flying a spacecraft on a trajectory that minimizes flight time and fuel usage (along with any other mission objectives) means that missions can be flown as cheaply and effectively as possible. The first trajectory optimization began with chemical propellant, or impulsive thrust, but with the advent of electrical propulsion and other "low thrust" propulsive techniques, new methods were derived and created to facilitate trajectory optimization.

1.1 Problem Statement

The topic of trajectory optimization is often broken into two sub-problems: High Thrust and Low Thrust. High Thrust, or "impulsive", trajectories use powerful chemical propellant that burns for a very short amount of time when compared to the time of flight of a trajectory. Oftentimes due to the large mass of chemical propellant, the optimal trajectory is seen as the one that requires the least ΔV, or change in velocity. For trajectories between two planets, this optimization is relatively easy. Using the Lambert's Method [8], one can brute force solve all potential solutions over large windows of departure and arrival dates for optimal trajectories quite quickly. This is due to the novelty of Lambert's Method, which allows the user to find trajectory ΔV's without needing to propagate the spacecraft through its mission. In the case of trajectories that include gravity assists, however, finding solutions that are optimal in

all legs is much more difficult and time consuming because of the significantly larger search space.

In response to the difficulty of impulsive trajectory optimization with gravity assists, Tim Fitzgerald created the Spacecraft Trajectory Optimization Suite (STOpS) [1] which uses a different approach than brute force optimization. He used the same Lambert's Method [8] along with some heuristic algorithms to find the optimal trajectories for user-defined missions. The heuristic algorithms, most of which will be described in Chapter 3, allowed STOpS to systematically search the Departure Date and Times of Flight windows for trajectories that required the lowest ΔV. This tool proved successful, eventually finding trajectories with even lower ΔV than the original test cases, showing that using heuristic algorithms is a viable option for impulsive thrust trajectory optimization.

The Low Thrust sub-problem of trajectory optimization is much more difficult. Because of the long duration of thrust over the course of a mission (often 80% or more of the time of flight), a spacecraft's burns can no longer be deemed "impulsive." Therefore, engineers and researchers must propagate a spacecraft continuously with a thrust magnitude and thruster angle in order to accurately define its trajectory. This is much more computationally expensive and requires new methods of variable transcription that will be covered in Section 2.1.

As an extension of Fitzgerald's STOpS [1], Shane Sheehan wrote Low Thrust-STOpS (LT-STOpS) [2] using the existing framework and adapting it to accommodate low thrust variable strings. However, while this work achieved its goal of optimizing trajectories for single-leg low thrust missions, it did not have the capability of optimizing low thrust trajectories with gravity assists. Griff Malloy [3] followed suit and adapted LT-STOpS to include gravity assists with Multiple Gravity Assist Low Thrust STOpS (MGALT-STOpS). This work aims to improve the performance of

MGALT-STOpS, namely the run time, final solution optimality and attitude control requirements, and prevention of premature discarding of promising solutions.

1.2 Purpose of Study

MGALT trajectories are becoming an integral part of scientific research missions of our solar system. With the current development of more powerful and more efficient electric propulsion systems along with the inherent wet mass reduction, low thrust options have now become viable for most robotic missions touring the planets. Given the previous work by Fitzgerald [1], Sheehan [2], and Malloy [3], MGALT-STOpS is nearing its potential as a preliminary low thrust mission design tool. This work aims to improve upon the foundation of MGALT-STOpS, introducing techniques to systematically reduce the search spaces and improve the final solutions.

1.3 Current Optimization Programs

STOpS is not the first program made to optimize either high or low thrust trajectories. In fact, there are many suites in existence today that are used daily by industry professionals. Unfortunately, not many trajectory optimization suites are freely available to the public. NASA JPL, for example, has compiled many such programs over the years, all with different strengths, approaches, and fidelities. A list of said programs and their fidelities is available in Table 1.1 [9, 2]. Table 1.2 [9, 2] shows the availability of each program as well.

As is evident by the tables, there are not many options for members of the public who want to practice trajectory optimization. While technically all of the suites (except Mystic) may be available to students depending on their school, only

Table 1.1: JPL Low Thrust Trajectory Tools

Name	Description	Fidelity
MALTO	Mission Analysis Low-Thrust Optimization	Medium
Mystic	Optimization of trajectory or entire mission	High
Copernicus	Generalized spacecraft trajectory design and optimization system	High
OTIS	Optimal Trajectory by Implicit Simulation	High
SNAP	Spacecraft N-body Analysis Program	High
CHEBYTOP	Chebyshev (Polynomial) Trajectory Optimization Program	Low
VARITOP	Variational Calculus Trajectory Optimization Program	Medium
SEPTOP	VARITOP based Solar Electric Propulsion Trajectory Optimization Program	Medium
NEWSEP	(new) VARITOP based Solar Electric Propulsion Trajectory Optimization Program	Medium
Sail	VARITOP customized for solar sails	Medium

Table 1.2: Availability of JPL Low Thrust Trajectory Tools

Name	Availability
MALTO	Freely available to NASA contractors, civil service, and academia. Commercial license available for a fee
Mystic	NASA employees only
Copernicus	NASA center, government contractors, and universities with contractual affiliations with NASA
OTIS	Anybody in government, academia, and industry. Subject to export control regulations
SNAP	Anybody in government, academia, and industry. Subject to export control regulations
CHEBYTOP	General public
VARITOP	NASA employees only, and universities with contractual affiliations to NASA
SEPTOP	NASA employees only, and universities with contractual affiliations to NASA
NEWSEP	NASA employees only, and universities with contractual affiliations to NASA
Sail	NASA employees only, and universities with contractual affiliations to NASA

CHEBYTOP is freely available to the general public. However, it is worth noting that CHEBYTOP runs on Microsoft Excel and FORTRAN and was developed over 50 years ago. There are certainly more recent and user friendly tools that should be available to the public.

One of the goals of this work was to implement techniques used in the previously mentioned programs such that computation time is saved and that potentially optimal solutions are kept throughout the optimization process. Unfortunately, due to the proprietary nature of all of the programs mentioned previously, [9] is one of the only publicly available documents outlining the general process that each of the suites undergoes. At best, these descriptions are surface level and do not go into any real detail besides a three or four sentence summary. Therefore, this work will not be able to use performance-improving techniques based on what is known to be used in JPL programs.

1.4 Structure of Paper

The background information section of this work starts in Chapter 2 where important information regarding the inner working of MGALT-STOpS optimization is described. This includes the variable transcription methods, the Multiple Forward/Backward Shooting Method, and the cost function. Followed next in Chapter 3 is a brief description of the metaheuristic algorithms used in the suite. Chapter 4 then dives into the idea of constraint pruning, specifically how it is commonly used and why a version of it should be implemented in MGALT-STOpS. Chapter 5 describes how clustering is observed in high thrust trajectories and how MGALT-STOpS can use clustering to reduce its search space. The last background section, Chapter 6, contains a brief summary of Multi-Objective Optimization and how its principles can

be leveraged to keep track of promising solutions that might otherwise be removed from algorithm populations.

The verification of the implemented methods begins in Chapter 7 with a description of the test case used for said verification. Chapter 8 covers the re-verification of the suite after it was ported from MATLAB to Python and Chapter 9 contains the verification process of Thruster Angle Pruning. Chapter 10 then verifies Cluster Pruning along with using both Cluster Pruning and Thruster Angle Pruning at the same time. Chapter 11 then follows suit with the verification process of Best Leg Tracking and its potential combinations with Cluster Pruning and Thruster Angle Pruning.

Chapter 12 contains the conclusion and final remarks along with the recommendations for future work. Appendix A is an in depth user guide for the suite in Python. Appendix B contains all of the figures and important data for the runs used in this work's verification processes. The author would like to note that Appendix B accounts for most of the length of this document. Finally, Appendix C walks the potential user through a brief demo of the suite.

Chapter 2

MGALT-STOPS OPTIMIZATION METHODOLOGY

This chapter will discuss the way that MGALT-STOpS creates, propagates, and scores trajectories. Such background is necessary to understand how the performance improvements proposed in this work operate and are evaluated.

2.1 Variable Transcription Methods

In order to represent a potential trajectory, MGALT-STOpS must create variable strings that can be propagated and optimized. MGALT-STOpS uses the same variable transcription methods utilized by LT-STOpS in the Direct Method [10] and the Indirect Method [11]. These methods are also sometimes referred to as the Segmented Method and Costate Method, respectively. The interested reader is encouraged to read the LT-STOpS [2] and MGALT-STOpS [3] theses for more information.

2.1.1 Direct Method

The Direct Method makes the valid assumption that the change in thrust (T) and thruster angle (ϕ) is slow, allowing for the trajectory array to be simplified into an array of N segments where each segment corresponds to 1/Nth of the transfer's time of flight. For each segment, a corresponding constant thrust and thruster angle are produced, resulting in a variable array for each transfer given by the form in Eq. (2.1).

$$[t_0 \ T_1 \ \phi_1 \ T_2 \ \phi_2 \ \ ... \ \ T_n \ \phi_n \ ToF] \tag{2.1}$$

where t_0 is the Departure Date and time, T_n is the Nth segment's thrust, ϕ_n is the Nth segment's thruster angle, and ToF is the trajectory's Time of Flight.

The metaheuristic algorithms that will be discussed in Chapter 3 have access to all the variables in said array. One of the strengths of this method is that is can change the amount of thrust for a given segment, potentially allowing it to coast for long periods of time. However, with the ability to optimize the entire array comes the issue of a large search space. Because each segment has two associated variables, the number of search space dimensions scale roughly by 2N. For a single-leg transfer with 30 segments, the search space is already 62 dimensions.

2.1.2 Indirect Method

The Indirect Method employs a very different technique from that of the Direct Method. Instead of reducing the trajectory into segments, it keeps the thruster angle continuous and fixes thrust. Additionally, instead of using a potentially large trajectory array, the Indirect Method always has five variables for a single transfer trajectory. For a detailed derivation, the interested reader is encouraged to review "Spacecraft Trajectory Optimization" [11]. Through said derivation, the variable array takes the form given in Eq. (2.2).

$$[t_0 \;\; \lambda_1 \;\; \lambda_2 \;\; \lambda_3 \;\; ToF] \tag{2.2}$$

where the λ_n's are costate variables representing the thruster angle throughout the trajectory. This method [11] also uses equations of motion, shown in Eqs. (2.3-2.5).

$$\dot{\lambda}_1 = -\lambda_3 + \frac{V\lambda_2}{R} \tag{2.3}$$

$$\dot{\lambda}_2 = \frac{-2V\lambda_1 + U\lambda_2}{R} \tag{2.4}$$

$$\dot{\lambda}_3 = \frac{V^2\lambda_1 - UV\lambda_2}{R^2} - \frac{2\mu\lambda_1}{R^3} \tag{2.5}$$

where the $\dot{\lambda}_n$'s are the rate of change for their respective λ_n, R is the instantaneous distance from the Sun to the spacecraft, U is the instantaneous velocity of the spacecraft in the radial direction, V is the instantaneous tangential velocity of the spacecraft, and μ is the Gravitational Parameter of the Sun, or $132,712,428,000$ km^3s^{-2} [12].

The thruster pointing angle is then given by Eqs. (2.6-2.7) [11].

$$\cos\phi = -\frac{\lambda_2}{\sqrt{\lambda_1^2 + \lambda_2^2}} \tag{2.6}$$

$$\sin\phi = -\frac{\lambda_1}{\sqrt{\lambda_1^2 + \lambda_2^2}} \tag{2.7}$$

where ϕ is the instantaneous thruster angle.

And lastly, a "free-time" constraint must be added [11], written in Eq. (2.8). This constraint allows the Indirect Method to not be forced to find trajectories that arrive at a planet at a specific time. As a note, MGALT-STOpS does constrain the Times of Flight for a trajectory but is able to work around this constraint with the Multiple Forward/Backward Shooting Method as explained in Section 2.2 of this chapter.

$$H(t_f) + \frac{\partial\Phi}{\partial t_f} = 0 \xrightarrow{\text{yields}} \frac{T}{m}\sqrt{\lambda_1 t_f^2 + \lambda_2 t_f^2} - 1 = 0 \tag{2.8}$$

The state vector is then adjusted from a traditional 2D polar form [11] to look like Eq. (2.9).

$$X = [\lambda_1 \ \lambda_2 \ \lambda_3 \ R \ \theta \ U \ V \ m]^t \tag{2.9}$$

While the derivation and mathematics behind the Indirect Method is much more involved than the Direct Method, it comes with the benefit of a much smaller search space. Since a single-leg trajectory can now be modeled using only five (5) variables, the search space is only five (5) dimensions. This means that the suite has an easier time optimizing the Indirect Method. On the downside, this method is constrained to using constant thrust. Therefore, Indirect Method trajectories will often have a much higher propellant mass fraction than those of the Direct Method.

2.2 Multiple Forward/Backward Shooting Method

LT-STOpS propagated spacecrafts using the Forward Shooting Method. This method is often used when modeling spacecraft trajectories over time, as one starts with initial conditions and simply propagates forward in time. However, when converting to MGALT trajectories, Malloy [3] quickly discovered it was quite rare that a spacecraft successfully arrived at a flyby planet, let alone its arrival planet. This is because not all possible variable combinations in a low thrust variable string produce valid trajectories. In order to solve this issue, he implemented the Forward/Backward Shooting Method which instead propagates spacecrafts forward from a departure planet and backwards from an arrival planet, each for half of the transfer's time of flight. This ensured that, at a bare minimum, spacecrafts were meeting the required boundary conditions necessary for an MGALT trajectory. The final implementation of the Multiple Forward/Backward Shooting Method (MFBSM) simply added the capability to perform multiple transfers by tracking previous times of flight such that the next transfer starts/ends at the right times and locations. A visual example created and used by Malloy [3] is shown in Figure 2.1.

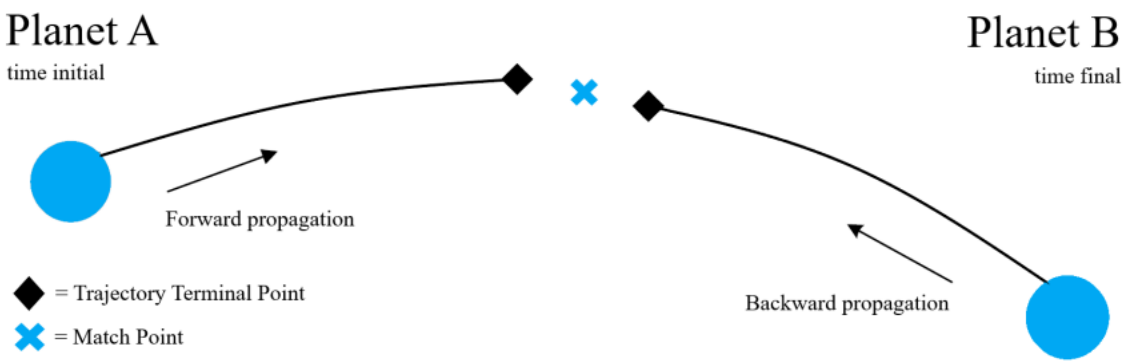

Figure 2.1: Forward/Backward Shooting Method Example [3]. The spacecraft is propagated forward and backward for half the time of flight each, and optimal trajectories are continuous across the match point.

In Sections 2.1.1 and 2.1.2, the variable arrays were constructed specifically for Forward Shooting, single-leg trajectories. In order to modify the arrays for the MF-BSM, they must be adapted to represent both the Forward/Backward approach and an MGALT trajectory. For an Earth-Mars-Jupiter trajectory, the overall format of a variable array, regardless of the transcription method, is given by Eq. (2.10).

$$\text{member} = [\text{Departure Flyby Arrival}] \tag{2.10}$$

where for the Indirect Method specifically:

$$\text{Departure} = [\text{JD}_{\text{Dep}} \;\; \lambda_{1f} \;\; \lambda_{2f} \;\; \lambda_{3f}] \tag{2.11}$$

$$\text{Arrival} = [\lambda_{1b} \;\; \lambda_{2b} \;\; \lambda_{3b} \;\; \text{ToF}] \tag{2.12}$$

$$\text{Flyby} = [\lambda_{1b} \;\; \lambda_{2b} \;\; \lambda_{3b} \;\; C_1 \;\; C_2 \;\; C_3 \;\; \text{ToF} \;\; \text{JD}_{\text{Flyby}} \;\; \lambda_{1f} \;\; \lambda_{2f} \;\; \lambda_{3f}] \tag{2.13}$$

For Eqs. (2.11-2.13), note that the departure string of the member is only represented by the Julian date of the departure and the forward section of the first transfer. Then, the flyby contains the backward section of the first transfer, coefficients for the gravity assist, the leg Time of Flight, and the forward section of the second transfer. Finally, the arrival string of the member mirrors the departure string in that it contains the backward section and arrival date of the second transfer. For missions containing multiple gravity assists, there will be multiple flyby strings in the trajectory array.

The Direct Method is modified similarly. The only difference is that the λ's are replaced by the N thrust and thruster angle variables. For a two-legged transfer trajectory with 30 segments each, this results in a variable array with 127 components.

2.3 Cost Function

A cost function is what tells MGALT-STOpS how optimal a spacecraft's trajectory is. Due to the nature of the problem, Multiple Forward/Backward Shooting Method, and meta-heuristic algorithms, the cost function for MGALT-STOpS is represented by how closely the parameters of the endpoints of the trajectories match (Note that due to the MFBSM, the "endpoints" could also be seen as "midpoints"). For a given trajectory, a spacecraft's endpoint radial and angular positions as well as its endpoint radial and tangential velocities are evaluated. There is also a term that evaluates how closely the times of flight match the user's desired times of flight. These cost function parameters are shown in Eqs. (2.14-2.19).

$$J_R = \frac{(R_f - R_b)^2}{\text{tol}_R^2} \tag{2.14}$$

$$J_\theta = \frac{(\theta_f - \theta_b)^2}{\text{tol}_\theta^2} \tag{2.15}$$

$$J_U = \frac{(U_f - U_b)^2}{\text{tol}_U^2} \tag{2.16}$$

$$J_V = \frac{(V_f - V_b)^2}{\text{tol}_V^2} \tag{2.17}$$

$$J_{tt} = \frac{-(\text{tt}_{\text{alg}} - \text{tt}_{\text{desired}}) \mid \text{tt}_{\text{alg}} - \text{tt}_{\text{desired}} \mid}{w_{tt}^2} \tag{2.18}$$

$$J = J_R + J_\theta + J_U + J_V + J_{tt} \tag{2.19}$$

where the "f" subscript means forward, the "b" subscript means backward, tt_{alg} refers to a solution's Time of Flight, and $\text{tt}_{\text{desired}}$ refers to the user's desired Time of Flight.

It is important to note that a perfectly optimized trajectory will have a cost of 0. However, it is not expected that MGALT-STOpS will find 0 cost solutions due to the complex search space. Instead, well optimized trajectories, particularly for the test case described in Chapter 7, are expected to have costs that range from 100 to 7,000, where single leg costs can be as low as around 10 or as high as 5,500. These might sound like high costs initially, but it is important to note that poorly optimized trajectories throughout the optimization process are often seen with costs around 10,000,000 - 100,000,000. Additionally, it is not until costs start to rise above 10,000 that the discontinuities in the MFBSM are noticeable to the user. Therefore, as explained in Chapter 7, final trajectories are only considered valid if they have costs below 10,000.

Chapter 3

OPTIMIZATION ALGORITHMS

MGALT-STOpS uses four (4) meta-heuristic algorithms in the Genetic Algorithm (GA), Differential Evolution Algorithm (DE), Particle Swarm Optimization Algorithm (PSO), and Monotonic Basin Hopping Algorithm (MBH). These algorithms use their individual techniques to evaluate the trajectory search space and try to find the optimal solution. Because these algorithms have all been previously used in MGALT-STOpS, they will not be covered in length. The interested reader is encouraged to visit the previous works of Fitzgerald [1], Sheehan [2], and Malloy [3] for a more detailed description.

3.1 Genetic Algorithm

The Genetic Algorithm is considered a global optimization method, meaning that it is designed to get close to or exactly find the most optimal solution in the entire search space. Fitzgerald [1] included the GA in the original STOpS and modeled the method after David E. Goldberg's "Genetic Algorithms in Search, Optimization, & Machine Learning" [13] and "Practical Genetic Algorithms" by Randy L. Haupt and Sue Ellen Haupt [14]. As GA's are traditionally designed to mimic Darwin's "Survival of the Fittest" theory, there are different selection methods (which solutions survive), mating methods (how solutions combine to produce children), and mutation methods (how children are randomly altered to better explore the search space). Sheehan [2] then adapted the GA to work with low thrust trajectories for LT-STOpS, which mainly involved allowing the algorithm to handle low thrust variable arrays. Finally,

Malloy [3] was able to further modify the variable array handling to include MGALT trajectories.

3.2 Differential Evolution

The Differential Evolution Algorithm was adapted from "Differential Evolution: A Practical Approach to Global Optimization" by Kenneth Price, Rainer M. Storm, and Jouni A. Lampinen [15]. This algorithm is similar to the GA in how it generates the initial population and selects which solutions survive. However, the main difference is how the solutions evolve over generations. In the DE, there is no mating; instead, solutions are altered by a mutant vector, $\overrightarrow{V}$. There are three generation methods for selecting $\overrightarrow{V}$. The first is random. The second is "best so far," where the base vector, $\overrightarrow{x_{r0}}$, is a vector directing to the best-so-far solution multiplied by some scaling factor between 0 and 1. The third is a blend, where some randomness is thrown into the base vector originally pointing to the best-so-far solution. This can be seen in the Eq. (3.1), where $\overrightarrow{x_{r1}}$ and $\overrightarrow{x_{r2}}$ represent the difference vector that acts similar to a random mutator and where F is the scaling factor between 0 and 1.

$$\overrightarrow{V_i} = \overrightarrow{x_{r0}} - F(\overrightarrow{x_{r1}} - \overrightarrow{x_{r2}}) \tag{3.1}$$

The application of the mutant vector allows the Differential Evolution algorithm to span large areas of the search space while also having a tendency to trend towards known "good" solutions. This tendency results in a behavior similar to a local search algorithm towards the end of optimization where promising areas of the search space are explored with a high density of solutions.

As with the GA, Fitzgerald [1] used the DE in his STOpS and Sheehan [2] was able to adapt it to accept low thrust variable arrays. Malloy [3] then followed by adding the capability for MGALT variable arrays.

3.3 Particle Swarm Optimization

The Particle Swarm Optimization Algorithm is based off of "Particle Swarm Optimization" by Maurice Clerc [16]. This algorithm initially starts off with a randomly generated population with randomly generated "velocity" vectors. Interestingly, instead of populations of solutions being generated after every member has been evaluated, the PSO propagates its original members around the search space over "time" with the "velocity" vector mentioned earlier. Throughout optimization, each population member keeps track of its own best-so-far solution and the best-so-far solution shared to it by an informant (population members that can share potential solutions). This memory alters the population's velocities and with some scaling factors called c_1 and c_{max}, it eventually converges on a global optimum. This algorithm is inspired by the ways the bees find sources of nectar, and mimics the way they communicate and swarm.

Once again, Fitzgerald [1] originally used this algorithm for STOpS, Sheehan [2] adapted it for low thrust variable arrays, and Malloy [3] adapted it once again to allow for MGALT variable arrays.

3.4 Monotonic Basin Hopping

The Monotonic Basin Hopping Algorithm is the only meta-heuristic algorithm used by MGALT-STOpS that is new to STOpS as a whole. Malloy [3] included this

algorithm because of "the principle that many real-world problems have a structure where individual local optima, or 'basins,' tend to cluster together into 'funnels', where one local optima is better than the rest" [17]. He decided that this algorithm will act alongside the GA, DE, and PSO algorithms.

As described by Malloy [3], the MBH algorithm operates by generating a random population and evaluating one at a time. If the algorithm evaluates a population member and returns a solution that satisfies a feasibility constraint, that original member will be slightly perturbed and run again. If that perturbed member's solution is not better than the original member's, a counter is incremented. If the perturbed member's solution is better than the original member's, the perturbed member is then perturbed again and the counter is set back to 0. This continues until the algorithm cannot find a better solution within a user-defined N counts. That final member is then archived and the algorithm continues on with the initial randomly generated population.

The term "Basin Hopping" refers to the ability of the MBH to move from one local optima to another. Due to the fact that there are many small basins located near one another, the MBH algorithm allows MGALT-STOpS to jump from one to another by injecting a small amount of randomness. This ensures that the solutions found are not only a local optima, but are the true optima of the local neighborhood.

3.5 Island Model Paradigm

All four of the algorithms used in MGALT-STOpS have their strengths and weaknesses. Used in isolation, they might eventually find themselves at the global optima. However, the novelty of the Island Model Paradigm ensures that the best solutions found throughout optimization always get more optimal.

The Island Model Paradigm treats each algorithm as if were its own island in an Archipelago. As the algorithms run, they all try to find the optimal solutions separately. Once they all have finished, the islands then share their best solutions according to a user-defined connectivity matrix. This process is called a migration. Then the process is continued for a user-defined number of migrations. Through testing, Malloy [3] found the Direct Method and Indirect Method provide acceptable results after 3 and 4 migrations, respectively.

Chapter 4

THRUSTER ANGLE PRUNING

This chapter and Chapter 5 will focus on the idea of "pruning." Pruning is the action taken by a program or programmer to limit the computational load of a problem [18]. This is particularly helpful when one is exploring large, many dimensional search spaces like the ones created by MGALT-STOpS. If done well, a search space can be drastically reduced, resulting in more optimal solutions.

4.1 Traditional Constraint Pruning

Constraint pruning is a general term that refers to any method that excludes regions of the search space in order to create "smaller areas where the optimal solutions are to be found" [19]. Common methods are often as simple as excluding trajectories based on ephemerides, or orbital parameters. For example, the 2nd Global Trajectory Optimization Competition (GTOC2) challenged participants to create mission trajectories that rendezvoused with four asteroids of their choice as long as they satisfied some other criteria. In their solutions, most teams' trajectories went in order of increasing Semi-Major Axis and automatically excluded asteroids if the difference in their Inclination and Right Ascension of the Ascending Node (RAAN) from that of the spacecraft's were higher than some arbitrary value [18]. The Inclination and RAAN assumptions were taken from the well-known fact that Inclination and RAAN changes are very ΔV heavy [20]. From the semi-major axis constraints alone, it is estimated that the potential asteroid combinations were reduced from over 41 billion to only about 3.5 billion, resulting in an approximate 92% simplification of the prob-

lem. While this example does not correlate directly to STOpS, the general idea can be applied to greatly simplify the search space.

A type of constraint pruning that can be used by MGALT-STOpS is that of Thruster Angle Pruning. From literature [4] and MGALT-STOpS [3], it is apparent that optimal, realistic trajectories often follow "smooth" thruster angle curves. An example of this behavior is shown in the GALLOP Test Case [4] in Figure 4.1, where the red lines are the direction of thrust at the beginning of each segment. This known optimal trajectory results in thruster angles that are always close to parallel with the velocity vector of the spacecraft, which is intuitive because thrusting in the velocity vector will raise the spacecraft's Semi-Major Axis. By constraining the allowed range of thruster angles, the search space can be drastically reduced while reasonably pruning large quantities of solutions which are unlikely to be optimal. In the future, a maximum thruster angle rate of change constraint can also be implemented. This work chose not to implement the maximum rate of change feature as the author wanted to test the ability of the suite to find the smooth thruster angle profile without it being forced upon it.

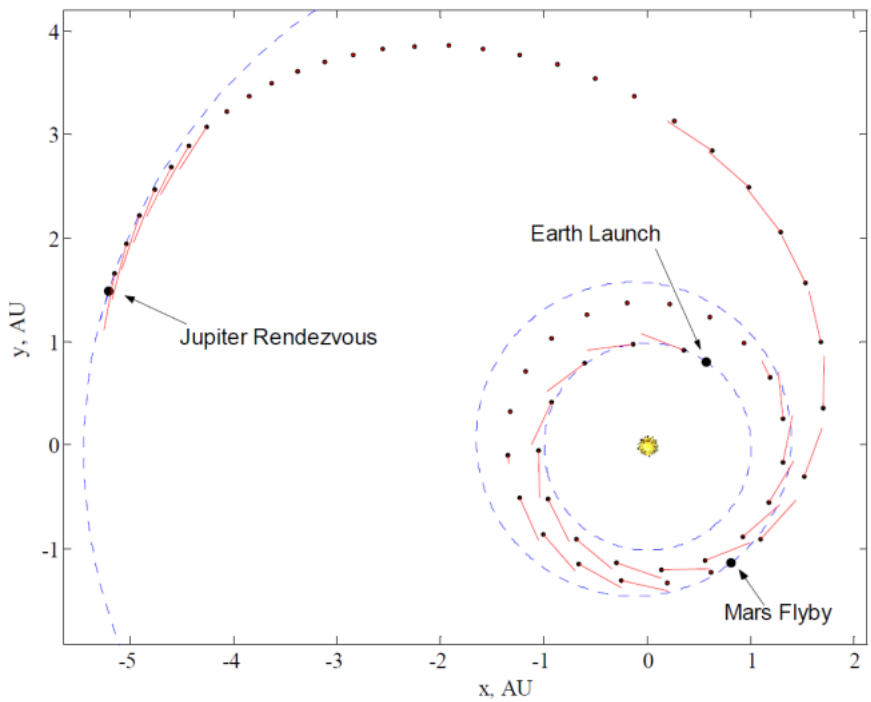

Figure 4.1: GALLOP Earth-Mars-Jupiter Optimal Trajectory [4]. Red lines represent thruster angle, black dots are segment start points.

4.2 Direct Method

Thruster Angle Pruning was implemented into the Direct Method via holding the thruster angles to "reasonable" values. Previously in MGALT-STOpS, thruster angles had the ability to range from 0-360° off of the velocity vector of the spacecraft. This led to MGALT-STOpS searching ranges of thruster angles across all segments that were both unlikely to be optimal and included unrealistic slews that would not be ideal for modern ADCS subsystems. Figure 4.2 shows a Direct Method trajectory produced by MGALT-STOpS [3] for the same Earth-Mars-Jupiter problem shown in Figure 4.1. Note that close to Earth, the initial thruster angles are against the velocity vector and then slew approximately 180° over two segments. Although this

results in a trajectory that meets well at the endpoints of the MFBSM, a trajectory such as this is not a good candidate for a real-world mission. It is also reasonable to assume that thrusters pointing in the opposite direction as velocity for a mission travelling to an outwards planet means that the departure date and/or time of flight is also sub-optimal.

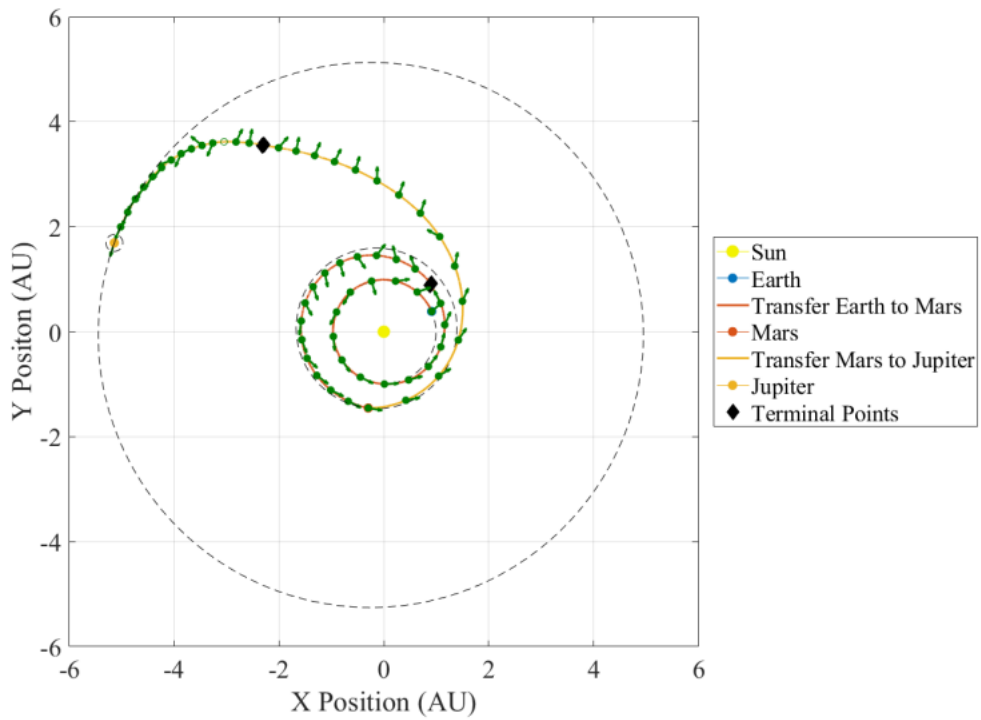

Figure 4.2: MGALT-STOpS Test Case 2 Earth-Mars-Jupiter Trajectory [3]. Green arrows represent thruster angles, green circles are segment start points.

Through examining trajectories in literature cited in MGALT-STOpS [3] and LT-STOpS [2] and testing multiple cases in MGALT-STOpS, including Earth-Mars-Jupiter, Earth-Mars-Saturn, and Earth-Venus-Mercury trajectories, ranges of realistic optimal thruster angles were determined. These ranges use the same orientation as

MGALT-STOpS has previously used, where positive thruster angles are measured clockwise from the velocity vector. Figure 4.3 shows a visual representation of how thruster angles are characterized versus the spacecraft's velocity vector.

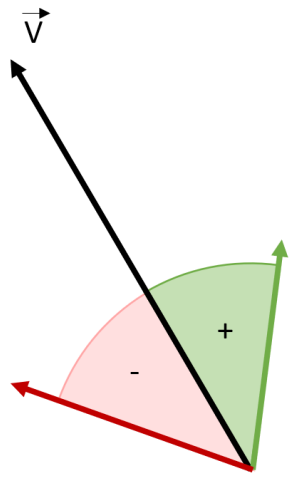

Figure 4.3: Direct Method Thruster Angles. Positive thruster angles are clockwise from the velocity vector. Negative thruster angles are counter-clockwise from the velocity vector.

For trajectories moving outwards in the solar system, a range of [-45°,75°] from the velocity vector of the spacecraft was found to appropriately contain the optimal thruster angles as well as provide 15° of additional margin. Similarly, for trajectories travelling inwards in the solar system, a range of [135°,255°] from the velocity vector accurately encompasses the optimal thruster angles and provides the same 15° of margin. Note that these angle ranges are exactly 180° apart, which acts as a conceptual check.

4.3 Indirect Method

Due to the fact that the equations of motion for the Indirect Method dictate both the thruster angle and its rate of change, it was not possible to implement Thruster Angle Pruning for this method. The author determined there were two options:

1. Cap the thruster angles at the ranges stated in Section 4.2, thereby losing the relationship between the costate variables and the trajectory.
2. Reject trajectories that leave the ranges stated in Section 4.2, thereby immediately losing most (if not all) solutions from the search space.

In the end, both options meant losing the natural integrity of the Indirect Method "control law," and it was decided to not implement Thruster Angle Pruning for this method. The author would also like to mention that the search space for the Indirect Method is significantly smaller than that of the Direct Method, so preserving the natural integrity of the Indirect Method was deemed more important than trying to reduce the search space.

Chapter 5

CLUSTER PRUNING

As a heuristic algorithm gets closer to finding the global optima, the solutions will begin to form clusters. "Clustering" is "the principle that many real-world problems have a structure where individual local optima, or "basins" tend to cluster together into "funnels" where one local optimum is better than the rest" [17]. In fact, Malloy [3] already observed this phenomenon when testing his MBH algorithm in MGALT-STOpS. By observing and exploiting clustering behavior in MGALT-STOpS, the search space can be reduced to only the volume in the cluster.

5.1 Traditional Cluster Pruning

For impulsive thrust, single-leg trajectories, a common method of finding the optimal departure and arrival dates is by creating a Porkchop Plot. Shown in Figure 5.1 is an example of a Porkchop Plot for Earth to Mars opportunities in 2005 from [5]. These plots are given their name by the shape of the contours of exit and entry velocities that directly correlate to the amount of ΔV required for a mission. As shown in Figure 5.1, there are two areas of ΔV minimization at the centers of the contour sections. Interestingly, these contour sections represent prograde and retrograde trajectories.

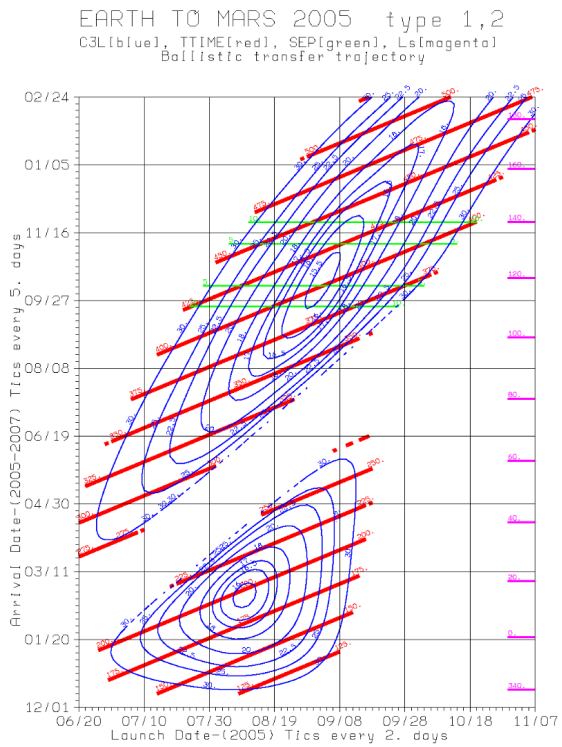

Figure 5.1: 2005 Earth-Mars Opportunities Porkchop Plot [5]. Blue contour lines are constant Characteristic Energy. Characteristic Energy directly correlates to the amount of ΔV required for a mission. The lowest Characteristic Energy solutions, and therefore lowest ΔV trajectories, are located in the center of the contours. Red lines are constant Time of Flight lines.

Figure 5.1 in particular was created using a brute force method where all potential departure and arrival dates are run through some Lambert's Solver [8] likely similar to the one that Fitzgerald used in STOpS [1]. This way, a fully solved search space can be examined by a mission designer to find the best launch windows and whether they want to travel prograde or retrograde. These contours also correlate to the synodic period of the planets and will repeat with the same frequency in the future.

If this search space were to be analyzed using the heuristic algorithms in MGALT-STOpS, the algorithms would eventually converge on at least one of the center sections

of the contours and the dispersion of the solutions would likely show clusters in the data. The theory of this work is that even though low thrust trajectories operate very differently, there might still be synodic behavior that can be exploited using cluster pruning in order to reduce the total search space to be optimized.

A previous master's thesis by Elmar Puts [21] has claimed that cluster pruning for single leg trajectories is feasible through use of the Skinny-dip Algorithm [6], although as of the writing of this work it is currently on a file embargo and cannot be considered verified without testing. Therefore, in the verification of Cluster Pruning, Section 10, the algorithms will not only be required to show reliable clustering behavior, but also that the clusters identified are accurately bounded and beneficial to the suite as a whole. The following subsections will explore the background of the Skinny-Dip Algorithm as well as the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithm [22].

5.1.1 Skinny-Dip Algorithm

The Skinny-Dip Algorithm [6] was the algorithm originally planned to be used by this work because of its previous implementation by Puts [21]. However, as will be explained later, it was not used in this work. The algorithm relies on the concavity of the Cumulative Distribution Function (CDF) of data over all N dimensions of the data set. For the example data set shown in Figure 5.2, the clusters are of varying sizes and densities, with those densities being significantly, but not overwhelmingly, larger than the density of the surrounding noise. Through the use of the CDF over the x-axis shown in Figure 5.3, the algorithm is able to identify where the clusters are in the data. This is done by marking windows where the CDF transitions from concave up to concave down, which in theory marks the beginning and ending of a cluster. The Skinny-Dip Algorithm will then run the same process over the bounds

found from the first step for all dimensions of the data, which in this case is two (2). Figure 5.4 shows the algorithm solving for two (2) clusters over the D Region of Figure 5.3.

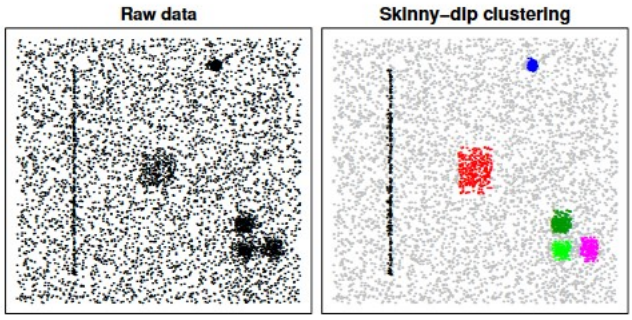

Figure 5.2: Skinny-Dip Algorithm Example Data Set [6]

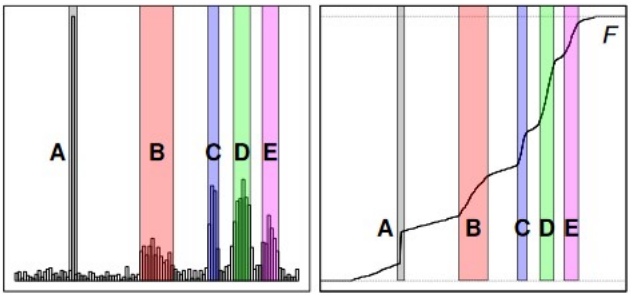

Figure 5.3: Histogram and Cumulative Density Function Over X-Axis [6]

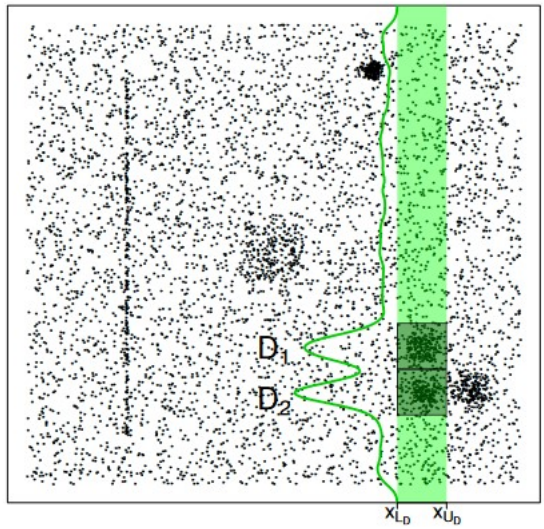

Figure 5.4: Finding Clusters Over Y-Axis [6]

Unfortunately, during the implementation phase of this algorithm, it became apparent that the clusters found in the data set were too dense compared to the surrounding data. In short, the clustering behavior contained approximately 99% of the data, and so the Cumulative Density Function was not sensitive enough to find where the clusters began and ended. Therefore, another algorithm needed to be used.

5.1.2 DBSCAN Algorithm

The DBSCAN Algorithm [22], or Density-Based Spatial Clustering of Applications with Noise Algorithm, takes a neighborhood approach to find clusters. The algorithm takes two global input parameters, ϵ and Minimum Points, where ϵ is a radius and Minimum Points is a value that describes the minimum number of data points that must be within ϵ units away in order for a data point to be considered part of a cluster.

The algorithm [22] starts by choosing an arbitrary point p in the data set and checking within the hyper-sphere given by radius ϵ to see if there are at least Minimum Points within it. There are two (2) possible outcomes for this initial point:

1. Noise: The point has less than the Minimum Points within ϵ units away.
2. Core Point: The point has greater than or equal to the Minimum Points within ϵ units away.

If p is considered noise, the algorithm [22] will then move onto the next arbitrary point in the data set. However, if p is a Core Point, the algorithm will then do the same calculation for the points found within the hyper-sphere around p and continue until all points in the cluster are found. This will eventually yield a third type of point in the data set:

3. Border Point: The point has less than the Minimum Points within ϵ units away, but neighbors at least one point that is a Core Point.

Once all of the points in the data set are evaluated for the number of points they contain within ϵ units away, the algorithm [22] has compiled a list of all clusters and noise in the data set. The DBSCAN Algorithm, as will be shown later in Section 10, is able to handle the large densities of points found in the populations of the Differential Evolution Algorithm. MGALT-STOpS will then automatically set bounds for the data set that constrains the rest of the optimization process to only search the region of the search space that the cluster was found in.

Chapter 6

MULTI-OBJECTIVE OPTIMIZATION

Multi-Objective Optimization is the process of optimizing a scenario that contains multiple cost functions. While MGALT-STOpS currently only optimizes one final cost function, each trajectory leg can be seen as its own individual cost. This section will discuss the issues faced by Malloy [3] during the creation of MGALT-STOpS and the solution this work proposes as they relate to Multi-Objective Optimization.

6.1 Premature Pruning

One of Malloy's [3] main concerns while writing MGALT-STOpS was that of prematurely pruning solutions that were promising. Said promising solutions often had at least one well-optimized leg, but the others were poorly optimized; they were therefore removed due to the high cost of the poorly optimized leg outweighing the low cost of the well-optimized leg. This problem arises from the fact that the current cost function simply adds together the costs of trajectory legs and the algorithms take this single value as a measure of optimality. An example of some promising solutions being prematurely pruned is shown in Figure 6.1. This figure is a visual of the by-leg costs of solutions during the final migration of optimization where the x-axis is an Earth to Mars transfer and the y-axis is a Mars to Jupiter transfer. The red circles encompass promising solutions that were removed from the Differential Evolution Algorithm population. If these solutions had been kept, Figure 6.1 would likely show other solutions in similar locations. This work will explore potential solutions to the issue of premature pruning.

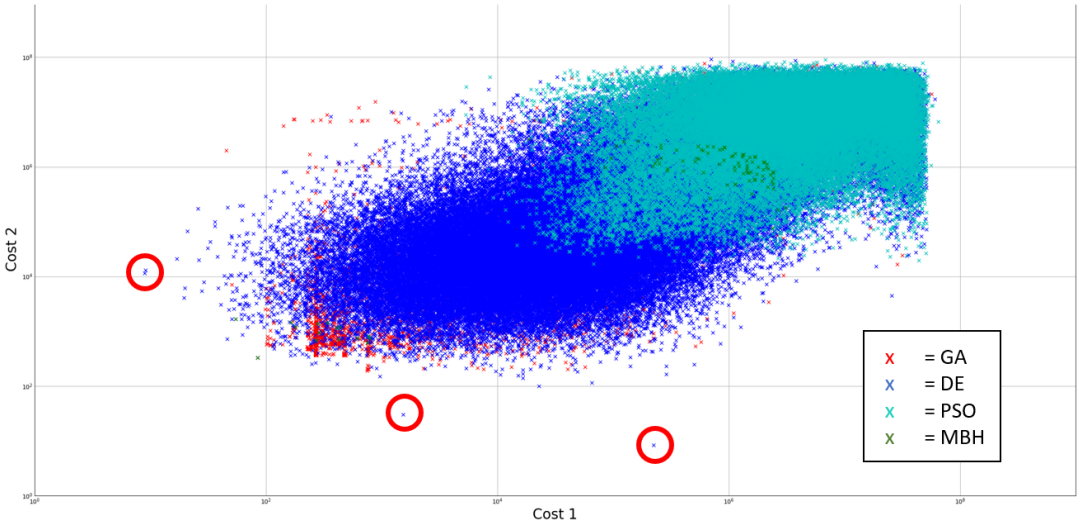

Figure 6.1: Premature Pruning for Indirect Method on Final Migration. Each point represents the by-leg cost breakdown of a solution evaluated during the final migration of optimization. The points are colored according to the algorithm they appeared in. The red circles encompass solutions that were exceptionally good in one leg but poor in another, and were therefore removed from their respective population.

6.2 Pareto Fronts

Generating a Pareto Front is often used as a tool to measure the trade-offs between multiple variables for a Multi-Objective Optimization problem [7]. Often times in engineering problems, decreasing the cost in one variable will lead to increasing the cost in another. For a simplified, non-engineering example, imagine two broke logicians planning where they want to eat dinner. They obviously want good food but cannot afford an expensive meal. While they suggest restaurants in their local area, they find that the cheaper they try to be, the worse the food becomes. At the same time, recommending better food options increases the price. In the end, they must collectively make a decision that weighs the trade-offs in prices and flavor. Being

logicians, they realize that creating a Pareto Front is their best option for visualizing their decision.

Pareto Fronts operate on the principle of Pareto Dominance and Non-Dominance. Pareto Dominated solutions are those in which there exist solutions more optimal than them in all variables, while Pareto Non-Dominated solutions are those where for all solutions in a data set, there are none that are more optimal in all variables. An arbitrary example is shown in Figure 6.2 where both variables are optimal when minimized.

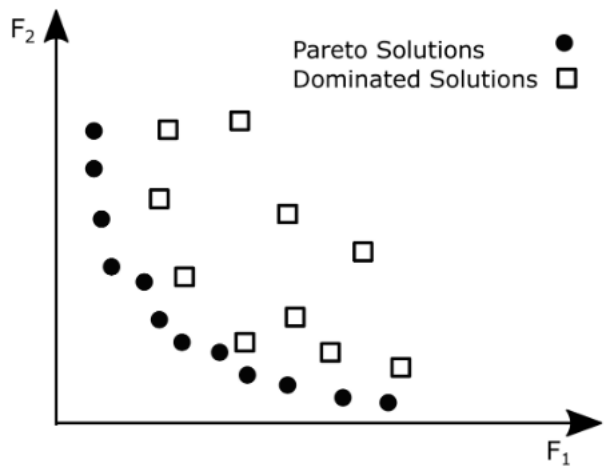

Figure 6.2: Arbitrary Min-Min Pareto Front [7]. Pareto Solutions are those for where there exist no solutions that are more optimal in all dimensions.

By keeping track of Pareto Non-Dominated solutions, also sometimes called Pareto Solutions, MGALT-STOpS could feed promising trajectories back into the optimization process as discussed in Sections 6.2.1 and 6.3.

6.2.1 Pareto Front Modification

While the Pareto Front served well in inspiration, it did not function appropriately for MGALT-STOpS. The reason for this was that the observed Pareto Fronts did not contain enough solutions to effectively feed back all promising solutions. Referencing back to Figure 6.1, it can be seen that a true Pareto Front similar to the one seen in Figure 6.2 would only contain about 10 solutions. It was therefore decided that instead of archiving all Pareto Non-Dominated solutions, it was more important to continually track the top 100 solutions for each leg. These solutions would then be fed back into the algorithms via Substitution as discussed in Section 6.3. For the rest of this work, the act of archiving and substituting legs back in will be referred to as Best Leg Tracking.

6.3 Algorithm Population Substitution

In an effort to still use elements of Multi-Objective Optimization without the use of a Pareto Front, the method of Substitution was implemented. Substitution was inspired by the idea of genetic crossover and mutation as seen in the Genetic Algorithm. During the GA mating process, there are probabilities for crossover and mutation where a population member's variable string can be altered. This allows for better global search space exploration and potentially landing a member close to the global optimum.

In Substitution, all population members have a small chance of inheriting a transfer leg from one of the archived best 100 solutions. Importantly, each leg can only inherit a substituted leg that correlates to the same planetary transfer. For example, a solution can only substitute a top 100 Earth→Mars leg for its Earth→Mars

transfer; substituting a Mars→Jupiter leg for an Earth→Mars leg would likely result in a very poor solution. This method was implemented for all four (4) algorithms in the suite, although the implementation in the MBH algorithm was limited to just the generation of the initial population.

Chapter 7

VERIFICATION APPROACH

In an attempt to simplify the verification of the topics of the following sections, all tests will be run under the conditions described in this section unless otherwise specified. These conditions are the same as Test Case 2 from Malloy's MGALT-STOpS Thesis [3] that was verified against the Earth to Jupiter via Mars trajectory in "Design of Low-Thrust Gravity-Assist Trajectories to the Outer Planets" by Yam, McConaghy, Chen, and Longuski [4]. All runs conducted in this work were performed on a Razer Blade 15 with 16GB RAM and an Intel i7-11800H processor, and the Spyder IDE was the only application running in the foreground or background.

7.1 Problem

This test case is based off of the proposed Jupiter Icy Moons Orbiter (JIMO) mission which, if flown, would have entered Jupiter's Sphere of Influence and studied its moons. This mission was planned to use electric propulsion and was a trial run to discover the feasibility of using low thrust and gravity assists to explore the outer solar system. The suites from literature that MGALT-STOpS verified against were the Gravity Assisted Low-thrust Local Optimization Program (GALLOP) [4] and Satellite Tour Design Program Low-Thrust Gravity Assist (STOUR-LTGA) [4], where they used the Direct and Indirect Methods, respectively.

From the literature [4], JIMO was planned to use five (5) Nuclear Electric Xenon Ion System (NEXIS) thrusters. MGALT-STOpS currently does not support the capability to model multiple thrusters, however variables like the thrust and mass flow rate

can be summed together to create values that represent the same physics. Therefore, MGALT-STOpS and this work will use the following parameters:

$$T_{max} = 2.26 \text{ N}$$
$$I_{sp} = 6000 \text{ s}$$
$$m_o = 20000 \text{ kg}$$
$$\dot{m} = 3.84e^{-5} \text{ kg/s}$$
$$V_{Dep} = 0.2 \text{ km/s}$$

where T_{max} is the maximum thrust in Newtons, I_{sp} is the specific impulse in seconds, m_o is the initial mass in kilograms, $\dot{m}$ is the mass flow rate in kilograms per second, and V_{Dep} is the excess velocity due to the launch vehicle at the time of departure in kilometers per second.

7.1.1 Direct Method

The Direct Method uses variable thrust in its attempt to optimize trajectories, as in reality, many spacecrafts use variable thrust as a way to throttle the effect of the thrusters over the course of the trajectory. Therefore, the Direct Method's segments will use thrust values of anywhere between 0 and 2.26 Newtons during flight.

Due to the large search space of the Direct Method, mostly caused by the use of 60 segments, the Departure Date and Times of Flight windows for this method only span a couple months each. These windows were used by Malloy [3] as they encompass the optimal solution found in literature while still forcing MGALT-STOpS to effectively search the search space. They are as follows:

Departure: Sep. 16, 2021 - Jan. 16, 2022

$$\text{ToF}_{\text{E}\rightarrow\text{M}}\text{: } 855 \pm 50 \text{ Days}$$

$$\text{ToF}_{\text{M}\rightarrow\text{J}}\text{: } 1346 \pm 50 \text{ Days}$$

7.1.2 Indirect Method

The Indirect Method unfortunately does not have the capability to simulate variable thrust. Instead, it simulates constant thrust throughout the trajectory. Therefore, the thrust magnitude of the thruster will always be 2.26 N.

While the constant thrust may be a limitation of the Indirect Method, one of its advantages is that it has a much smaller total search space. Therefore, the Departure Date and Times of Flight windows are larger than that of the Direct Method. They are as follows:

$$\text{Departure: Nov. 6, 2020 - Nov. 6, 2022}$$

$$\text{ToF}_{\text{E}\rightarrow\text{M}}\text{: } 752 \pm 100 \text{ Days}$$

$$\text{ToF}_{\text{M}\rightarrow\text{J}}\text{: } 1448 \pm 300 \text{ Days}$$

7.2 Settings

MGALT-STOpS used different Island Model and algorithm parameters for the Direct and Indirect Methods. In general, the Indirect Method was given a larger population size and more migrations because it was used as a preliminary trajectory solver while the Direct Method was used to refine the Indirect Method's result. The parameters for this test case are shown in Tables 7.1-7.5 and are the same as shown in [3].

Table 7.1: Island Model Parameters

Parameter	Indirect Method	Direct Method
Migrations	4	3
Islands	GA, DE, PSO, MBH	GA, DE, PSO, MBH
Selection Policy	Natural Selection	Natural Selection
Number Selected per Island	15	15
Replacement Policy	Best N	Best N
Number Replaced per Island	15	15
Connection	All Connected	All Connected

Table 7.2: Genetic Algorithm Parameters

Parameter	Indirect Method	Direct Method
Population	450	300
Generations	180	115
Probability for Crossover	0.8	0.8
Probability for Mutation	.005	.01
Generation Method	Tournament	Tournament
Members per Tournament	15	10
Number of Elite Members	1	1
Mate Method	Random Crossover	Random Crossover
Cross Points	30	30

Table 7.3: Differential Evolution Parameters

Parameter	Indirect Method	Direct Method
Population	450	300
Generations	180	115
Probability for Crossover	0.8	0.8
Selection Method	Random Best Blend	Random Best Blend
Scale Factor Method	Jitter	Jitter
Scale Factor Bounds	0.5 - 0.9	0.5 - 0.9
Survivor Method	Tournament	Tournament
Members per Tournament	3	3

Table 7.4: Particle Swarm Optimization Parameters

Parameter	Indirect Method	Direct Method
Population	350	250
Iterations	150	100
Max Velocity	0.7	0.7
Informants	5	5
Confidence in Self	0.9	0.9
Max Confidence	0.7	0.7

Table 7.5: Monotonic Basin Hopping Parameters

Parameter	Indirect Method	Direct Method
Population	15,000	10,000
Iterations	1,500	800
Basin Population	200	175
Basin Iterations	125	100
Percent Feasibility	75, 10, 2.20, 1.30, 1.00	20, 5, 2.25, 1.00
Percent Random	0.13, 0.09, 0.06, 0.03, 0.01	0.09, 0.06, 0.03, 0.01

7.3 Verification Criteria

Due to the large, many-dimensional search spaces that MGALT-STOpS handles in addition to cost functions that differ from literature, it is not expected that solutions will be replicated exactly. In Malloy's work [3], neither of the Direct or Indirect Methods matched the literature exactly, but they were close enough that they served as a good start for preliminary mission design. Because of this variation, all cases tested in the following chapters were run five (5) times in order to evaluate the proposed performance improvements' consistencies. Additionally, in order for the results in the following chapters to be considered verified, they must meet the following criteria:

1. Final solution costs shall be under 10,000.

- Costs above this signify that the methods have failed to solve for trajectories continuous across the MFBSM terminal points.

2. Final solutions shall exhibit behaviors appropriate to the proposed method of improvement.

   - MGALT-STOpS consistently displays some optimal and sub-optimal behaviors, as will be explained in detail later. Most proposed methods will aim to minimize sub-optimal behaviors. However, Chapter 8 will only aim to reproduce trajectories. Therefore, similar sub-optimal behavior will be considered proof of verification.

3. Proposed Methods shall be beneficial to the user.

   - The goal of the performance improvements is to provide options to the user to further refine and improve their trajectories. If a proposed method of improvement does not offer anything beneficial to the user, it should not be included in the suite.

Chapter 8

PYTHON VERIFICATION

In an effort to make MGALT-STOpS more accessible to the public, it was decided to port the code from MATLAB into Python 3. This work is the first work on the low thrust side of STOpS to be ported into Python, although previous works have been written in Python on the impulsive side of STOpS [23, 24]. Before any performance improvements were implemented into MGALT-STOpS, it was first necessary to verify the same results were replicable in Python. This section will cover the results of the switch to Python and changes made to the suite, specifically the Monotonic Basin Hopping Algorithm. The specific criteria for Python verification are as follows:

1. Final solution costs shall be under 10,000.
2. Final solutions shall exhibit behaviors previously seen in MGALT-STOpS.
3. MGALT-STOpS in Python shall have lower run times than in MATLAB.

8.1 MBH Changes

While porting MGALT-STOpS from MATLAB to Python, it became apparent that some minor changes needed to take place to prevent consistent memory overload. In short, the MBH function tracks more variables per solution than the other algorithms. Due to its need to check for feasibility of its solutions, it retains a dictionary which contains the entire propagated trajectory and associated planetary locations for each population member. When storing this for all 10,000+ trajectories, the cumulative data storage required is too much for a 16GB RAM machine. Therefore, it was

deemed necessary to check for the feasibility of population members periodically and then only keep the top 100 best-so-far solutions. This drastically reduced the amount of storage required and fixed the memory error issue. Crucially, it does not affect the performance of the MBH algorithm as it only changes the order of operations while keeping the amount of computation the same. Additionally, only recording the top 100 best-so-far solutions does not change the results of the algorithm as when sharing solutions to other islands via migration, only the top 15 solutions are reported. This was verified through testing as the MBH algorithm consistently reported similar top 100 costs with and without the changes implemented. Additionally, all evaluated trajectory legs are then checked against the Top 100 best-so-far solutions if Best Leg Tracking is implemented so that no promising solutions are lost.

The author would like to note before proceeding that in the following verification chapters, there will be figures that show the trajectory costs for each algorithm's population. The MBH algorithm will appear under-represented because of its comparable lack of solutions. However, while the number of solutions shown will be fewer, it is important to keep in mind that for the MBH algorithm, only the top 100 solutions are shown.

8.2 Direct Method Results

After porting MGALT-STOpS to Python, it was apparent that the newly coded suite performed similarly to the one in MATLAB. Table 8.1 shows the costs, Departure Dates, Flyby and Arrival Times of Flight, Total Time of Flight, and Propellant Mass Fraction for one of the Python Direct Method runs compared to MATLAB.

It is important to note that this chosen Python trajectory is not considered special, but is instead a trajectory that exhibits average behavior when compared to the other

Table 8.1: MGALT-STOpS Direct Method MATLAB/Python Comparison

	MATLAB	**Python**
Cost	3722.30	3907.36
Departure Date	Oct 16, 2021	Aug 1, 2021
Flyby ToF (days)	805	825
Arrival ToF (days)	1396	1396
Total ToF (years)	6.03	6.08
Propellant Mass Fraction	34.2%	36.85%

four (4) runs. The average cost of all five (5) Python Direct Method trajectories is 3,800, with none reaching a cost higher than 4,297. Therefore, the cost requirement for verification of the Python Direct Method is satisfied.

Along with the trajectory parameters, the exhibited behaviors are similar to those seen in MATLAB. Figure 8.1 shows the MATLAB trajectory and Figure 8.2 shows the same Python trajectory as tabulated in Table 8.1.

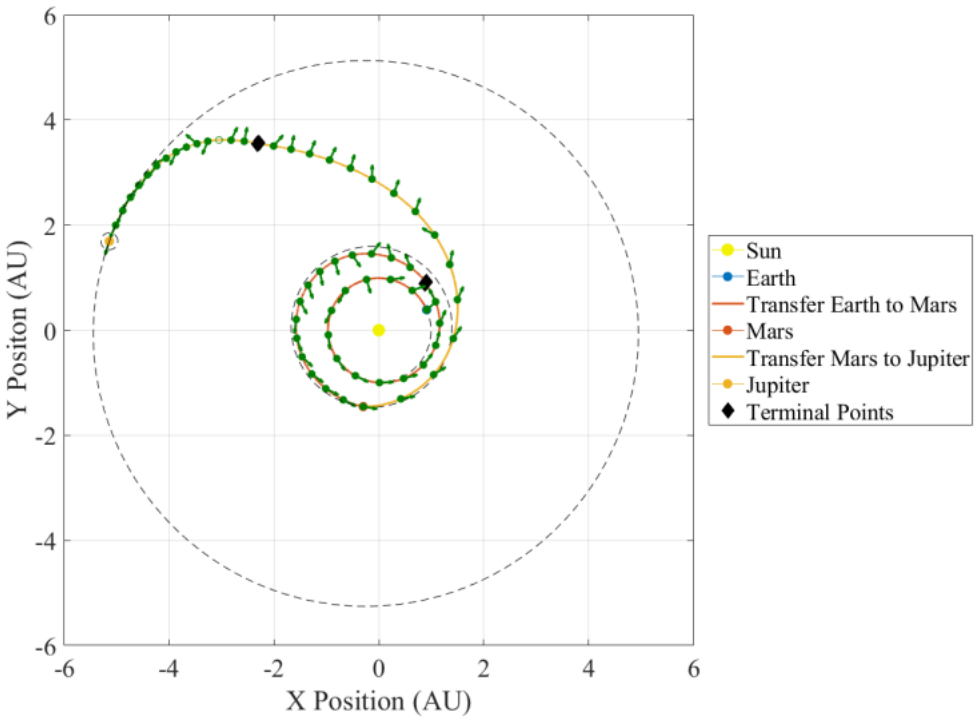

Figure 8.1: MATLAB MGALT-STOpS Direct Method Final Solution

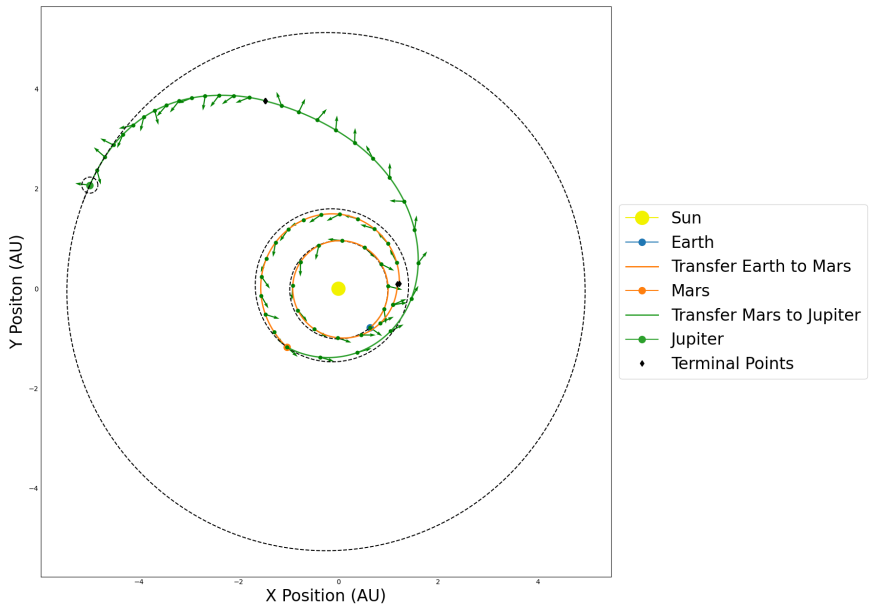

Figure 8.2: Python MGALT-STOpS Direct Method Final Solution

While not exactly the same, the chosen Python trajectory reaches Mars and Jupiter in a similar fashion as the one in MATLAB. Both trajectories complete approximately $1\frac{3}{4}$ heliocentric orbits before reaching Mars and less than one more before reaching Jupiter. Additionally, the thruster angles vary widely at times before smoothing out and gradually changing. The sub-optimal thruster angle behavior, while not ideal, is seen as proof that the suites behave similarly.

It is noted that after the gravity assist at Mars both suites produce trajectories that fly below Mars' orbit, although some are more drastic than others. This is a tendency commonly seen in the Direct Method for this test case. The behavior is likely due to the selected Time of Flight for the Mars to Jupiter transfer, where MGALT-STOpS might see flying below Mars as the best way to change its angular position quickly. This trend will be seen later in Chapter 10 where the algorithms will tend to search for solutions at the upper limit of the Mars to Jupiter transfer. The occurrence of this behavior and the sub-optimal thruster angles is enough for the Direct Method to be considered verified.

The final metric for verification of the Python Direct Method is that of lowering run time. The current MATLAB Direct Method run time for the device used for this work is approximately 6 hours. After porting to Python, the run time dropped to an average of around 1.5 hours, which is an approximate 75% decrease in time spent patiently waiting. Because of this massive decrease in run time, the third requirement for the Python Direct Method verification is satisfied.

8.3 Indirect Method Results

The Indirect Method also performed similarly in Python as it does in MATLAB. Table 8.2 shows the important trajectory parameters of both the MATLAB and one of the Python Indirect Method runs.

Table 8.2: MGALT-STOpS Indirect Method MATLAB/Python Comparison

	MATLAB	**Python**
Cost	2429.53	1704.77
Departure Date	Sept 19, 2021	Sep 1, 2021
Flyby ToF (days)	756	708
Arrival ToF (days)	1507	1748
Total ToF (years)	6.11	6.72
Propellant Mass Fraction	37.0%	40.75%

Again, this Python Indirect Method trajectory was chosen because it performed about average when compared to the other ones used for verification. The average cost of all of the runs was 2,537, with none reaching a cost higher than 5,558. Therefore, the cost verification requirement for Indirect Method can be considered satisfied.

Figures 8.3 and 8.4 show the MGALT-STOpS MATLAB and Python trajectories as tabulated in Table 8.2, respectively. Again, there are similar exhibited behaviors.

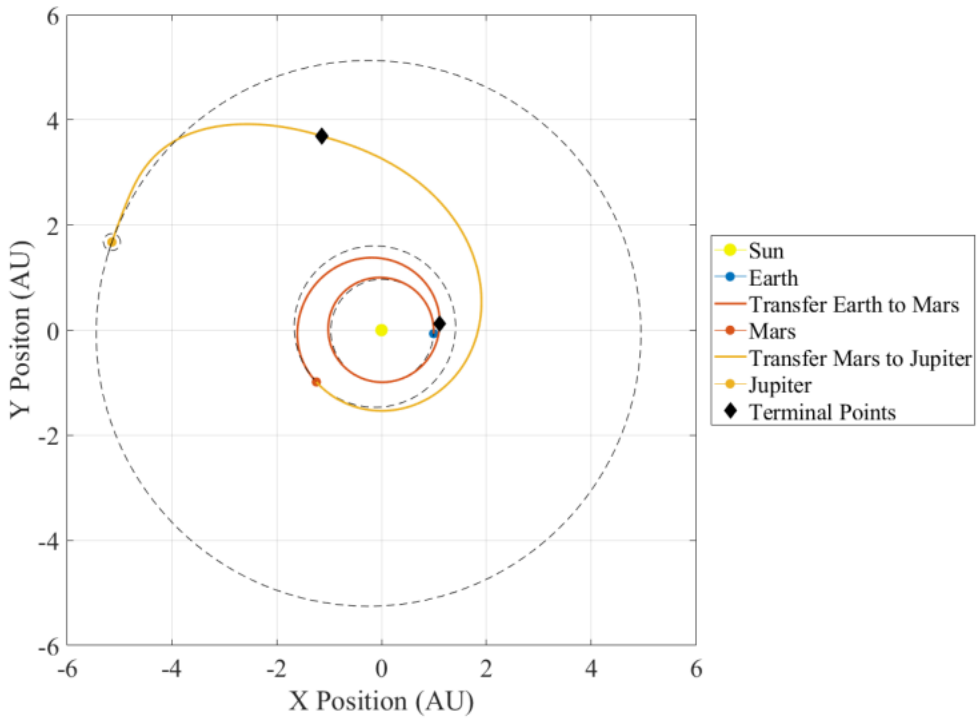

Figure 8.3: MATLAB MGALT-STOpS Indirect Method Final Solution

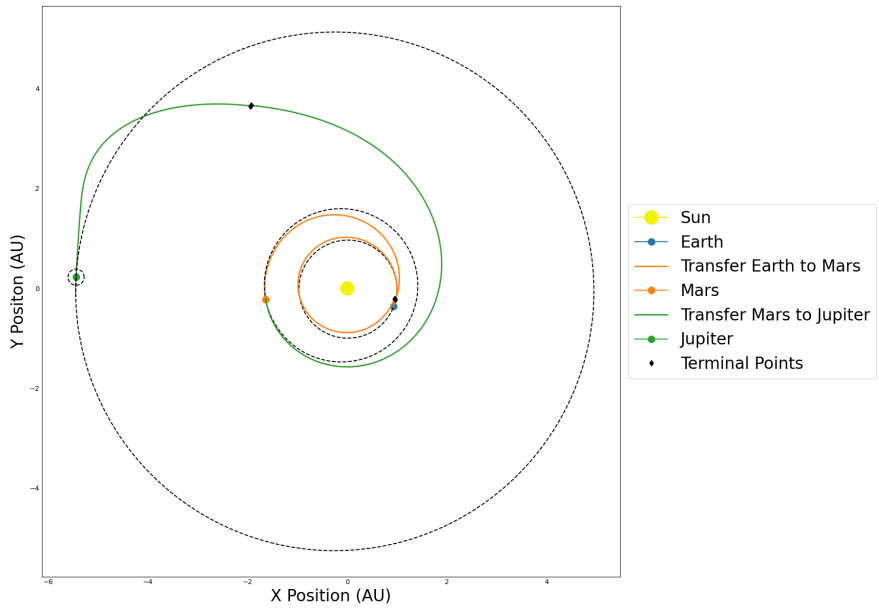

Figure 8.4: Python MGALT-STOpS Indirect Method Final Solution

The two main pieces of evidence that the MATLAB and Python codes work similarly is the issue of returning below Earth's orbit and overshooting Jupiter. Malloy [3] noted these issues in his work, and this work sees the same behavior. Additionally, both of these runs perform over one and a half heliocentric revolutions when heading to Mars, and almost one more when headed to Jupiter. Because of the similarities in these trajectories, another case is made for Python Indirect Method verification.

Note: The author notes that it is a fairly common occurrence that the Indirect Method provides solutions that depart in early 2021. This trend was seen in both the MATLAB and Python versions. However, if one were to shrink the departure window to exclude early 2021 trajectories, the suites will more frequently converge on a date in late 2021.

Finally, the run time improvement for the Python suite is also significant. On the device used for this work, the MATLAB Indirect Method required approximately 9 hours of run time. However, the Python Indirect Method ran for approximately 3 hours, resulting in about 66% less time required for optimization. It is worth noting that a large part of the difference between the run times for the Direct and Indirect Methods stems from the combination of more migrations, higher populations sizes, and more generations for the Indirect Method. The individual methods take approximately the same time to run if given equivalent parameters.

8.4 Final Remarks

Overall, the results and behaviors seen from the Python Direct and Indirect Methods mirror the results from the original MGALT-STOpS suite in MATLAB while also

being significantly faster. Therefore, the Python suite is considered verified. This was a crucial step in this work, as it was now possible to incorporate other methods of improvement such as Thruster Angle Pruning, Cluster Pruning, and Best Leg Tracking. For the following chapters, all mentions of a baseline Direct or Indirect Method run will be referring to the Python runs displayed in this chapter.

Chapter 9

THRUSTER ANGLE PRUNING VERIFICATION

The goal of Thruster Angle Pruning is to constrain thruster angles to ranges that would be considered realistic for a flown mission. This not only reduces the search space significantly, but also reduces a potential user's frustration by lowering the chances of solutions nothing short of an attitude control nightmare. This chapter will cover the results and analysis of the implementation of Thruster Angle Pruning for the Direct Method, as the formulation of the thruster ranges has already been discussed in Section 4.2. As such, the verification criteria are as follows:

1. Final solution costs shall be under 10,000.
2. Final solutions shall exhibit behaviors previously seen in MGALT-STOpS.
3. Final solutions shall consistently exhibit more realistic thruster angles and improve on sub-optimal thruster behaviors.

9.1 Direct Method Results

The goal of Thruster Angle Pruning was to hold the spacecraft thruster angles to reasonable values based on literature [4] and MGALT-STOpS trajectories [3]. The hope was that it would force MGALT-STOpS to search areas of the search space that were more likely to contain the optimal trajectory. Figures 9.1 and 9.3 show Python MGALT-STOpS trajectories with and without Thruster Angle Pruning, respectively. Figures 9.2 and 9.4 show the thruster angles for those same trajectories, respectively.

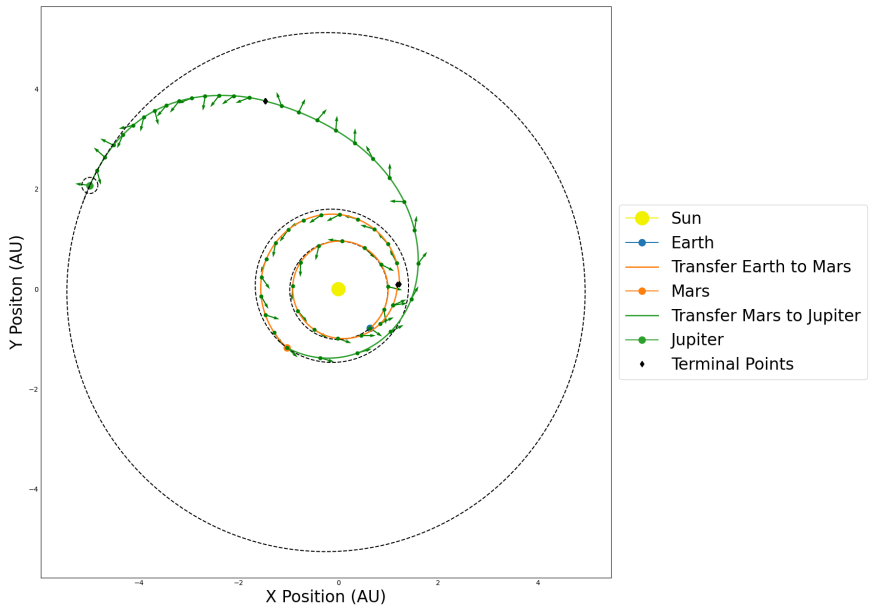

Figure 9.1: Direct Method Trajectory without Thruster Angle Pruning

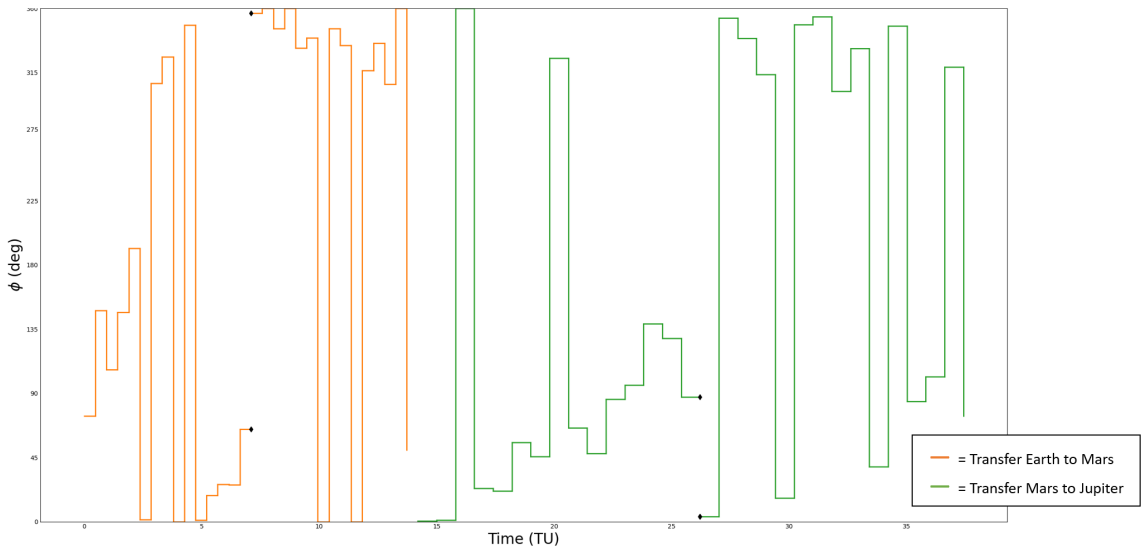

Figure 9.2: Direct Method Thruster Angles without Thruster Angle Pruning. Thruster Angles below 75° and above 315° are considered in the optimal range for this work. The orange segments are the Earth to Mars transfer, and the green segments are the Mars to Jupiter transfer.

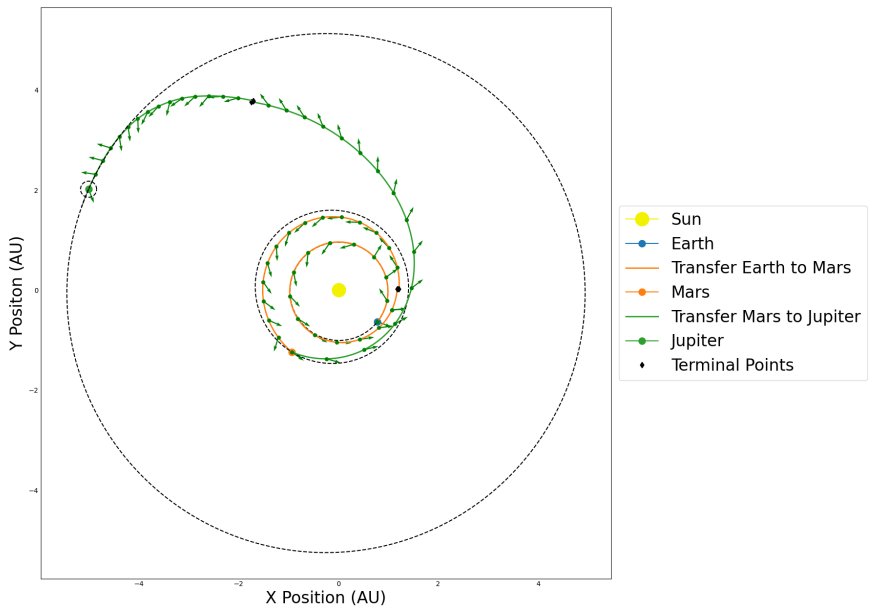

Figure 9.3: Direct Method Trajectory with Thruster Angle Pruning. The visibly smoother thruster profile is evidence of a more practical trajectory.

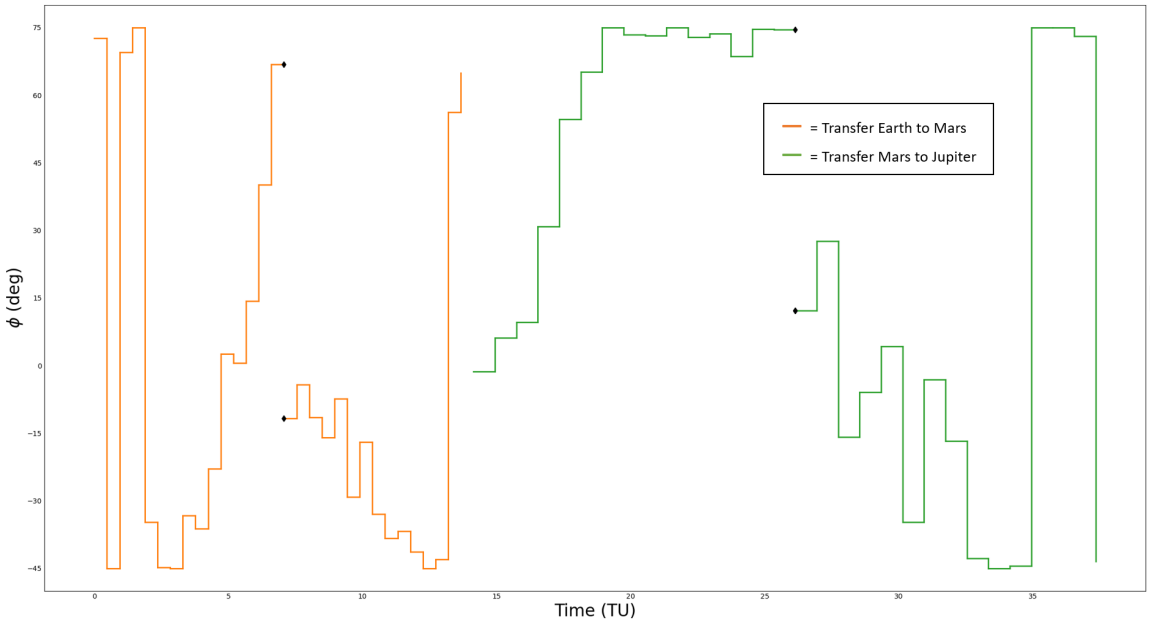

Figure 9.4: Direct Method Thruster Angles with Thruster Angle Pruning. On a scale from -45° to 75°, there is significantly reduced variation between thruster angles for most of the trajectory.

For Figure 9.2, it is important to remember that it has a thruster angle range of [0°,360°] while Figure 9.4 has a range of [-45°,75°]. Therefore, for Figure 9.2, any thruster angle lower than 75° or higher than 315° is considered inside the optimal range. With that in mind, it is apparent that the start of the trajectory along with some points in between are poorly optimized. In other final solutions, poorly optimized thruster angles also tend to occur at the end of the mission. Additionally, there is a lot of variation between the thruster angles of adjacent segments. It is assumed that these behaviors are likely due to the extremely large search space of the Direct Method and lack of emphasis on thruster angles in the cost function.

In Figure 9.4, there is much less thruster angle variation between segments, where the only large changes occur at the beginning and very end. Specifically comparing the forward shooting portion of the Mars to Jupiter transfers for both trajectories, it is apparent that the Thruster Angle Pruned Trajectory is significantly more realistic as the thruster angles change very smoothly. This is likely due to the constrained search space, as with this form of Thruster Angle Pruning, the search space volume was reduced by a factor of $(\frac{1}{3})^{60}$. In other words, MGALT-STOpS does favor trajectories with smoothly changing thruster angles when it has a smaller search space volume to cover.

Although this behavior is not obviously shown in Figure 9.2, Thruster Angle Pruning also removed the tendency of the Direct Method to thrust against the velocity vector of the spacecraft when leaving Earth. This behavior led the author to believe that there were optimal departure dates later than the one proposed by MGALT-STOpS, as it is sub-optimal to initially slow down before speeding back up. This will also be discussed later in Chapter 10. Thruster Angle Pruning has validated

this theory as the resulting trajectories, which can be seen in Appendix B.2.1, use departure dates that align more consistently with the optimal trajectories displayed by literature [4] and Malloy [3].

While the implementation of Thruster Angle Pruning did drastically reduce the search space volume and remove the tendency of the suite to thrust against the velocity vector, it unfortunately did not reduce the total costs. This is likely because when the suite was able to use all 360°, it compensated for sub-optimal thruster angles by using unrealistic ones. The average cost for the Direct Method with Thruster Angle Pruning was 4,772 with a high of 6,111, which are 974 and 1,814 higher than the runs without it, respectively. Although this outcome might suggest that Thruster Angle Pruning hurts the final results output by MGALT-STOpS, these costs are still well below the 10,000 cutoff. There is also an important trade-off to consider between the performance of the FBSM for preliminary trajectory design and the performance required for the spacecraft. For this reason, Thruster Angle Pruning is verified as a viable option for users to improve their thruster angles after they have determined viable Departure Dates and Times of Flight for their proposed mission.

9.2 Indirect Method Results

This section serves as a brief reminder to the reader that Thruster Angle Pruning for the Indirect Method was not implemented in this work. The author encourages any future work to explore other ways of constraining thruster angles for the Indirect Method, but also warns that tampering with the relationship between the thruster angle and the costate variables risks losing reproducibility of solutions.

9.3 Final Remarks

Thruster Angle Pruning for the Direct method has been verified as a beneficial tool for the user. It may not reliably lower the cost of trajectories, but it does serve as a way to make the reported solutions more realistic. It is recommended for the user to use Thruster Angle Pruning only after they have confidently refined their Departure Date and Times of Flight window to within a few months. This should ensure a trajectory that is somewhat realistic for today's spacecrafts.

The Indirect Method unfortunately was not able to accommodate this type of Thruster Angle Pruning because of its similarities to a control law. The costate variables are simply not controllable by a user and they will follow the guidance of their mathematical derivation. Even if the user were to cap thruster angles, the trajectories would be impossible to reproduce on another suite, which would make the solutions useless.

Chapter 10

CLUSTER PRUNING VERIFICATION

Because of the unverified nature of the DBSCAN Algorithm in regards to MGALT trajectories, it is first necessary to verify that it can accurately find any clusters in the data. From there, Cluster Pruning must be proven to work for both the Direct and Indirect Methods. Finally, the combination of Cluster Pruning with Thruster Angle Pruning for the Direct Method will be evaluated. The verification criteria for Cluster Pruning are as follows:

1. Final solution costs shall be under 10,000.
2. Final solutions shall exhibit behaviors previously seen in MGALT-STOpS.
3. Final solutions shall help the user refine their Departure Date and Times of Flight windows.

10.1 DBSCAN Algorithm Results

The first step to verifying the DBSCAN Algorithm's potential for pruning an MGALT search space is to find clusters in the populations. In order to stay true to the inspiration of impulsive thrust clusters and minimize the scanned dimensions, it was decided to analyze the Departure, Flyby, and Arrival dates for the trajectories with the theory that throughout optimization, the algorithms will begin to converge on a range of optimal dates. This proved somewhat successful, as the Differential Evolution Algorithm, as will be seen later, shows obvious clustering behavior. This is likely due to its strengths as a global search algorithm as well as solutions constantly

being moved around by a mutant vector. In contrast, the Genetic Algorithm nicely spread its solutions around the search space as well, but because the only methods of change are crossover and mutation, the clusters that formed were often simply the best solutions repeated on top of each other.

Unfortunately, while clusters existed, the pruning of the Flyby and Arrival dates proved very difficult to work with as the suite itself defines trajectories based on the Departure Date and Times of Flight. Therefore, in order to keep the integrity of the user-defined search windows, clusters were searched for over the Departure Date and Times of Flight. This proved fruitful and much easier to work with. After the first migration and second run-through of the DE Algorithm, the eventual solution was found within the cluster 10 out of 10 times for both the Direct and Indirect Methods without any pruning implementation. This meant that finding the cluster and setting bounds around it was a very promising way to prune the search space.

For the Direct Method, the values of ϵ and Minimum Points that most accurately found the clusters in the Differential Evolution Algorithm were 7.5 and 10, respectively. Figure 10.1 shows a Differential Evolution, Direct Method cluster found by the DBSCAN Algorithm. As can be seen, the cluster contains a significant majority of the population. The green box around it signifies the bounds of the cluster, which shrinks the Departure Date and Times of Flight search space by approximately 84.2% in this case. Ideally, the bounding box would have been even smaller, but decreasing the parameters any more caused some inconsistent clustering identification between runs.

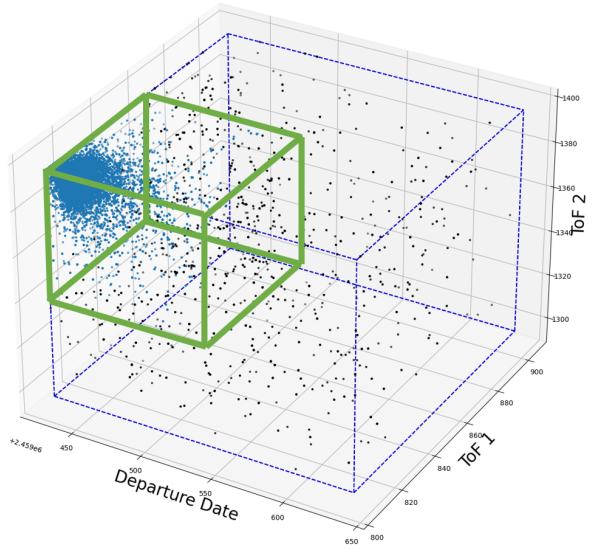

Figure 10.1: Direct Method DE Cluster Found by DBSCAN. The blue points are points in the cluster, and the green box represents the bounds of the cluster. After MGALT-STOpS finds the cluster, the Departure Date and Times of Flight Windows search space is reduced to the box.

The values of ϵ and Minimum Points that most accurately found the clusters in the Indirect Method Differential Evolution Algorithm were also 7.5 and 10, respectively. Figure 10.2 shows a cluster found in the Differential Evolution population for the Indirect Method. Again, the cluster contains most of the population and the bounds eliminate around 96.7% of the Departure Date and Times of Flight search space. It is important to note that the Indirect Method Clusters have fewer respective data points inside them. This is because for this test case, the Indirect Method has a much larger Departure Date and Times of Flight search space. If the search space mirrored that of the Direct Method's, the clusters would likely contain approximately the same amount of data points.

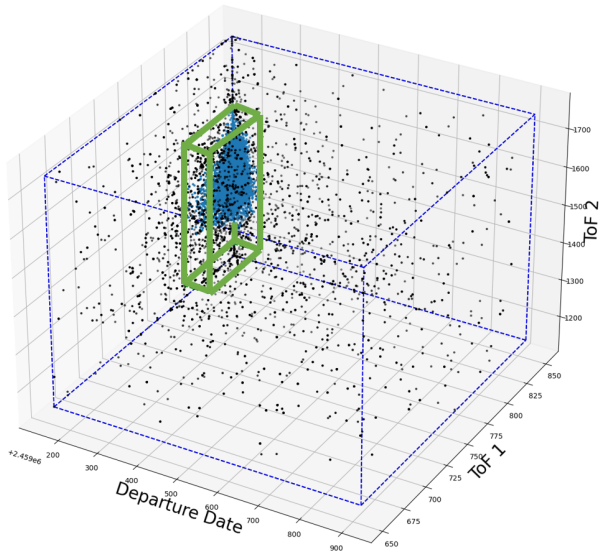

Figure 10.2: Indirect Method DE Cluster Found by DBSCAN

Since the clusters in both methods have been accurately bounded, the DBSCAN Algorithm has been verified to be a viable method of cluster identification. Once the bounds in the Departure Date and Times of Flight have been set, the rest of the optimization process deals with trajectories that launch, flyby, and arrive within these windows.

10.2 Direct Method Results

The Direct Method with Cluster Pruning produced trajectories with an average cost of 4,468 and maximum cost of 5,191. The average cost was 670 higher than runs without Cluster Pruning, with the maximum cost being 894 higher. Additionally, the standard deviation of the results were consistent, with the Cluster Pruned trajectories deviating by 470 and the original Python suite varying by 493. This trend, while suggesting that Cluster Pruning likely does not effectively improve the continuity of

solutions across the FBSM match points, does allow the suite to focus on areas smaller portions of the search space that likely contain the optimal solution. However, it is important to note the consistency of the cluster locations in the Differential Evolution Algorithm.

Figure 10.4 shows a solution proposed by MGALT-STOpS using Cluster Pruning, and Figure 10.3 shows the correlating cluster found in the Differential Evolution Algorithm. Note that the cluster is located in the top left corner of the Departure Date and Times of Flight search space. Every final solution returned from the Direct Method with Cluster Pruning reported an obvious cluster in this general area, hinting that there might be some validity to the idea of a synodic "sweet-spot" existing in the search space. However, it is also important to note that isolating the cluster by varying the Departure Date and Times of Flight windows was difficult, as other clusters began to form when the search space was changed. These other clusters reported similar final costs, so the question of whether the clustering is really due to the existence of synodic periods remains unanswered.

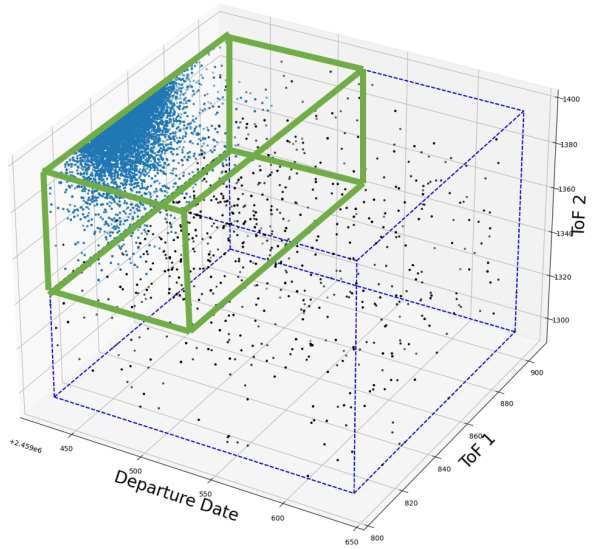

Figure 10.3: Direct Method Cluster Found with Cluster Pruning

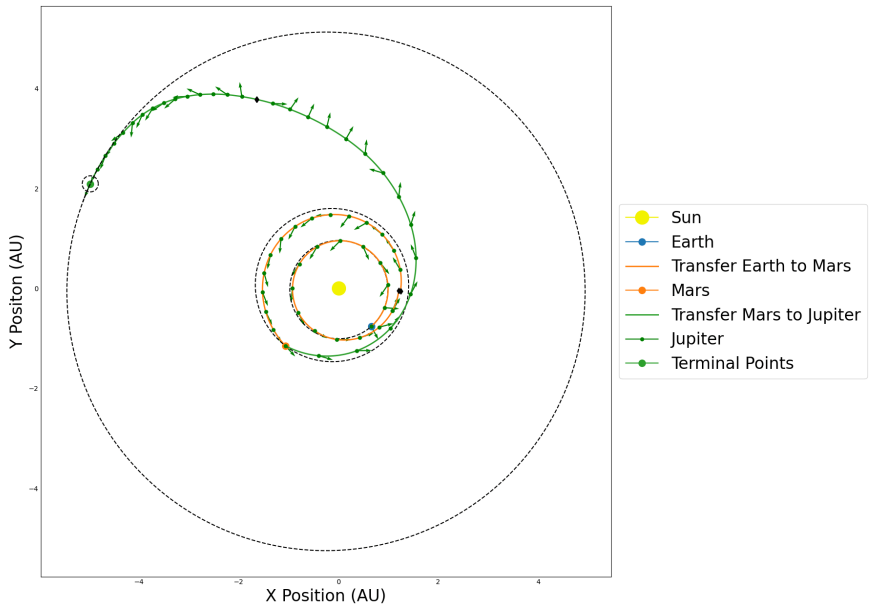

Figure 10.4: Direct Method Trajectory with Cluster Pruning

Cluster Pruning via the DBSCAN Algorithm on the Direct Method's Differential Evolution Algorithm data set proved consistent, but not particularly advantageous to a potential user. However, because of the tendency to find clusters in the same general location in the search space, it is still recommended to the user as a way to learn and visualize where optimal trajectories might exist. In the future, if cost function terms are added to provide more context on what makes a trajectory optimal, the clustering behavior might change in a way that is beneficial to a preliminary trajectory designer.

10.2.1 Cluster Pruning with Thruster Angle Pruning

Because the techniques used for Cluster Pruning and Thruster Angle Pruning are mutually exclusive, it is possible for the user to activate both for a given Direct

Method run. When using both performance improvements, the average cost over the five (5) runs was 4,831 with a maximum cost of 5,762. Interestingly, these results seem to combine the cost characteristics seen from Cluster Pruning and Thruster Angle Pruning. The average cost was only 59 higher than the Thruster Angle Pruned runs, which with the small sample size of five (5) runs is marginal. Additionally, the standard deviation of these results was 494, which is 4 higher than the original suite and 24 higher than Cluster Pruning only. This means that the same conclusions can be drawn from these tests as when the suite is run with only Cluster Pruning or Thruster Angle Pruning; the costs might rise, but the trajectories are still valid and it is beneficial for the user to regulate the spacecraft's attitude performance and see better potential Departure Date and Times of Flight windows.

The trajectories produced by MGALT-STOpS showed the same behaviors as those that only used Cluster Pruning or Thruster Angle Pruning. Figures 10.5-10.7 show the trajectory, cluster, and thruster profile, respectively, for one of the runs. Interestingly, it appears that when using Cluster Pruning and Thruster Angle Pruning together, the suite tends toward finding a cluster in the same location in the search space consistently. The cluster that appears in Figure 10.6 also appears in all five (5) runs, which implies that Thruster Angle Pruning does constrain MGALT-STOpS to certain areas of the Departure Date and Times of Flight search space. This was one of the hopes of Thruster Angle Pruning, as it was believed that removing sub-optimal thruster behavior would push the suite to find more optimal Departure Dates and Times of Flight. The consistent clustering location seen in these runs is also close to the optimal solution provided by Malloy [3] and literature [4], meaning that the combination of the two performance improvements does push MGALT-STOpS towards the known optimal solution.

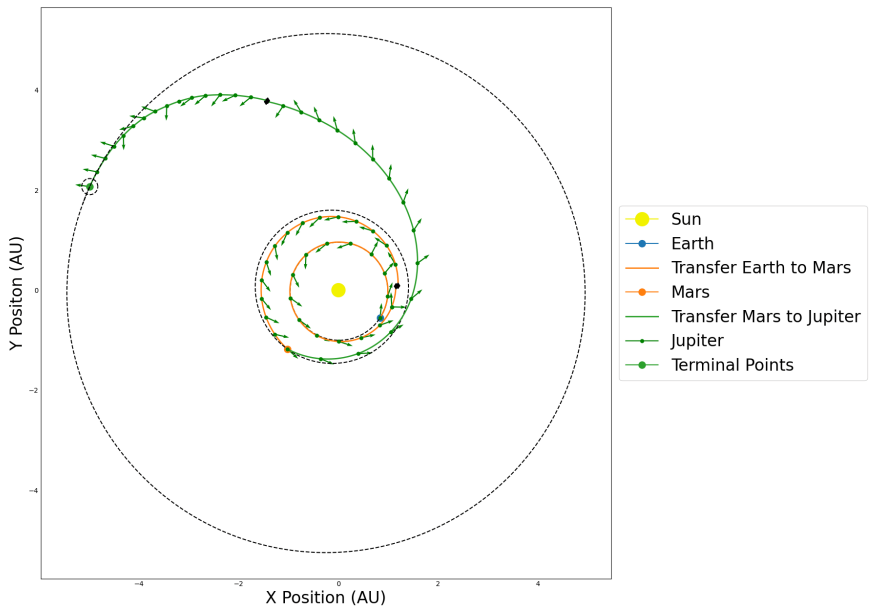

Figure 10.5: Direct Method Trajectory with Cluster Pruning and Thruster Angle Pruning. This cluster appeared consistently in all Cluster and Thruster Angle Pruned trajectories.

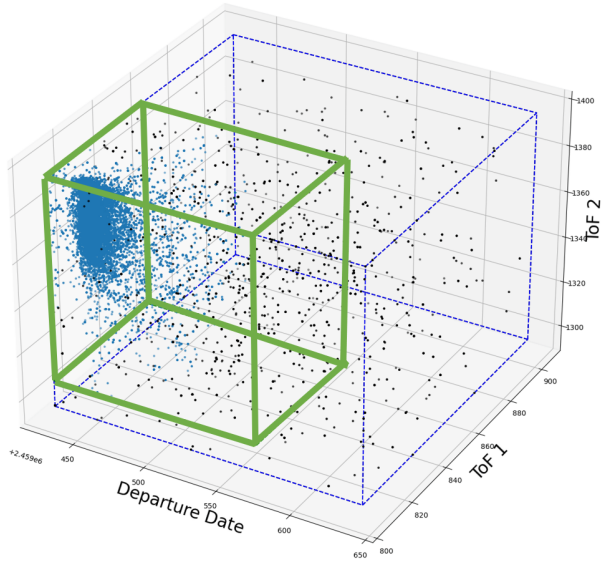

Figure 10.6: Direct Method Cluster Pruning and Thruster Angle Pruning Cluster

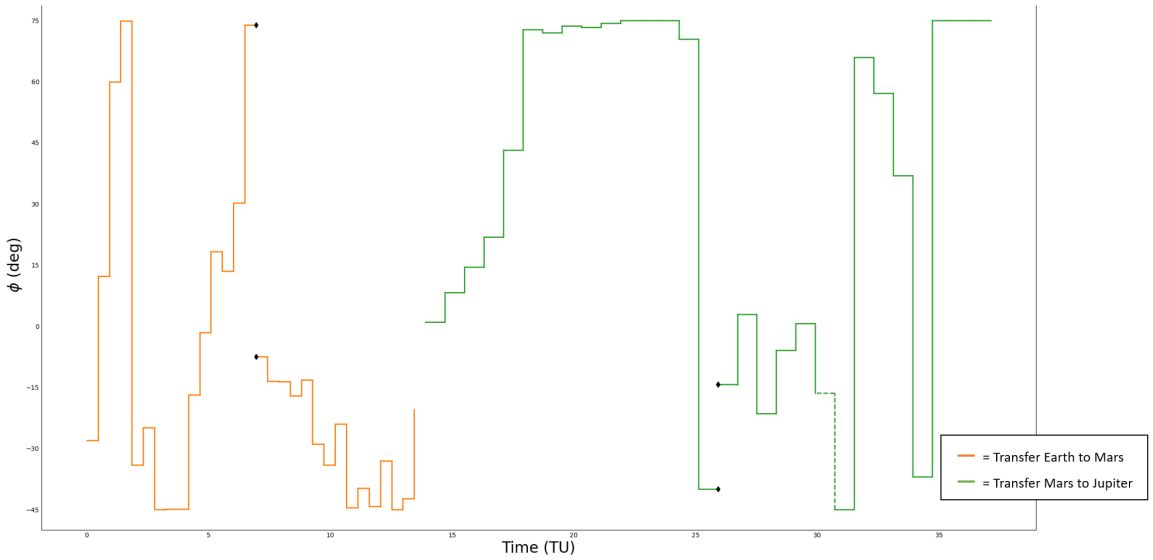

Figure 10.7: Direct Method Cluster Pruning with Thruster Angle Pruning Thrust

The thruster profile is also consistent with previous thruster profiles seen thus far, meaning that the use of Cluster Pruning does not impact the performance of the Thruster Angle Pruning. In this example, the thruster angles do vary a bit more than others, but this is not consistent across the board for all runs under these conditions. Therefore, it is a valid assumption that Cluster Pruning and Thruster Angle Pruning for the Direct Method act independently of each other.

Although the combination of Cluster Pruning and Thruster Angle Pruning does not result in decreased costs, it does provide the user with an increased chance of finding solutions close to the optimal ones found by Malloy [3] and literature [4]. Therefore, the combination of Cluster Pruning and Thruster Angle Pruning is verified as a viable option for the potential user to refine their Departure Date and Times of Flight windows as well as provide more attitude control compliant thruster profiles.

10.3 Indirect Method Results

When the Indirect Method was run with Cluster Pruning, the resulting trajectories showed some marginal cost improvement. In total, the average cost of the five (5) runs was 2,320, which was about 217 better than the original Python suite. Additionally, the highest single cost with Cluster Pruning was 4,624, while the highest without it was 5,558. The fact that the average and maximum costs are both lower is evidence that Cluster Pruning does provide useful search space pruning, although the reasoning behind that trend is unclear.

The original goal of Cluster Pruning was to be able to find the synodic periods of the planets for a given mission. This, in theory, would assist MGALT-STOpS in finding an optimal solution. However, it is difficult to determine whether or not this is truly happening in the Indirect Method. For example, the locations of the cluster in the Departure Date and Times of Flight search space are not the same every time. Figure 10.8 shows a cluster that is near the middle of the search space. In fact, this cluster encompasses the final solution given by Malloy [3] in his thesis. The final trajectory for this run, shown in Figure 10.9, is very similar. The final cost of this solution was 963.54, where 331.94 was from the Earth to Mars leg and 631.59 was from the Mars to Jupiter leg. Overall, this is a very low cost solution with relatively equally good transfer legs. On the other hand, Figure 10.2, previously shown in Section 10.1, shows a cluster appearing in a completely different location in the search space. Coincidentally, this run was actually even more optimal, totalling a cost of 782.82 with an Earth to Mars cost of 706.23 and Mars to Jupiter cost of 76.59.

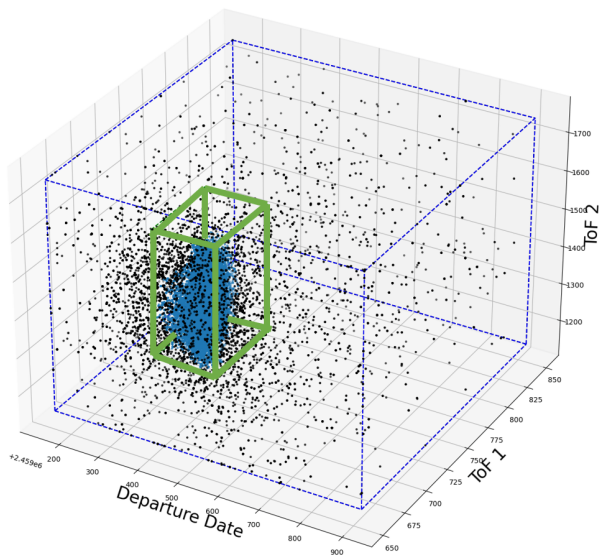

Figure 10.8: Indirect Method Cluster Found with Cluster Pruning. This cluster contains in its bounds the optimal solution found by literature.

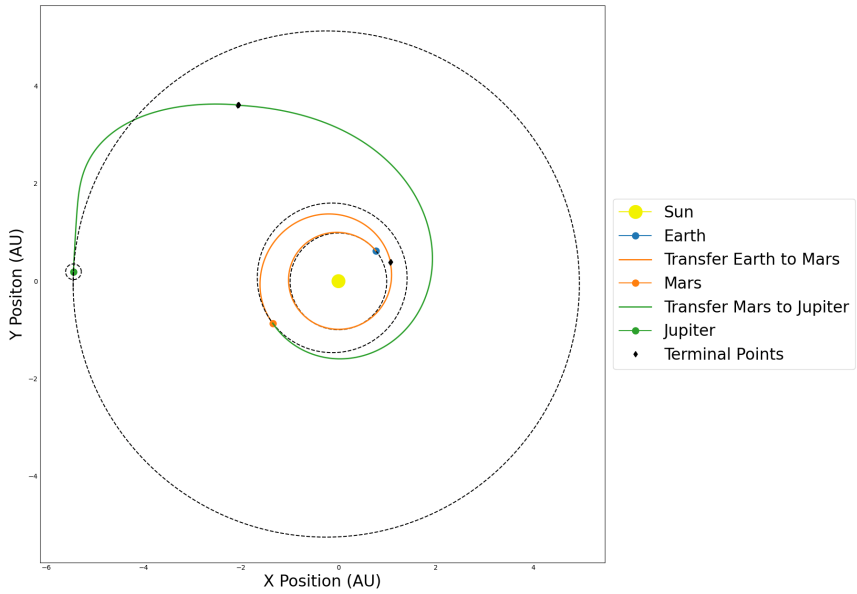

Figure 10.9: Indirect Method Trajectory with Cluster Pruning

The discrepancy between the locations of the clusters in the previously mentioned runs highlights the idea that Cluster Pruning via the Differential Evolution Algorithm might not be the most consistent way of identifying clusters due to synodic periods. After all, the Differential Evolution Algorithm only seems to target one cluster per run even though it has been shown that well optimized solutions can exist in very different parts of the search space.

In the end, Cluster Pruning for the Indirect Method has shown promise as a way to bound well-optimized solutions and allow MGALT-STOpS to focus on a small portion of the search space. Even though the locations of the clusters can vary, solutions have been shown to average lower costs, likely due to the extended time spent on a smaller search space volume. In the future, if an additional cost function were to be added to rate characteristics of solutions besides the continuity of the FBSM, clusters might become more consistent.

10.4 Final Remarks

The DBSCAN Algorithm proved to accurately identify clusters in the Differential Evolution data set on a very consistent basis. With the final solution always occurring within the bounds of the cluster, it is recommended that the user to implement Cluster Pruning if desired. However, it is not expected that using Cluster Pruning will significantly improve the final solution, if at all. At a minimum, it can inform the user where clusters exist and then they can alter their Departure Date and Times of Flight window accordingly. Additionally, the user may activate Cluster Pruning alongside Thruster Angle Pruning without fear of jeopardizing their results, although it is recommended to only activate Thruster Angle Pruning when trying to refine a trajectory in an already reduced search space.

Chapter 11

BEST LEG TRACKING VERIFICATION

The purpose of Multi-Objective Optimization is to allow MGALT-STOpS to handle multiple cost functions. While the approach described in this work is not truly Multi-Objective Optimization, it serves as a valid method of reintroducing well-optimized legs that may have been removed from populations too early. In future works, modifying the algorithms to handle multiple cost functions is strongly recommended. This is discussed more in Sections 12.2.1 and 12.2.2. This chapter will focus on the verification of Best Leg Tracking, along with the verifications of the combinations of Best Leg Tracking, Thruster Angle Pruning, and Cluster Pruning. For this chapter, the verification criteria are as follows:

1. Final solution costs shall be under 10,000.
2. Final solutions shall exhibit behaviors previously seen in MGALT-STOpS.
3. Final solutions shall have costs consistently lower than solutions without Best Leg Tracking.

11.1 Direct Method Results

Best Leg Tracking for the Direct method did reduce the costs of trajectories, averaging a cost of 3,107 with a maximum cost of 5,273. These were 691 lower and 976 higher than the respective values for the original Python suite, which implies that Best Leg Tracking alone can be a valid method of cost improvement. As a note, the highest run alone raised the average of the Best Leg Tracked solutions by 541. The

next worst cost was 3,616, which was actually lower than the average cost seen in all of the original Python Direct Method runs. As will be discussed in the following paragraphs, this means that Malloy [3] was correct when he mentioned the possibility of near optimal transfer legs being rejected because of poorly optimized legs in the same population member. Figures 11.1-11.3 show the trajectory, thruster profile, and by-leg costs for the last migration, respectively, for one of the Direct Method Best Leg Tracking runs.

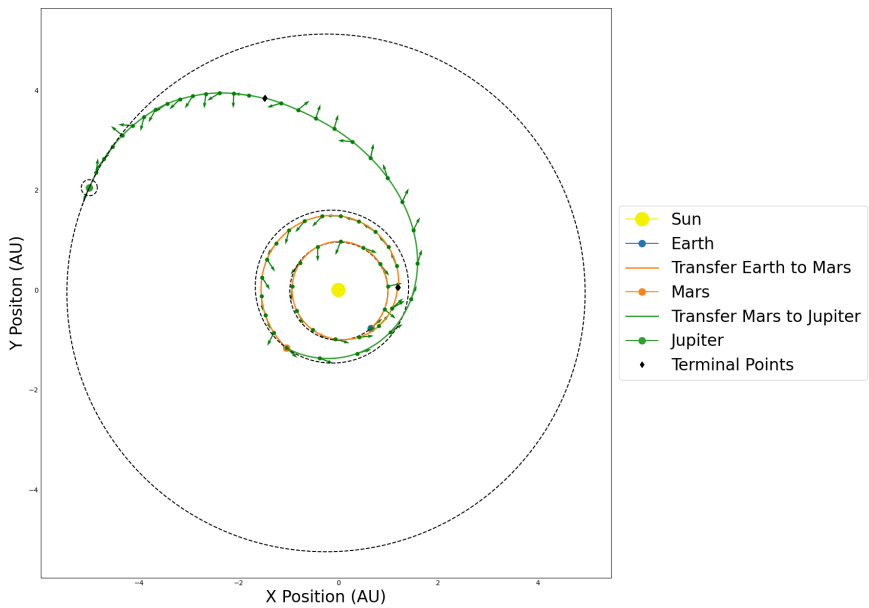

Figure 11.1: Best Leg Tracking Direct Method Trajectory

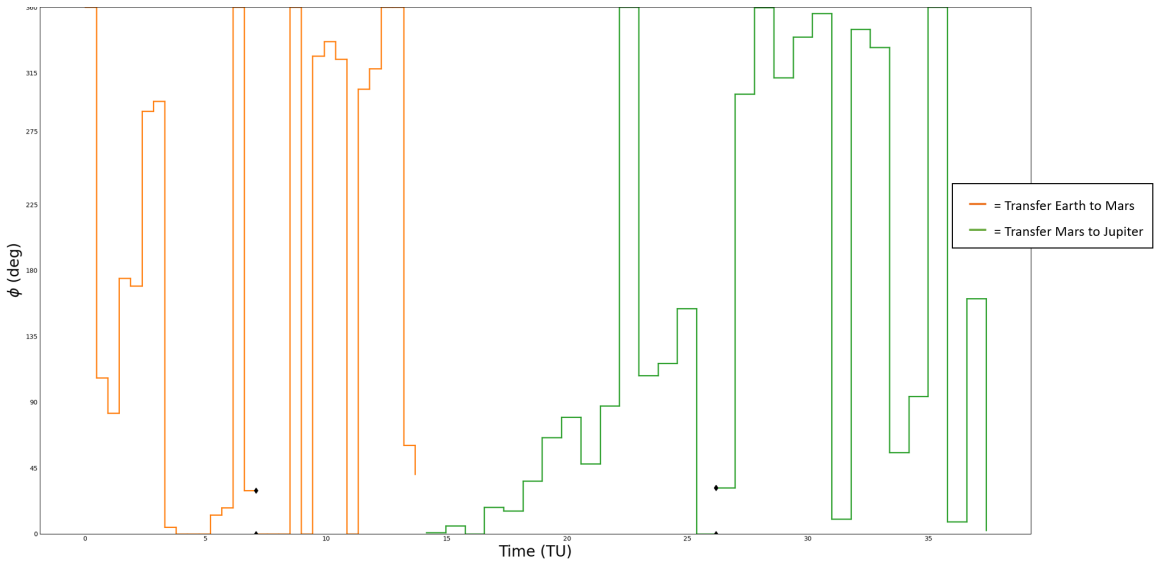

Figure 11.2: Best Leg Tracking Direct Method Thrust

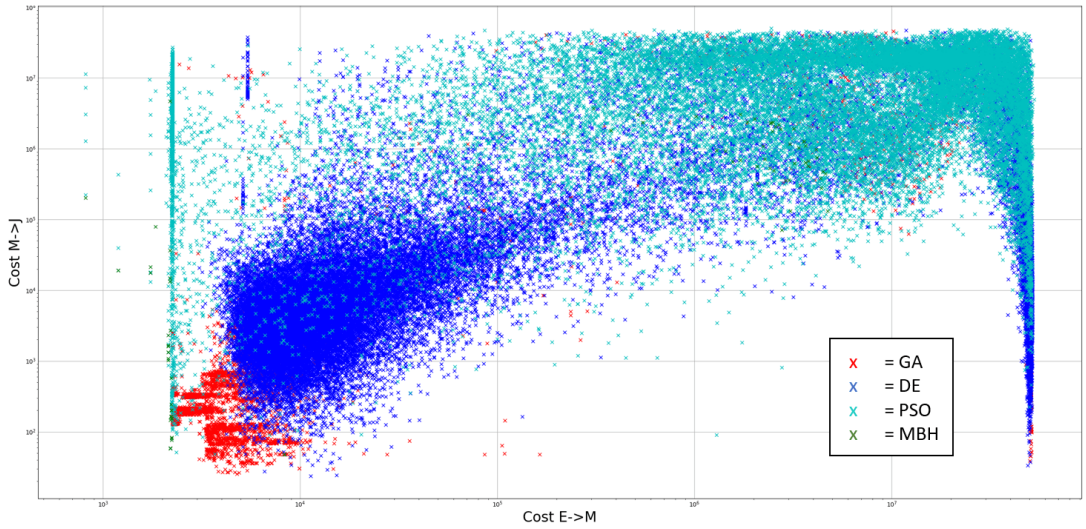

Figure 11.3: Best Leg Tracking Direct Method Leg Tracking. Vertical lines indicate substituted Earth to Mars legs. Horizontal lines indicate Mars to Jupiter leg Substitution.

While neither the trajectory nor thruster profile show anything unique, the by-leg costs show the Best Leg Tracking at work. Throughout the optimization process, the best 100 variable strings for each transfer are fed back into the populations. This is what causes the linear behavior in Figure 11.3. The vertical lines in the data represent promising Earth to Mars solutions being repeated, while the horizontal lines show the Mars to Jupiter solutions being repeated. In this specific example, it is apparent that there was one Earth to Mars leg in particular that showed promise and its corresponding Mars to Jupiter leg was continually improved. In fact, this solution ended up being the very close to the eventual solution, which was found by the MBH Algorithm slightly perturbing the solution to reach a basin close by.

There also exist some solutions to the left of the main vertical line, probably representing promising Earth to Mars legs that were found late in optimization. This assumption is based on the fact that they do not seem to be explored in depth. It is likely that these solutions needed one more migration to be fully examined. By that same thread, it is also worth noting that this work did not implement a check to prevent the same leg from being included in the top 100 multiple times. This decision was made because it seemed appropriate to allow the most promising solutions to be explored more over time; the more a leg is substituted in, the more spots in the top 100 it occupies. However, it is important to recognize that a leg cannot occupy the entirety of the top 100 unless it is currently the best leg known to the suite. That way there is always room for the suite to continue checking different promising solutions no matter how "lucky" one variable string might have been.

In the end, Direct Method Best Leg Tracking has shown to decrease total costs and continue to retain promising solutions that might have otherwise been prematurely pruned. This performance improvement is absolutely beneficial to a potential user,

and is recommended to be activated often. Therefore, Best Leg Tracking for the Direct Method is considered verified.

11.1.1 Best Leg Tracking with Thruster Angle Pruning

The effects of combining Thruster Angle Pruning and Best Leg Tracking were similar to the effects of adding Thruster Angle Pruning to the original Python suite. The costs were raised on average by a few hundred, returning the average cost to be 4,173. Similarly to the previous section, the highest cost was 6,019, which raised the average of these solutions by 462. Without this outlier, the average cost would have been on par with the original Python suite without any pruning. However, that does not mean that introducing Thruster Angle Pruning into Best Leg Tracked runs is not beneficial to the user. As can be seen in Figures 11.4 and 11.5, the thruster angles are mostly smoothly changing with some sub-optimal behavior still existing at the beginning of the trajectory. Even though the cost might be somewhat more with Thruster Angle Pruning activated, the original conclusion from Chapter 9 remains that Thruster Angle Pruning provides a more ADCS compliant trajectory that is still valuable to a user.

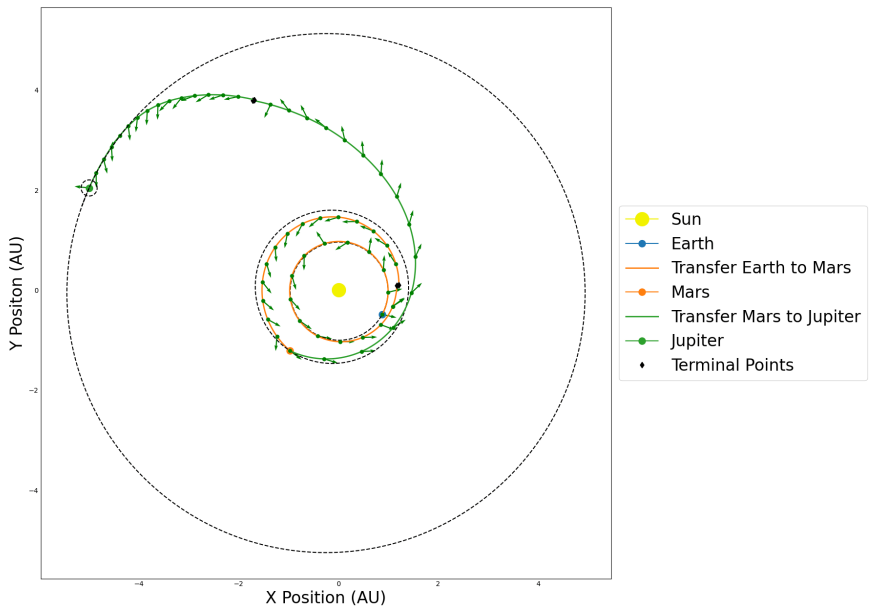

Figure 11.4: Direct Method Best Leg Tracking with Thruster Angle Pruning Trajectory

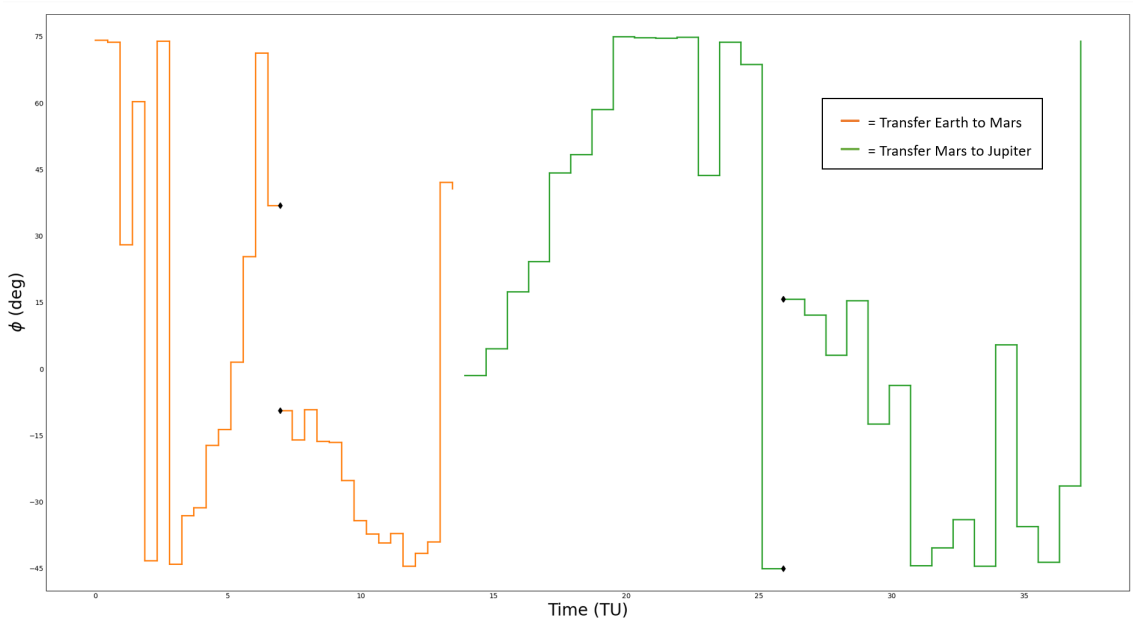

Figure 11.5: Direct Method Best Leg Tracking with Thruster Angle Pruning Thrust

Figure 11.6 shows the by-leg costs for this run. Interestingly, the effect of Best Leg Tracking is much harder to see, but it is assumed that the final solution provided by the Genetic Algorithm spike at the bottom is an extension of the vertical line above the Differential Evolution data. All of the runs for the combination of Thruster Angle Pruning and Best Leg Tracking had this same trend, so it is likely that Thruster Angle Pruning diminishes the cost lowering effectiveness of Best Leg Tracking. The mechanism behind this effect is not known for sure, but it is the assumption of the author that since the thruster angles are constrained, it is harder for the suite to find exceptionally lower costs to explore. There are a couple of solutions that appear in the top left and seem to be adequately explored, so it is likely that the algorithms simply could not find a matching Mars-Jupiter trajectory that made the total transfer costs competitive.

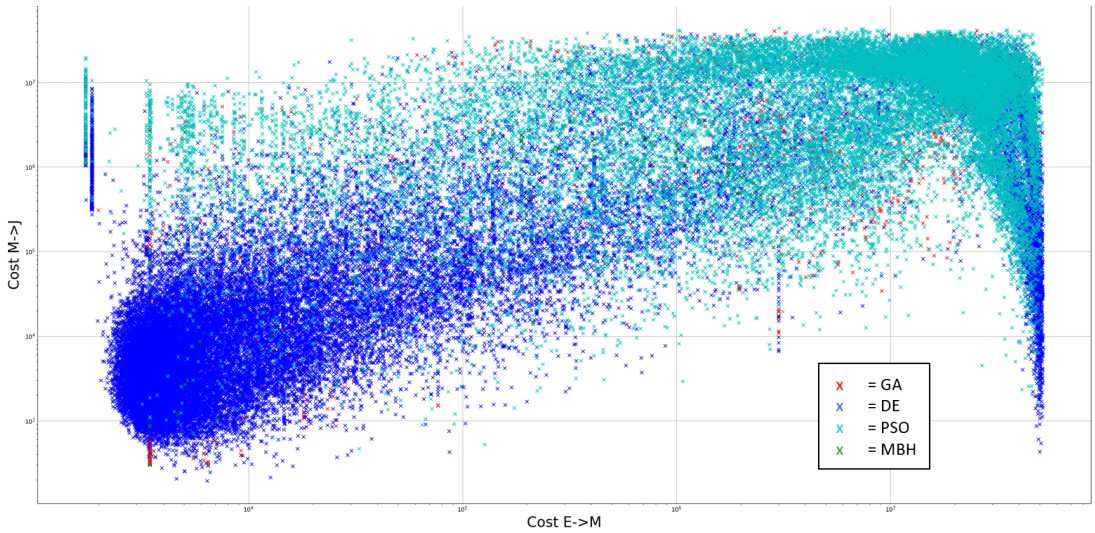

Figure 11.6: Direct Method Best Leg Tracking with Thruster Angle Pruning Leg Tracking. Based on the lack of obvious vertical and horizontal lines, the effects of Best Leg Tracking seem to be reduced.

While the costs of trajectories created by MGALT-STOpS with Thruster Angle Pruning and Best Leg Tracking enabled is generally about the same as the original Python suite, it is still beneficial to the user to implement both. However, it is recommended that the user first use only Best Leg Tracking as a method to reduce the Departure Date and Times of Flight windows before activating Thruster Angle Pruning. This way the suite will have a smaller search space and will likely find good trajectories with realistic thruster angles.

11.1.2 Best Leg Tracking with Cluster Pruning

The combination of Best Leg Tracking with Cluster Pruning resulted in runs with an average cost of 3,431 and a maximum cost of 4,733. Overall, these costs were largely consistent with the Best Leg Tracked only solutions. Following a similar thread to the previous sections, there was one solution that greatly raised the average cost even though it was still acceptable. Importantly, Best Leg Tracking with Cluster Pruning outperformed the solutions with only Cluster Pruning by averaging costs about 1000 less. Although the behavior of Best Leg Tracking for the Direct Method seems to occasionally produce results on par with solutions without Best Leg Tracking, it seems that the continuity of trajectories across the FBSM match points is greatly improved with this combination.

Importantly, the final trajectories and clusters were very similar to those produced when Best Leg Tracking and Cluster Pruning were implemented by themselves. All Departure Dates were around the beginning of August 2021 and the Times of Flight were around 820 days and 1395 days for Earth to Mars and Mars to Jupiter, respectively. The consistency between the methods working alone and together means that they can be used at the same time with no cost to the user. Figures 11.7-11.10 show

the trajectory, thrust profile, cluster, and by-leg costs during the final migration for one of the Direct Method runs using both Best Leg Tracking and Cluster Pruning.

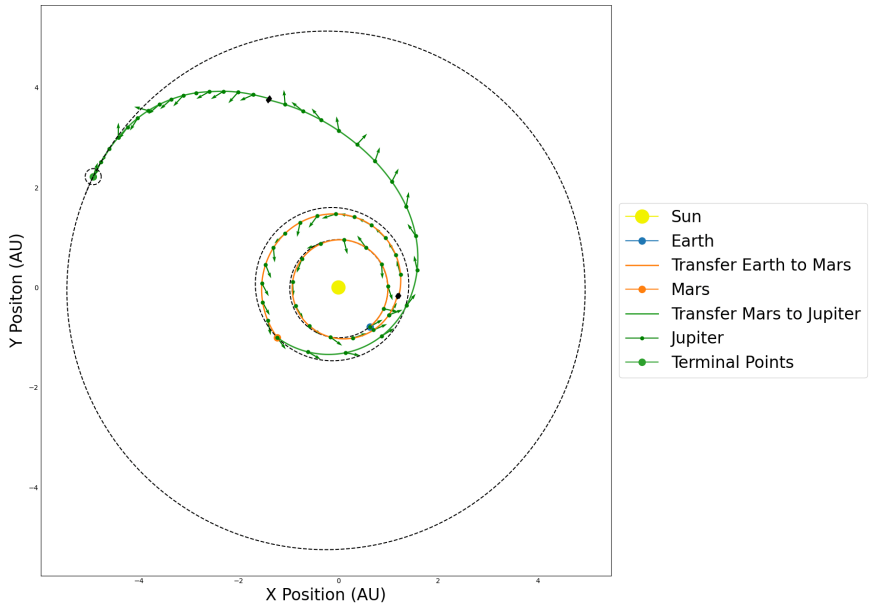

Figure 11.7: Direct Method Trajectory with Best Leg Tracking and Cluster Pruning

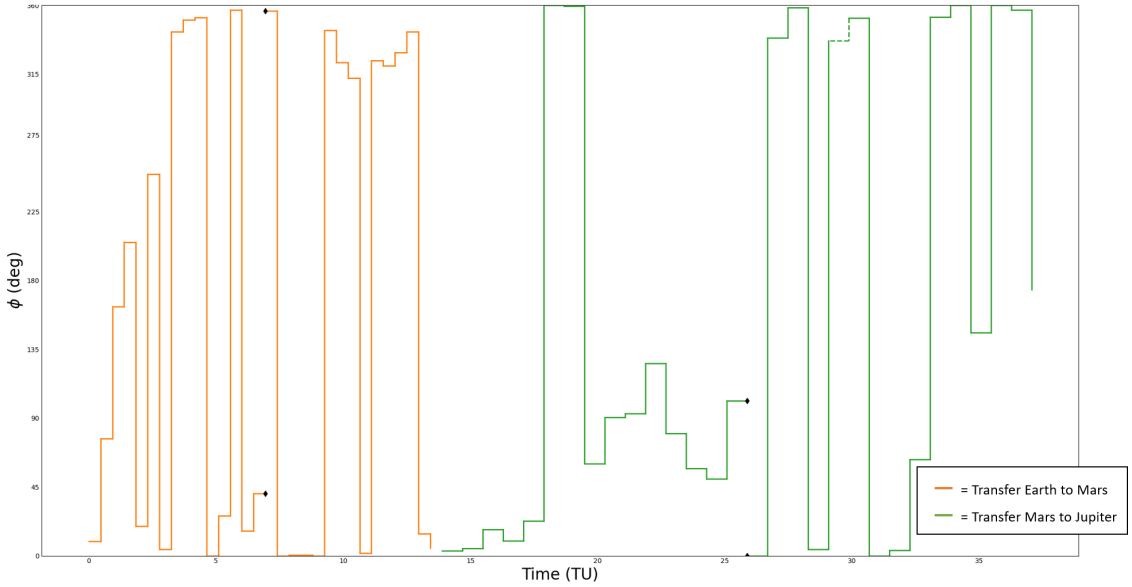

Figure 11.8: Direct Method Best Leg Tracking with Cluster Pruning Thrust

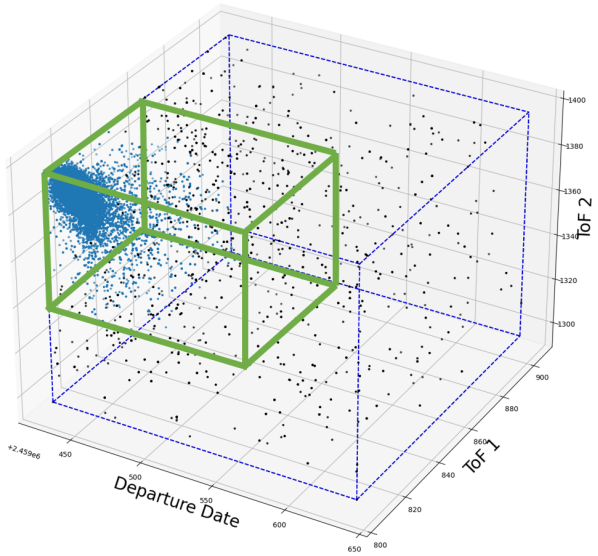

Figure 11.9: Direct Method Clustering with Best Leg Tracking and Cluster Pruning

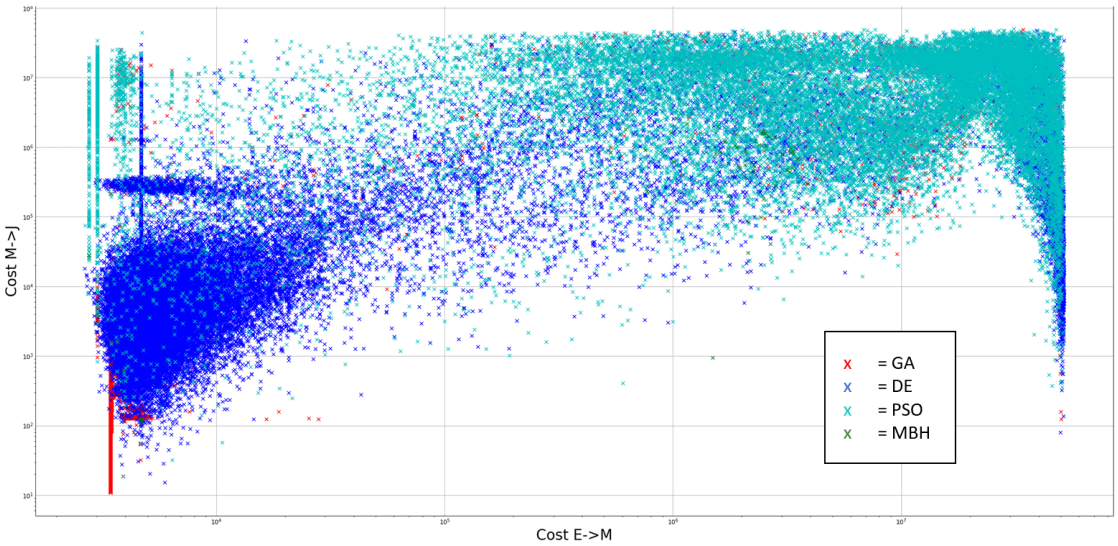

Figure 11.10: Direct Method Final Migration Costs with Best Leg Tracking and Cluster Pruning

From the figures, it can be seen that this trajectory seems like any other from these methods. The trajectory behavior is familiar, the cluster location is where it was with only Cluster Pruning activated, and the thruster profile is sporadic at times and often sub-optimal. However, the benefits of Best Leg Tracking are apparent as the Genetic Algorithm was able to converge on a solution that resulted in a Mars to Jupiter transfer cost of approximately 10 when the rest of the solutions were hovering between 100 and 10,000. It did not end up being the final solution as other population members existed with significantly lower Earth to Mars costs, but the Best Leg Tracking was able to significantly outperform the rest of the suite in this specific instance.

Because of the slightly decreased average cost when compared to the original Python suite, the user does not uniquely benefit from having both Best Leg Tracking and Cluster Pruning turned on for the Direct Method. However, because the methods are able to work in parallel, they still both have the potential to be helpful for the user. It has been shown that the Best Leg Tracking can still optimize transfer legs, although in this small sample size there was no instance of it leading to the optimal solution. Also, Cluster Pruning still offers the user an idea of where the optimal solution might lie, and therefore can let them reduce their Departure Date and Times of Flight search space accordingly.

11.1.3 Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning

Activating Best Leg Tracking, Thruster Angle Pruning, and Cluster Pruning resulted in solutions similar to those with just Thruster Angle Pruning and Cluster Pruning used together. With an average cost of 4,564 and maximum cost of 5,634, they resulted in a decrease in the average and maximum costs of 267 and 128, respectively. It is likely that the highly constrained search space due to the Cluster

Pruning and Thruster Angle Pruning reduces the effectiveness of Best Leg Tracking. However, the average cost of the solutions is still lower. Plus, the two best solutions using all three pruning methods resulted in costs between 3,000 and 4,000, while the best solutions without Best Leg Tracking had costs in the low 4,000's.

Figures 11.11-11.14 show the trajectory, thruster profile, clustering, and by-leg costs for one result of the combination of all three performance improvements. Overall, the trajectory displays a lot of the typical sub-optimal behavior such as undershooting Mars' orbit after the flyby and wildly varying thruster angles, particularly during the backwards shooting portion of the Mars to Jupiter transfer. This solution had a cost of 4,797, which is slightly higher than the average among this test case scenario.

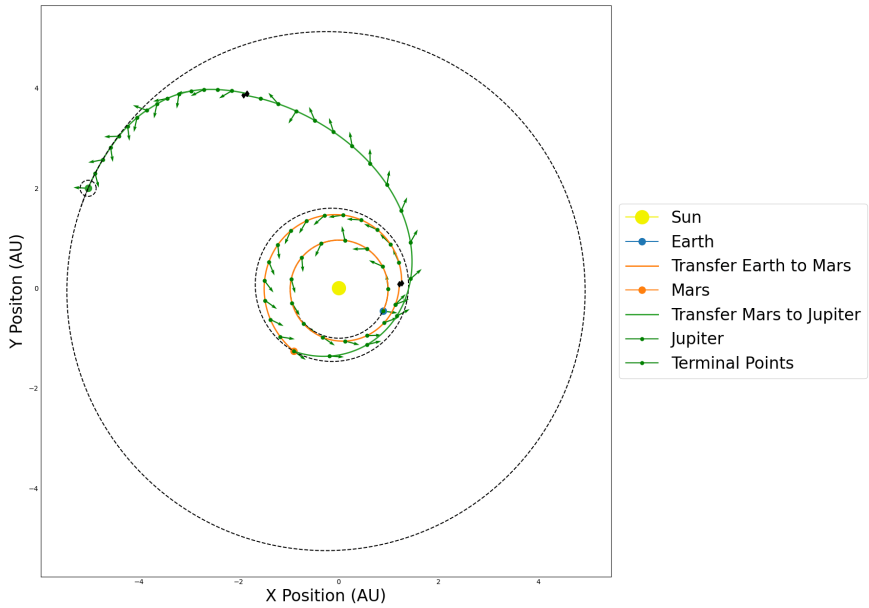

Figure 11.11: Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Trajectory

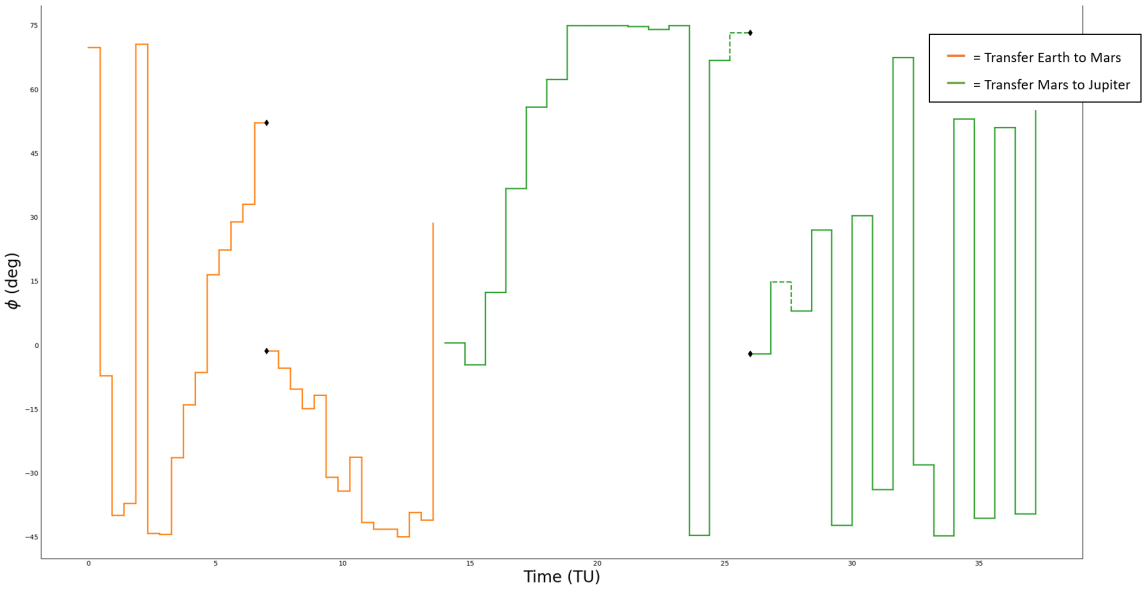

Figure 11.12: Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Thrust

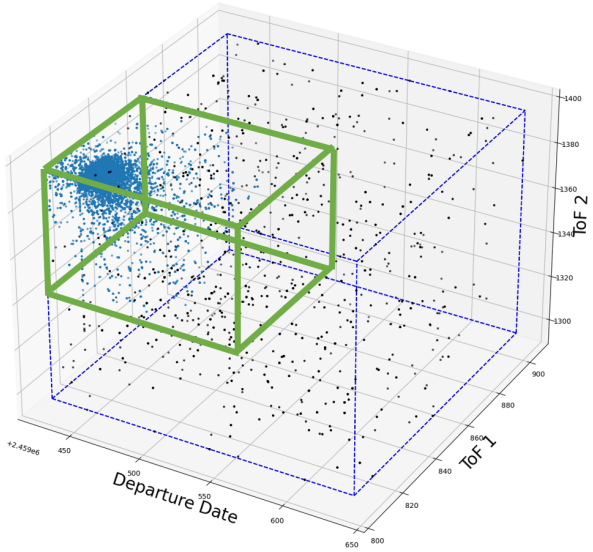

Figure 11.13: Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Clustering

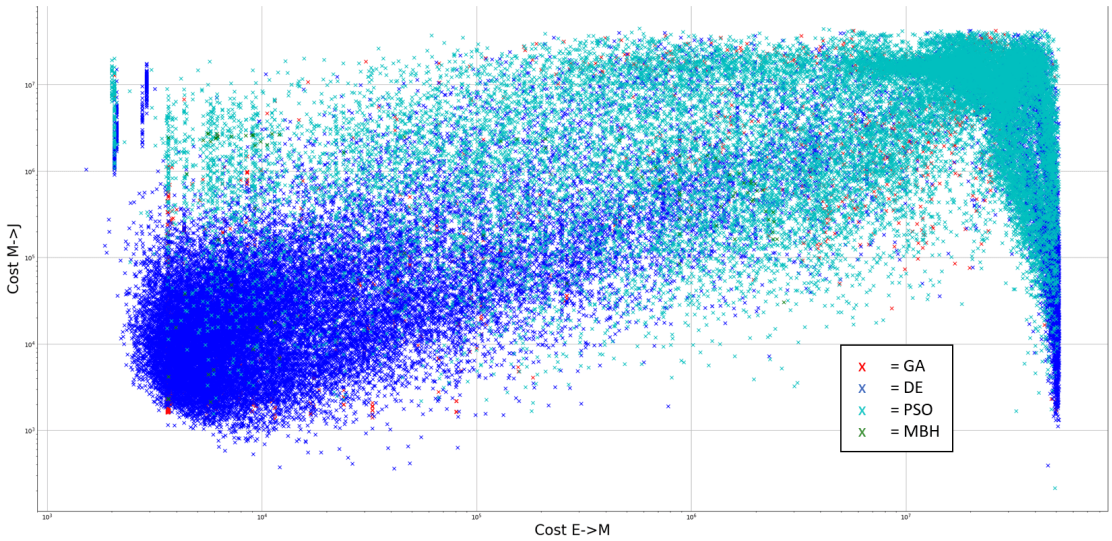

Figure 11.14: Direct Method Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning Final Migration Costs

From a cost perspective, using Best Leg Tracking along with Cluster Pruning and Thruster Angle Pruning seems to raise costs to be about 750 higher than the original Python suite. However, the solutions still benefit the user as the pruning methods continue to offer their unique advantages. Thruster Angle Pruning performed very well in providing a smooth thruster profile until the midpoint of the Mars to Jupiter transfer, and Cluster Pruning found the consistent cluster that has been mentioned previously for the Direct Method. Although it may affect the cost reducing ability of Best Leg Tracking, the combination of all three performance improvements is verified as beneficial to the user.

11.2 Indirect Method Results

Best Leg Tracking for the Indirect Method was by far the most beneficial single performance improvement seen in this work. The original Python suite produced trajectories averaging a cost of 2,537 with a maximum cost of 5,558. The trajectories with Best Leg Tracking, however, averaged a cost of 298 with a maximum cost of 473. That is an overall improvement of nearly one order of magnitude. Additionally, the trajectories seemed to very much favor the Departure Dates around the optimal solution provided in literature [4]. Figure 11.15 shows the by-leg costs of solutions for the last migration of the Python Indirect Method Verification run shown in Chapter 8, and Figure 11.16 shows the by-leg costs of solutions during the final migration for one of the Indirect Method runs with Best Leg Tracking. Figure 11.17 then shows the corresponding trajectory for the solution using Best Leg Tracking.

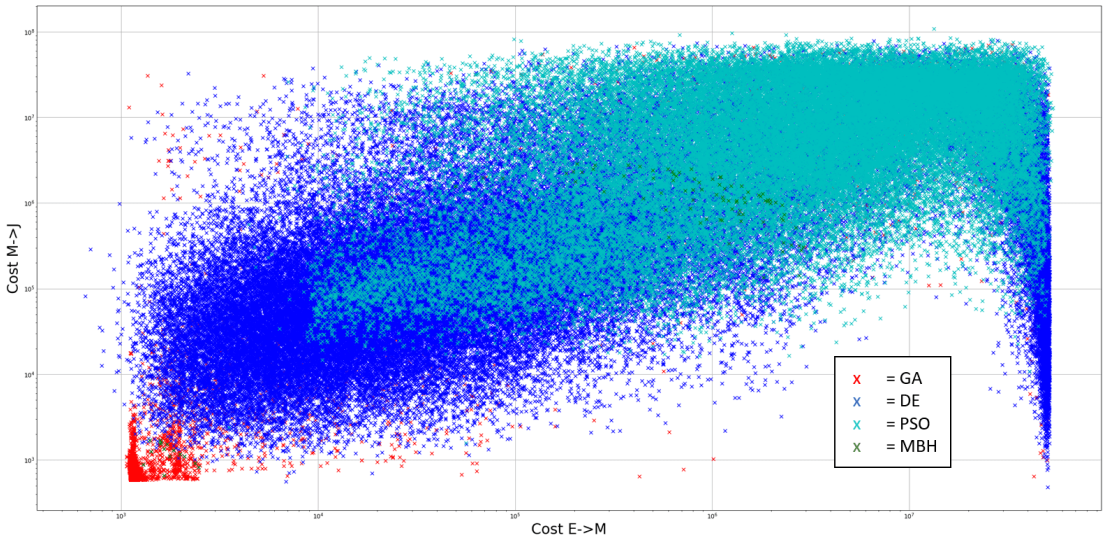

Figure 11.15: Indirect Method Final Migration Costs without Best Leg Tracking. Note the lack of exploration of low cost Earth to Mars solutions. These solutions were lost because of their poor Mars to Jupiter legs.

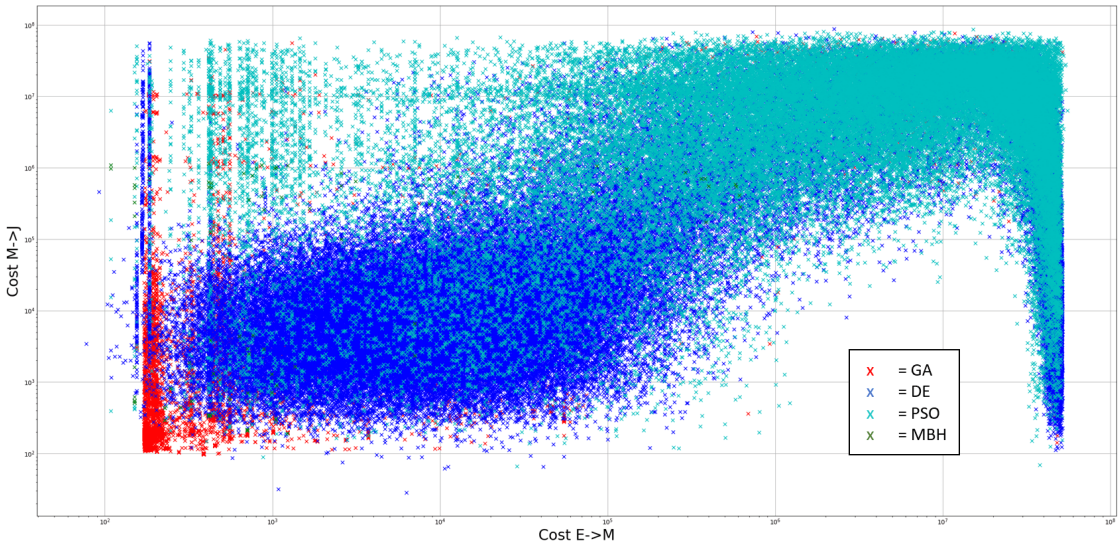

Figure 11.16: Indirect Method Final Migration Costs with Best Leg Tracking

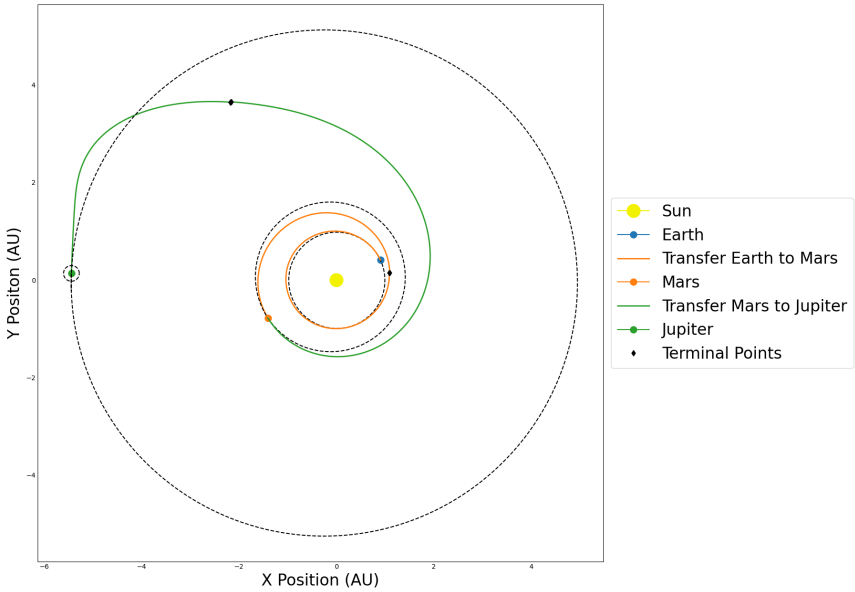

Figure 11.17: Indirect Method Trajectory with Best Leg Tracking

In comparing Figure 11.15 and Figure 11.16, it is shown that there is a stark difference between the behaviors of the suite as it tries to find an optimal trajectory. The vertical lines that appear in Figure 11.16 are the result of repeated attempts to optimize promising solutions found in the Earth to Mars transfer. This added behavior ensures that well-performing legs are never prematurely rejected from optimization. It is important to note that this behavior occurs throughout the optimization process, so if a promising solution is found very early on, it is explored thoroughly by the end of the run.

The author would like to note the lack of obvious horizontal lines, as in theory they would also appear in a similar fashion as the ones seen previously for the Direct Method. This is likely due to the fact that the Mars to Jupiter transfer is not a completely independent variable even though it does have its own independent variable string. Because the Earth to Mars leg controls the positions of the planets at the time of the Mars flyby, an optimal Mars to Jupiter leg will rely on that timing. If a Mars to Jupiter leg were fed into a solution where the Departure Date and Earth to Mars transfer Time of Flight is different than when it originally returned a low cost, the transfer will no longer be as optimal as it once was. This is likely what causes the higher volume of solutions that fall in the bottom right of Figure 11.16.

The much improved costs of the Indirect Method with Best Leg Tracking is evidence that Malloy [3] was correct in his belief that solutions were occasionally removed from the search space before being fully explored. Due to the smaller size of the Indirect Method's search space, the repeated substitution of the well-optimized legs proves to be extremely beneficial to the potential user. Therefore, Best Leg Tracking for the Indirect Method is verified as a performance improvement.

11.2.1 Best Leg Tracking with Cluster Pruning

Best Leg Tracking combined with Cluster Pruning for the Indirect Method produced very low cost results, but not quite as low with only Best Leg Tracking. These runs averaged a cost of 537, but that average is skewed high by one trajectory with a cost of 1,143. Without this outlier, the average would have been 385. Compared to Best Leg Tracking without Cluster Pruning, this is an increase in the average of 239. This is not a particularly concerning increase, but does raise some questions about the effectiveness of Cluster Pruning in tandem with Best Leg Tracking for the Indirect Method. Due to Best Leg Tracking's ability to improve solutions well into optimization, it is likely that restricting the Departure Date and Times of Flight search space by upwards of 90% early in the process is not optimal. However, it still remains that the solution costs are significantly lower than those without either Best Leg Tracking or Cluster Pruning.

With that statement in mind, Figures 11.18-11.20 show one particularly interesting solution found by MGALT-STOpS with Cluster Pruning and Best Leg Tracking activated. It might not seem out of the ordinary at first, but as shown by the Figure 11.18, this trajectory does not overshoot Jupiter. This means that it is possible for the Indirect Method to resolve the overshooting issue on its own, and in the future additional cost function terms could be added to make the suite prefer solutions that do not overshoot or undershoot planetary orbits. This behavior was only shown twice in the five (5) trajectories produced, but it is still notable behavior that is promising for the future of this program.

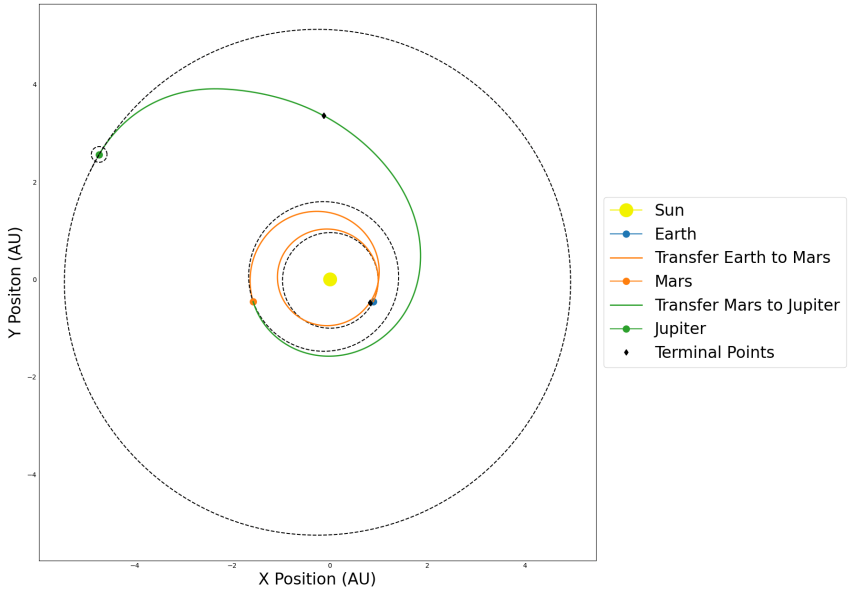

Figure 11.18: Indirect Method Trajectory with Best Leg Tracking and Cluster Pruning

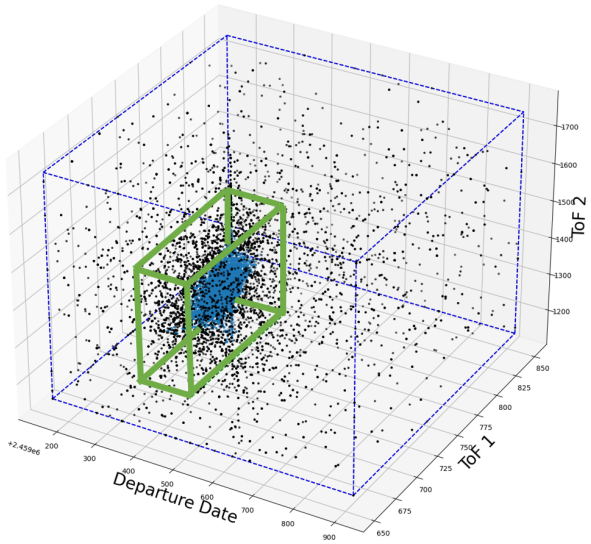

Figure 11.19: Indirect Method Clustering with Best Leg Tracking and Cluster Pruning

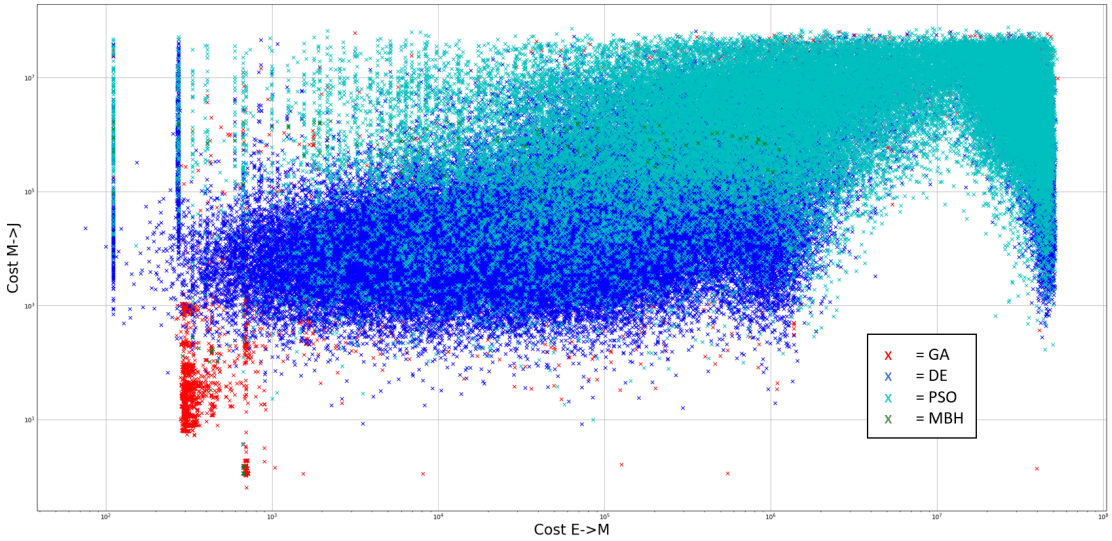

Figure 11.20: Indirect Method Final Migration Costs with Best Leg Tracking and Cluster Pruning

The clusters found when implementing Cluster Pruning with Best Leg Tracking were all in a similar vicinity, yet the actual Departure Dates and Times of Flight varied widely. This was peculiar as it demonstrated different behavior than when run with just Cluster Pruning alone. Originally, the clusters were found in very different locations, yet similar clusters yielded similar results. Now, however, similar clusters provided results that greatly differed in their reported optimal Departure Dates and Times of Flight. This could be an effect of a relatively small sample size, but it is likely that reducing the search space provided the suite with enough trajectories in a smaller space to find solutions that were different from the previously generated ones.

Even with the surprising variation in Departure Dates and Times of Flight, Best Leg Tracking with Cluster Pruning for the Indirect Method is still seen as a viable combination for a user. The Best Leg Tracking does most if not all of the significant cost reduction, and the Cluster Pruning informs the user of the likely location of an

optimal solution. Therefore, Best Leg Tracking with Cluster Pruning is considered verified.

11.3 Final Remarks

Best Leg Tracking proved to drop the average cost by a couple hundred for the Direct Method. However, because of the tendencies of Thruster Angle Pruning and Cluster Pruning, that decrease in cost was quickly nullified and the suite performed nominally with the original Python suite when combined with the other proposed performance improvements. Importantly, Cluster Pruning and Thruster Angle Pruning still performed well even though they slightly raised the cost. Therefore, given the user wants to use Cluster Pruning and/or Thruster Angle Pruning for the Direct Method, they should feel free to do so in tandem with Best Leg Tracking. For the Indirect Method, Best Leg Tracking showed vast cost reduction, lowering the average cost tenfold. With or without Cluster Pruning, it is seen as an almost mandatory option.

Chapter 12

CONCLUSION

12.1 Closing Thoughts

The implementation of four (4) performance improvements to MGALT-STOpS allows the user to access the suite for free, reduce run time, find trajectories with more ADCS compliance than before, refine their Departure Dates and Times of Flight, and lower trajectory costs. These all prove beneficial to a potential user, as these are all important steps in preliminary trajectory optimization. Tabulated summaries of solution characteristics for all performance improvements are shown in Tables 12.1 and 12.2.

Table 12.1: Python MGALT-STOpS Direct Method Cost Performance Breakdown

	No PI's	**TAP**	**CP**	**TAP & CP**	**BLT**	**BLT & TAP**	**BLT & CP**	**BLT, TAP & CP**
Average Cost	3,798	4,772	4,468	4,831	3,107	4,173	3,430	4,564
Max. Cost	4,297	6,111	5,191	5,762	5,273	6,019	4,733	5,634
Min. Cost	2,857	3,866	3,719	4,294	1,982	3,313	2,838	3,176
Std. Dev.	493	752	470	494	1,219	1,020	673	965

Table 12.2: Python MGALT-STOpS Indirect Method Cost Performance Breakdown

	No PI's	CP	BLT	BLT & CP
Average Cost	2,537	2,320	298	537
Max. Cost	5,558	4,624	473	1,143
Min. Cost	190	782	105	292
Std. Dev.	1,752	1,628	135	328

Porting the code from MATLAB to Python proved a great way for the user to reduce run time and access MGALT-STOpS as an open source suite. It allows the user to quickly input parameters and get solutions in a matter of a couple hours rather than the previous 5-10. The Python code was verified against the MATLAB code and showed the same behaviors and costs.

In this work, Thruster Angle Pruning was verified for the Direct Method as a viable option to find trajectories that showed more realistic variation in thruster angles over time. However, it could likely still be improved with the implementation of a maximum change in thruster angle between segments. This would remove the last bits of widely varying thruster profiles and force a smooth change over time. Because of the attitude control benefit, it is recommended that the user first find Departure Dates and Times of Flight that work for a potential mission before refining their thruster angles.

In order to find the best Departure Dates and Times of Flight, the user is recommended to use Cluster Pruning. While it does not always find clusters in the same locations every time, the clusters it does find prove to always contain the eventual solutions reported by the suite. Therefore, it was verified as it was deemed useful for the user to see where optimal trajectories likely lie on a timeline. In the future as cost

function terms are added, it is likely that the clusters will become more consistent as currently near-optimal clusters become penalized. When used together with Thruster Angle Pruning the final costs did rise, but the benefits of both the attitude control requirements and informing the user of useful Departure Dates and Times of Flight counteracted the elevated costs. Therefore, Cluster Pruning is recommended to be used both alone and together with Thruster Angle Pruning (Direct Method only).

The last individual performance improvement implemented was that of Best Leg Tracking. Best Leg Tracking proved somewhat beneficial for the Direct Method as it managed to reduce costs by recycling promising solution legs. Unfortunately, however, this benefit was shown to diminish as Thruster Angle Pruning and Cluster Pruning were used in tandem. For the Indirect Method, it consistently reduced the trajectory costs by around an order of magnitude. Therefore, it is strongly recommended as an option to reduce Indirect Method costs, with or without Cluster Pruning.

In the end, what makes an optimized trajectory "optimal" is up to the user's discretion. The performance improvements presented in this work all benefit the user in their own ways, and it is up to them to decide what they want to use. Thruster Angle Pruning, Cluster Pruning, and Best Leg Tracking simply act as tools to help the user decide when, where, and how they want to explore the solar system. As the suite is updated to include new cost function terms, remove 2D assumptions, add gravitational perturbations, and other improvements mentioned later in Section 12.2, MGALT-STOpS will improve in its ability to find trajectories for a variety of low thrust missions.

12.2 Recommendations

This section represents potential topics of future work that were thought of by the author while working on this project. They are roughly listed in the order of the author's perceived importance, and are NOT listed in the order of "ease of implementation." For additional insight on ideas for future work, see the MGALT-STOpS thesis by Malloy [3], the LT-STOpS thesis by Sheehan [2], and the original STOpS thesis by Fitzgerald [1].

12.2.1 Further Multi-Objective Optimization

The current form of Multi-Objective Optimization simply substitutes transfer legs back into a population. While this did improve MGALT-STOpS' performance, a future work could take a more involved Multi-Objective Optimization approach. Engineers, scientists, and researchers have developed techniques for algorithms to handle multiple cost functions at the same time, which would likely be ideal for the MGALT search space. Exploring the feasibility of this modification for all algorithms is strongly recommended. This, combined with the following subsection, would likely be a topic for a future thesis.

12.2.2 Additional Cost Function

In addition to the previous recommendation, it is apparent that MGALT-STOpS should use additional terms in the cost function. This could be in the form of more terms or a second cost function entirely. The current cost function only measures how well the Multiple Forward/Backward Shooting Method performs, and provides

no insight into how "flyable" a trajectory really is. Some variables that might be included are as follows:

- End-of-Transfer Propellant Mass Fraction
- Total Time of Flight (Although important, Low Thrust trajectories are not expected to be quick)
- Total change in thruster angles

The implementation of Multi-Objective algorithms is a precursor to the implementation of an additional cost function. This will likely be time consuming to weigh all variables appropriately and implement well.

12.2.3 3D Orbit Optimization

Currently MGALT-STOpS uses a 2-Dimensional approximation for its transfers. This is an appropriate assumption for a low-fidelity suite, but in the future should be adapted into 3 dimensions. This would entail changing the equations of motion as well as adapting the variable strings to allow for 3-Dimensional thruster angles. This improvement will likely not be the subject of an entire thesis, but will be time consuming nonetheless.

12.2.4 Gravitational Perturbations

MGALT-STOpS currently uses a gravitational assumption called Patched Conics, where the only gravity that affects the spacecraft is that of the sun until it reaches a flyby. In the real world, gravitational perturbations from other planets will affect the trajectory of the spacecraft throughout its mission. Additionally, the implementation

of gravitational perturbations would act as a check to make sure that the spacecraft does not unintentionally fly too close (or through) planets along its journey. Adding this feature is not expected to be time-intensive, as it would only affect the equations of motion.

12.2.5 Non-Planetary Targets

Real-world missions do not only fly to other planets. Other potential targets could include asteroids, comets, Lagrange points, or even other non-planetary orbits. Allowing MGALT-STOpS to target non-planetary targets would serve as a major improvement to the versatility of the suite. In the future, users could identify target ephemeris data via input or by referring to a catalog such as the JPL catalog seen at https://ssd.jpl.nasa.gov/orbits.html. The implementation of this feature would likely take some time, however this work has already set up the ability to use the JPL data for planetary data.

12.2.6 Re-tuning Parameters

When initially coding the algorithms, Malloy [3], Sheehan [2], and Fitzgerald [1] decided on the best parameters by testing different combinations and evaluating their outputs. However, with the introduction of Best Leg Tracking, future works can make changes to the island parameters and directly see the effects throughout optimization. By using Best Leg Tracking, future works could potentially find more optimal algorithm parameters and converge on better final solutions. This task is not expected to be overly complicated, but it could take some time to run enough tests to make evidence-backed conclusions about the optimal island parameters.

By the same thread, the Forward/Backward Shooting Method was implemented and tested by Malloy but not modified by this work. Due to the complicated nature of low thrust trajectory optimization, it would be quite interesting to test the algorithms' performances when altering the 50/50 aspect of the FBSM. By changing the ratio of the forward to backward section time spans, there might be an optimal value that indirectly prunes out certain sub-optimal behaviors like over/under shooting planets.

12.2.7 Coding Improvements

Perhaps the most important improvement before publication of the suite is that of coding efficiency. Currently, MGALT-STOpS and this work were written by Aerospace Engineers who (hopefully) coded for readability instead of efficiency. Re-coding this suite with efficiency in mind would likely save a lot of run time, allowing for users to optimize their trajectories without leaving their computers for an hour or three. One suggestion would be to keep the user interface in Python while outsourcing some of the algorithms and propagation to a faster language.

12.2.8 Deep Space Flyby

The Multiple Forward/Backward Shooting Method currently sets the arrival planet as the end of the journey for the spacecraft. However, real missions such as the Voyager Probes, for example, continued out into space after flying by their final planet. To improve the versatility of the suite, it would be beneficial to allow for the arrival planet to act as a flyby planet.

12.2.9 Powered Flyby

Real-world missions often use powered flybys as it allows a spacecraft to gain a much larger velocity boost than when using an unpowered flyby. Currently MGALT-STOpS only performs instantaneous, unpowered flybys. STOpS-FLITE [24] is a previous implementation of high thrust powered flybys on the impulsive side of STOpS, done by Aaron Li. This work can likely be adapted into MGALT-STOpS, although a low thrust powered flyby optimizer/propagator will most definitely be its own thesis. In addition, low thrust powered flybys might also have the constraint of requiring flybys to be sun-side due to the high power draw of the thruster. This is a specific case of Additional Flyby Requirements covered in Section 12.2.12.

12.2.10 B-Plane Targeting

B-Plane Targeting is a method of calculating flybys in 3-Dimensional space. Real-world missions will design flybys and arrivals using B-Plane Targeting to achieve optimal pre-arrival and post-flyby trajectories. A previous thesis by Jared Graef [23] titled "B-Plane Targeting with the Spacecraft Trajectory Optimization Suite" has already implemented B-Plane Targeting on the impulsive side of STOpS, but it is not yet implemented in MGALT-STOpS due to the current 2D assumptions. Implementing 3D trajectory optimization capabilities are a prerequisite for this topic.

12.2.11 Re-Implementation of Forced Heliocentric Orbits

While this work greatly focused on introducing new methods of search space reduction, there was an early form that existed in LT-STOpS by Sheehan [2]. Sheehan added a feature that allowed the user to force a certain number of heliocentric orbits

over the duration of the trajectory. This forced LT-STOpS to only try to optimize a small portion of the search space that satisfied this requirement. While working on this project, the author chose not to reintroduce this feature due to the similarities between it and Thruster Angle Pruning. It was decided before implementation that Thruster Angle Pruning would likely yield better results. However, it may serve well to reduce the search space of the Indirect Method and guide it toward trajectories more likely to fly in the real-world. Implementation would likely require the user to prompt the number of preferred heliocentric revolutions for each transfer individually. Additionally, this constraint will need to be carried along throughout the optimization process, potentially embedded within the member variable strings themselves. In his future work, Malloy [3] recommended adding "a binary switch followed by a number, to indicate if a heliocentric orbit is supposed to take place followed by the number of orbits."

12.2.12 Additional Flyby Requirements

In addition to Section 12.2.8 and Section 12.2.9, real-world trajectories are often constrained by their mission goals. For example, spacecraft equipped with newly developed cameras or sensors might require sun-side flybys in order to achieve mission success. Such a mission might also require specific planetary locations during flybys such that the sun will not interfere with communications or scientific data. Currently, there is no functionality in MGALT-STOpS to include real-world constraints such as these. The implementation of this feature would likely be associated with a large cost penalty for trajectories that violate these user-defined constraints.

BIBLIOGRAPHY

[1] T. J. Fitzgerald. Spacecraft Trajectory Optimization Suite (STOpS): Optimization of Multiple Gravity Assist Spacecraft Trajectories Using Modern Optimization Techniques. Master's Thesis, California Polytechnic State University, San Luis Obispo, Dec. 2015.

[2] S. P. Sheehan. Spacecraft Trajectory Optimization Suite (STOpS): Optimization of Low-Thrust Interplanetary Spacecraft Trajectories Using Modern Optimization Techniques. Master's Thesis, California Polytechnic State University, San Luis Obispo, Sept. 2017.

[3] M. G. Malloy. Spacecraft Trajectory Optimization Suite (STOpS): Design and Optimization of Multiple Gravity-Assist Low-Thrust (MGALT) Trajectories Using Modern Optimization Techniques. Master's Thesis, California Polytechnic State University, San Luis Obispo, Dec. 2020.

[4] C. H. Yam, T. T. McConaghy, K. J. Chen, and J. M. Longuski, "*Design of Low-Thrust Gravity-Assist Trajectories to the Outer Planets," 55th International Astronautical Congress of the International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law*, 2004.

[5] Mars Exploration: Features.

[6] Samuel Maurus and Claudia Plant. Skinny-dip: Clustering in a Sea of Noise. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1055–1064, San Francisco California USA, August 2016. ACM.

[7] Shahryar Rahnamayan, Sedigheh Mahdavi, Kalyanmoy Deb, and Azam Asilian Bidgoli. Ranking Multi-Metric Scientific Achievements Using a Concept of Pareto Optimality. *Mathematics*, 8:956, June 2020.

[8] Dario Izzo. Revisiting Lambert's Problem. *Celestial Mechanics and Dynamical Astronomy*, 121(1):1–15, January 2015. arXiv:1403.2705 [astro-ph].

[9] L. Kos, T. Polsgrove, R. Hopkins, D. Thomas, and J. Sims, "Overview of the Development for a Suite of Low-Thrust Trajectory Analysis Tools," in AIAA/AAS Astrodynamics Specialist Conference and Exhibit, (Keystone, Colorado), American Institute of Aeronautics and Astronautics, Aug. 2006.

[10] Gerald A. Rauwolf and Victoria L. Coverstone-Carroll. "Near-optimal low-thrust orbit transfers generated by a genetic algorithm", Journal of Spacecraft and Rockets, Vol. 33, No. 6 (1996), pp. 859-862.

[11] Conway, Bruce A. Spacecraft Trajectory Optimization. Cambridge: Cambridge UP, 2014. Print.

[12] D. A. Vallado and W. D. McClain, *Fundamentals of Astrodynamics and Applications*. No. 21 in Space technology library, Hawthorne, Calif: Microcosm Press, 4. ed ed., 2013. OCLC: 869869951.

[13] Goldberg, David E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989. Print.

[14] Haupt, Randy L., and Sue Ellen Haupt. Practical Genetic Algorithms. 2nd ed. Hoboken, N.J: Wiley-Interscience, 2004. Print.

[15] Price, Kenneth V., Rainer M. Storn, and Jouni A. Lampinen. Differential Evolution: A Practical Approach to Global Optimization. Berlin: Springer, 2005. Print.

[16] Clerc, Maurice. Particle Swarm Optimization. London: ISTE, 2006. Print.

[17] J. Englander, B. Conway, and T. Williams, "Automated Interplanetary Trajectory Planning," AIAA/AAS Astrodynamics Specialist Conference, 2012.

[18] Kristina Alemany and Robert D Braun. DESIGN SPACE PRUNING TECHNIQUES FOR LOW-THRUST, MULTIPLE ASTEROID RENDEZVOUS TRAJECTORY DESIGN. page 13.

[19] Dario Izzo. Global Optimization and Space Pruning for Spacecraft Trajectory Design. In Bruce A. Conway, editor, *Spacecraft Trajectory Optimization*, pages 178–201. Cambridge University Press, 1 edition, August 2010.

[20] Curtis, Howard D. *Orbital Mechanics for Engineering Students*. 2nd ed. Amsterdam: Elsevier, Butterworth-Heinemann, 2010. Print.

[21] Elmar Puts. Cluster-based search space pruning in single leg low-thrust trajectory optimisation problems | TU Delft Repositories.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.

[23] J. Graef. B-Plane Targeting with the Spacecraft Trajectory Optimization Suite. Master's Thesis, California Polytechnic State University, San Luis Obispo, Dec. 2020.

[24] A. H. Li. Spacecraft Trajectory Optimization Suite: FLy-bys with Impulsive Thrust Engines (STOpS-FLITE). Master's Thesis, California Polytechnic State University, San Luis Obispo, Jun. 2022.

[25] Cal Poly Github. `http://www.github.com/CalPoly`.

[26] Jacob A Englander, Matthew A Vavrina, and David Hinckley. GLOBAL OPTIMIZATION OF LOW-THRUST INTERPLANETARY TRAJECTORIES SUBJECT TO OPERATIONAL CONSTRAINTS. page 20.

APPENDICES

Appendix A

USER GUIDE

Two (2) scripts are used to run MGALT-STOpS in Python. The first is "main.py" and the second is "STOpS.py." "main.py" is used to provide a user-friendly script that allows the user to directly call a specific case that has been created in "STOpS.py." Therefore, "STOpS.py" is where the user can create custom trajectories or run the test cases used in this work. The following sections include detailed documentation and discussion on options and parameters available in MGALT-STOpS.

A.1 Creating a Mission

Because each mission in Python is its own function in a class, the user can create a new scenario or modify an existing one. In order to create a new scenario, the user will scroll to the "presets" class in "STOpS.py" and define a new mission with the syntax shown in Figure A.1. From there, the user will define the options and parameters for the mission scenario as described in the following sections.

```
#%% Presets
class presets():

    def custom():
        bod = {}
        opts = {}

        return bod, opts
```

Figure A.1: Creating a Mission

A.2 Planets, Departure Date, and Times of Flight

The first step for the user is to define the planetary ephemeris data to be used by MGALT-STOpS. This is done by setting the opts[‘ephemeris’] variable. Currently, MGALT-STOpS can handle the inputs “JPL_Horizons” or “jplEphem.” However, it is strongly recommended to use “JPL_Horizons,” as the 2D approximation works better with that coordinate frame. When MGALT-STOpS starts to produce 3D trajectories, “jplEphem” will most likely be a better option.

The next step is to define the planets for a mission. This is done by setting the “bod[‘bodies’]” variable to be a tuple of the desired planets in chronological order of the mission. The currently supported options are the following: ‘Mercury’, ‘Venus’, ‘Earth’, ‘Mars’, ‘Jupiter’, ‘Saturn’, ‘Uranus’, ‘Neptune’, and ‘Pluto’. For an Earth-Mars-Jupiter trajectory, the tuple would be “(‘Earth’,‘Mars’,‘Jupiter’).”

Next up is the departure window. The user must set a lower and upper bound, correlating with the “bod[‘window1’]” and “bod[‘window2’]” variables, respectively. These variables must be in datetime format, so the syntax looks like the following: “bod[‘window1’] = datetime(YYYY,MM,DD)”.

The final settings to describe the trajectory options are the goal Times of Flight and their margins. MGALT-STOpS currently works by setting a target Time of Flight and then allowing the trajectories to explore options within a set margin. Both the goals and margins for Times of Flight are customizable for each leg of the trajectory. Therefore, the length of these variables must be one less than the length

of the bodies array. Below is an example for setting the Times of Flight or the same Earth-Mars-Jupiter-Saturn trajectory.

Goal ToF (days):

opts['tof_total'] = [(752),\
(1448)]

ToF Margin (days):

opts['tof_margin'] = [(100,100),\
(300,300)]

Once all of these settings are complete, the code should like Figure A.2

```
#%% Presets
class presets():

    def demo():
        bod = {}
        opts = {}

        opts['ephemeris'] = 'JPL_Horizons'

        bod['bodies'] = ('Earth','Mars','Jupiter')

        # Launch Window
        bod['window1'] = datetime(2020,11,1)       # earliest launch day    [year,month,day]
        bod['window2'] = datetime(2022,12,1)       # latest launch day      [year,month,day]

        # Time of Flight Options
        opts['tof_total'] = [(752),\
                             (1448)] # tof w/ gravity assists [days]

        opts['tof_margin'] = [(100,100),\
                              (300,300)] # margin on tof -/+     [days]

        return bod, opts
```

Figure A.2: Setting Bodies and Times of Flight

A.3 Solver Selection

The next steps in creating a mission is to define the solver. There are four options, all case sensitive. They are: 'LT_DIR_FSM_2D', 'LT_IN_FSM_2D', 'MGALT_DIR_FBSM_2D', and 'MGALT_IN_FBSM_2D'. While the two LT FSM methods are available, it is important to note that they are not MGALT solvers and can only be used for single leg trajectories. If these methods are input for an MGALT trajectory, the user will be prompted with an error message. This step is also where the user decides which variable transcription method they desire, as 'DIR' stands for Direct Method and 'IN' stands for Indirect Method. Whatever the user chooses, the desired solver must be set to the variable 'opts['solver']'. After this step, a custom mission should resemble Figure A.3.

```
#%% Presets
class presets():

    def demo():
        bod = {}
        opts = {}

        opts['ephemeris'] = 'JPL_Horizons'

        bod['bodies'] = ('Earth','Mars','Jupiter')

        # Launch Window
        bod['window1'] = datetime(2020,11,1)       # earliest launch day    [year,month,day]
        bod['window2'] = datetime(2022,12,1)       # latest launch day      [year,month,day]

        # Time of Flight Options
        opts['tof_total'] = [(752),\
                             (1448)] # tof w/ gravity assists [days]

        opts['tof_margin'] = [(100,100),\
                              (300,300)] # margin on tof -/+     [days]

        # Solver and Thrust Method
        opts['solver'] = 'MGALT_IN_FBSM_2D'

        return bod, opts
```

Figure A.3: Setting the Solver

A.4 Spacecraft Parameter Selection

The user now has the ability to set the desired spacecraft parameters, including the initial mass, thruster's specific impulse, thruster's mass flow rate, and most importantly the thrust method, among others. This is done by selecting one of the preset options (or creating one's own) in the 'optionsLowThrust.py' file. For a new user, it is recommended to first try one of the preset options, mainly 'Yam-STOUR' or 'Yam-GALLOP'. Note that all variable units are available at the top of the function in the comments.

While these options are fully customizable, there is one notable exception. The user must choose a thrust method that is consistent with the variable transcription method they have chosen. Therefore, if the user chooses the Indirect Method, it is imperative they also choose a thrust method that is 'constant'. The Direct Method has been shown in previous works [2, 3] to function best with 'variable' thrust, however it can also be used with the 'constant' thrust method. MGALT-STOpS also has the capability for 'solar_sail' thrust, which can conveniently be used by both the Direct and Indirect Methods. For a new user, if they wish to use the Indirect Method, it is recommended to try the 'Yam-STOUR' option. For the Direct Method, try the 'Yam-GALLOP' option. In order to call the thrust options, use the command seen in Figure A.4 where 'Yam-STOUR' is a placeholder for the intended spacecraft parameters case.

```
# Solver and Thrust Method
opts['solver'] = 'MGALT_IN_FBSM_2D'
opts['thrust'] = optionsLowThrust(opts,'Yam-STOUR')


return bod, opts
```

Figure A.4: Setting the Spacecraft Parameters

A.5 Island Model Parameters

Once the spacecraft parameters have been entered, the user can now select the Island Model parameters. This is done by visiting the 'optionsIsland.py' file. Similarly to 'optionsLowThrust.py', the user can decide to create new parameters or select the pre-existing options. The customizable options include the number of migrations, list of islands, island connectivity matrix, replacement policy, and selection policy. Instructions on how to format each variable is given in the document header. For reference, the Direct Method runs used in this work used the '3M-all' setting, while the Indirect Method runs used the '4M-all' setting. The decision to use these settings came from testing by Malloy [3] during his thesis. In the end, the command to use to set the island parameters is shown in Figure A.5, where '4M-all' is a placeholder for the user's selection.

```
# Solver and Thrust Method
opts['solver'] = 'MGALT_IN_FBSM_2D'
opts['thrust'] = optionsLowThrust(opts,'Yam-STOUR')

# Island Model
opts['island'] = optionsIsland('4M-all')


return bod, opts
```

Figure A.5: Setting the Island Model Parameters

A.6 Cost Function Parameters

The user can now change the cost function of the suite by visiting the 'optionsCost.py' file. There are two categories of cost functions depending on if the user selects the FSM or FBSM in their solver. Both take the exact same inputs, although they behave slightly differently in application due to their difference in propagation. There are two categories of inputs: tolerances and On/Off switches. The tolerances allow the user to emphasize how much weight is put on a specific term. The On/Off switches allow for the user to decide whether they want a specific term included in the cost function summation. The terms include the terminal point radial and angular positions of the spacecraft, along with terminal point radial and tangential velocities. Additionally, there is a time of flight term that penalizes a solution if it strays from the middle of the given time of flight window. The command for setting the cost function parameters can be seen in Figure A.6

```
# Solver and Thrust Method
opts['solver'] = 'MGALT_IN_FBSM_2D'
opts['thrust'] = optionsLowThrust(opts,'Yam-STOUR')

# Island Model
opts['island'] = optionsIsland('4M-all')

# Cost Function
[opts['cost'],opts['weighting'],opts['ode']] = optionsCost(opts,'default')



return bod, opts
```

Figure A.6: Setting the Cost Function Parameters

As a warning to the user, the current tolerances have been tested by Sheehan [2] and Malloy [3], and it is recommended to not change them significantly, as orders of

magnitude changes will likely disturb the balance they are currently in. Additionally, it is not recommended for the user to turn off any switches, but the author does recognize specific examples where it could be beneficial. For example, if the user does not care when in a time of flight window the trajectory lands, turning off the time of flight parameter could be useful. Also, for an FSM trajectory (this specific example does not apply to an FBSM trajectory), the user could turn off the angular position term if they only wish to arrive at a particular planetary orbit but not necessarily at the planet itself.

A.7 Algorithm Parameters

It is important for the user to have full knowledge of how to control each of the four (4) individual algorithms in the suite and tailor them for their needs. The following subsections will describe the options for each algorithm. Figure A.7 shows the syntax for calling the algorithm parameters.

```
# Algorithm Parameters
opts['GA'] = parametersGA('450_180')#     # Genetic Algorithm Island
opts['DE'] = parametersDE('450_180')#      # Differential Evolution Island
opts['PSO'] = parametersPSO('350_150')#     # Particle Swarm Island
opts['MBH'] = parametersMBH('15000_1500')#     # Particle Swarm Island
```

Figure A.7: Setting the Algorithm Parameters

A.7.1 Differential Evolution

This section covers the inner workings of the Differential Evolution Algorithm and how to customize its parameters. For a more detailed explanation of the Differential

Evolution Algorithm, the interested reader is invited to read Section 3.2 of this work and the previous STOpS works by Malloy [3], Sheehan [2], and Fitzgerald [1].

The Differential Evolution Algorithm operates on the principle of mutation by a mutant vector. The mutant vector is created by taking a current member and applying a perturbing member that is defined as a difference vector between two randomly chosen population members multiplied by a scalar value. The original member and its now perturbed member are considered "parent vectors" and are then combined to create two "trial vectors." This process of generating mutations via mutant vectors is repeated for all population members. Once all parent and trial vectors are created, there are more vectors in the population than are allowed. The algorithm then selects the next generation out of the oversized population based on one of the user-defined methods of natural selection, weighted random, and tournament. Finally, this process repeats until the user-defined number of generations has been reached.

Table A.1 outlines the customizable variables for the Differential Evolution Algorithm and their potential inputs in 'parametersDE.py'. A very similar table can also be referenced in [3]. In order to call this function, the user must input the command "opts['DE'] = parametersDE('450_180')" where "450_180" was the default parameters case used for the Indirect Method in this work.

Table A.1: Differential Evolution Parameters Guide

Parameter	Description
Npop	Number of members in each population. Integer.
Ngen	Number of generations to evaluate. Integer.
pc	Percent probability for crossover to occur. Float, between 0 and 1.
sel_method	Selection method for the base vector. Options are: 'random', 'best_so_far', and 'random_best_blend'. *All options are spelling and case sensitive.* String.
F_method	Method for applying the scale factor. Options are: 'constant', 'jitter', and 'dither'. *All options are spelling and case sensitive.* String.
F	Scaling factor when applying difference vector to base vector. If F method == 'constant': Float between 0 and 1. If F method == 'jitter' or 'dither': (1,2) Float, between 0 and 1 for each number. *Lower number must come first.* *Recommended F is 0.4 or greater for all cases.*
surv_method	Method for choosing survivors. Options are: 'natural_selection', 'weighted_random', and 'tournament'. *All options are spelling and case sensitive.* String.
T	Number of members to participate in each tournament if the survivor method is 'tournament'. Integer, 2 or greater
weight	How probabilities are calculated for the selection method. Options are: 'cost' and 'rank'. *All options are spelling and case sensitive.* String.

A.7.2 Genetic Algorithm

This section details the Genetic Algorithm and its parameters. For a more detailed explanation, the interested reader is encouraged to visit the works by Malloy [3], Sheehan [2], and Fitzgerald [1].

The Genetic Algorithm is built to mimic the idea of "Survival of the Fittest" as seen in nature. To do this, the population members' variable strings are seen like genes in DNA. Therefore, to create the next generation of solutions, members mate

and their genes combine in a process called Crossover. There is also a mechanic called Mutation that randomly alters some member's variables which allows for better search space exploration. Once a population has been evaluated, the algorithm removes solutions from the population via a generation method. It then randomly mates the remaining solutions together and creates two (2) new "children" each time. This way, the population is restored to its previous size. The process of evaluation, selection, and mating repeats for a user-defined number of generations.

Table A.2 outlines the customizable variables for the Genetic Algorithm and their potential inputs in 'parametersGA.py'. A very similar table can also be referenced in [3]. In order to call this function, the user must input the command "opts['GA'] = parametersGA('450_180')" where "450_180" was the default parameters case used for the Indirect Method in this work.

Table A.2: Genetic Algorithm Parameters Guide

Parameter	Description
Npop	Number of members in each population. Integer.
Ngen	Number of generations to evaluate. Integer.
pc	Percent probability for crossover to occur. Float, between 0 and 1.
pm	Percent probability for mutation to occur. Float, between 0 and 1.
gen_method	Generation method: how the algorithm selects which members move to the next generation. Options are: 'total_random_replacement', 'natural_selection', 'tournament', 'thresholding', 'weighted_random'. *All options are spelling and case sensitive.* String.
N_keep	Number of solutions to keep between generations if the generation method is 'natural_selection', 'thresholding', or 'weighted_random'. Integer.
T	Number of members to participate in each tournament if the generation method is 'tournament'. Integer, 2 or greater
elite	Number of elite solutions which automatically survive to prevent the best solution from being lost. Integer, recommended to be 1.
threshold	Cost threshold if the generation method is 'thresholding'. Integer.
weight	How probabilities are calculated if the generation method is 'thresholding', otherwise unused. Options are: 'cost' and 'rank'. *All options are spelling and case sensitive.* String.
mate_method	The mating method for two members. Options are: 'uniform_crossover', 'random_crossover', and 'blending'. *All options are spelling and case sensitive.* String.
cross_points	How many points are permitted between a member and mate if the mate method is 'random_crossover', otherwise unused. Integer.
OB	Out of bounds limit for blending. Float, between 0 and 1, recommended less than 0.2.

A.7.3 Particle Swarm Optimization

This section documents the Particle Swarm Optimization Algorithm and its parameters. For a more detailed explanation, the interested reader is encouraged to visit the works by Malloy [3] and Sheehan [2].

The PSO Algorithm is based on how bees will search for nectar as a group in nature. Each member of the population has a particular location and velocity in the search space at a given time. As the members move throughout the population, they will remember their best-so-far solution and that will influence their velocities; they will prefer to stay in the vicinity of good solutions. However, if this were the end of the algorithm, it would take quite a long time for the algorithm to converge on the global optimum. By introducing informant population members, best-so-far solutions can be shared around the population. This means that if a member receives a solution from an informant that is better than their own best-so-far solution, it will start to explore the informant's area of the search space instead.

There is a trade-off in raising the amount of informants, however. If too many informants share the same solution, then the algorithm will prematurely converge on sub-optimal solutions. On the other hand, too few informants will cause the population members to wander around the search space and prevent convergence at all. Therefore, it is recommended to keep the number of informants around four (4), as this has been decided by Sheehan [2] and Malloy [3] to work nicely for this search space.

By entering the 'parametersPSO.py' file, the user is able to customize many parameters of the PSO Algorithm. These parameters are listed in Table A.3. A very similar table can also be referenced in [3]. In order to call this function, the user must

input the command "opts['PSO'] = parametersPSO('350_150')" where "350_150" was the default parameters case used for the Indirect Method in this work.

Table A.3: Particle Swarm Optimization Parameters Guide

Parameter	Description
Npop	Number of members in each population. Integer.
tspan	How many iterations are evaluated per particle. Integer.
vmax	Max velocity for each particle. Float, between 0 and 1, recommended less than 0.5.
K	Number of informants for each particle. Integer, between 1 and 'Npop'.
cl	Particle confidence in own velocity. *Note, will decrease over time.* Float, between 0 and 1.
cmax	Particle confidence in others' velocity. *Note, larger numbers may cause premature convergence.* Float, between 0 and 1.

A.7.4 Monotonic Basin Hopping

This section details the Monotonic Basin Hopping Algorithm and its parameters. For a more detailed explanation, the interested reader is encouraged to visit the work by Malloy [3].

The MBH Algorithm operates by injecting random perturbations into its population. Often in low thrust trajectories, the search space is quite bumpy. This can lead to local searches finding themselves trapped in local minima. The goal of the MBH Algorithm is to take those solutions and try to kick them out of the small local basin they might be in. This involves repeatedly perturbing solutions until either a better solution has been found or a certain number of iterations has been reached. The algorithm is complete once all the population members have been tested.

The user may call the 'parametersMBH.py' file to control the parameters of the algorithm. These parameters are listed in Table A.4. A very similar table can also

be referenced in [3]. In order to call this function, the user must input the command "opts['MBH'] = parametersMBH('15000_1500')" where "15000_1500" was the default parameters case used for the Indirect Method in this work.

Table A.4: Monotonic Basin Hopping Parameters Guide

Parameter	Description
N1_outer	Number of members in each initial population. Integer.
N1_inner	Number of iterations to run in initial local search. *Will only run if member(s) from 'N1 outer' deemed feasible.* Integer.
N2_outer	Number of members to generate from each parent member for basin exploration. *Will only run if member(s) from 'N1 outer' deemed feasible. Integer.
N2_inner	Number of iterations to run in secondary local search for each member of 'N2 outer'. *Will only run if member(s) from 'N1 outer' deemed feasible.* Integer.
maxclst	Maximum number of clusters to explore if 'N1 outer' deemed feasible. Used in combination with 'N2 outer' to generate the size of the secondary population. If 'N2 outer' is 150 and 'maxclst' is 5, the size of the secondary population will be (750,n), where the first 150 members are derived from the first feasible solution found in 'N1 outer'. *Note, will decrease to fix maximum number of feasible members from 'N1 outer'.* Integer.
per_feas	Percentage values to apply to the parameter used to check feasibility. If using FSM, applied to the target planet's SOI. If using FBSM, applied to the match point between trajectories. See [3] for details on how this number works in MBH. *Recommended to start large and exponentially decrease values.*(1,island migrations-1) Float, recommended less than 200% for the first and less than 10% for later.
per_rand	Percentage values to apply to the perturbation limits generation of new members generated from 'N2 outer' and 'maxclst'. See [3] for details on how this number works in MBH. *Will only run if member(s) from 'N1 outer' deemed feasible.* *Recommended to start large and exponentially decrease values.* (1,island migrations-1) float, recommended less than 2% for the first and less than 0.1% for later.

A.8 Performance Improvement Control

The last step in setting up a mission is to implement Performance Improvement Control. In addition to parallel computing, this work detailed three performance improvement methods in Thruster Angle Pruning, Cluster Pruning, and Best Leg Tracking. It is imperative that the user reads Chapters 9, 10, 11, and 12 before beginning to activate any of these methods. Otherwise, particular combinations may not be allowed or beneficial to the user.

In order for the user to turn performance improvements on or off, they must define them all as either 'y' or 'n', where 'y' turns them on, and 'n' turns them off. Figure A.8 shows a possible command combination to implement the performance improvement methods correlating with the Indirect Method.

```
opts['parallel'] = 'y' # 'y' or 'n' #Decides Parallel Programming
opts['constraint_Pruning'] = 'n' # 'y' or 'n' # Recommended on for Direct Method, off for Indirect Method
opts['cluster_Pruning'] = 'y' # 'y' or 'n' # Recommended on for longer runs, off for short runs
opts['best_Leg_Tracking'] = 'y'  # 'y' or 'n' # Decides Best Leg Solution Tracking

return bod, opts
```

Figure A.8: Setting the Performance Improvements

A.9 Final Form

The user is now equipped with all necessary knowledge and commands to create a mission. Figure A.9 shows the completed mission preset (without documentation) for the MGALT-STOpS Demo. Happy optimizing!!

```python
#%% Presets
class presets():

    def demo():
        bod = {}
        opts = {}

        opts['ephemeris'] = 'JPL_Horizons'

        bod['bodies'] = ('Earth','Mars','Jupiter')

        # Launch Window
        bod['window1'] = datetime(2020,11,1)       # earliest launch day    [year,month,day]
        bod['window2'] = datetime(2022,12,1)       # latest launch day      [year,month,day]

        # Time of Flight Options
        opts['tof_total'] = [(752),\
                             (1448)] # tof w/ gravity assists [days]

        opts['tof_margin'] = [(100,100),\
                              (300,300)] # margin on tof -/+     [days]

        # Solver and Thrust Method
        opts['solver'] = 'MGALT_IN_FBSM_2D'
        opts['thrust'] = optionsLowThrust(opts,'Yam-STOUR')

        # Island Model
        opts['island'] = optionsIsland('4M-all')

        # Cost Function
        [opts['cost'],opts['weighting'],opts['ode']] = optionsCost(opts,'default')

        # Algorithm Parameters
        opts['GA'] = parametersGA('450_180')#     # Genetic Algorithm Island
        opts['DE'] = parametersDE('450_180')#      # Differential Evolution Island
        opts['PSO'] = parametersPSO('350_150')#     # Particle Swarm Island
        opts['MBH'] = parametersMBH('15000_1500')#     # Particle Swarm Island

        opts['parallel'] = 'y' # 'y' or 'n' #Decides Parallel Programming
        opts['constraint_Pruning'] = 'n' # 'y' or 'n' # Recommended on for Direct Method, off for Indirect Method
        opts['cluster_Pruning'] = 'y' # 'y' or 'n' # Recommended on for longer runs, off for short runs
        opts['best_Leg_Tracking'] = 'y'  # 'y' or 'n' # Decides Best Leg Solution Tracking

        return bod, opts
```

Figure A.9: Custom Mission Final Form

Appendix B

ADDITIONAL VERIFICATION RESULTS

This appendix will show all runs used in the verification of the MGALT-STOpS performance improvements. For all following sections, Run 1 will be the run that was shown in the respective verification section.

B.1 Python Verification

B.1.1 Direct Method

Table B.1: MGALT-STOpS Direct Method Python Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3907.36	3886.12	4042.86	2857.35	4297.02
Departure Date	Aug 1, 2021	Aug 4, 2021	Aug 2, 2021	Aug 3, 2021	Aug 1, 2021
Flyby ToF (days)	825	820	809	810	807
Arrival ToF (days)	1396	1396	1396	1396	1396
Total ToF (years)	6.08	6.07	6.04	6.04	6.00
Propellant Mass Fraction	36.85%	36.75%	36.58%	36.59%	36.53%

Table B.2: MGALT-STOpS Direct Method Python Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3907.36	3886.12	4042.86	2857.35	4297.02
Earth to Mars Cost	3882.78	3874.02	3774.13	2846.58	4282.50
Mars to Jupiter Cost	24.57	12.10	268.72	10.77	14.52

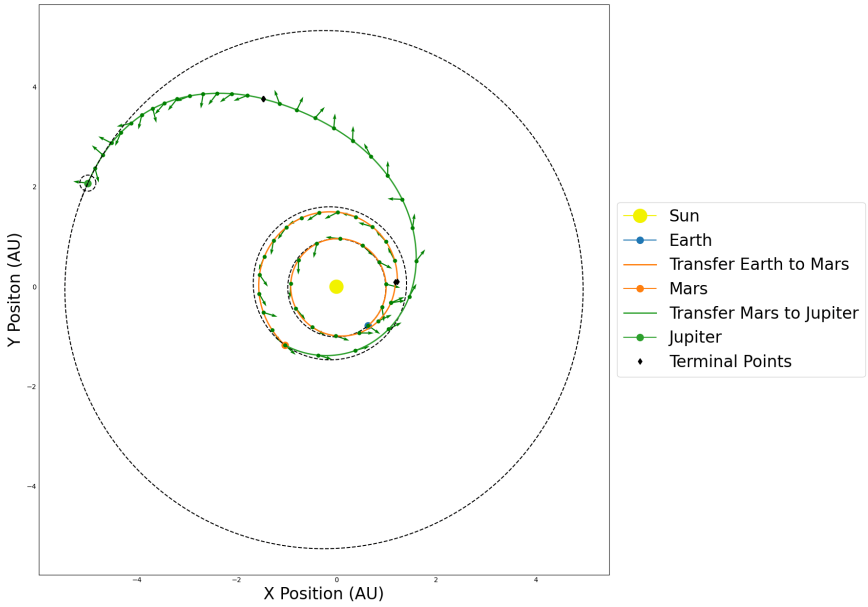

Figure B.1: Python MGALT-STOpS Direct Method Optimal Solution 1 Trajectory

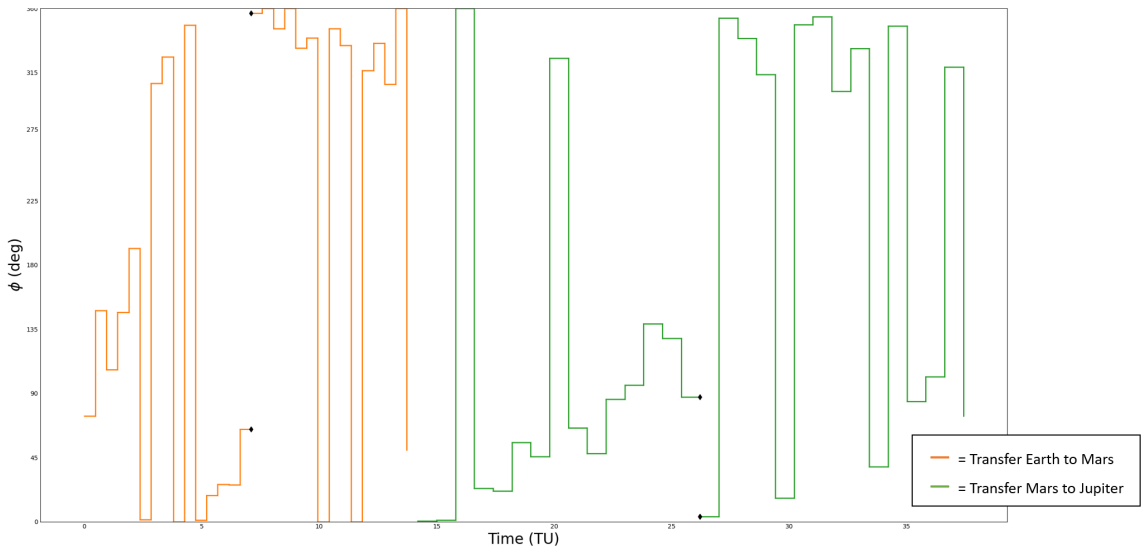

Figure B.2: Python MGALT-STOpS Direct Method Optimal Solution 1 Thrust

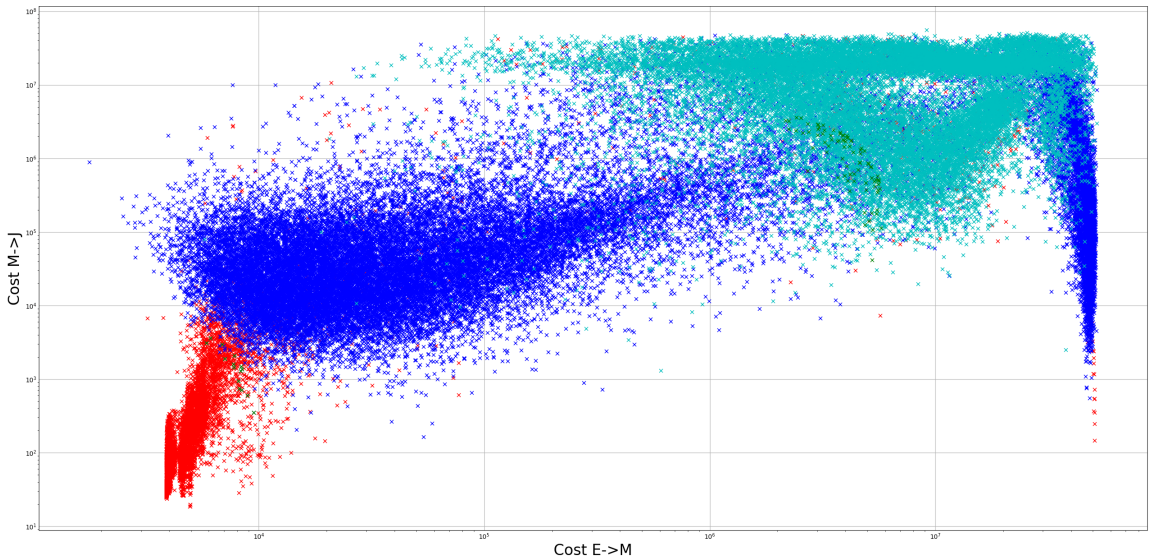

Figure B.3: Python MGALT-STOpS Direct Method Optimal Solution 1 Leg Tracking

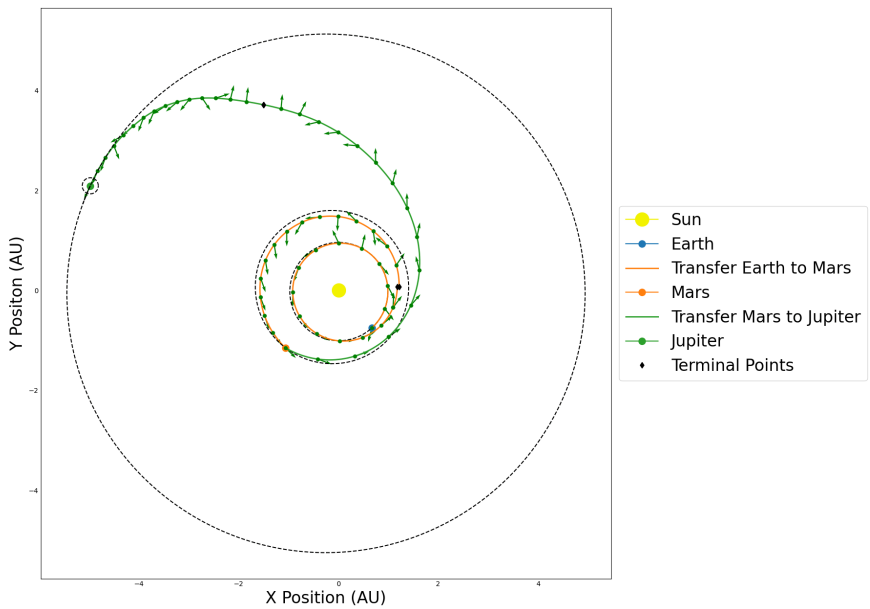

Figure B.4: Python MGALT-STOpS Direct Method Optimal Solution 2 Trajectory

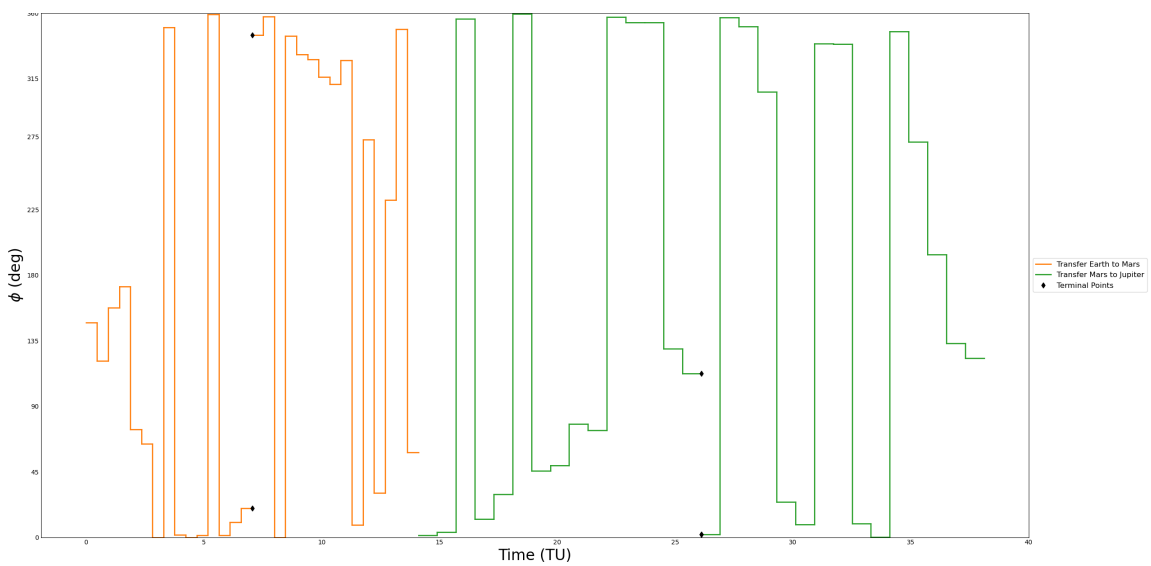

Figure B.5: Python MGALT-STOpS Direct Method Optimal Solution 2 Thrust

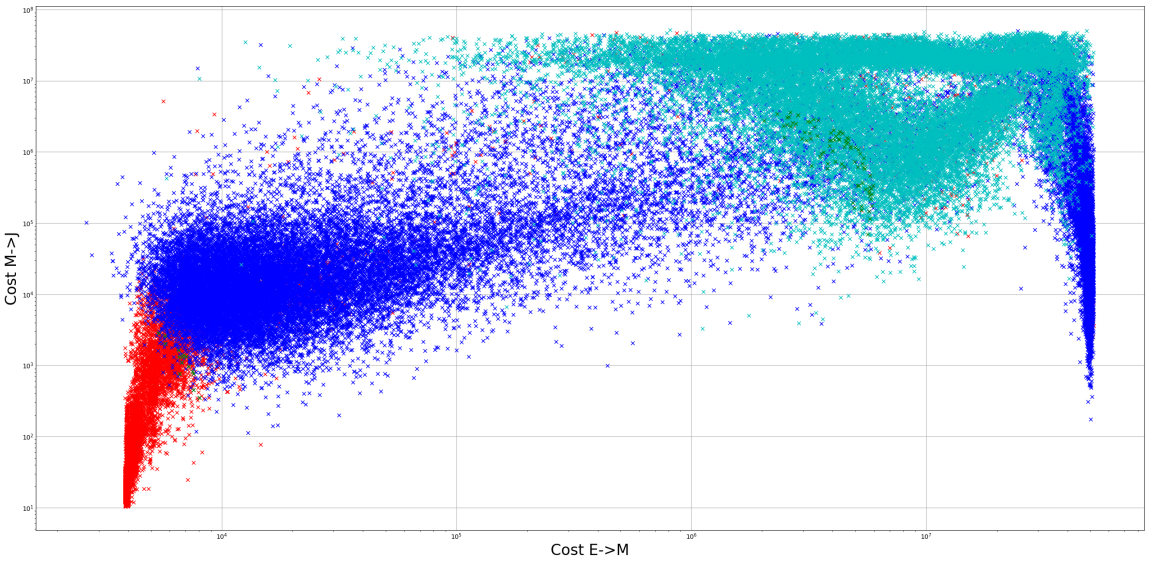

Figure B.6: Python MGALT-STOpS Direct Method Optimal Solution 2 Leg Tracking

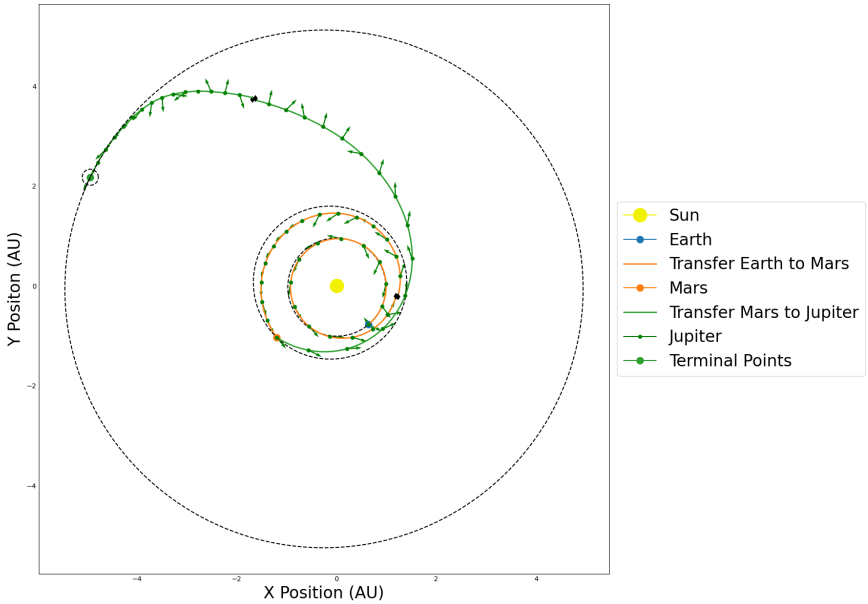

Figure B.7: Python MGALT-STOpS Direct Method Optimal Solution 3 Trajectory

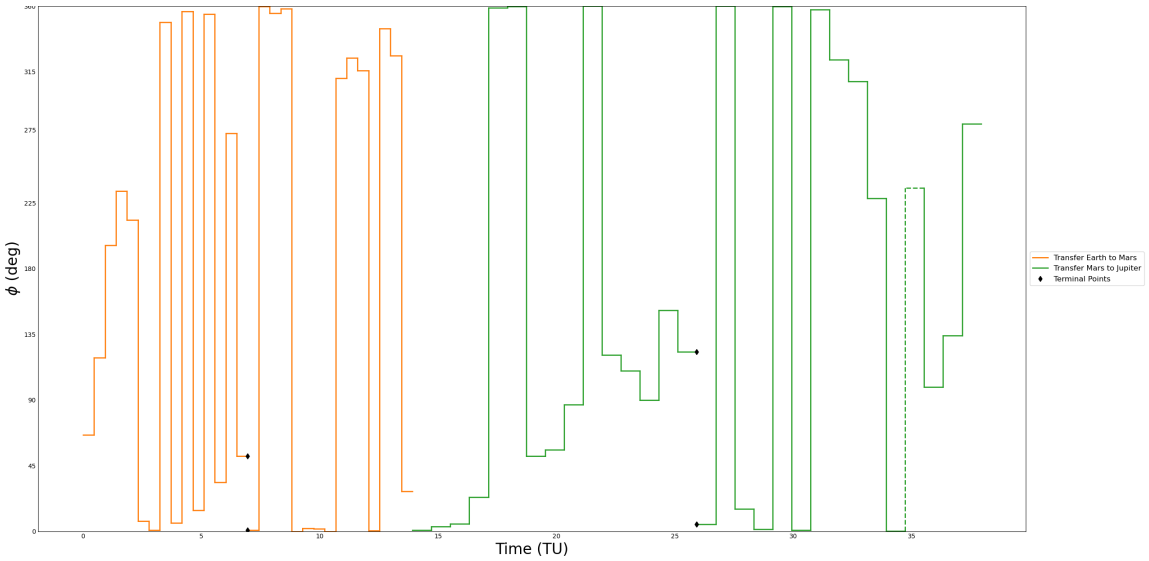

Figure B.8: Python MGALT-STOpS Direct Method Optimal Solution 3 Thrust

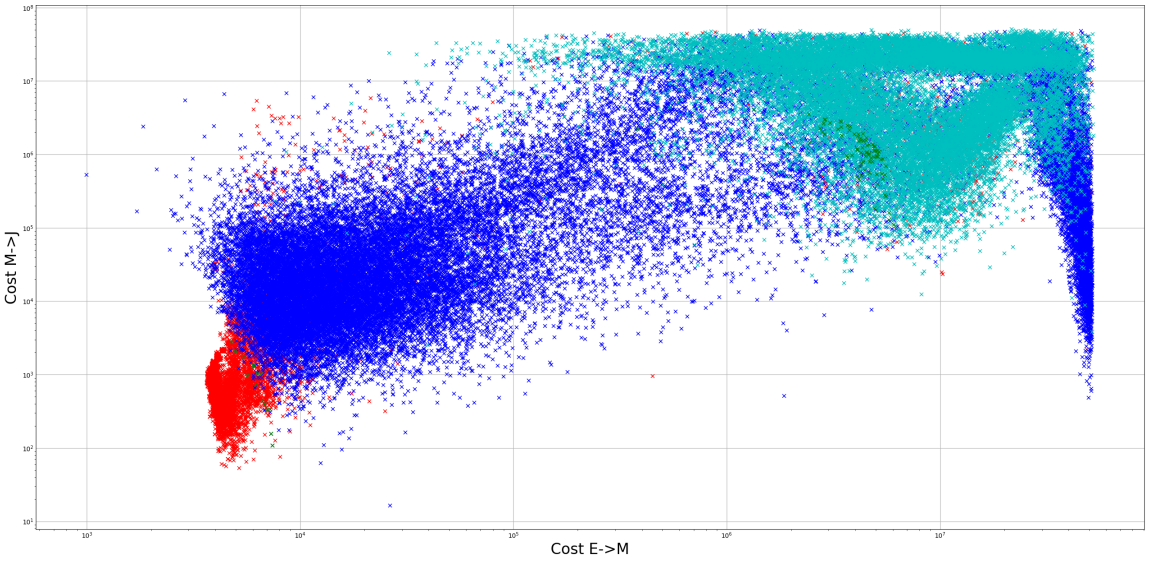

Figure B.9: Python MGALT-STOpS Direct Method Optimal Solution 3 Leg Tracking

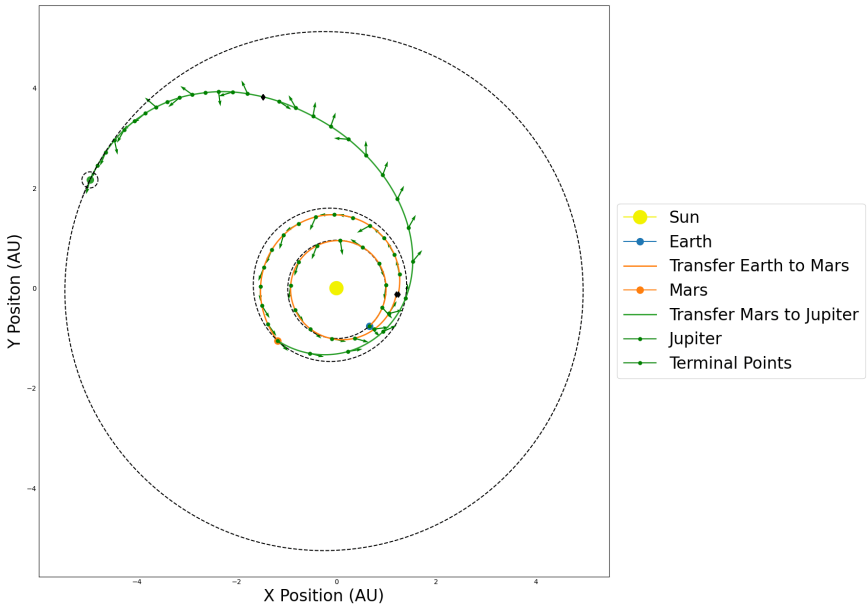

Figure B.10: Python MGALT-STOpS Direct Method Optimal Solution 4 Trajectory

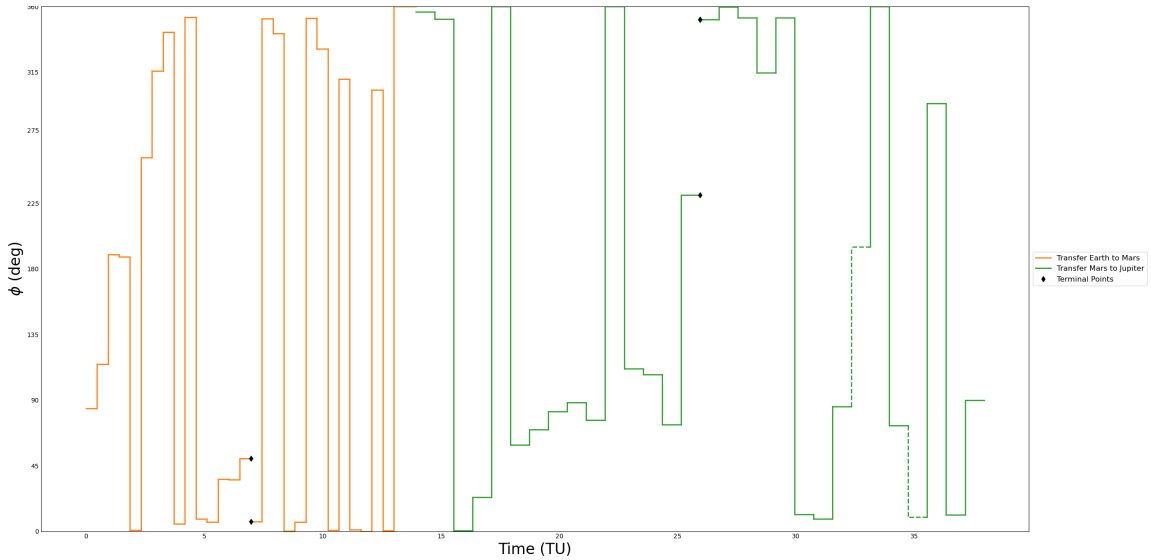

Figure B.11: Python MGALT-STOpS Direct Method Optimal Solution 4 Thrust

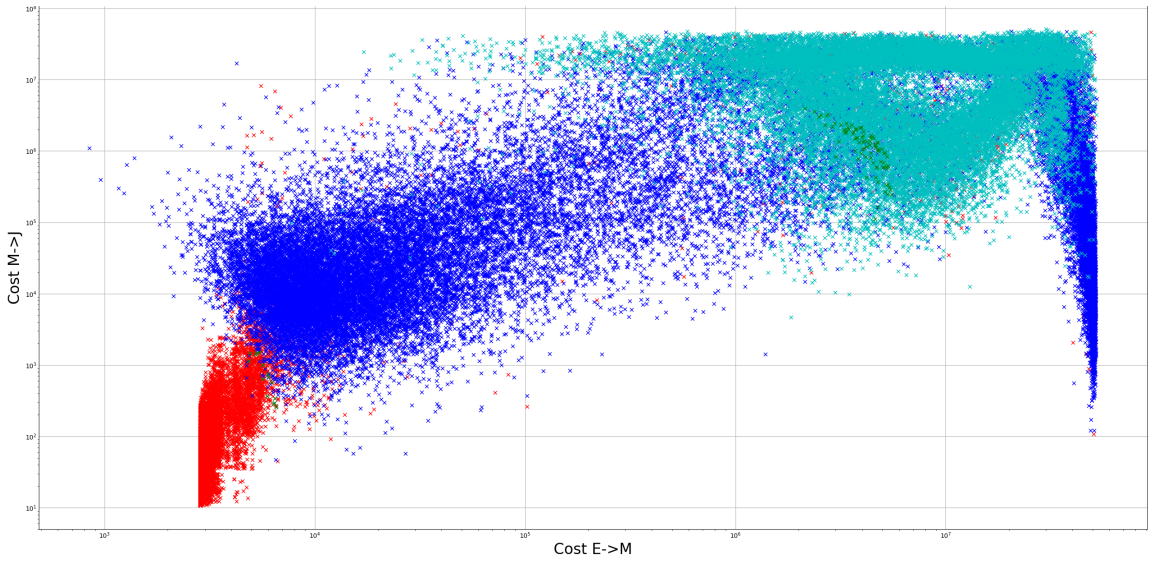

Figure B.12: Python MGALT-STOpS Direct Method Optimal Solution 4 Leg Tracking

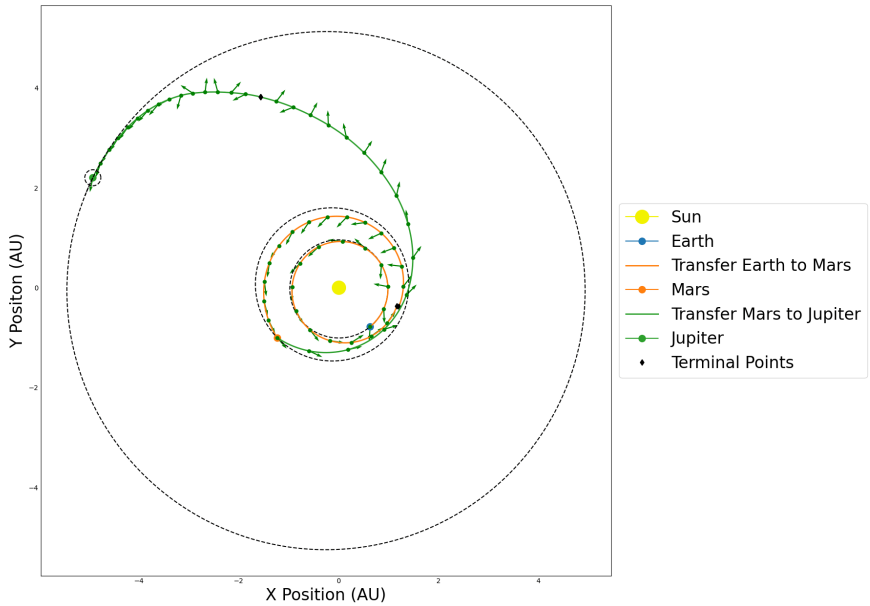

Figure B.13: Python MGALT-STOpS Direct Method Optimal Solution 5 Trajectory

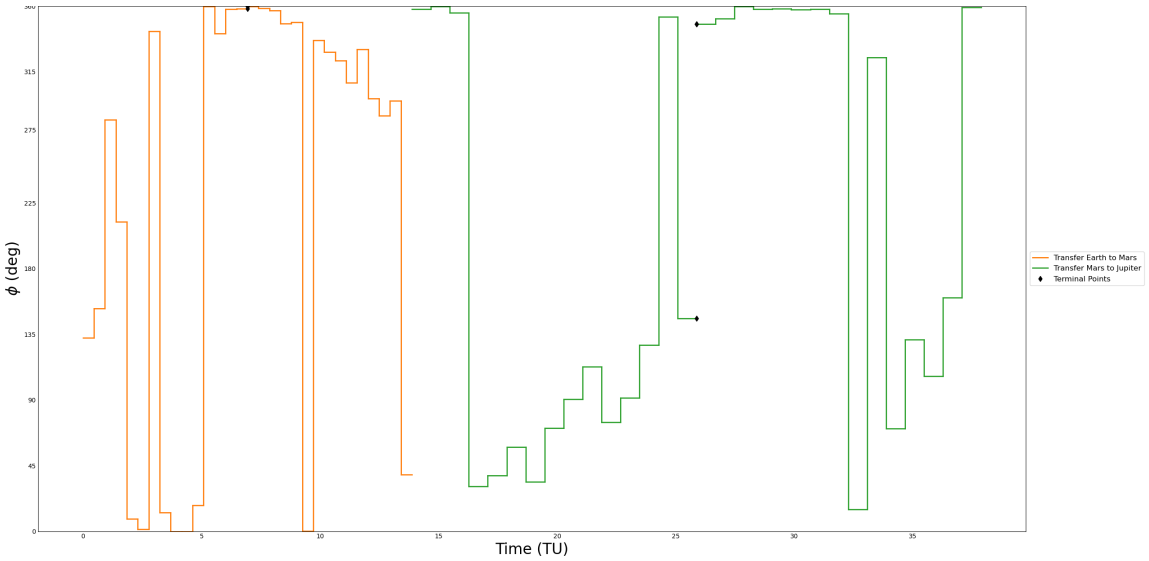

Figure B.14: Python MGALT-STOpS Direct Method Optimal Solution 5 Thrust

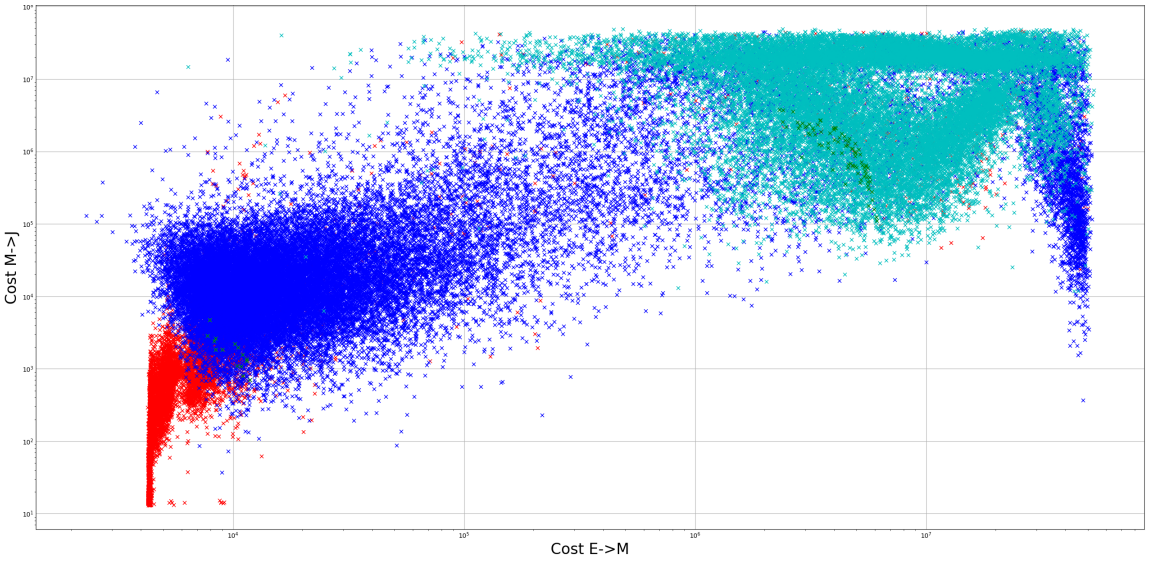

Figure B.15: Python MGALT-STOpS Direct Method Optimal Solution 5 Leg Tracking

B.1.2 Indirect Method

Table B.3: MGALT-STOpS Indirect Method Python Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	1704.77	190.77	2536.06	5558.64	2697.68
Departure Date	Sep 1, 2021	Jun 8, 2021	Jan 19, 2021	Feb 19, 2021	Feb 11, 2021
Flyby ToF (days)	708	845	848	834	846
Arrival ToF (days)	1748	1732	1745	1652	1748
Total ToF (years)	6.72	7.06	7.10	6.81	7.10
Propellant Mass Fraction	40.75%	42.81%	43.00%	41.23%	43.03%

Table B.4: MGALT-STOpS Indirect Method Python Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	1704.77	190.77	2536.06	5558.64	2697.68
Earth to Mars Cost	1121.25	56.21	16.51	772.05	825.90
Mars to Jupiter Cost	583.52	134.55	2519.55	4786.58	1871.78

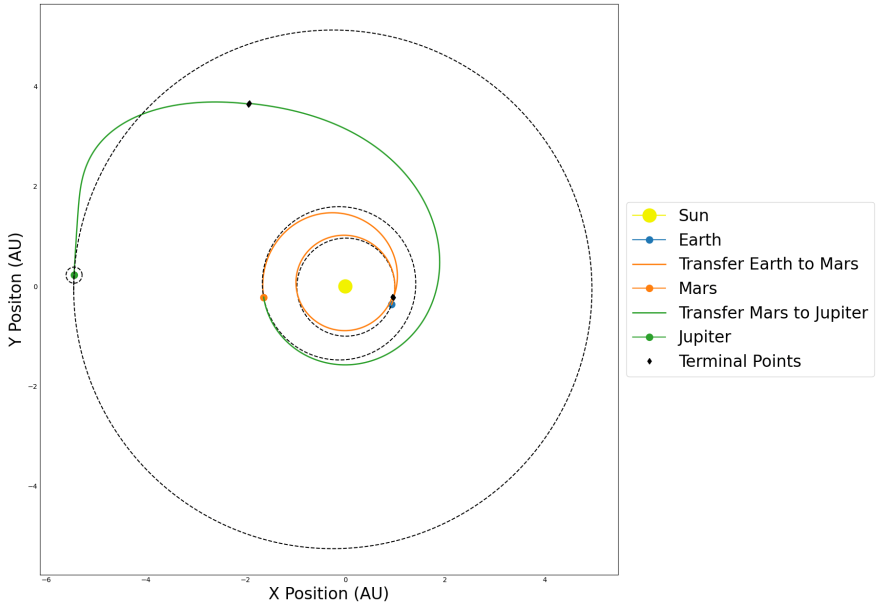

Figure B.16: Python MGALT-STOpS Indirect Method Optimal Solution 1 Trajectory

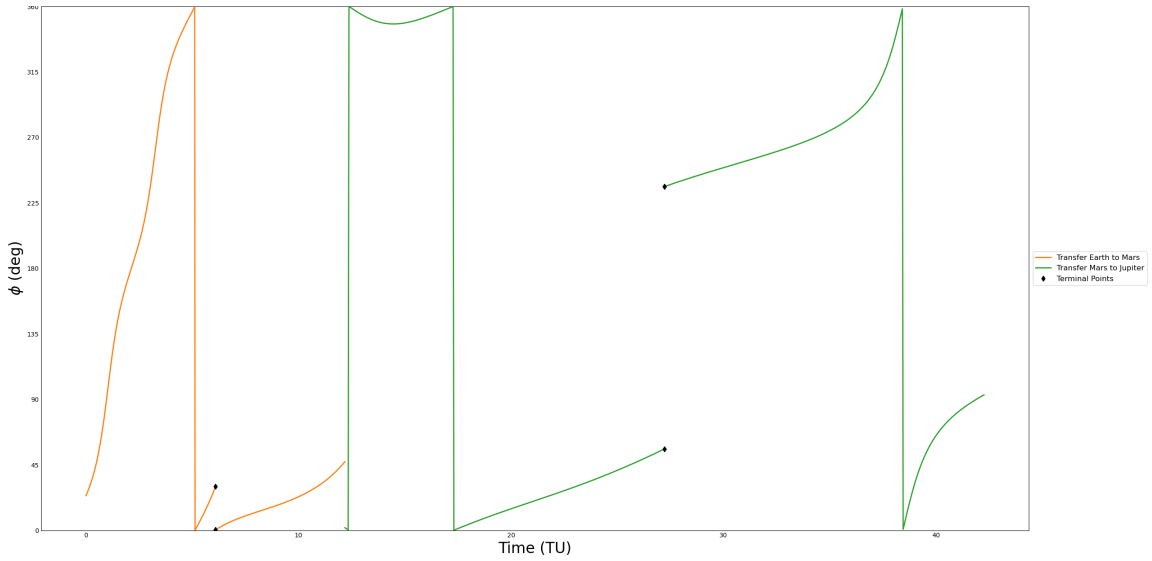

Figure B.17: Python MGALT-STOpS Indirect Method Optimal Solution 1 Thrust

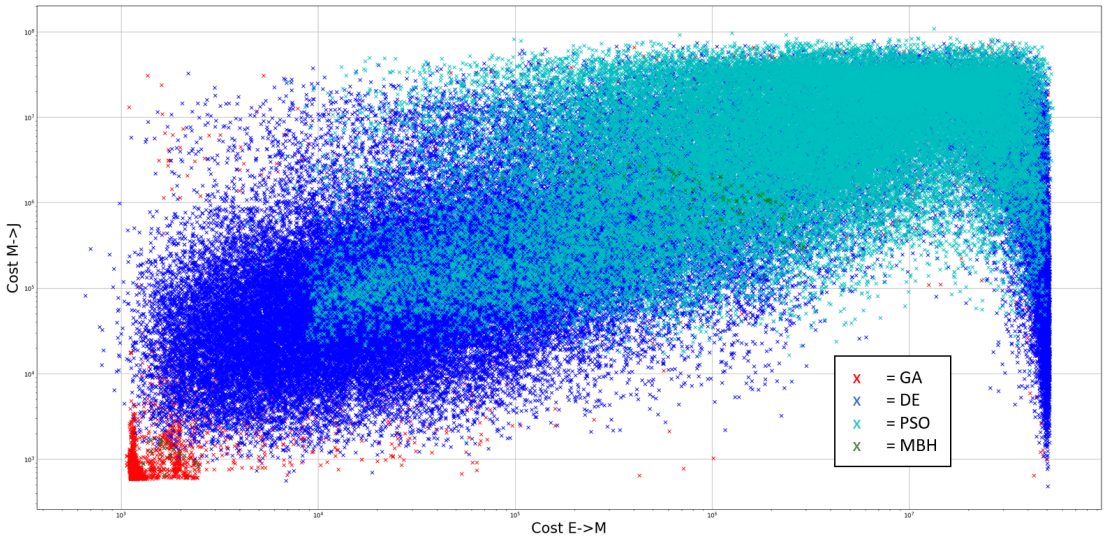

Figure B.18: Python MGALT-STOpS Indirect Method Optimal Solution 1 Leg Tracking

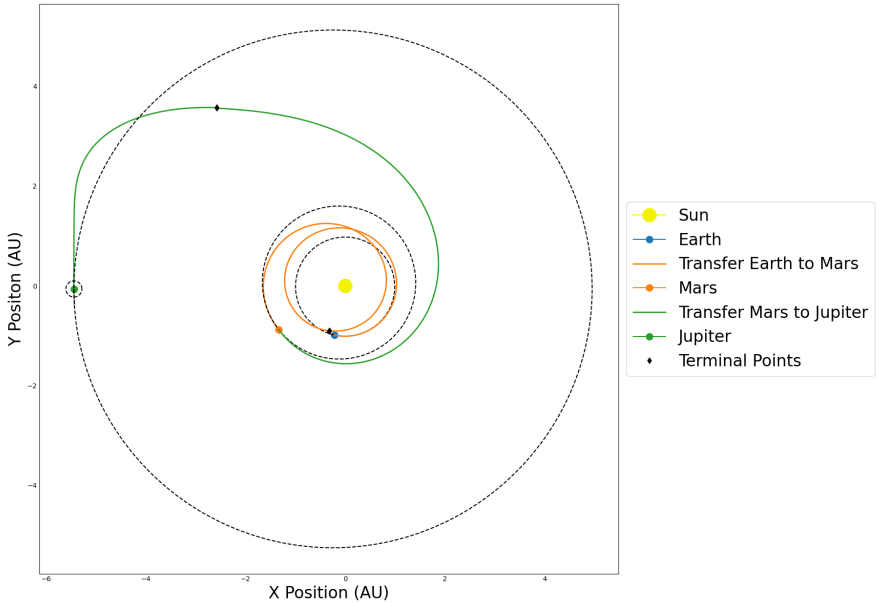

Figure B.19: Python MGALT-STOpS Indirect Method Optimal Solution 2 Trajectory

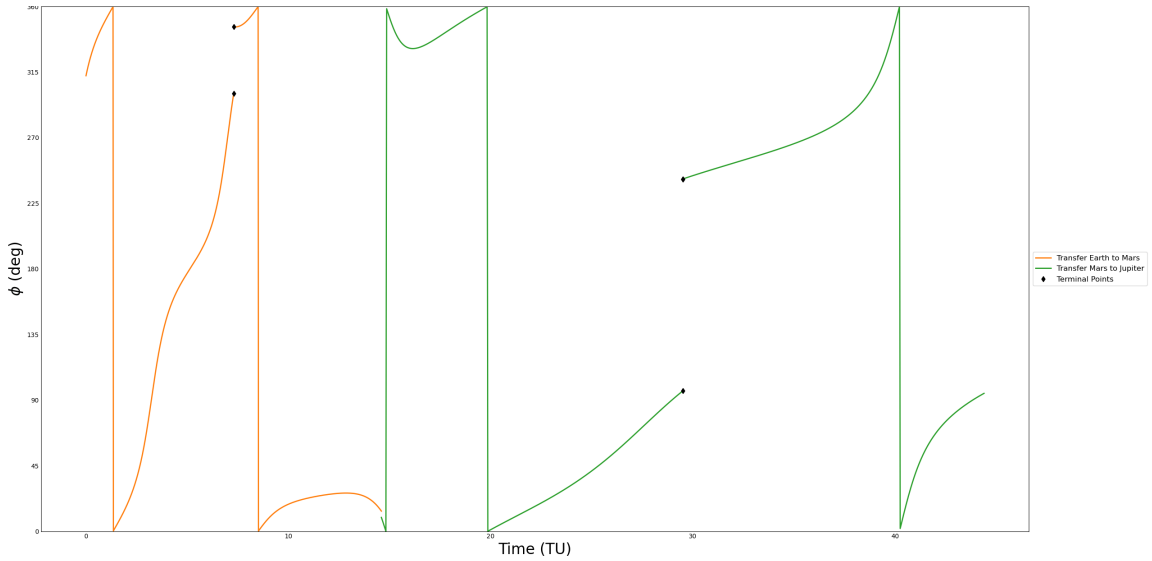

Figure B.20: Python MGALT-STOpS Indirect Method Optimal Solution 2 Thrust

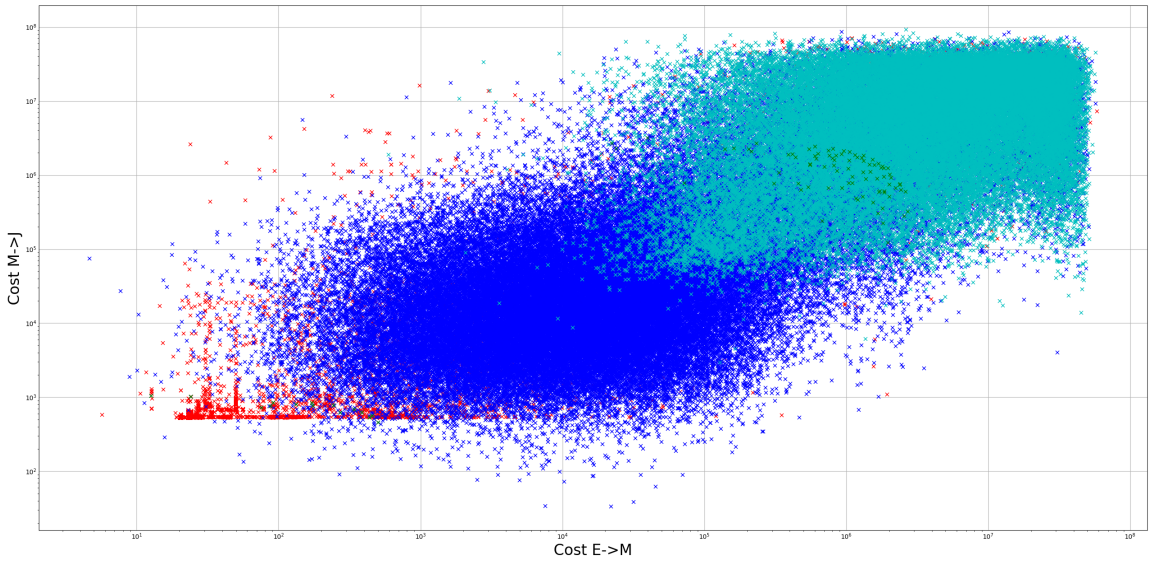

Figure B.21: Python MGALT-STOpS Indirect Method Optimal Solution 2 Leg Tracking

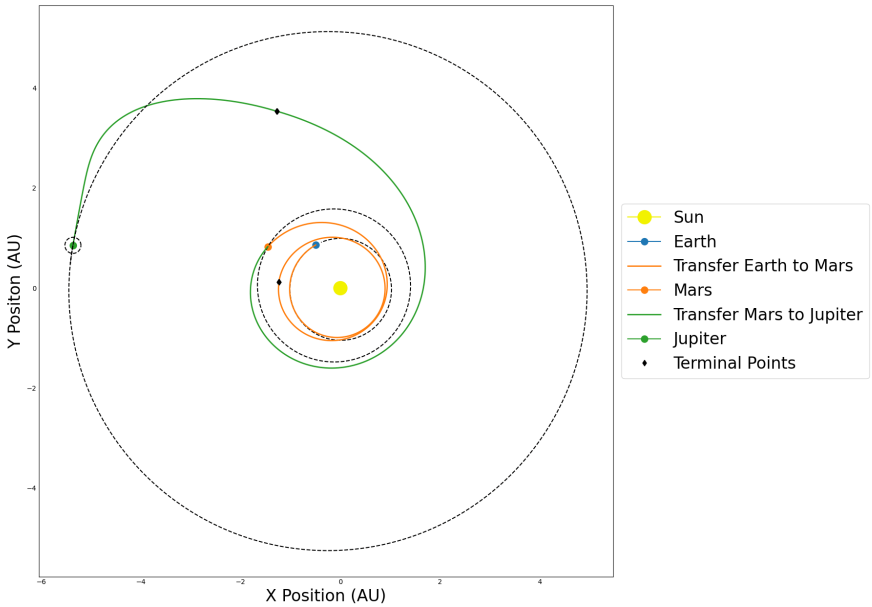

Figure B.22: Python MGALT-STOpS Indirect Method Optimal Solution 3 Trajectory

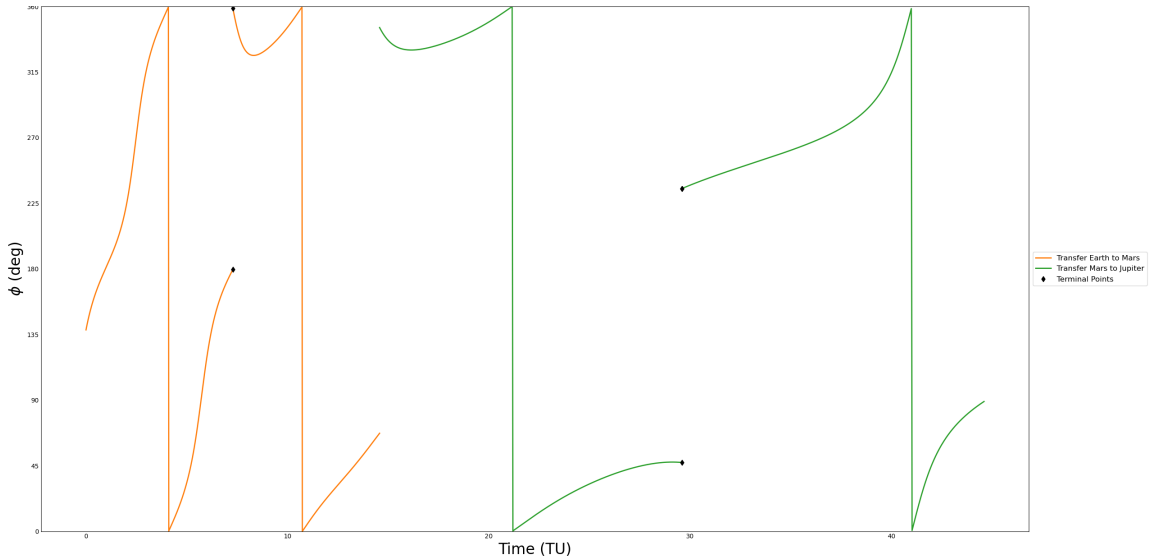

Figure B.23: Python MGALT-STOpS Indirect Method Optimal Solution 3 Thrust

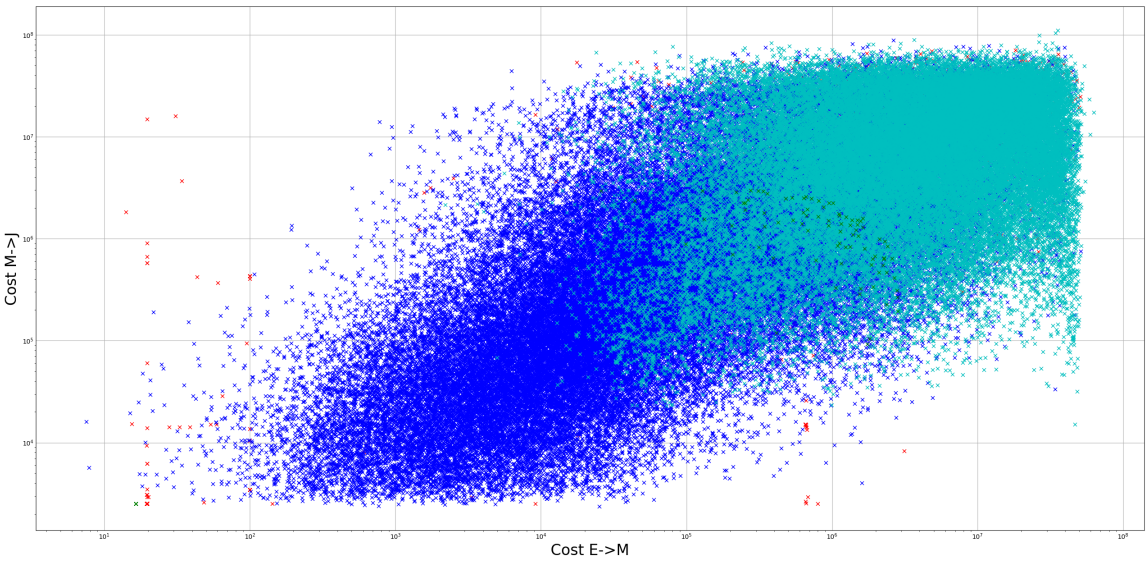

Figure B.24: Python MGALT-STOpS Indirect Method Optimal Solution 3 Leg Tracking

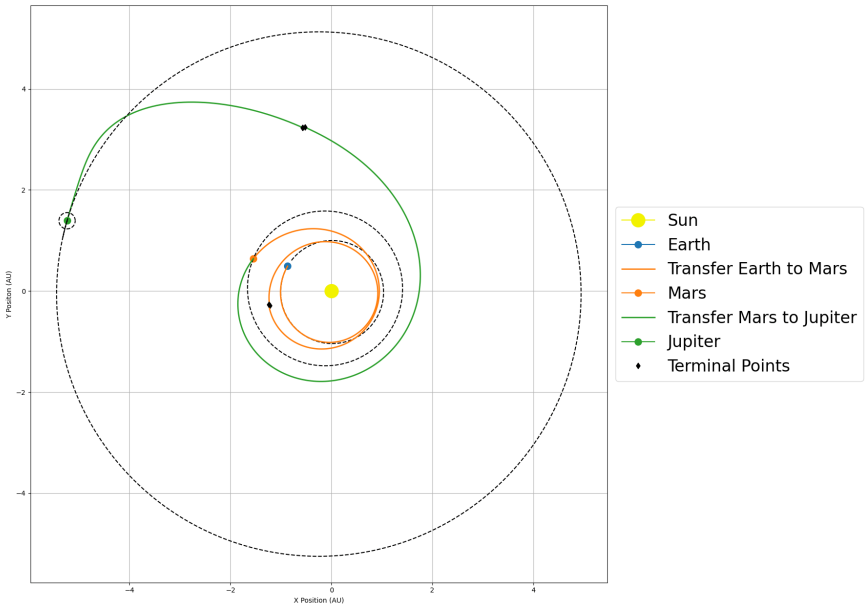

Figure B.25: Python MGALT-STOpS Indirect Method Optimal Solution 4 Trajectory

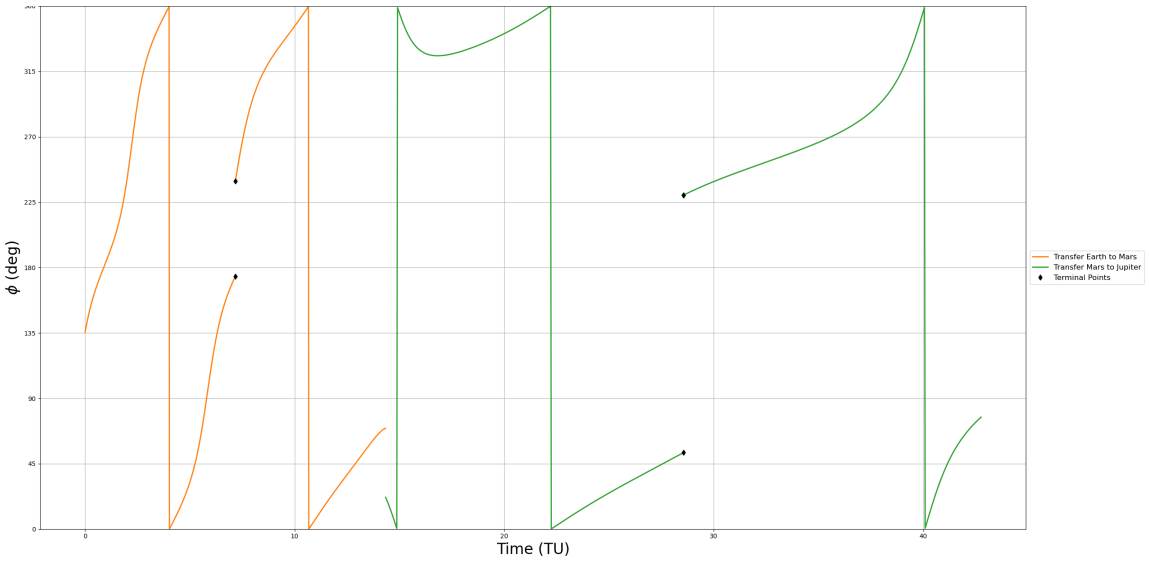

Figure B.26: Python MGALT-STOpS Indirect Method Optimal Solution 4 Thrust

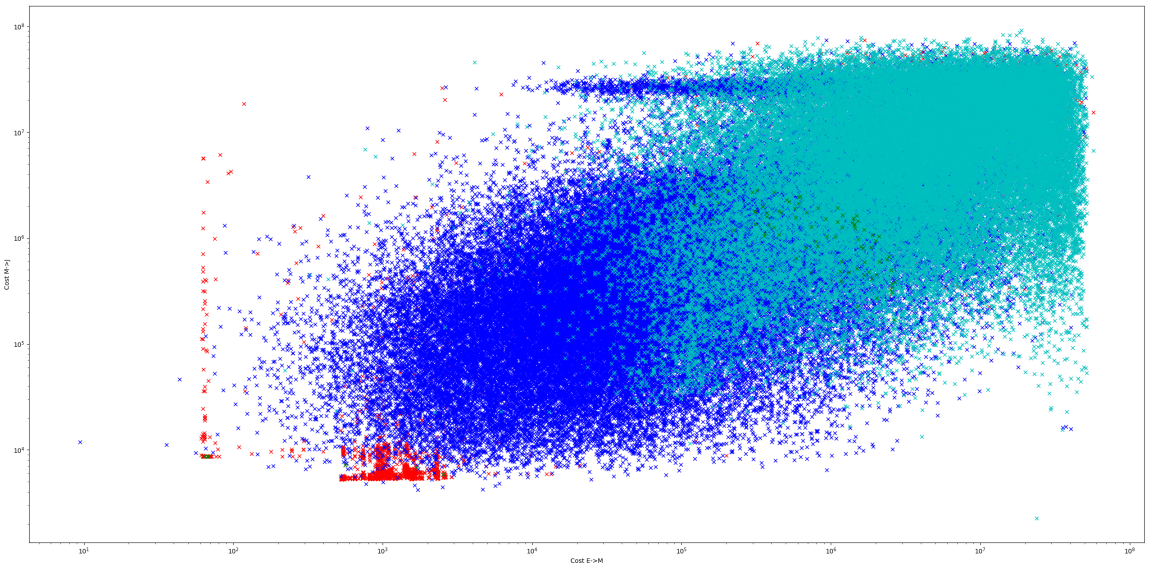

Figure B.27: Python MGALT-STOpS Indirect Method Optimal Solution 4 Leg Tracking

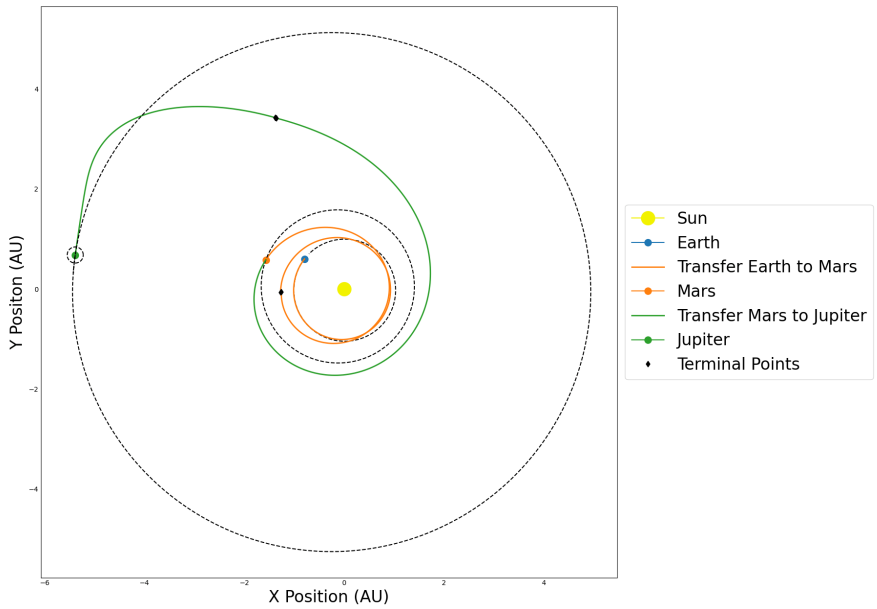

Figure B.28: Python MGALT-STOpS Indirect Method Optimal Solution 5 Trajectory

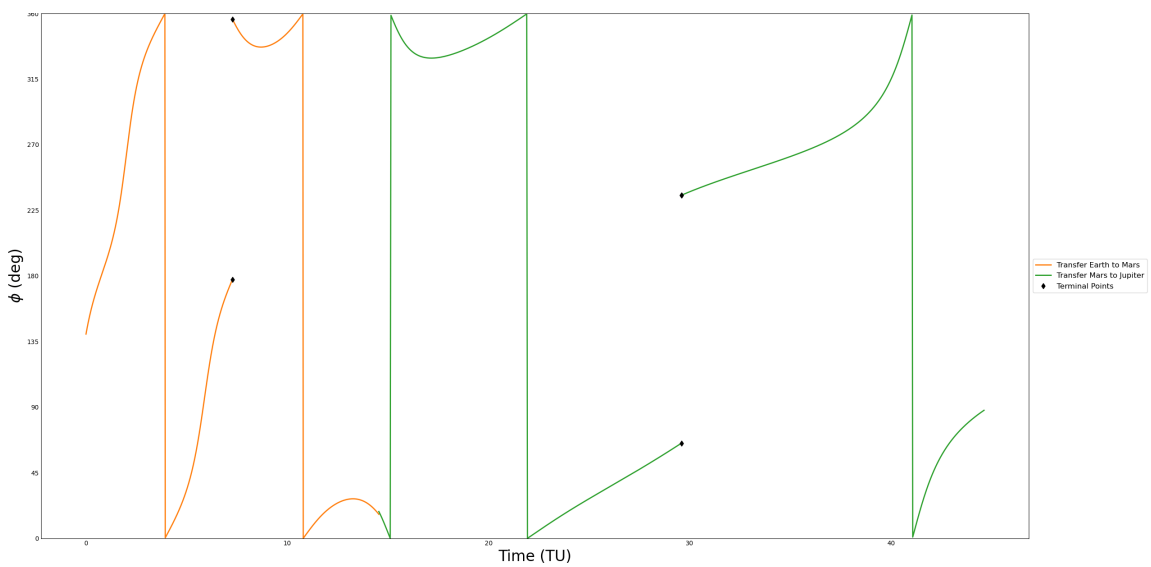

Figure B.29: Python MGALT-STOpS Indirect Method Optimal Solution 5 Thrust

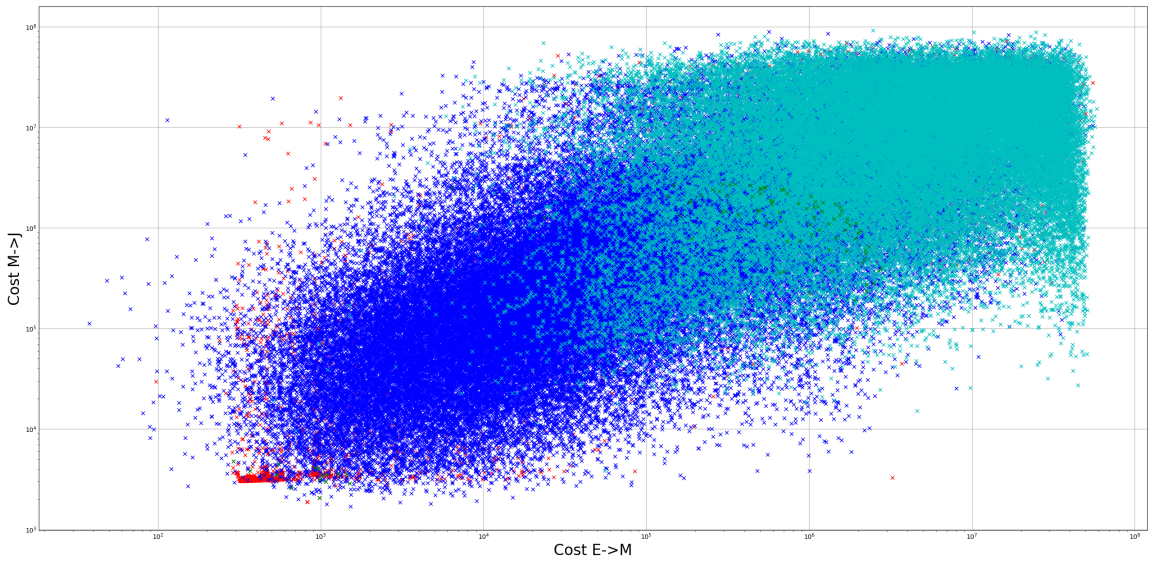

Figure B.30: Python MGALT-STOpS Indirect Method Optimal Solution 5 Leg Tracking

B.2 Thruster Angle Pruning Verification

B.2.1 Direct Method

Table B.5: MGALT-STOpS Direct Method Thruster Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	4331.56	4882.04	6111.83	4670.03	3866.06
Departure Date	Aug 14, 2021	Aug 19, 2021	Aug 30,2021	Sep 5, 2021	Aug 19, 2021
Flyby ToF (days)	822	831	817	806	828
Arrival ToF (days)	1395	1396	1396	1396	1393
Total ToF (years)	6.07	6.10	6.06	6.03	6.08
Propellant Mass Fraction	36.78%	36.95%	36.71%	35.75%	36.84%

Table B.6: MGALT-STOpS Direct Method Thruster Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	4331.56	4882.04	6111.83	4670.03	3866.06
Earth to Mars Cost	2767.85	4171.03	4947.32	3910.09	2987.96
Mars to Jupiter Cost	1563.72	711	1164.50	759.94	878.10

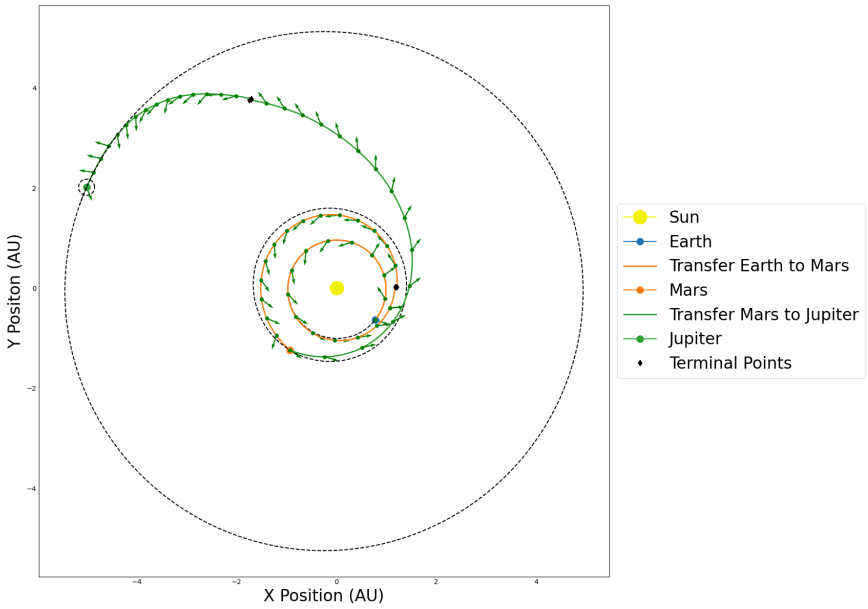

Figure B.31: Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory

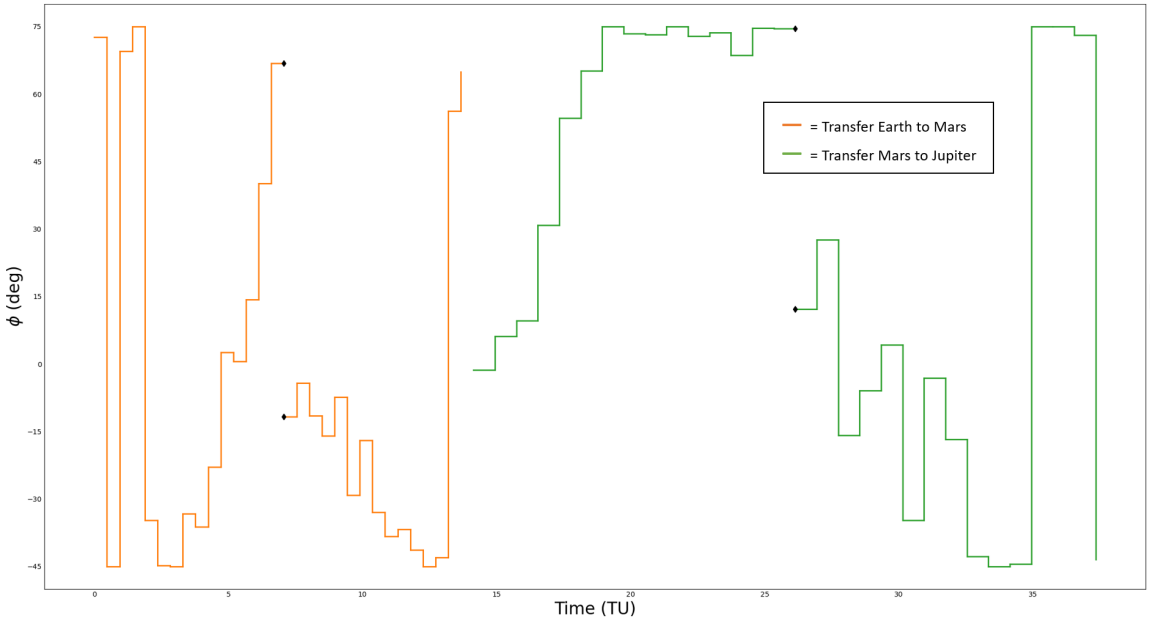

Figure B.32: Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust

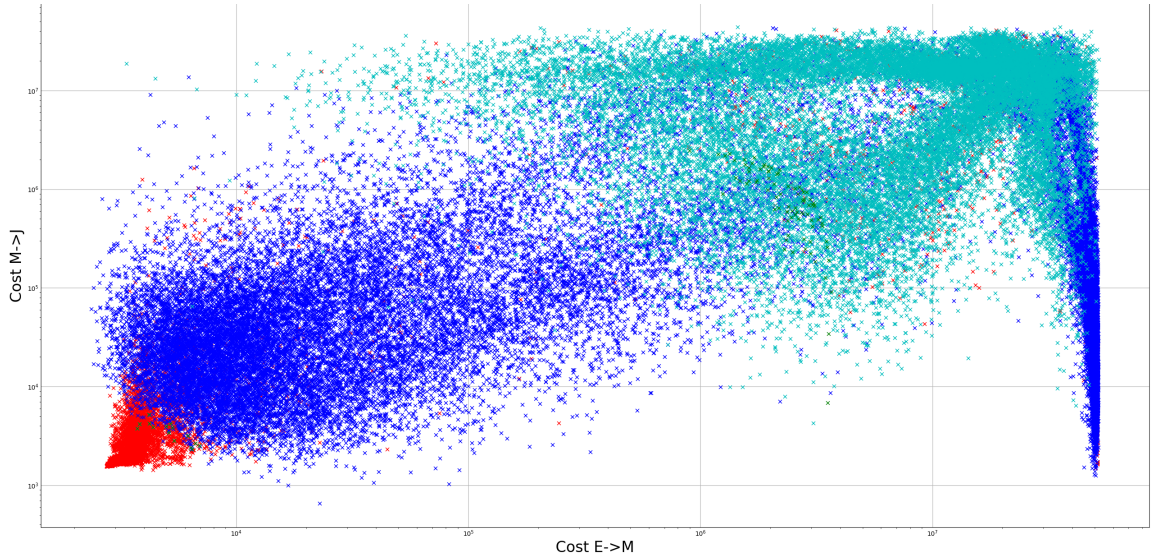

Figure B.33: Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking

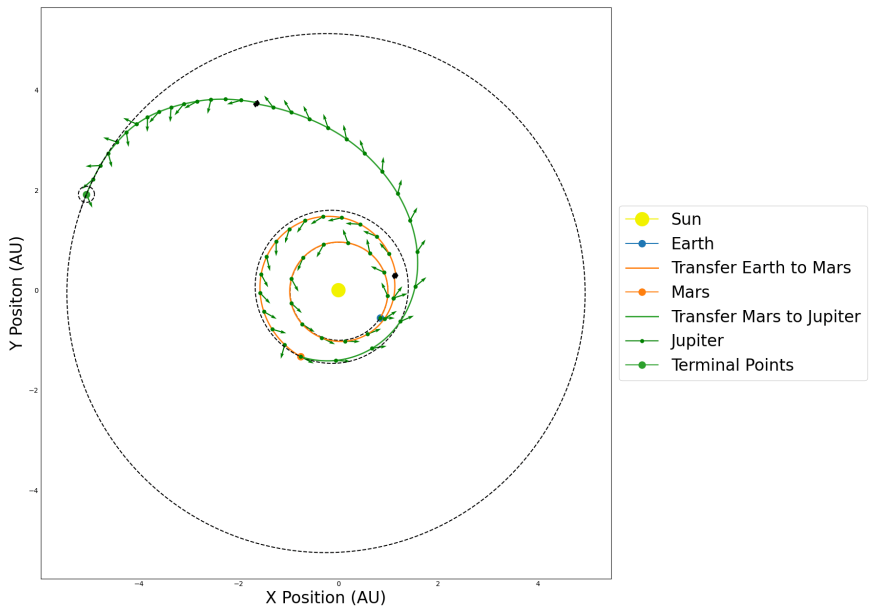

Figure B.34: Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory

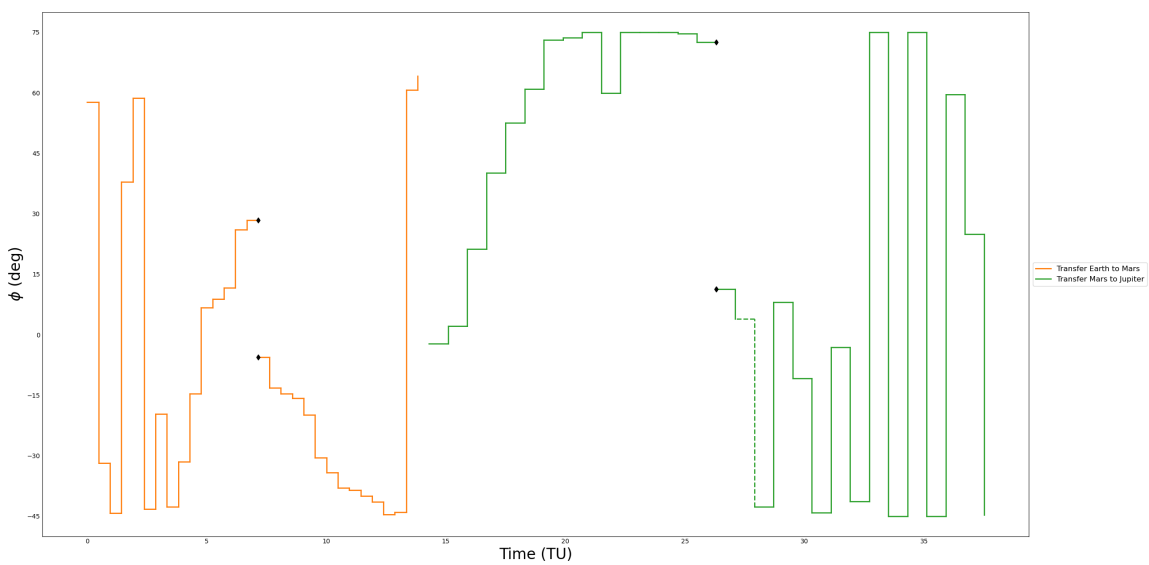

Figure B.35: Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust

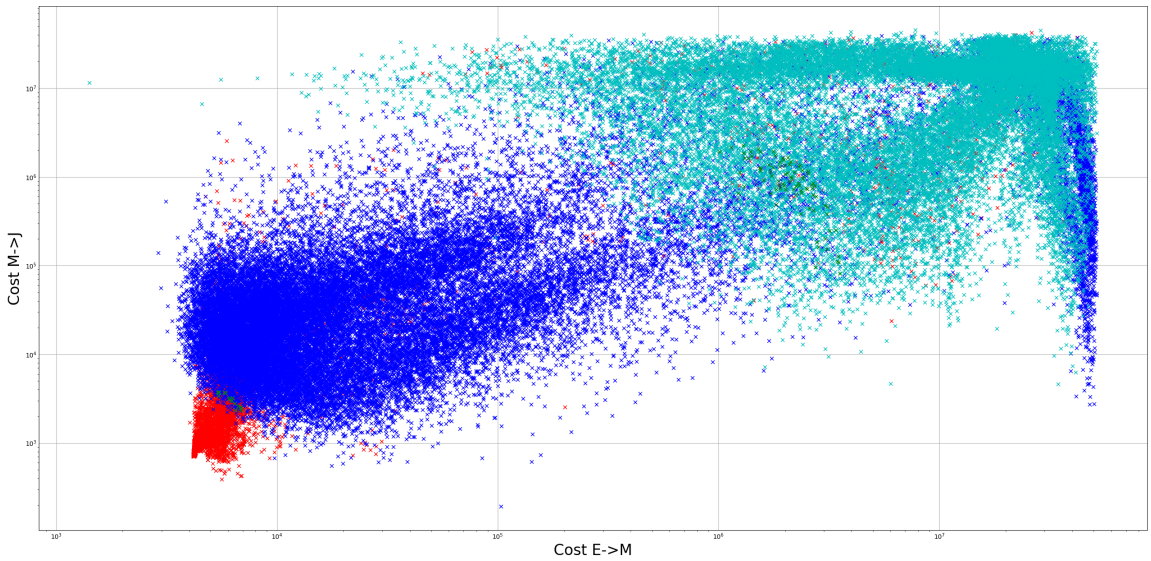

Figure B.36: Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking

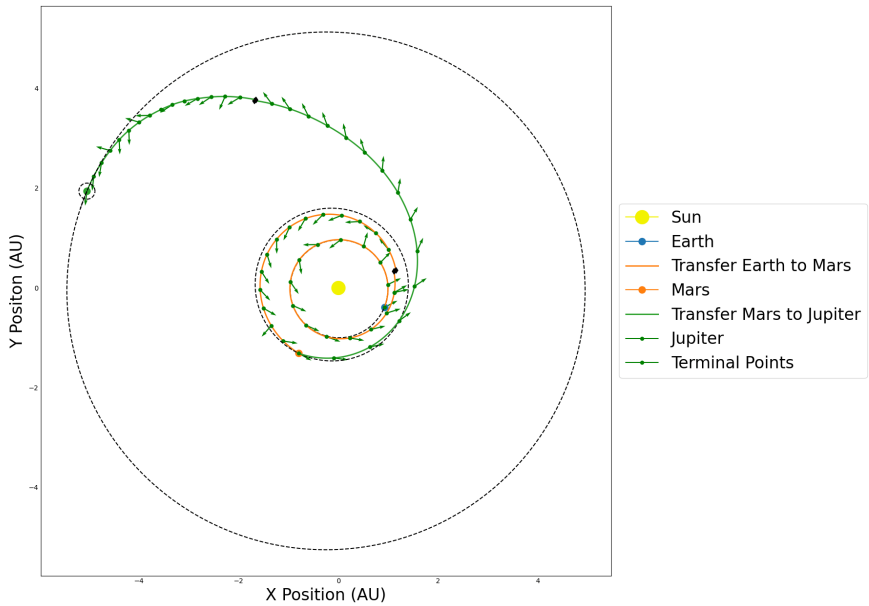

Figure B.37: Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory

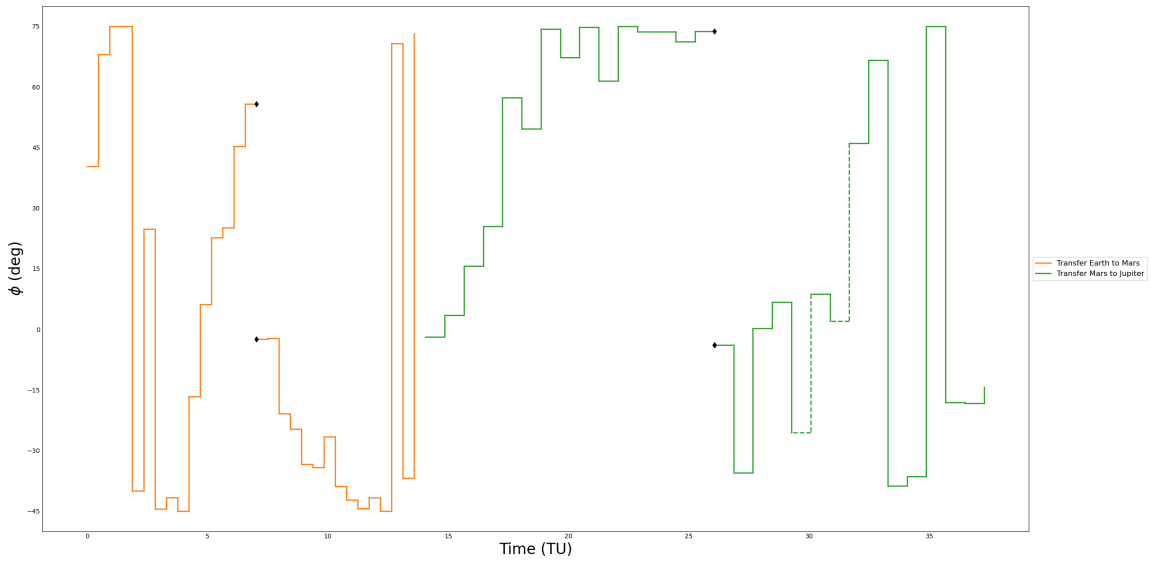

Figure B.38: Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust

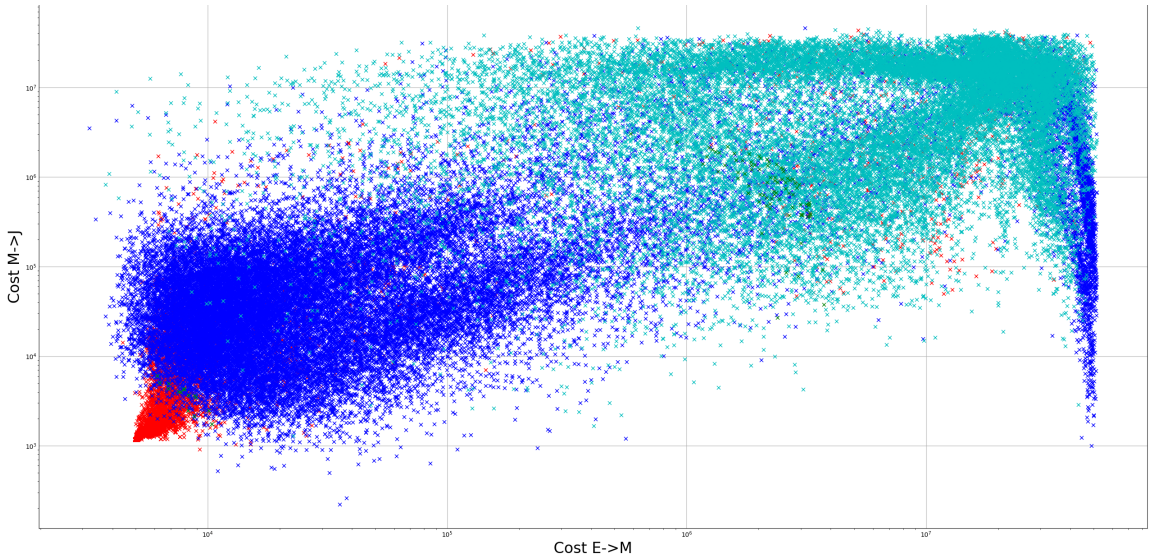

Figure B.39: Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking

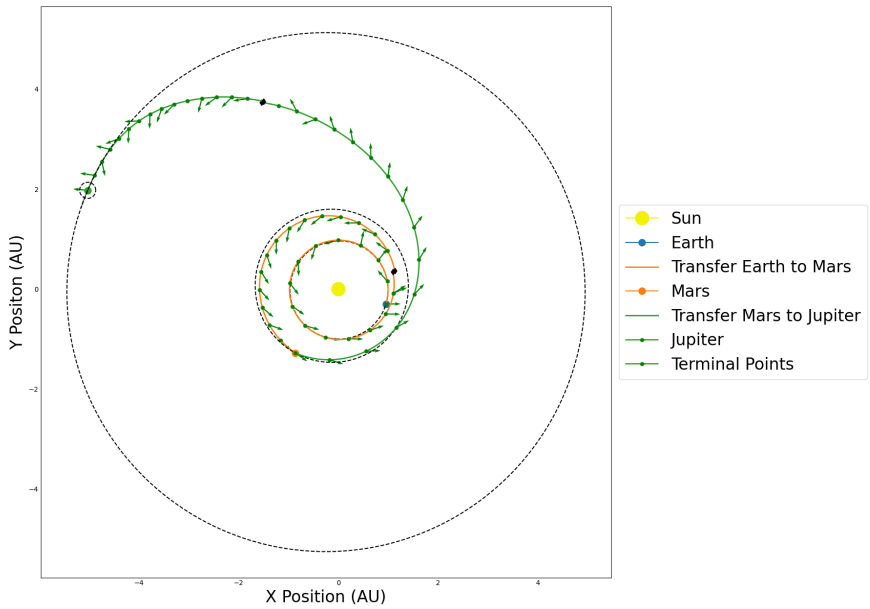

Figure B.40: Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory

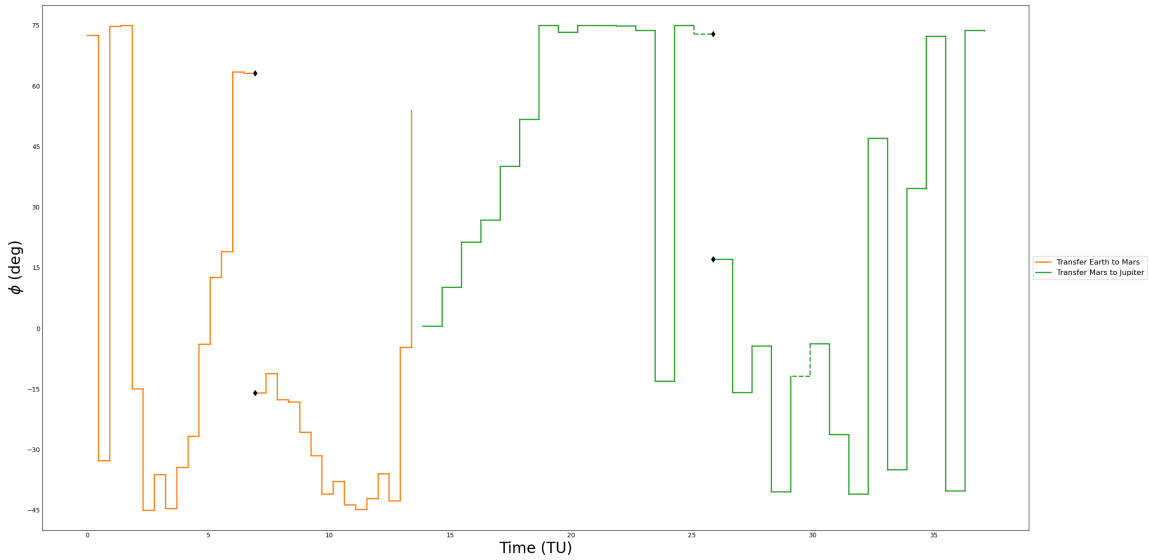

Figure B.41: Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust

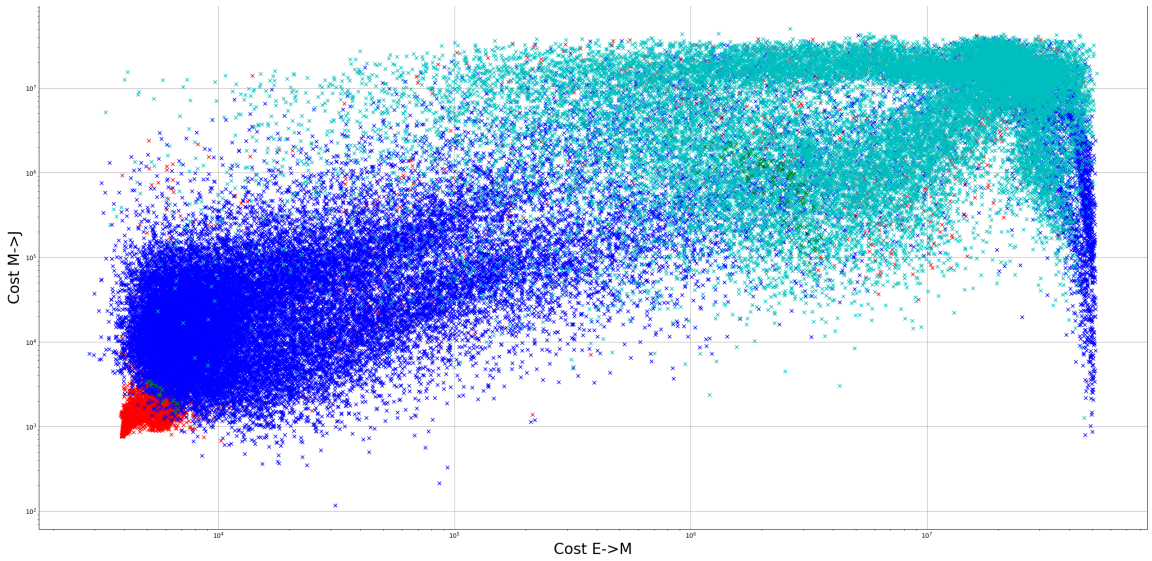

Figure B.42: Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking

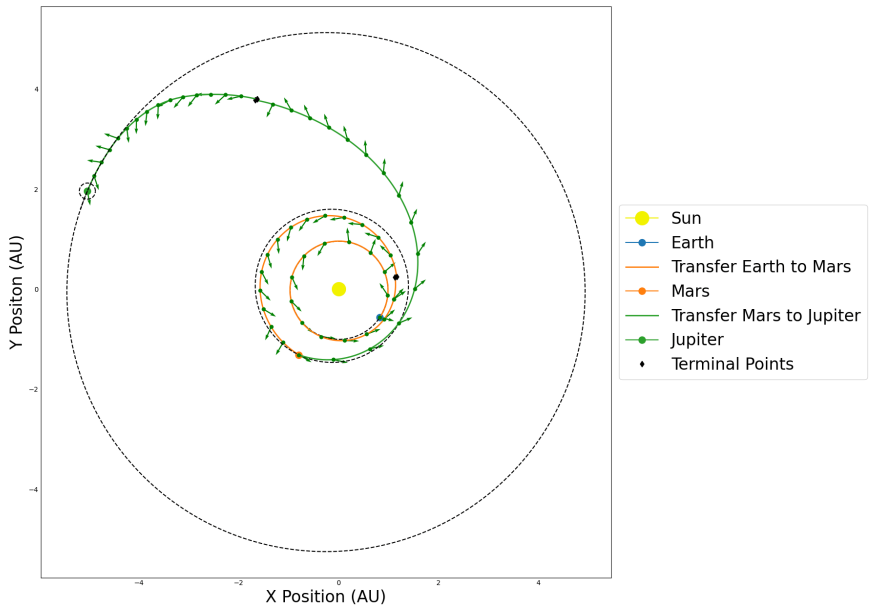

Figure B.43: Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory

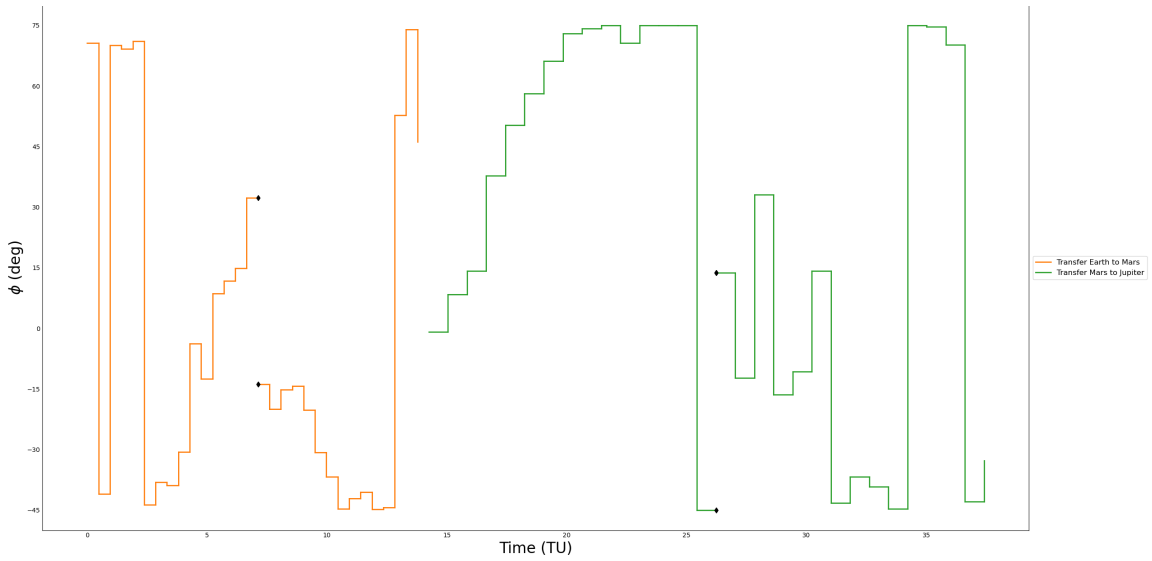

Figure B.44: Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust

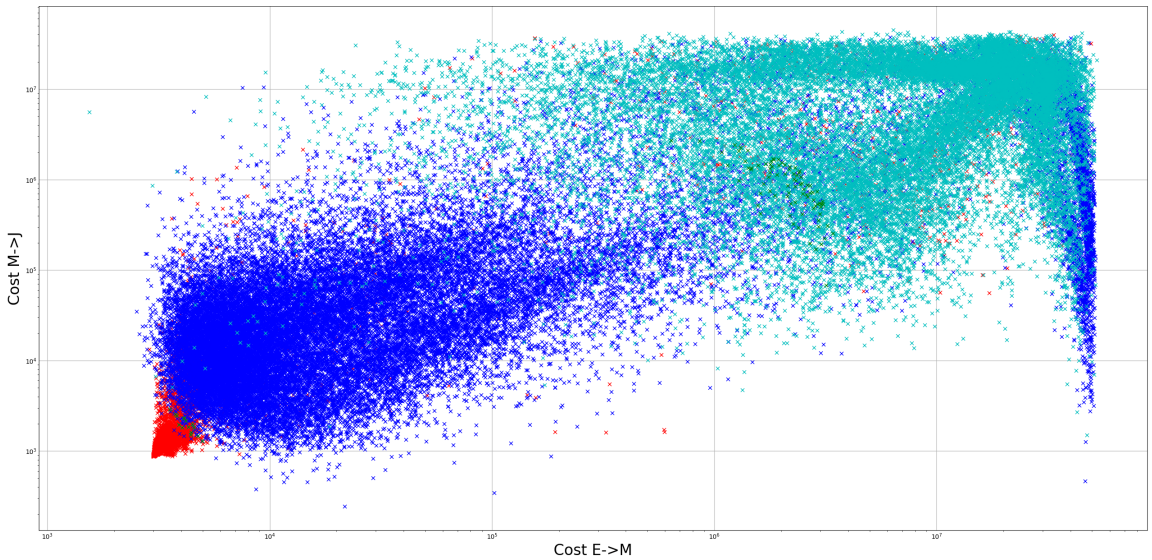

Figure B.45: Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking

B.2.2 Indirect Method

Thruster Angle Pruning was not implemented for the Indirect Method. Therefore, there is no data in this section.

B.3 Cluster Pruning Verification

B.3.1 Direct Method

Table B.7: MGALT-STOpS Direct Method Cluster Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3719.68	5191.83	4440.74	4396.18	4593.42
Departure Date	Aug 3, 2021	Aug 8, 2021	Aug 1, 2021	Aug 1, 2021	Aug 2, 2021
Flyby ToF (days)	821	814	838	866	848
Arrival ToF (days)	1396	1395	1396	1393	1394
Total ToF (years)	6.07	6.05	6.12	6.19	6.14
Propellant Mass Fraction	36.78%	36.64%	37.07%	37.48%	37.20%

Table B.8: MGALT-STOpS Direct Method Cluster Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3719.68	5191.83	4440.74	4396.18	4593.42
Earth to Mars Cost	3703.85	5171.60	4407.59	4279.62	4534.26
Mars to Jupiter Cost	15.84	20.23	33.15	116.56	59.15

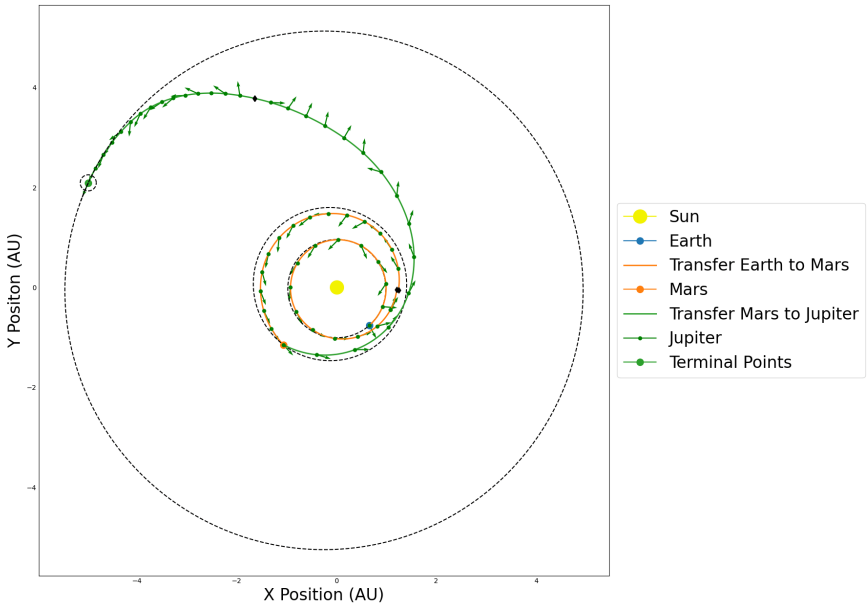

Figure B.46: Cluster Pruning Direct Method Optimal Solution 1 Trajectory

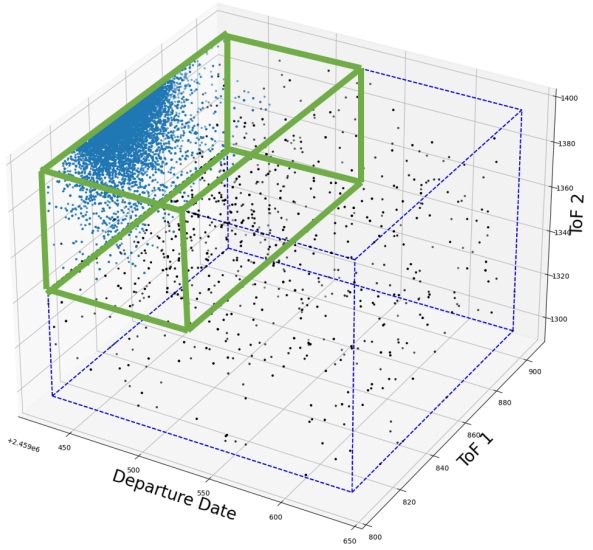

Figure B.47: Cluster Pruning Direct Method Optimal Solution 1 Clustering

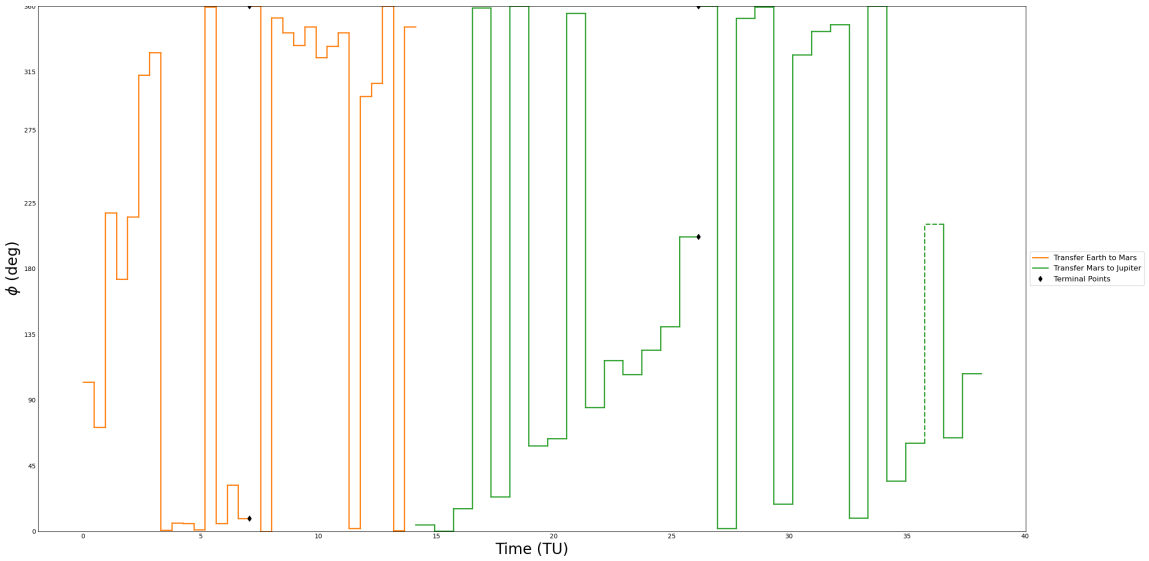

Figure B.48: Cluster Pruning Direct Method Optimal Solution 1 Thrust

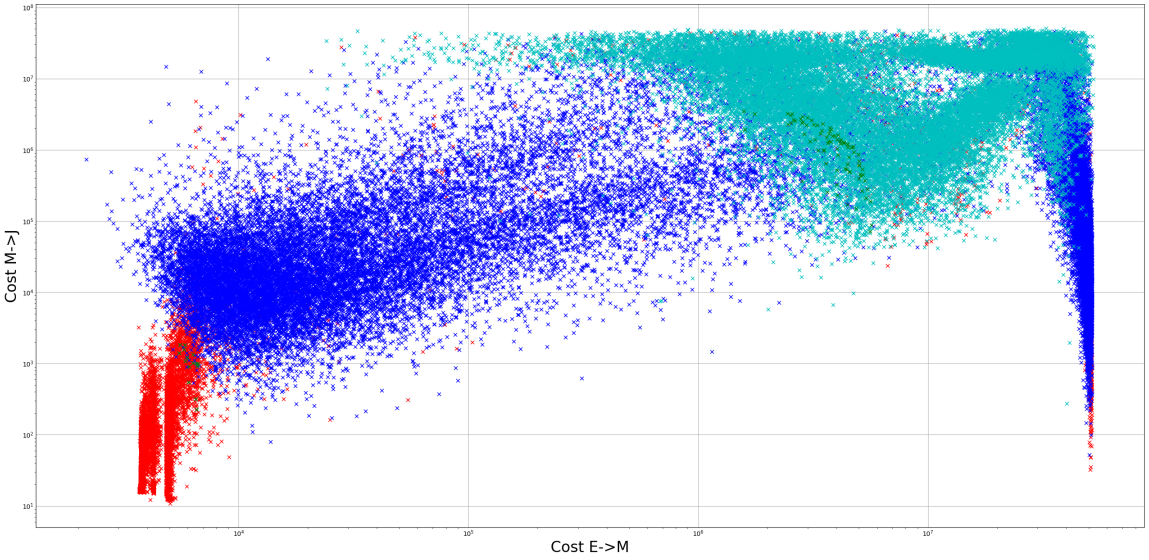

Figure B.49: Cluster Pruning Direct Method Optimal Solution 1 Leg Tracking

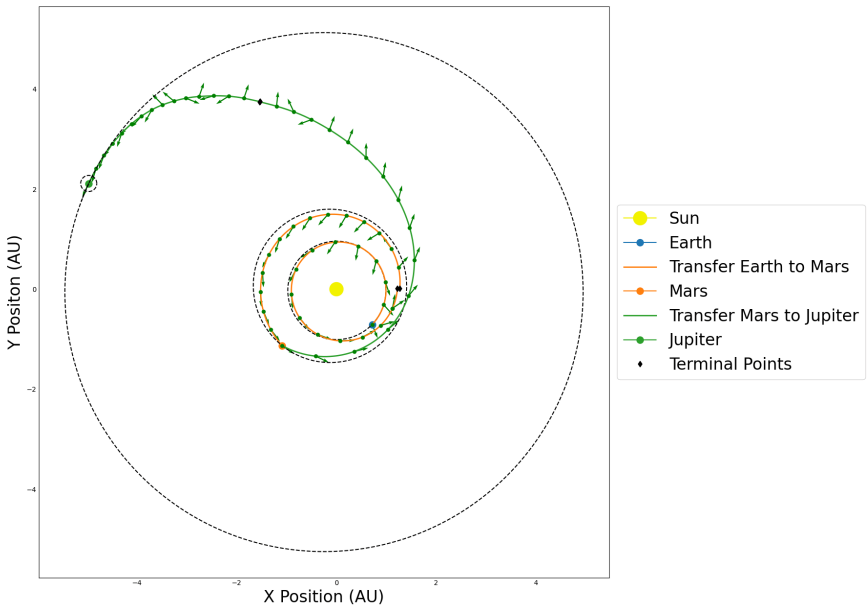

Figure B.50: Cluster Pruning Direct Method Optimal Solution 2 Trajectory

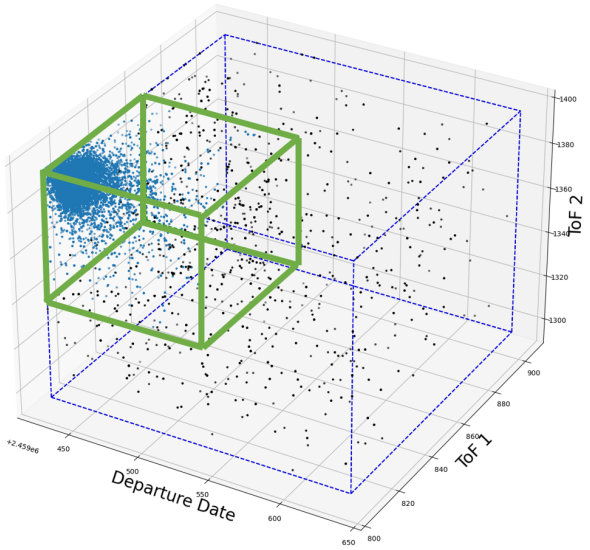

Figure B.51: Cluster Pruning Direct Method Optimal Solution 2 Clustering

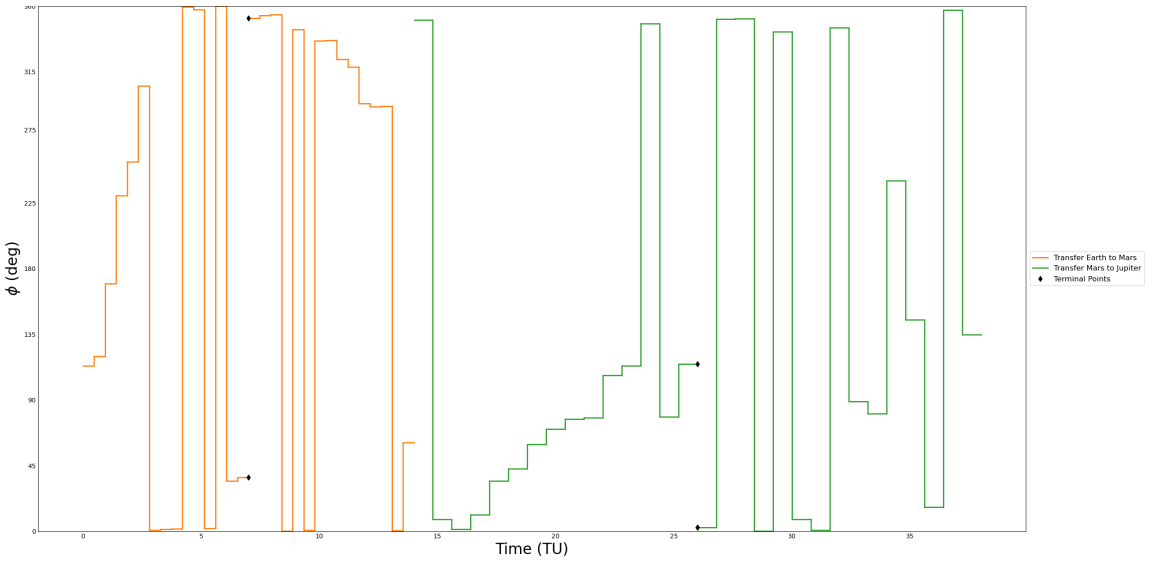

Figure B.52: Cluster Pruning Direct Method Optimal Solution 2 Thrust

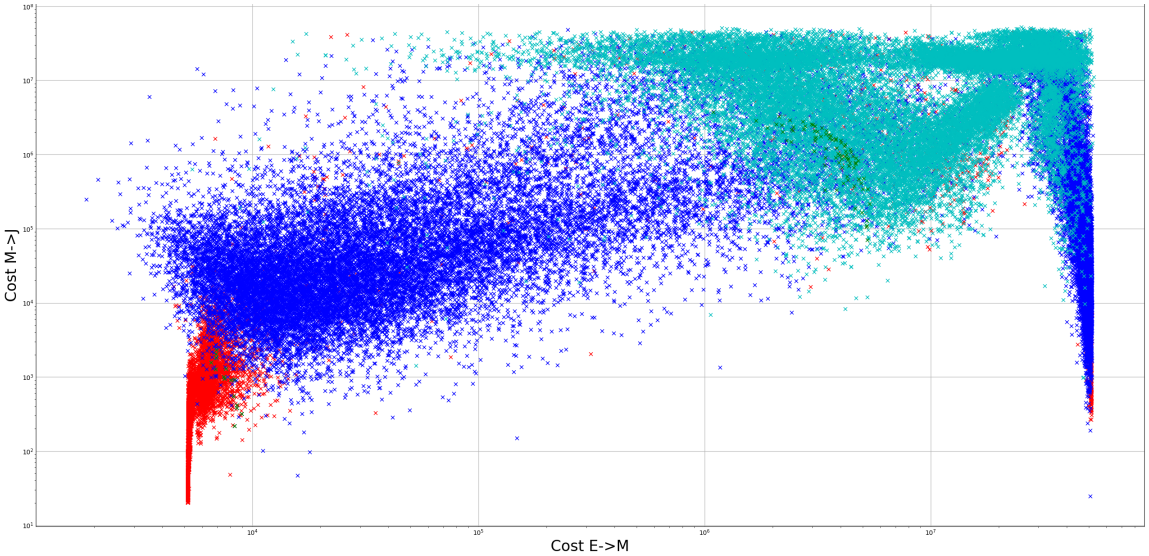

Figure B.53: Cluster Pruning Direct Method Optimal Solution 2 Leg Tracking

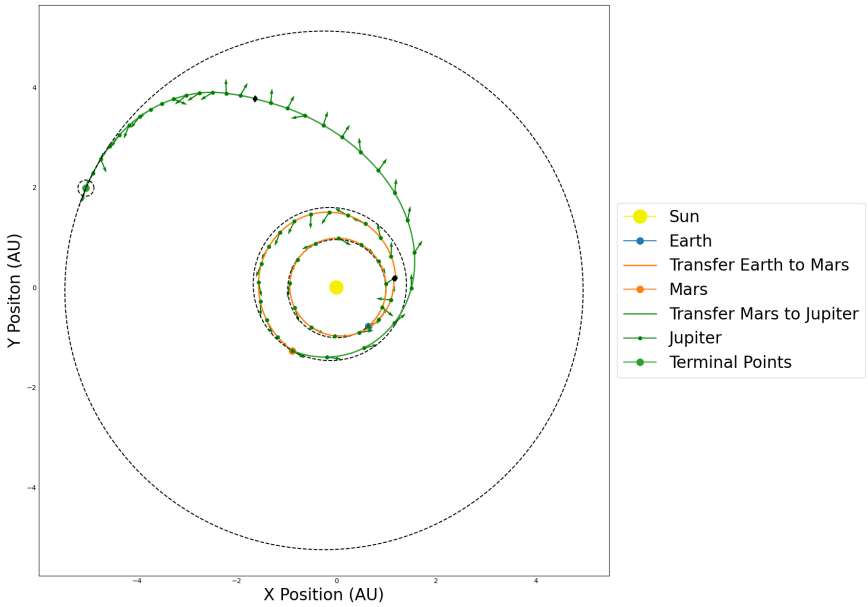

Figure B.54: Cluster Pruning Direct Method Optimal Solution 3 Trajectory

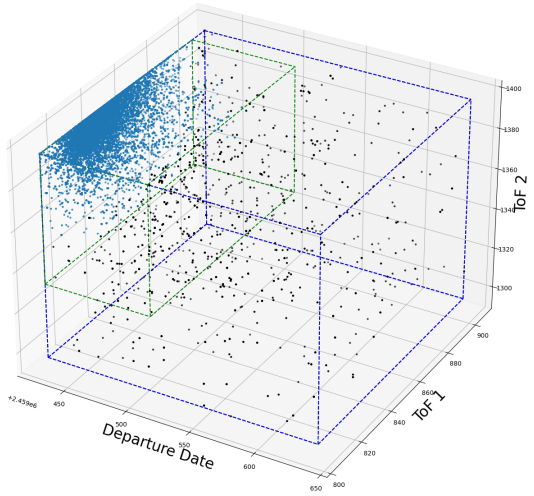

Figure B.55: Cluster Pruning Direct Method Optimal Solution 3 Clustering

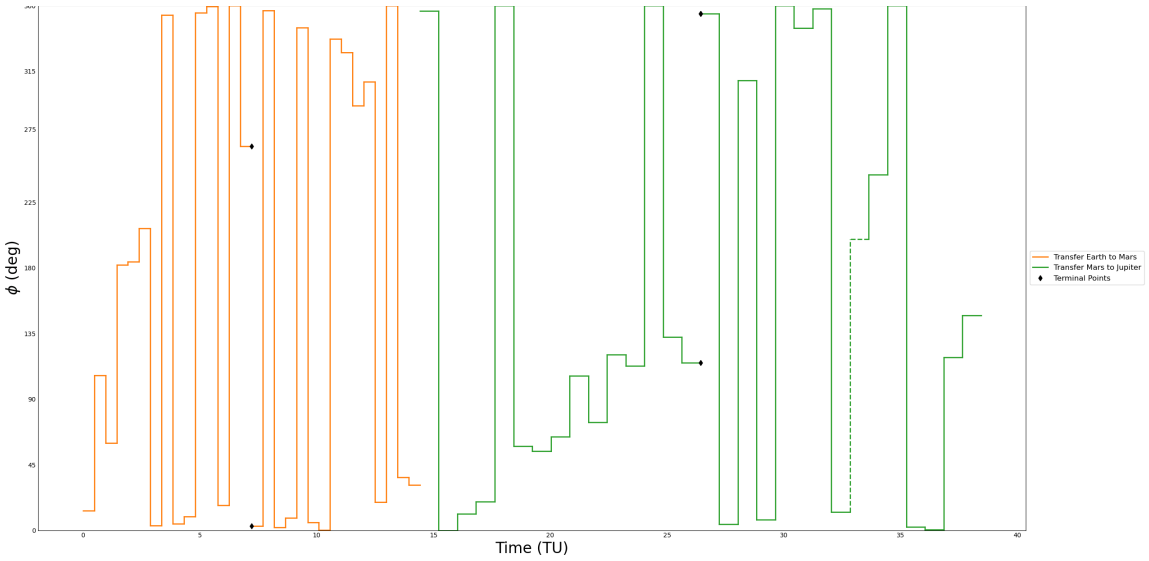

Figure B.56: Cluster Pruning Direct Method Optimal Solution 3 Thrust

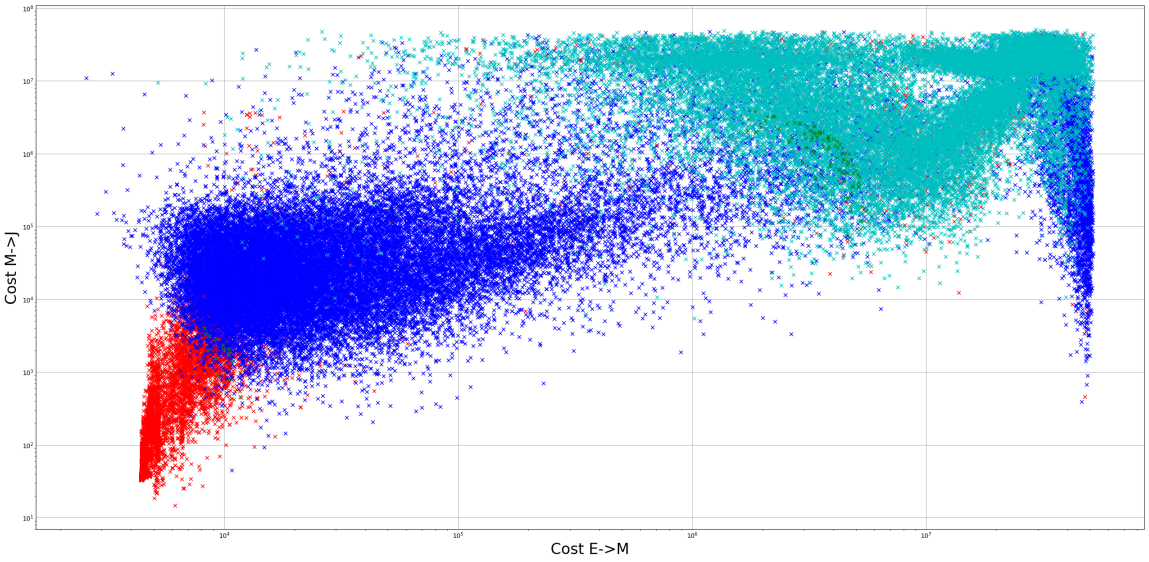

Figure B.57: Cluster Pruning Direct Method Optimal Solution 3 Leg Tracking

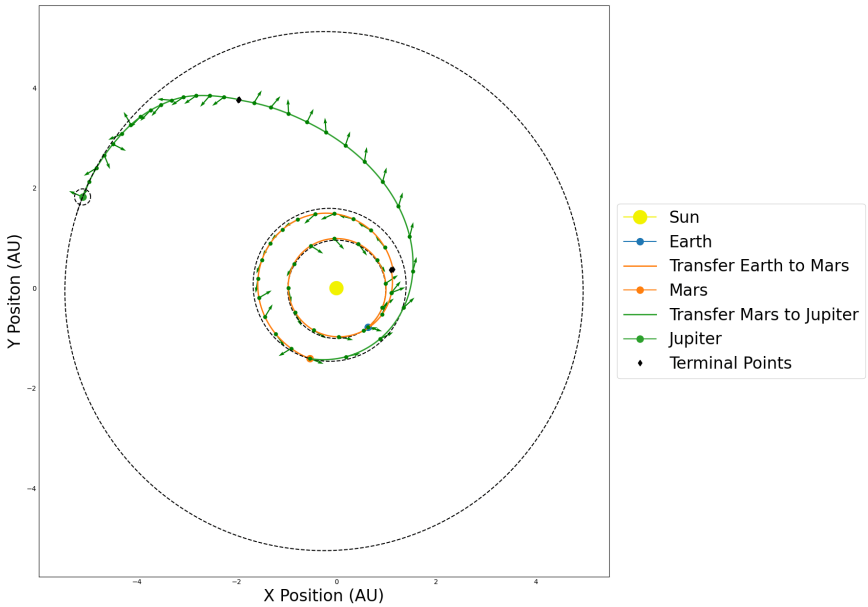

Figure B.58: Cluster Pruning Direct Method Optimal Solution 4 Trajectory

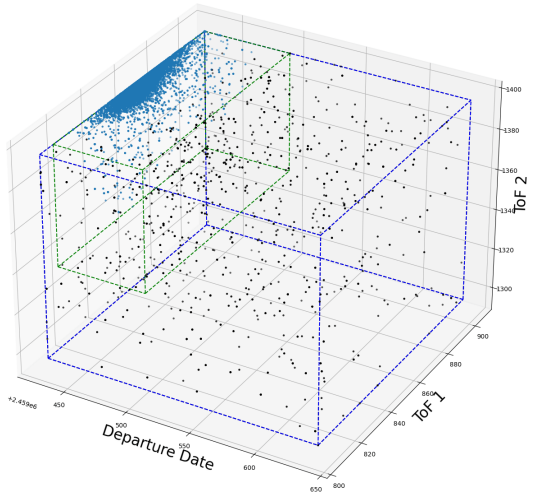

Figure B.59: Cluster Pruning Direct Method Optimal Solution 4 Clustering

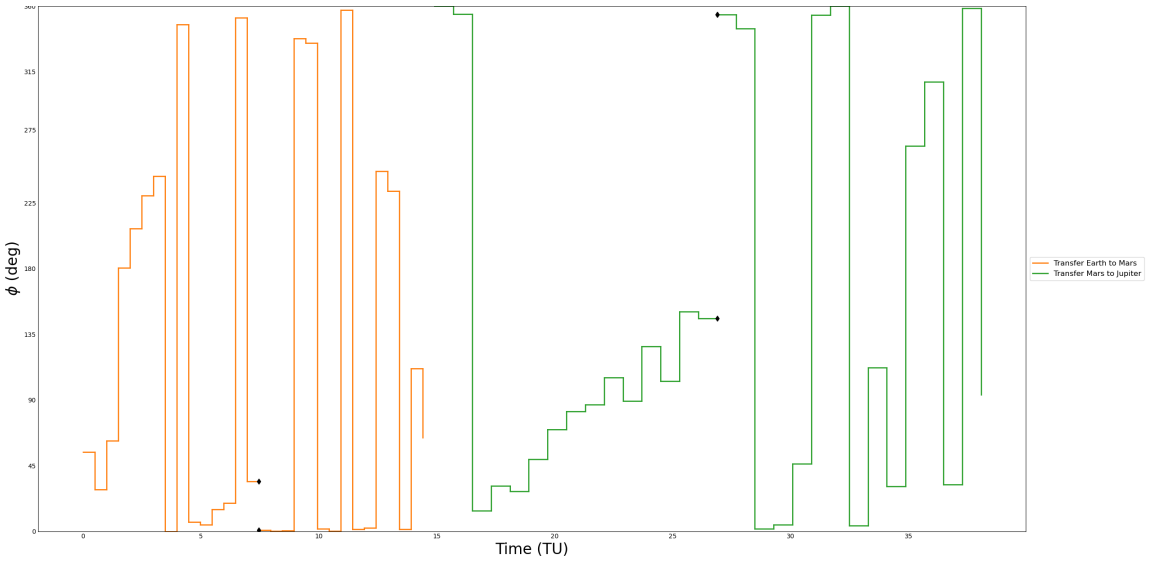

Figure B.60: Cluster Pruning Direct Method Optimal Solution 4 Thrust

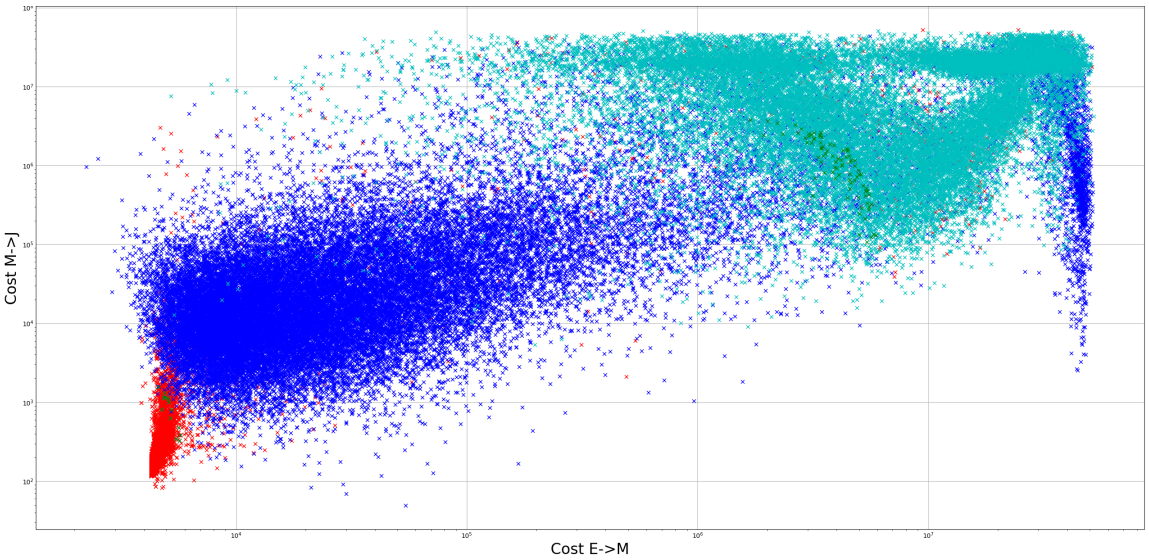

Figure B.61: Cluster Pruning Direct Method Optimal Solution 4 Leg Tracking

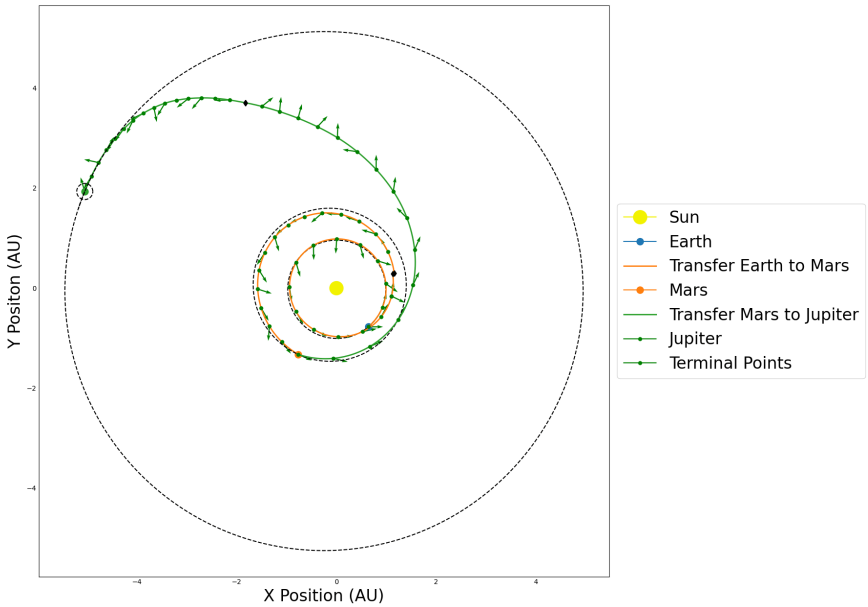

Figure B.62: Cluster Pruning Direct Method Optimal Solution 5 Trajectory

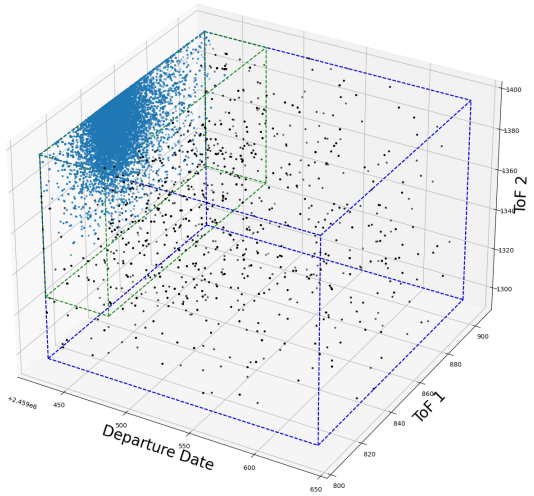

Figure B.63: Cluster Pruning Direct Method Optimal Solution 5 Clustering

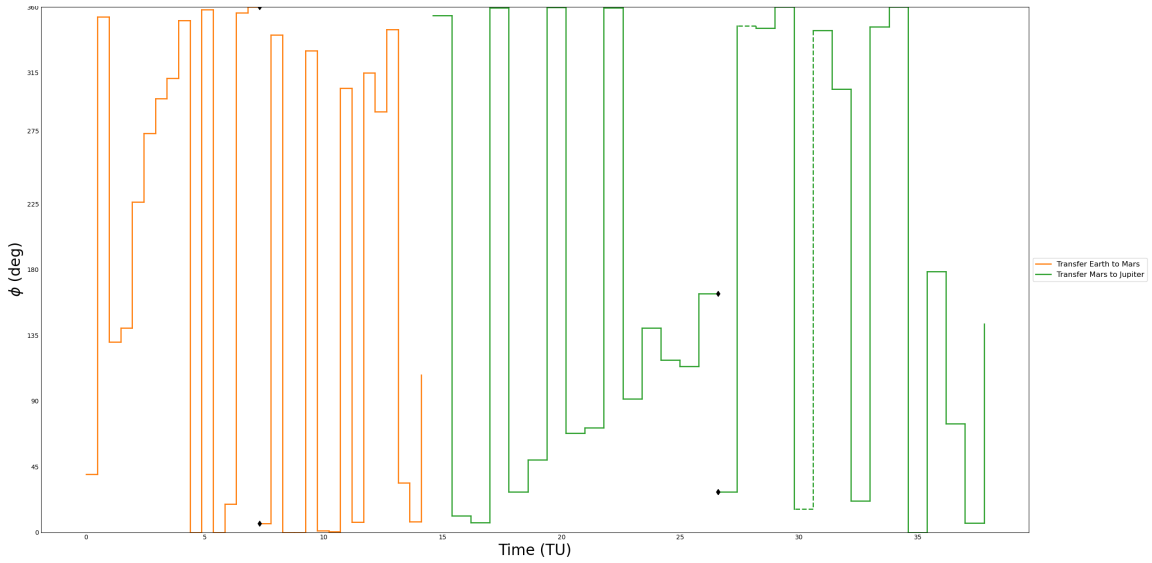

Figure B.64: Cluster Pruning Direct Method Optimal Solution 5 Thrust

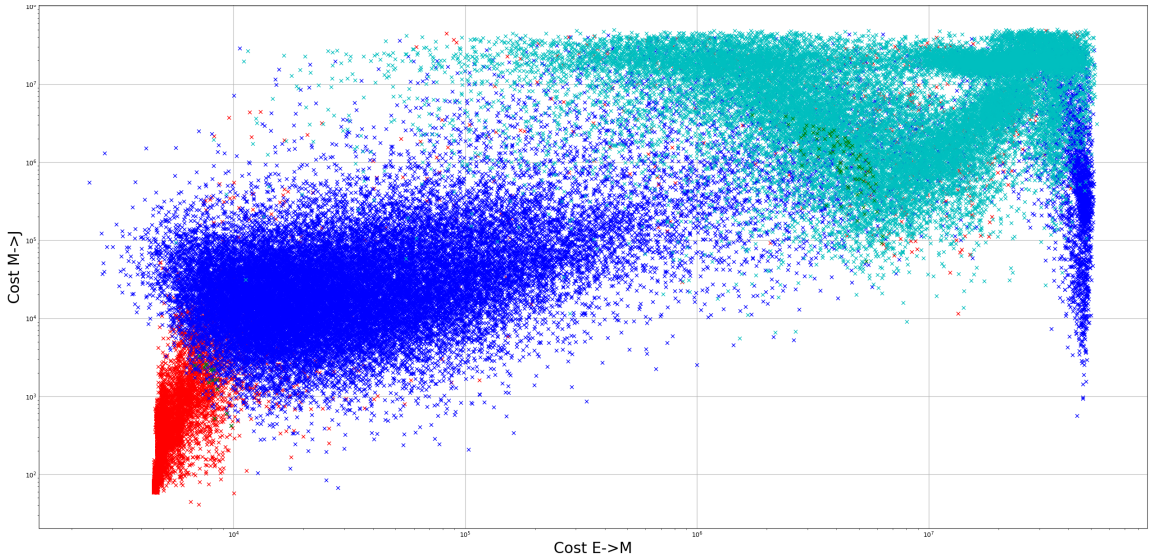

Figure B.65: Cluster Pruning Direct Method Optimal Solution 5 Leg Tracking

B.3.1.1 Cluster Pruning with Thruster Angle Pruning

Table B.9: MGALT-STOpS Direct Method Cluster Pruning with Thruster Angle Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	4758.73	5762.23	4599.81	4740.43	4294.75
Departure Date	Aug 19, 2021	Aug 31, 2021	Aug 29, 2021	Aug 27, 2021	Aug 11, 2021
Flyby ToF (days)	808	806	806	810	809
Arrival ToF (days)	1395	1396	1396	1396	1395
Total ToF (years)	6.03	6.03	6.03	6.04	6.03
Propellant Mass Fraction	36.56%	36.53%	36.52%	35.83%	36.56%

Table B.10: MGALT-STOpS Direct Method Cluster Pruning with Thruster Angle Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	4758.73	5762.23	4599.81	4740.43	4294.75
Earth to Mars Cost	4360.82	3392.21	4307.41	2956.40	2952.13
Mars to Jupiter Cost	397.92	2370.02	292.41	1784.03	1342.62

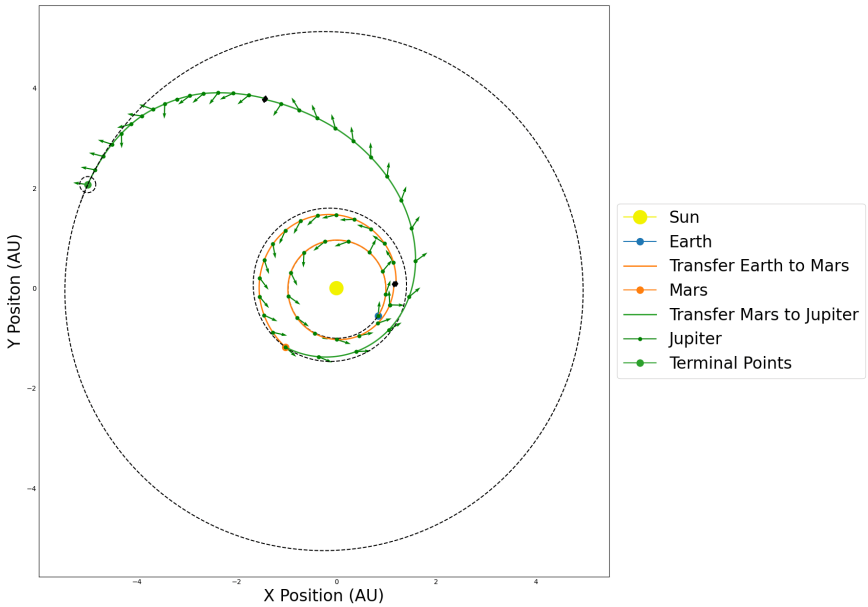

Figure B.66: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory

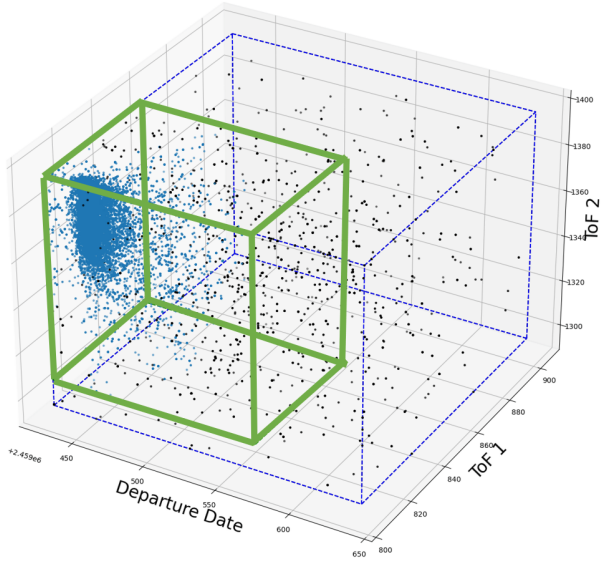

Figure B.67: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Clustering

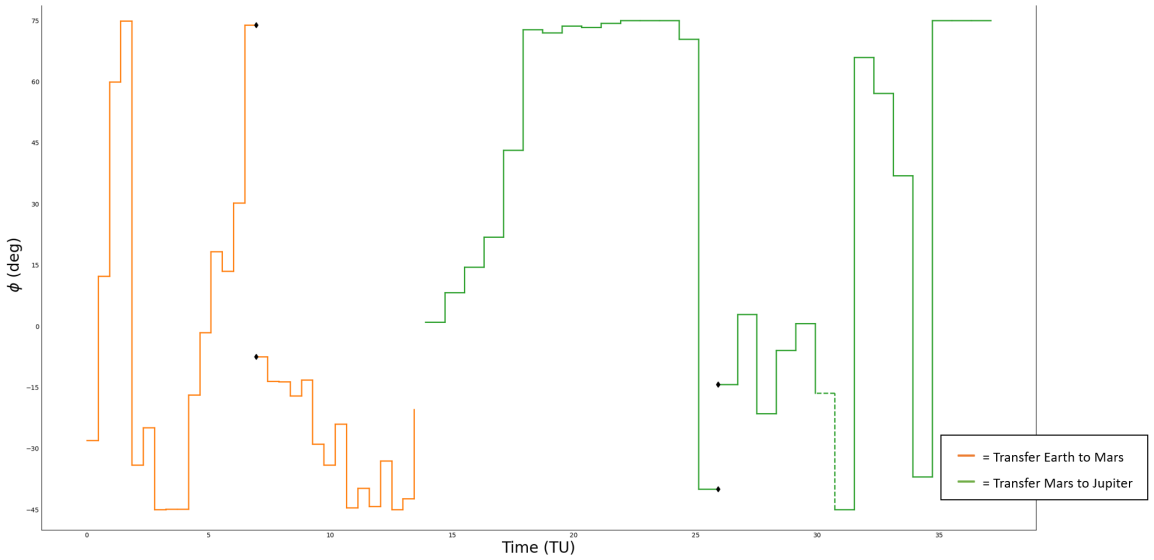

Figure B.68: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust

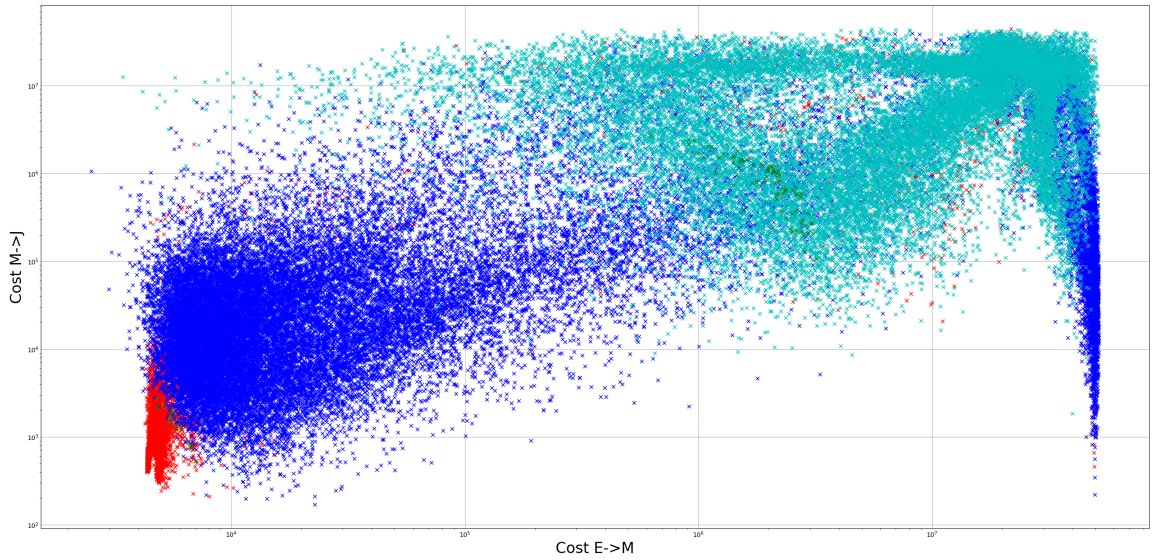

Figure B.69: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking

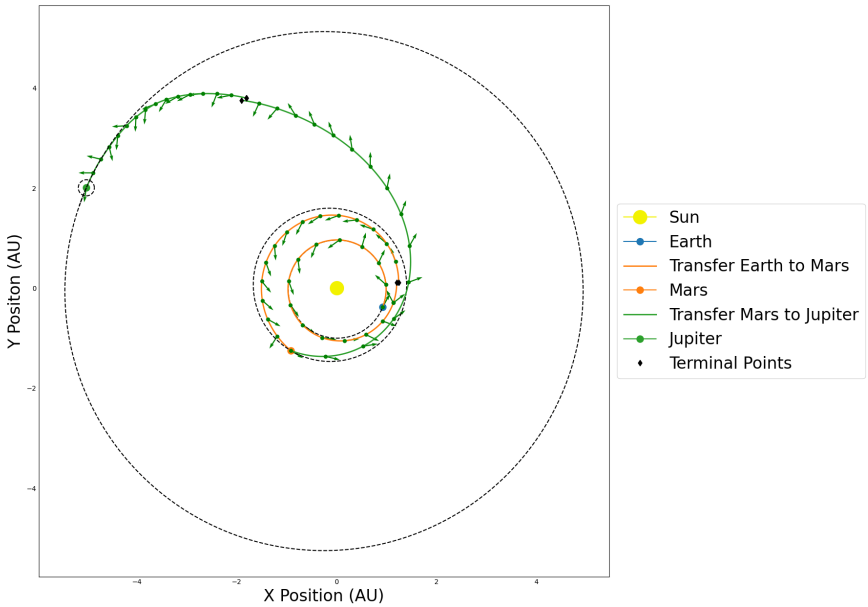

Figure B.70: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory

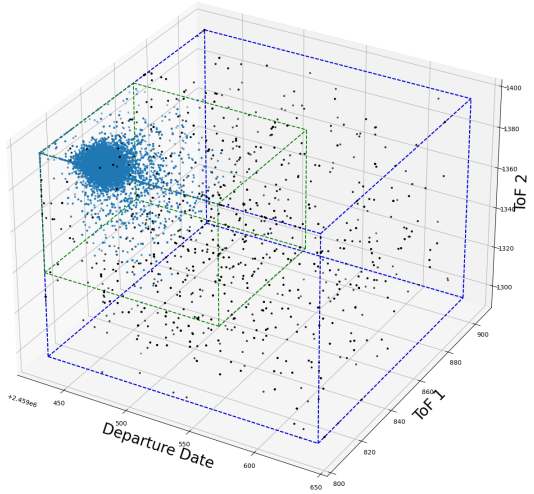

Figure B.71: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Clustering

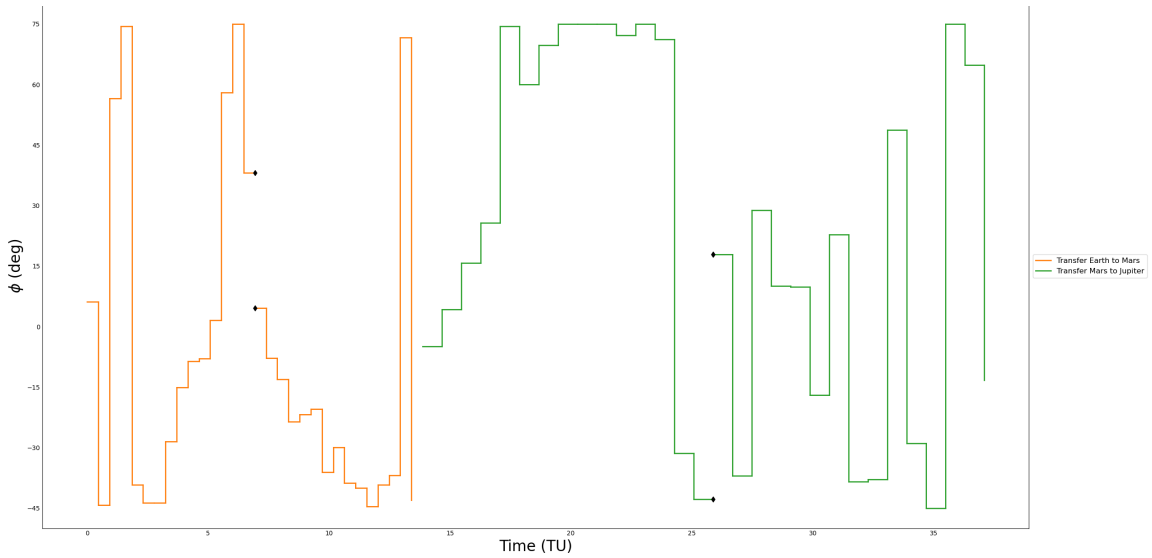

Figure B.72: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust

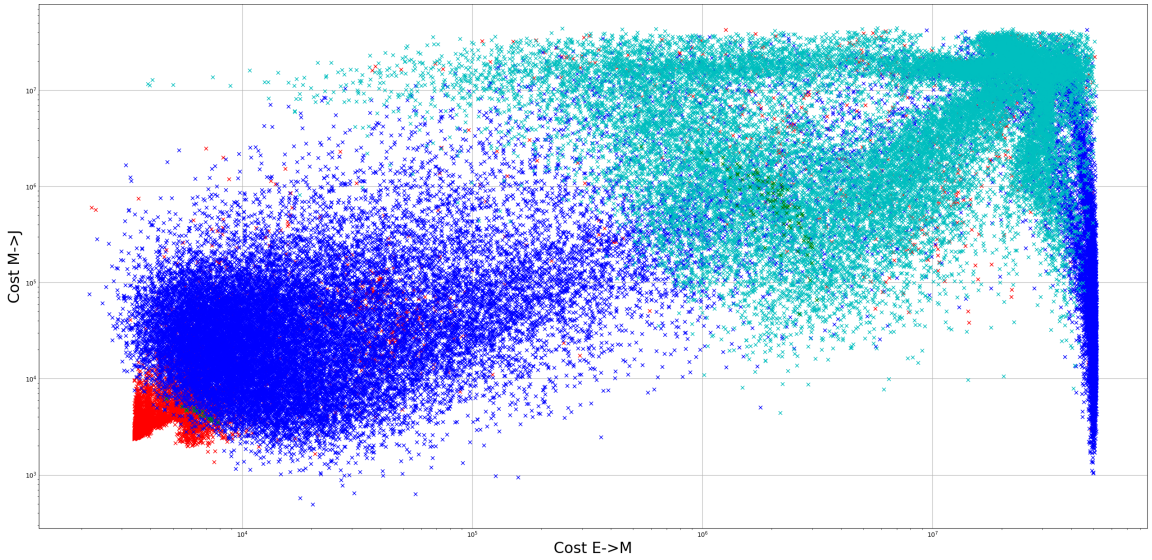

Figure B.73: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking

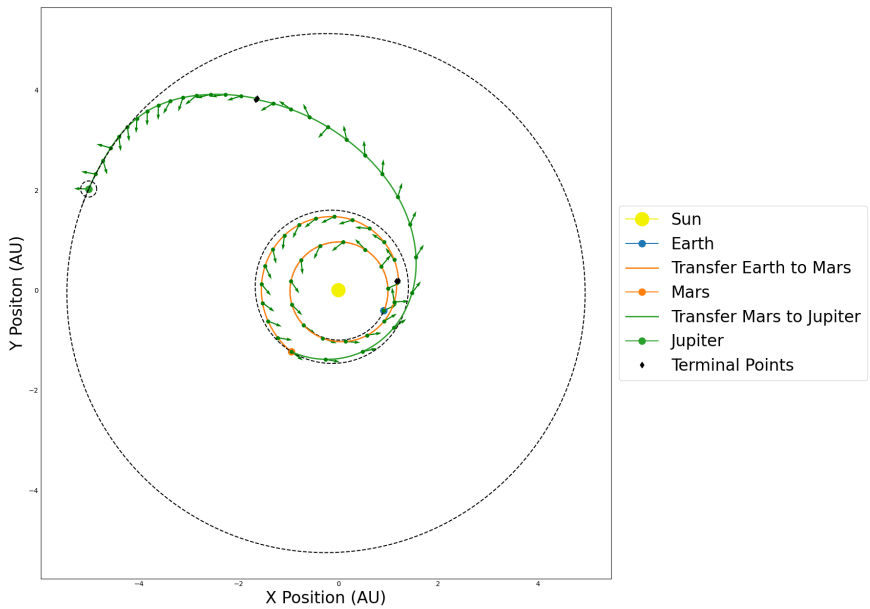

Figure B.74: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory

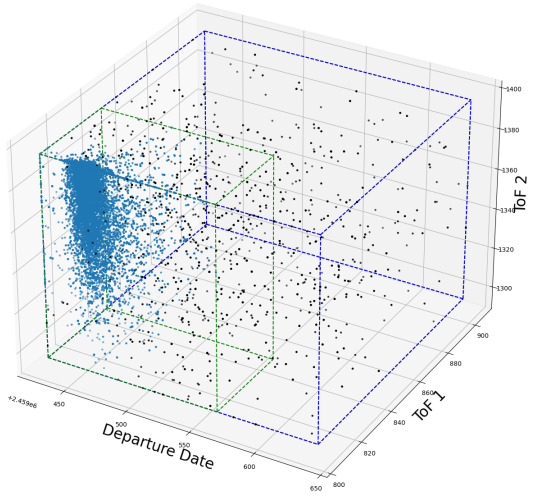

Figure B.75: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Clustering

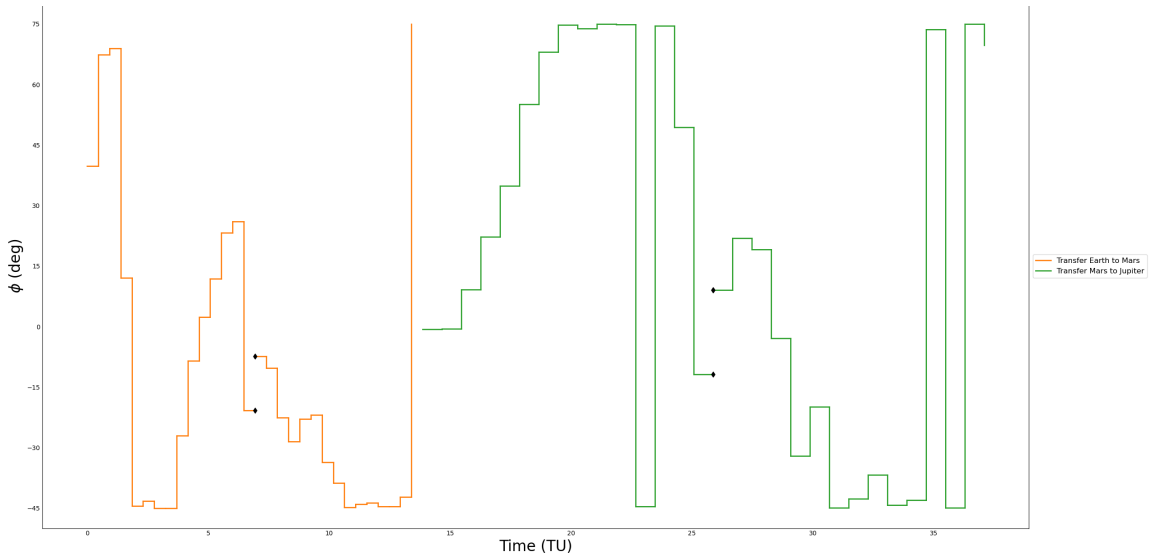

Figure B.76: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust

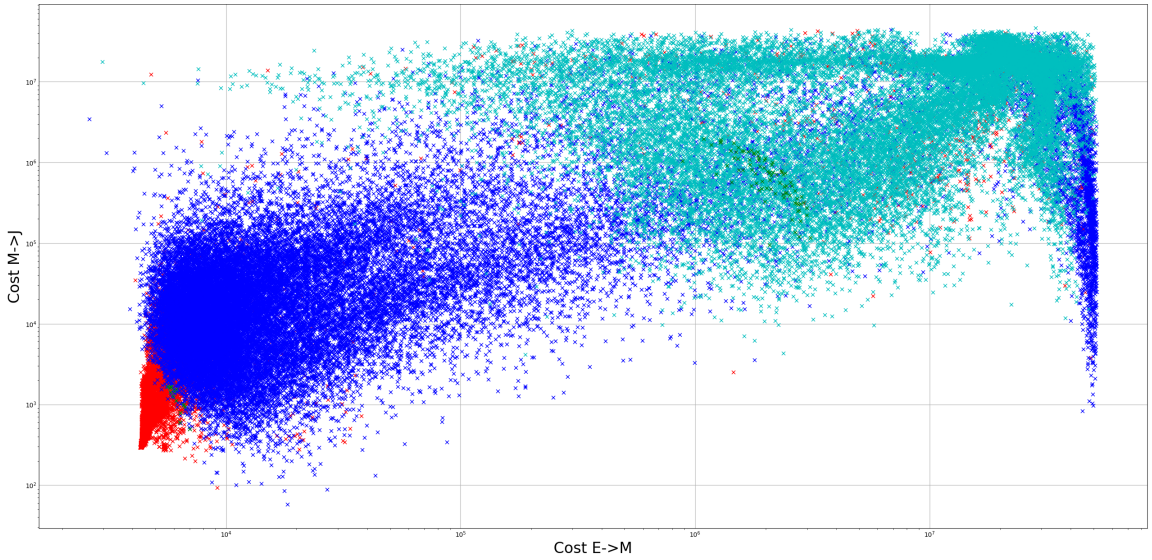

Figure B.77: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking

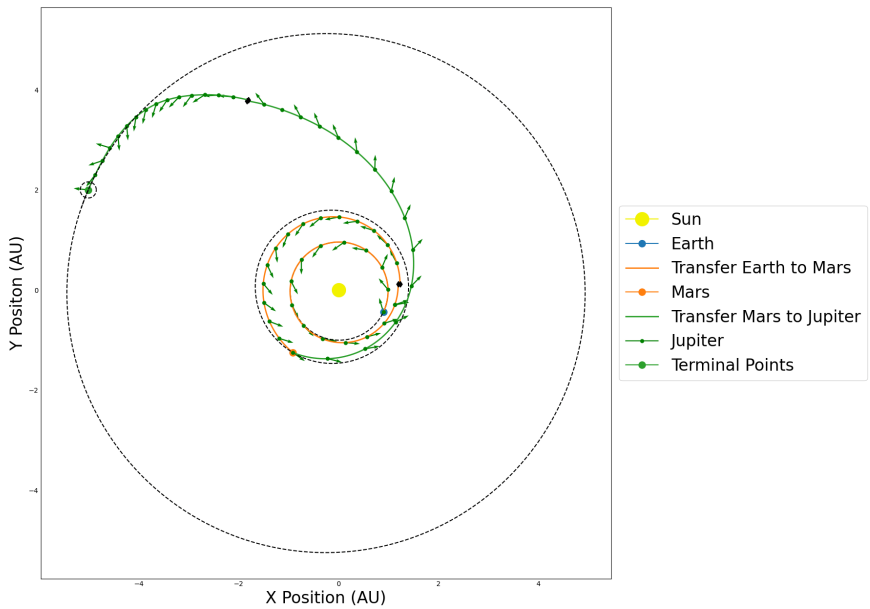

Figure B.78: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory

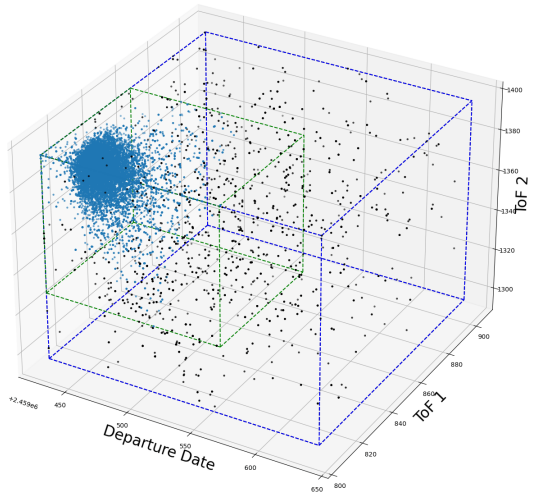

Figure B.79: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Clustering

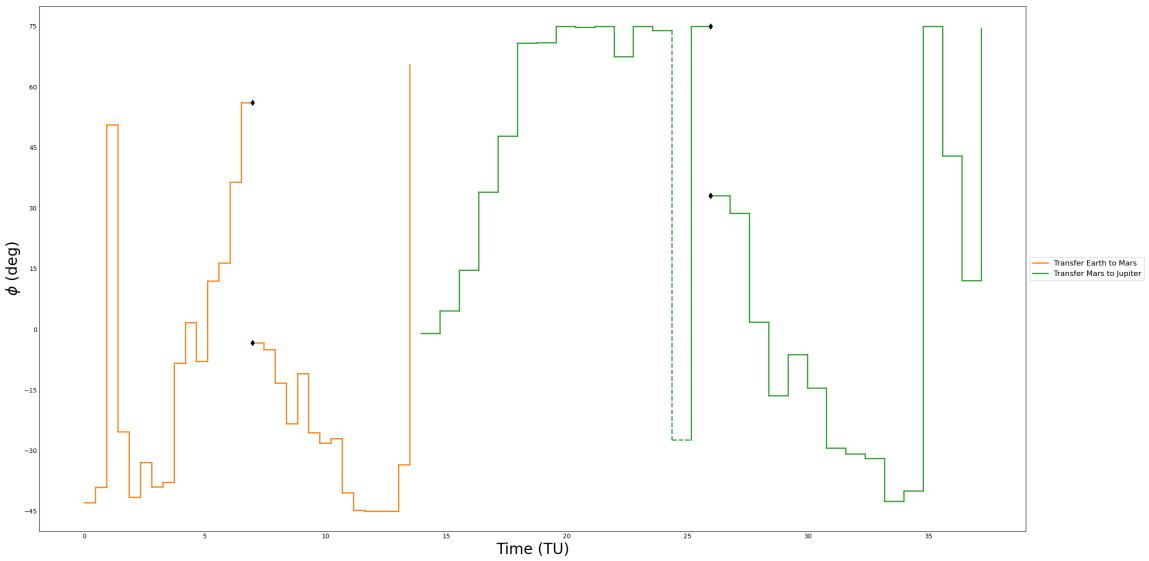

Figure B.80: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust

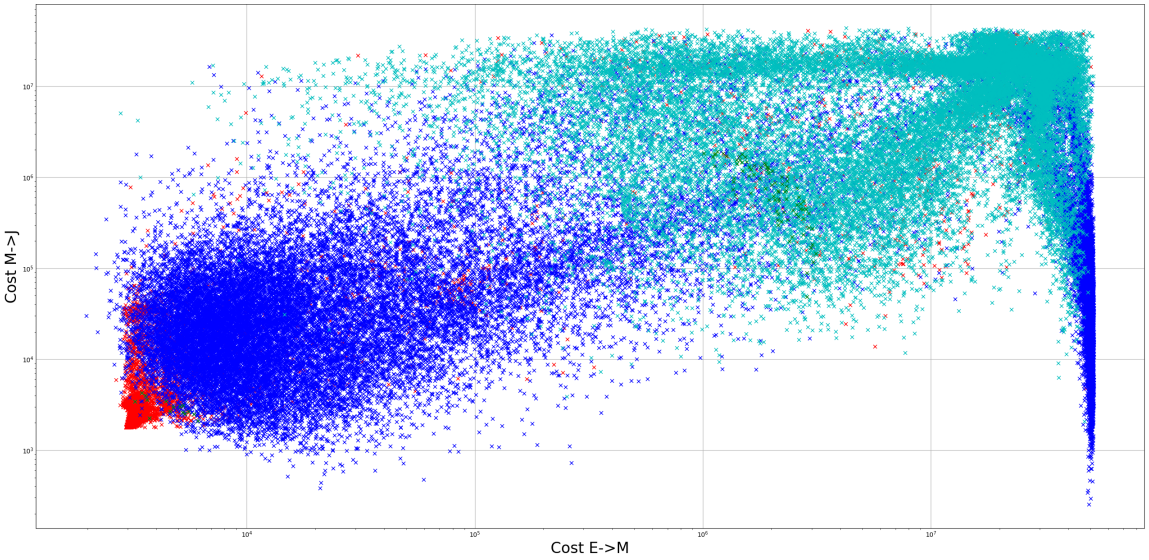

Figure B.81: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking

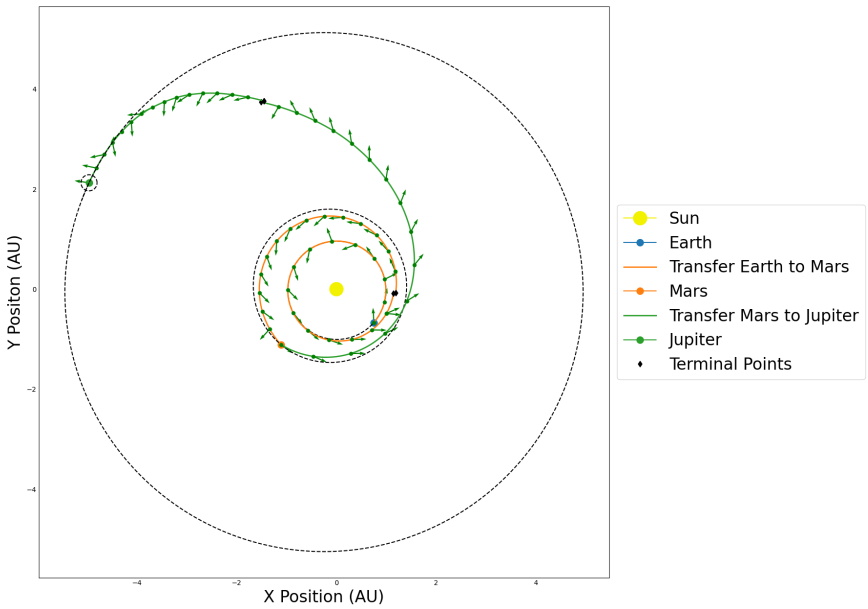

Figure B.82: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory

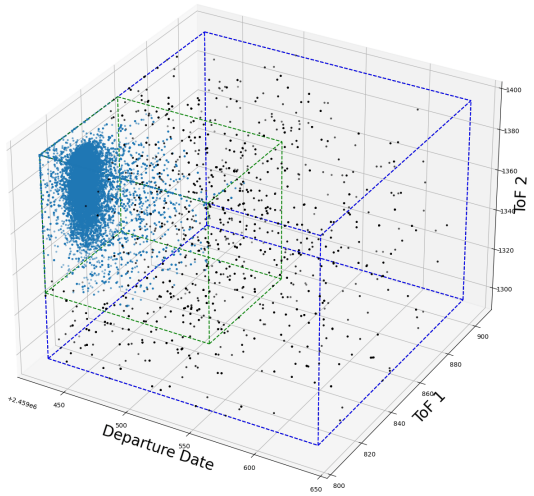

Figure B.83: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Clustering

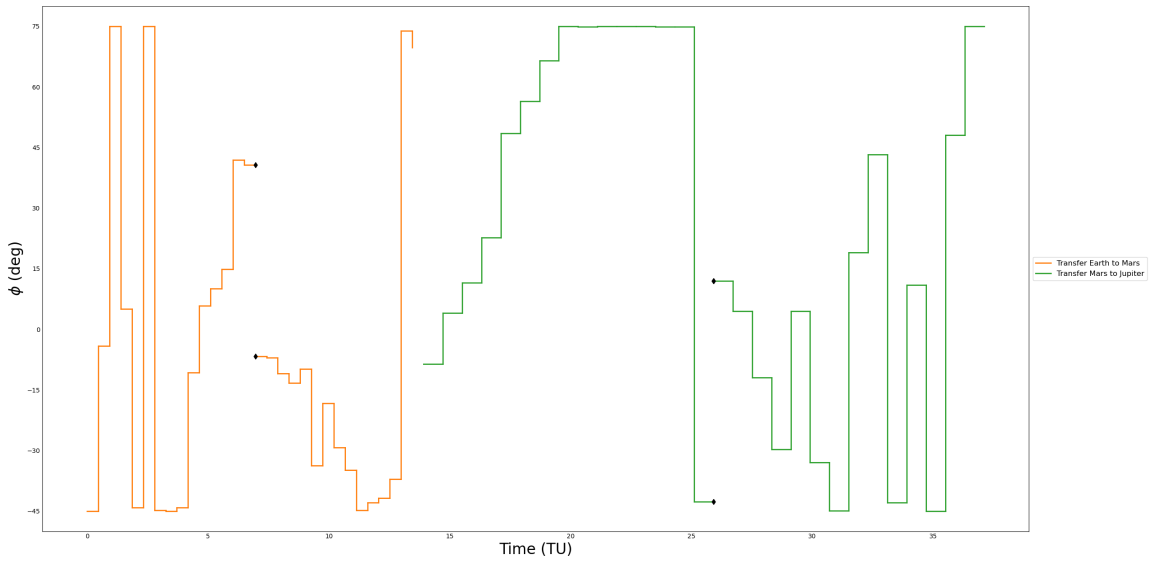

Figure B.84: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust

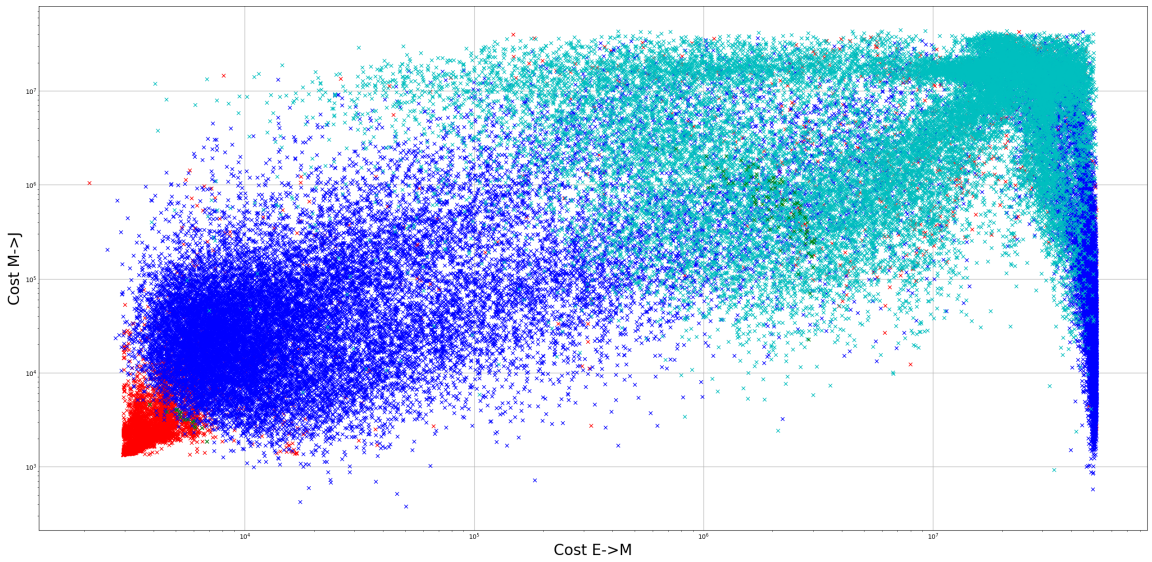

Figure B.85: Cluster Pruning with Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking

B.3.2 Indirect Method

Table B.11: MGALT-STOpS Indirect Method Cluster Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	963.54	782.82	4624.26	1282.84	3950.93
Departure Date	Oct 31, 2021	Nov 6, 2020	Aug 27, 2021	Nov 30, 2022	Jan 23, 2021
Flyby ToF (days)	702	852	745	852	852
Arrival ToF (days)	1698	1403	1559	1533	1748
Total ToF (years)	6.58	6.18	6.31	6.53	7.12
Propellant Mass Fraction	39.81%	37.40%	38.21%	39.57%	43.13%

Table B.12: MGALT-STOpS Indirect Method Cluster Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	963.54	782.82	4624.26	1282.84	3950.93
Earth to Mars Cost	331.94	706.23	207.72	1272.33	1377.45
Mars to Jupiter Cost	631.59	76.59	4416.54	10.51	2573.48

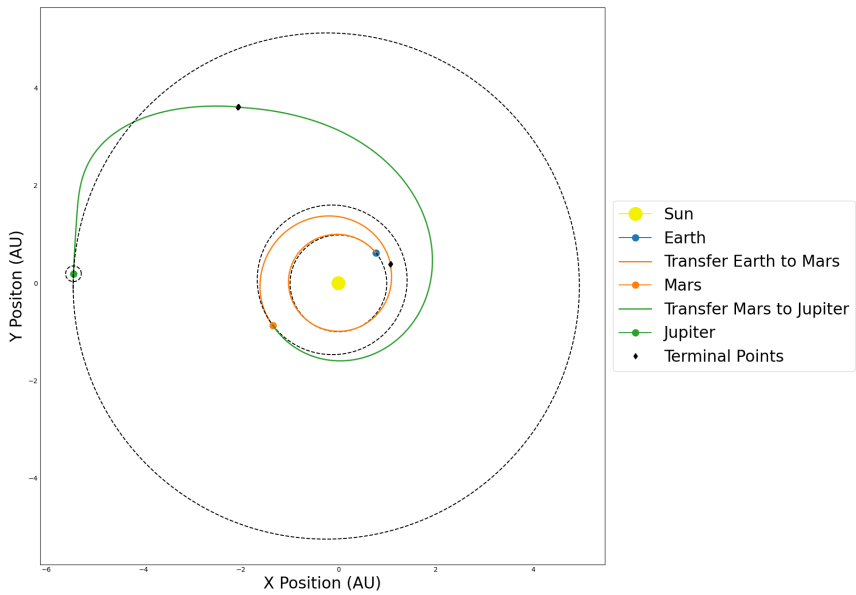

Figure B.86: Cluster Pruning Indirect Method Optimal Solution 1 Trajectory

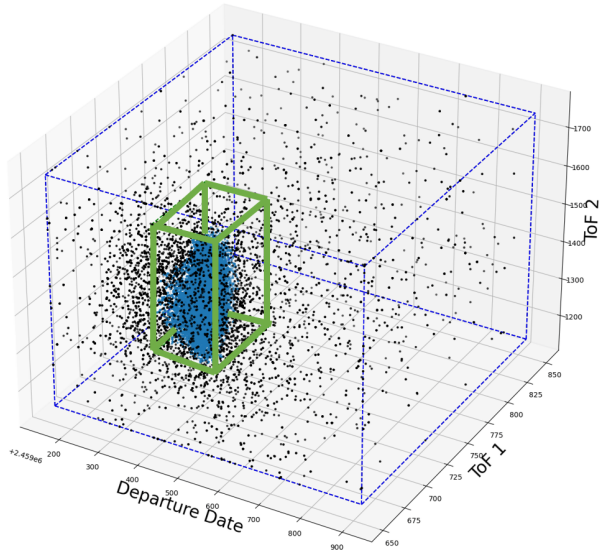

Figure B.87: Cluster Pruning Indirect Method Optimal Solution 1 Clustering

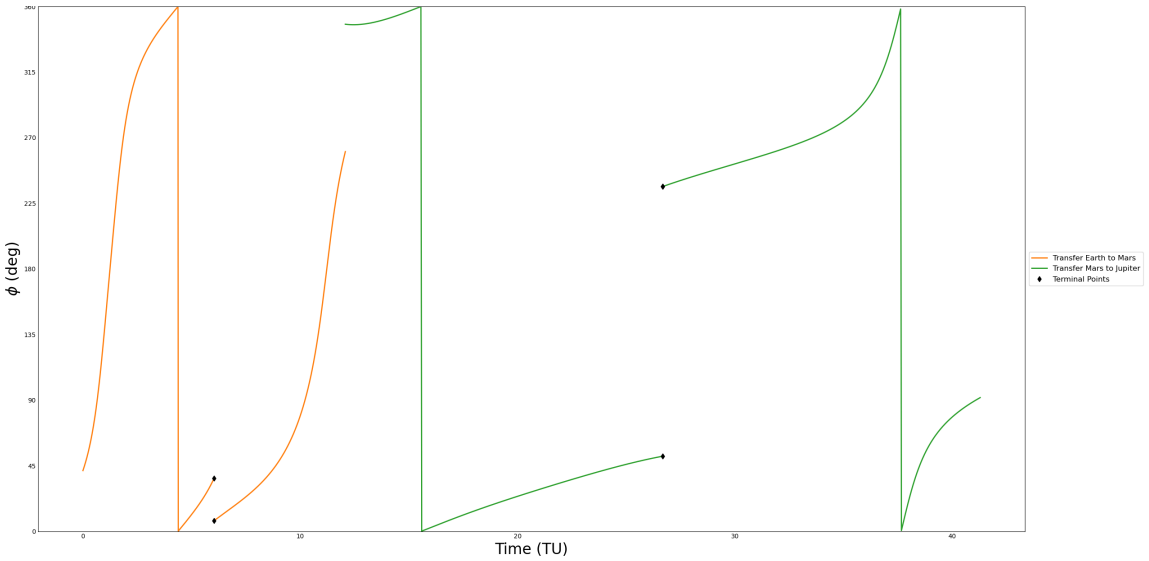

Figure B.88: Cluster Pruning Indirect Method Optimal Solution 1 Thrust

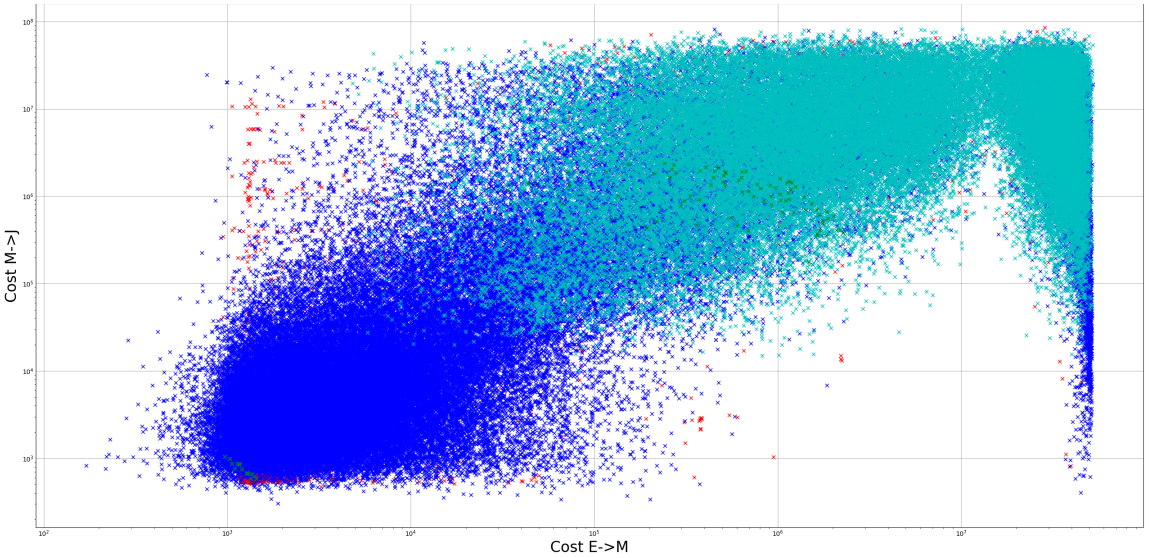

Figure B.89: Cluster Pruning Indirect Method Optimal Solution 1 Leg Tracking

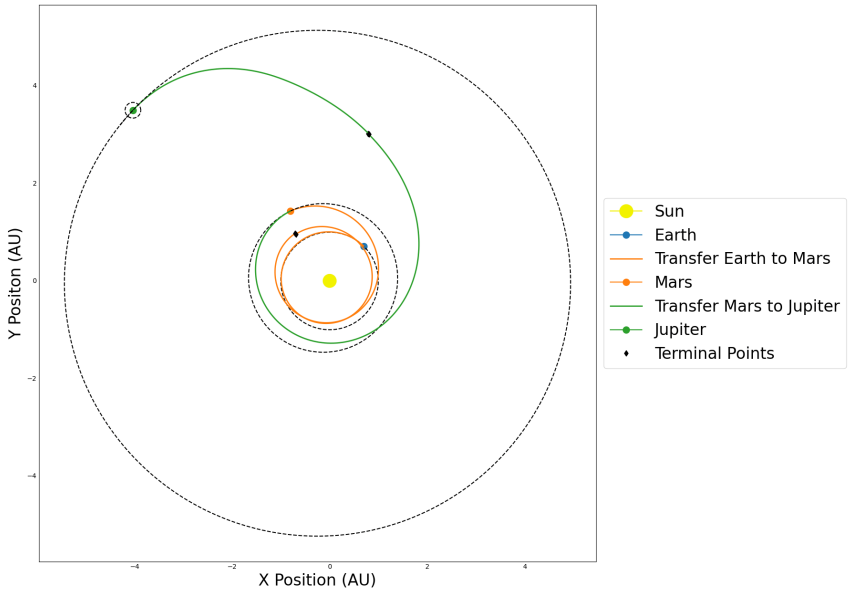

Figure B.90: Cluster Pruning Indirect Method Optimal Solution 2 Trajectory

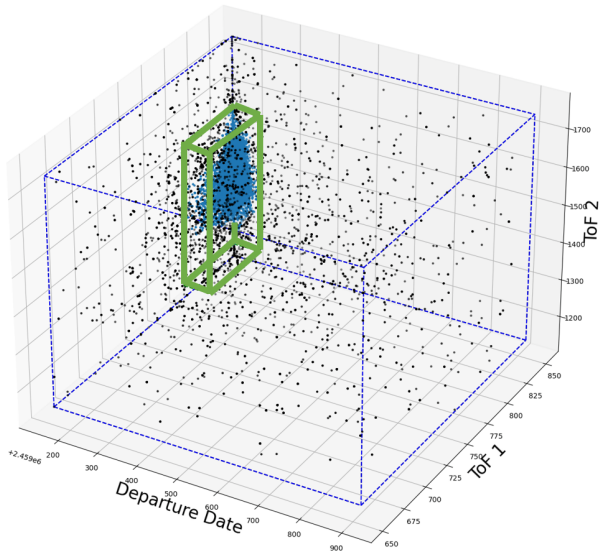

Figure B.91: Cluster Pruning Indirect Method Optimal Solution 2 Clustering

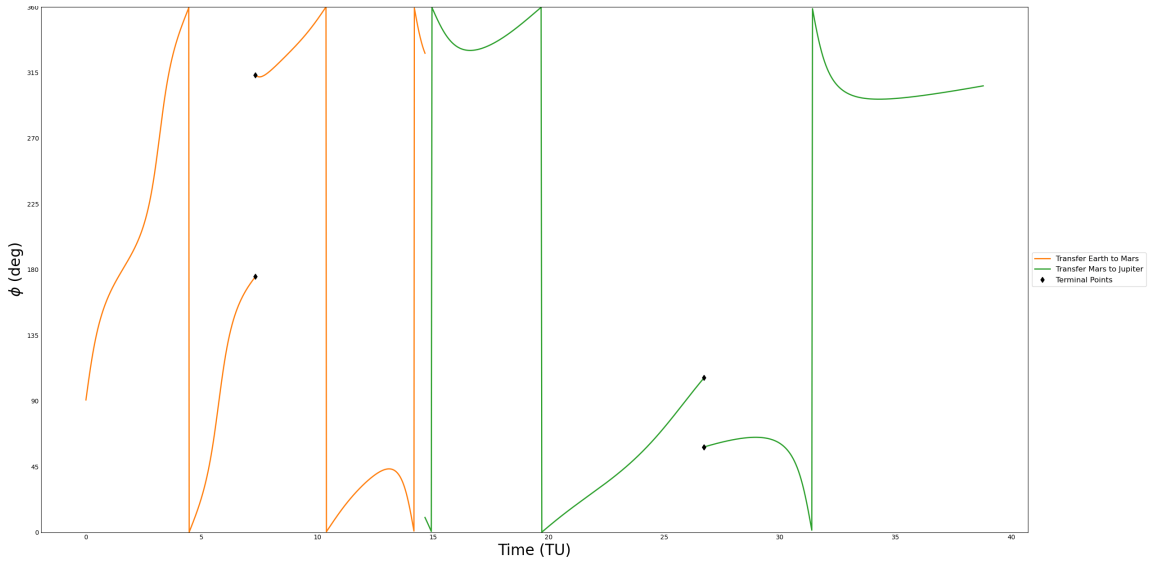

Figure B.92: Cluster Pruning Indirect Method Optimal Solution 2 Thrust

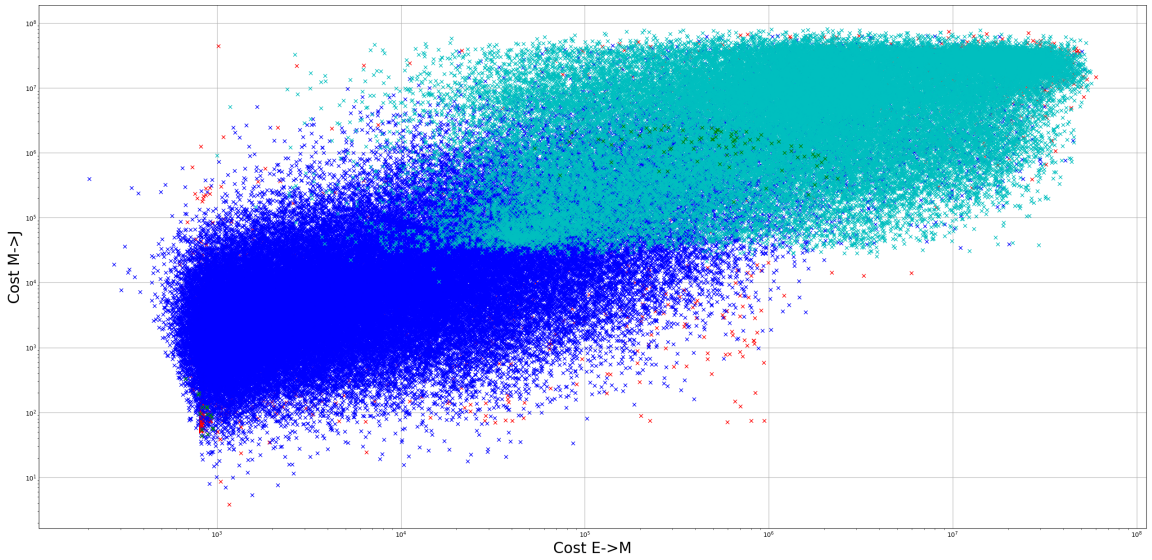

Figure B.93: Cluster Pruning Indirect Method Optimal Solution 2 Leg Tracking

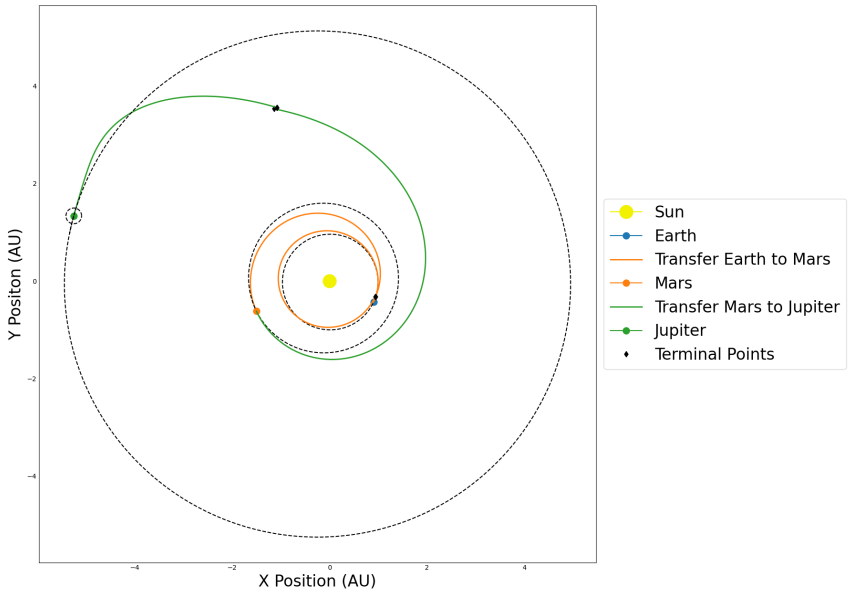

Figure B.94: Cluster Pruning Indirect Method Optimal Solution 3 Trajectory

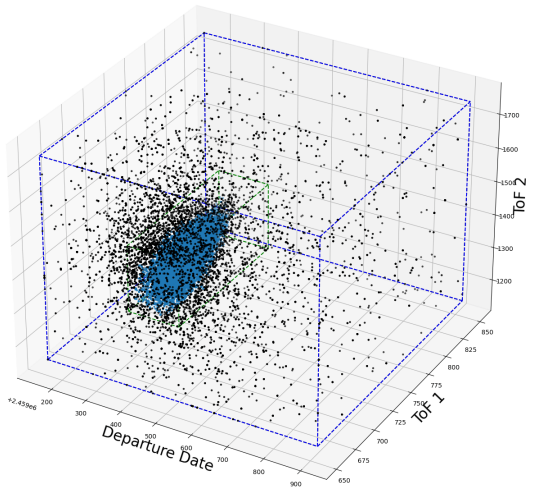

Figure B.95: Cluster Pruning Indirect Method Optimal Solution 3 Clustering

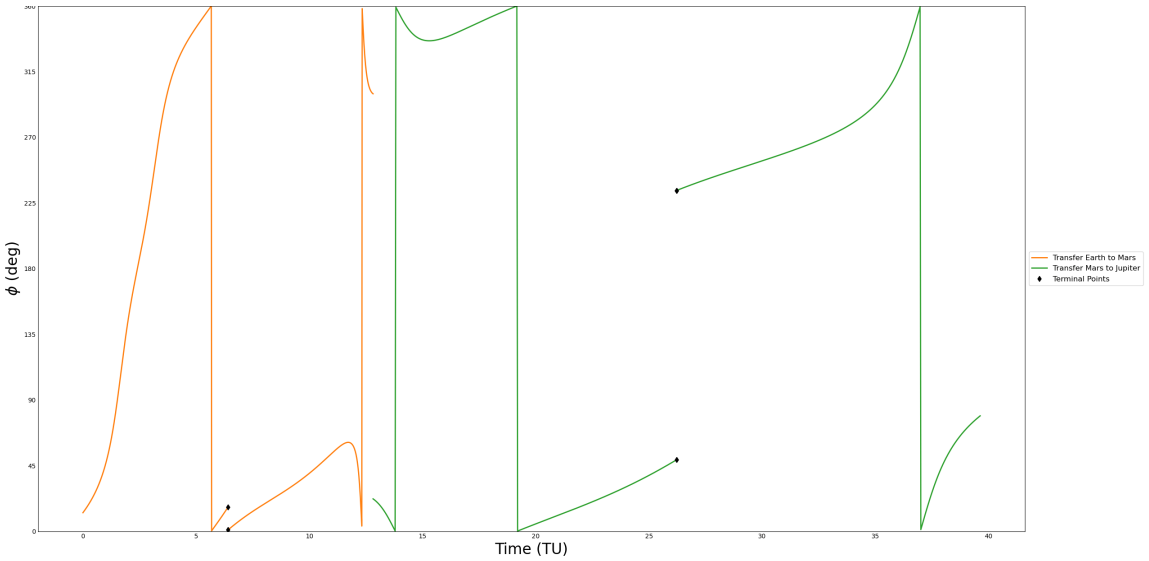

Figure B.96: Cluster Pruning Indirect Method Optimal Solution 3 Thrust

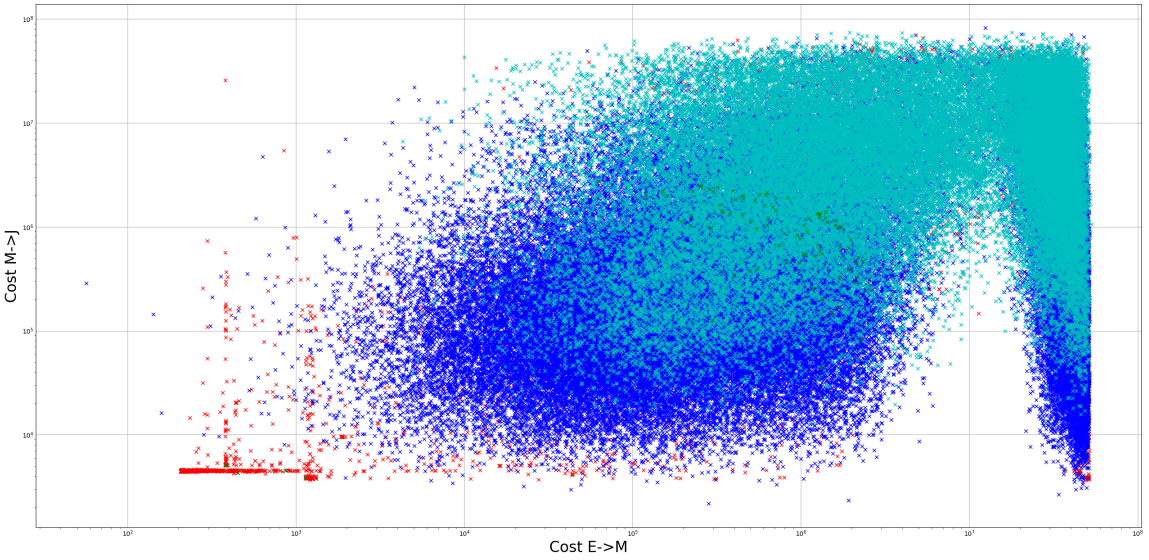

Figure B.97: Cluster Pruning Indirect Method Optimal Solution 3 Leg Tracking

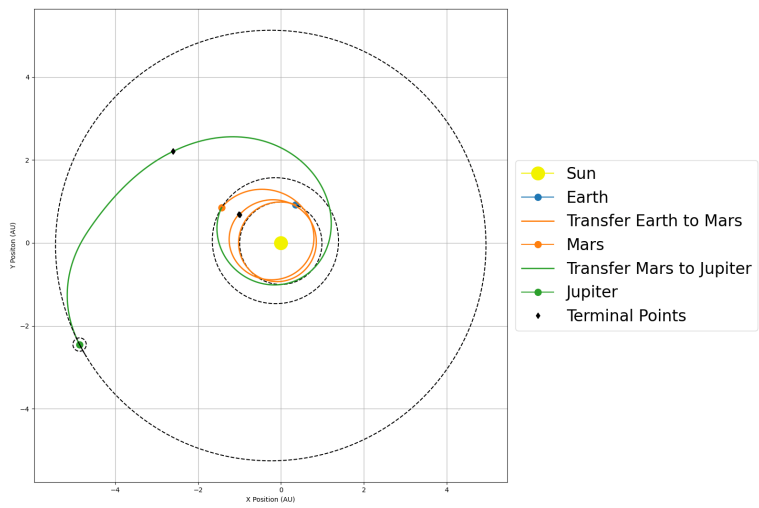

Figure B.98: Cluster Pruning Indirect Method Optimal Solution 4 Trajectory

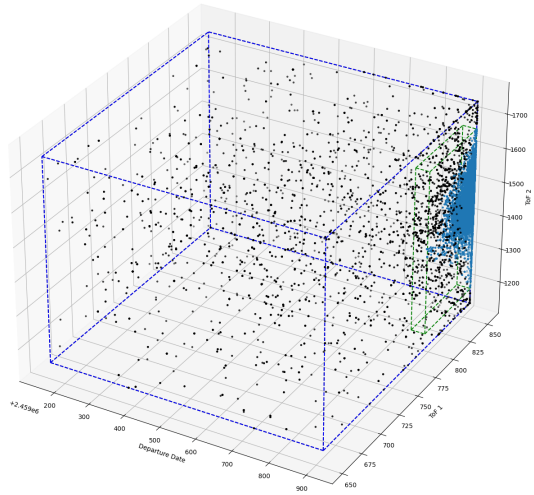

Figure B.99: Cluster Pruning Indirect Method Optimal Solution 4 Clustering

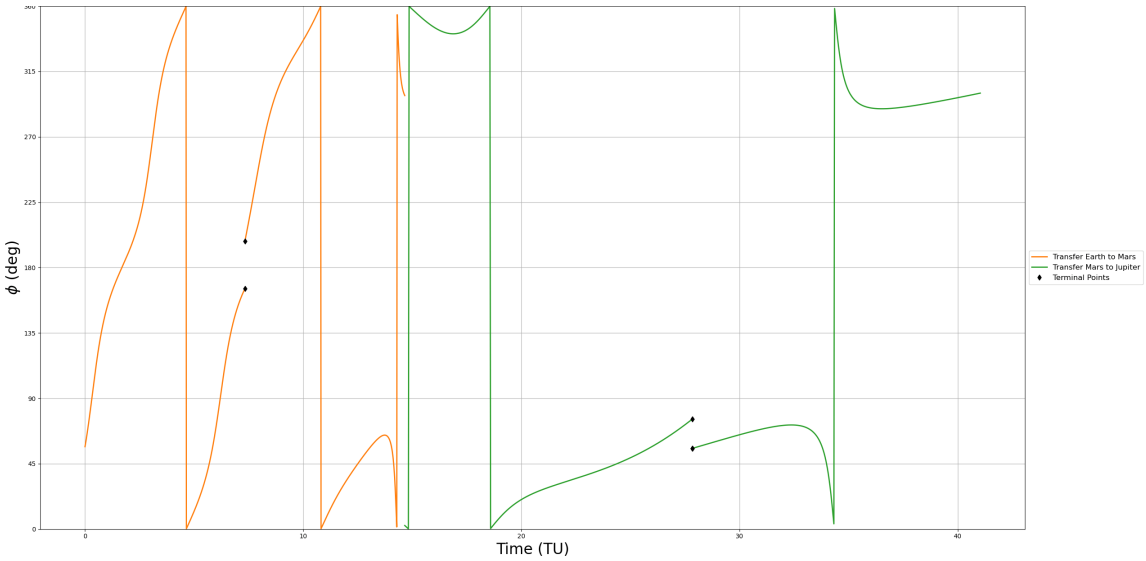

Figure B.100: Cluster Pruning Indirect Method Optimal Solution 4 Thrust

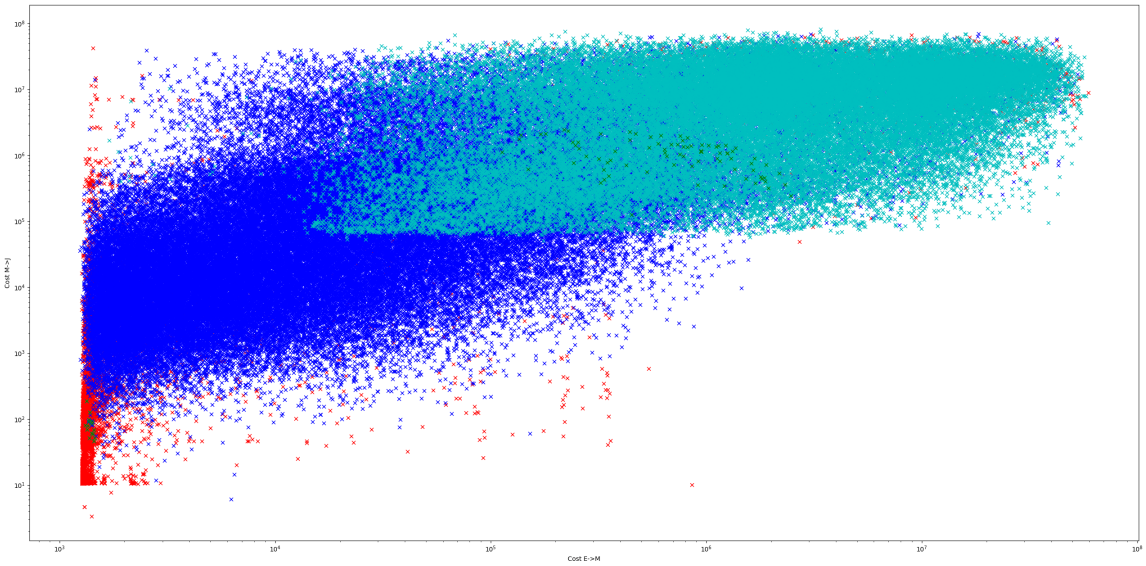

Figure B.101: Cluster Pruning Indirect Method Optimal Solution 4 Leg Tracking

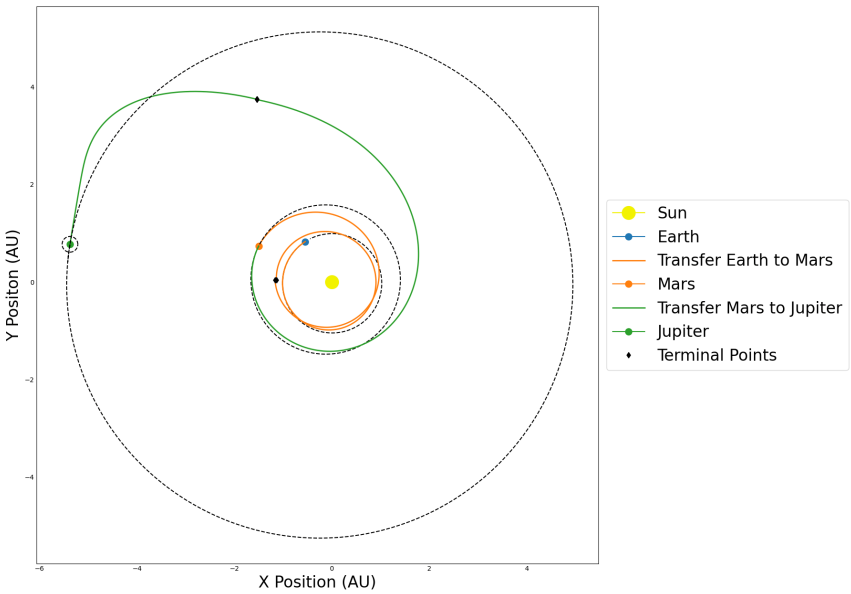

Figure B.102: Cluster Pruning Indirect Method Optimal Solution 5 Trajectory

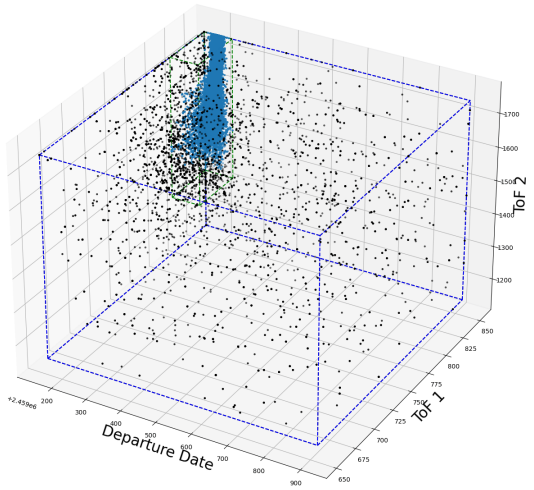

Figure B.103: Cluster Pruning Indirect Method Optimal Solution 5 Clustering

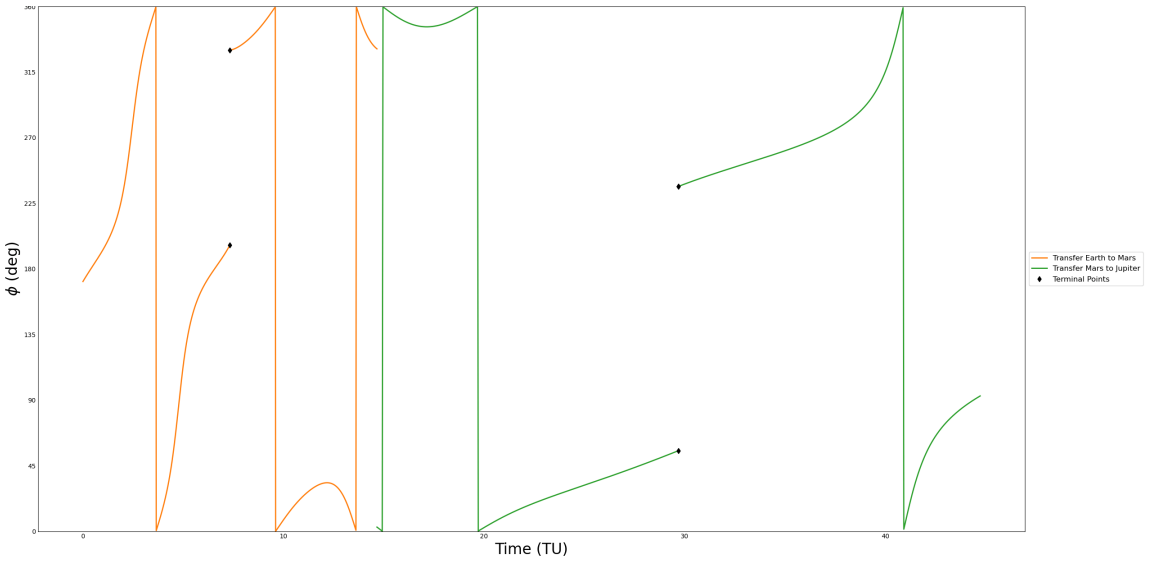

Figure B.104: Cluster Pruning Indirect Method Optimal Solution 5 Thrust

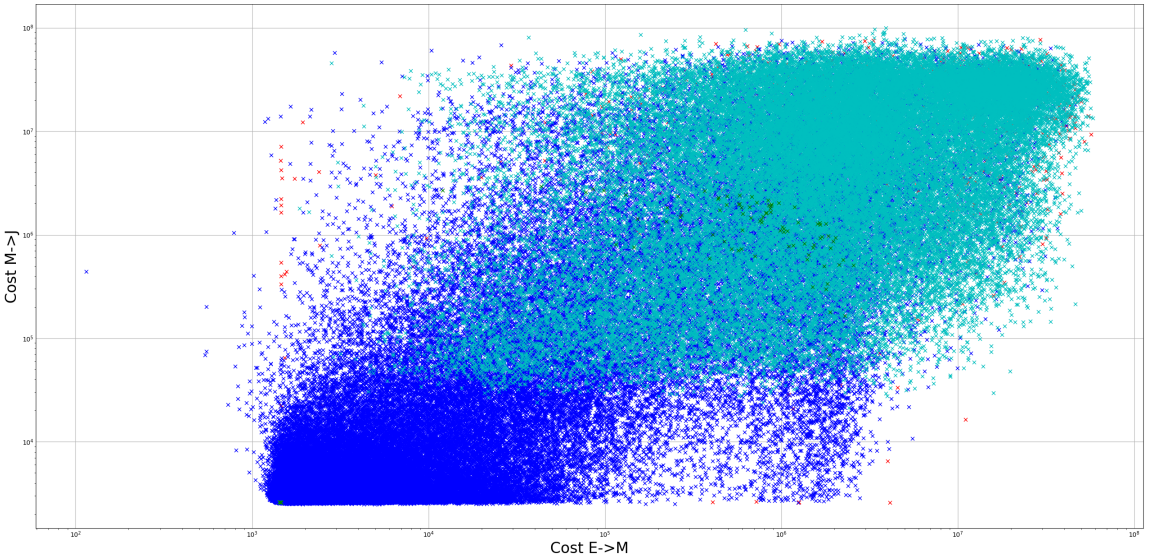

Figure B.105: Cluster Pruning Indirect Method Optimal Solution 5 Leg Tracking

B.4 Best Leg Tracking Verification

B.4.1 Direct Method Results

Table B.13: MGALT-STOpS Direct Method Best Leg Tracking Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3616.34	2252.09	5273.66	1982.96	2413.40
Departure Date	Aug 2, 2021	Aug 2, 2021	Aug 9, 2021	Aug 2, 2021	Aug 1, 2021
Flyby ToF (days)	810	823	810	842	816
Arrival ToF (days)	1396	1396	1396	1385	1396
Total ToF (years)	6.04	6.08	6.04	6.10	6.05
Propellant Mass Fraction	36.58%	36.82%	36.60%	36.94%	36.69%

Table B.14: MGALT-STOpS Direct Method Best Leg Tracking Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3616.34	2252.09	5273.66	1982.96	2413.40
Earth to Mars Cost	3630.04	2194.17	4948.08	1958.70	2367.26
Mars to Jupiter Cost	10.30	57.92	325.58	24.25	46.14

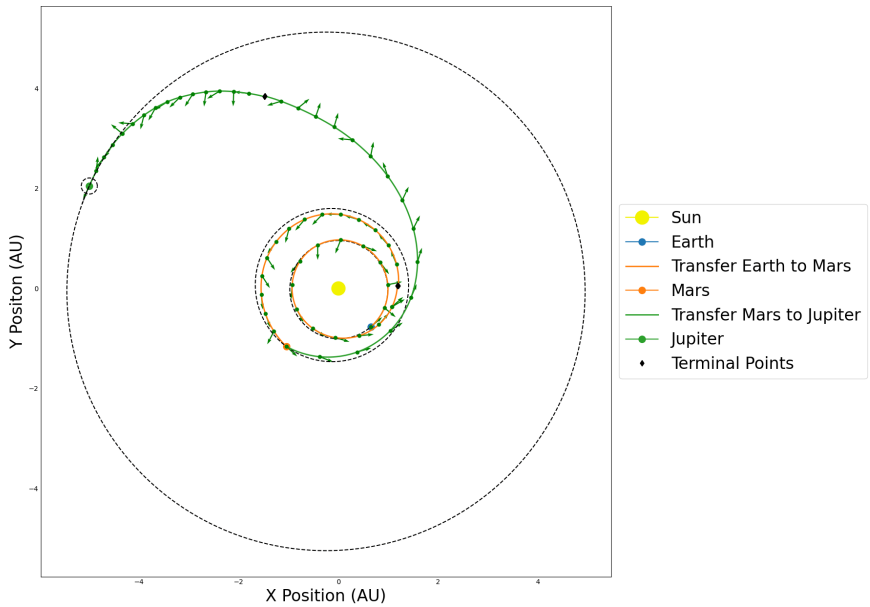

Figure B.106: Best Leg Tracking Direct Method Optimal Solution 1 Trajectory

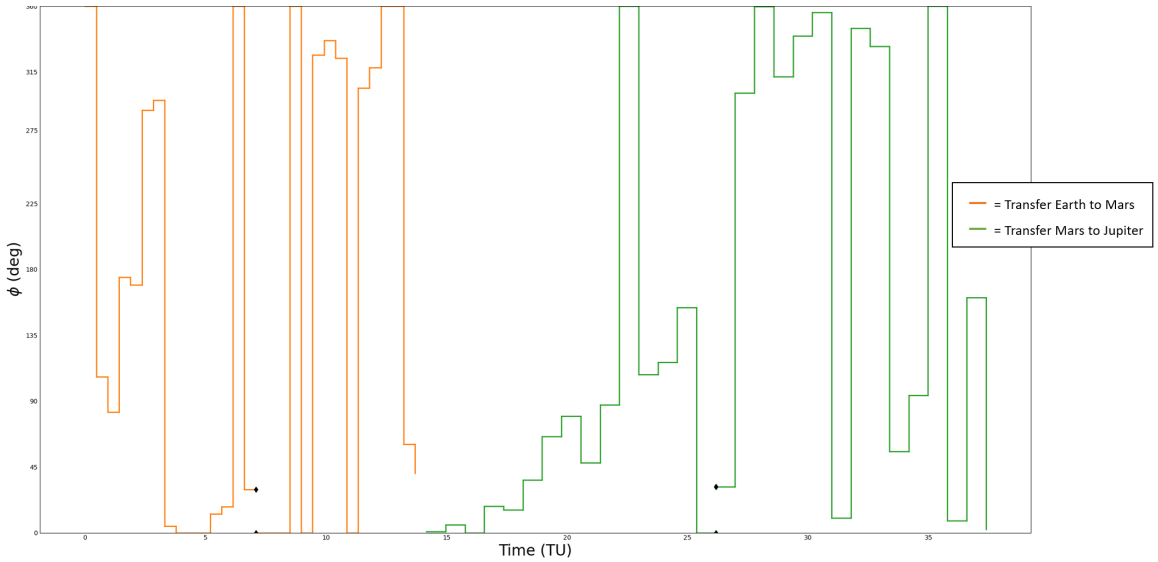

Figure B.107: Best Leg Tracking Direct Method Optimal Solution 1 Thrust

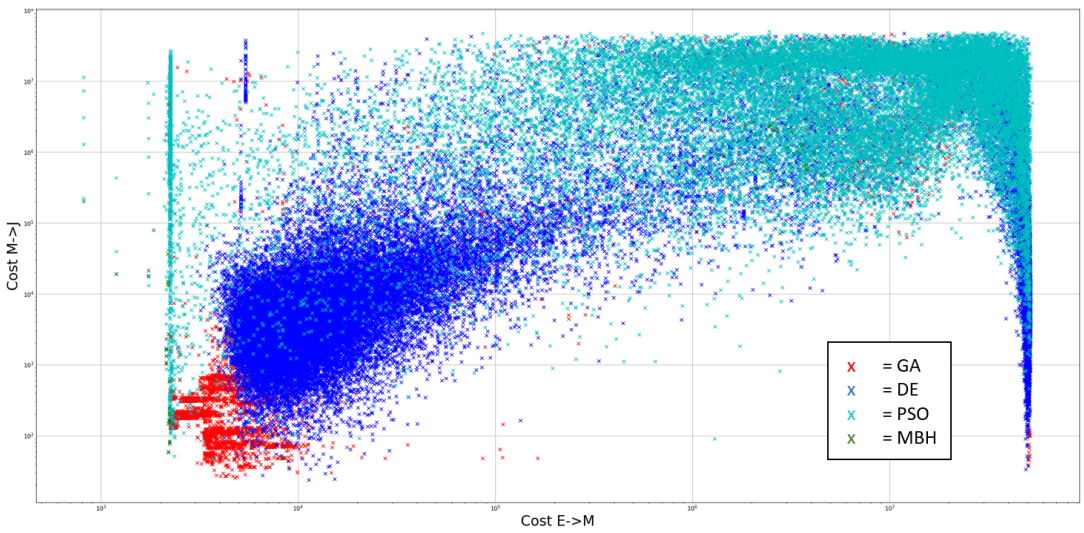

Figure B.108: Best Leg Tracking Direct Method Optimal Solution 1 Leg Tracking

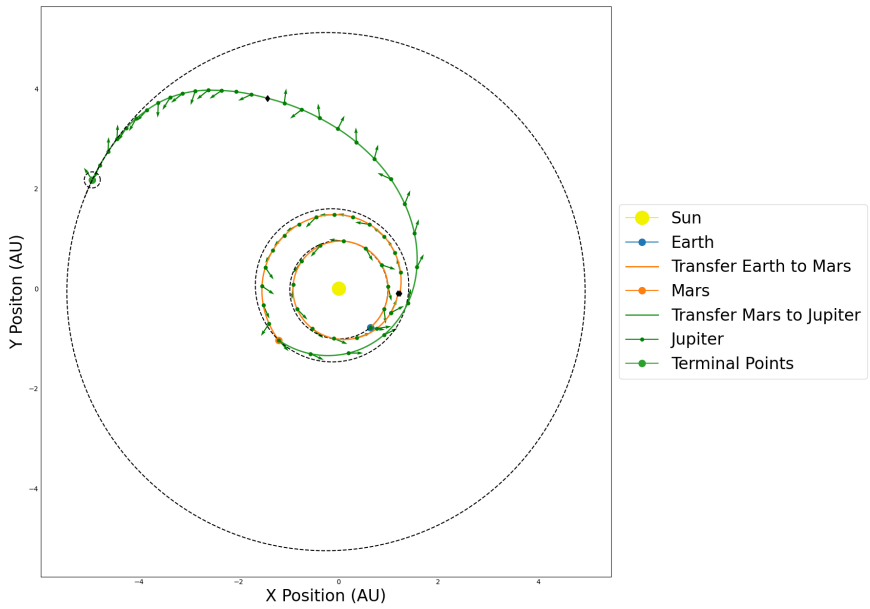

Figure B.109: Best Leg Tracking Direct Method Optimal Solution 2 Trajectory

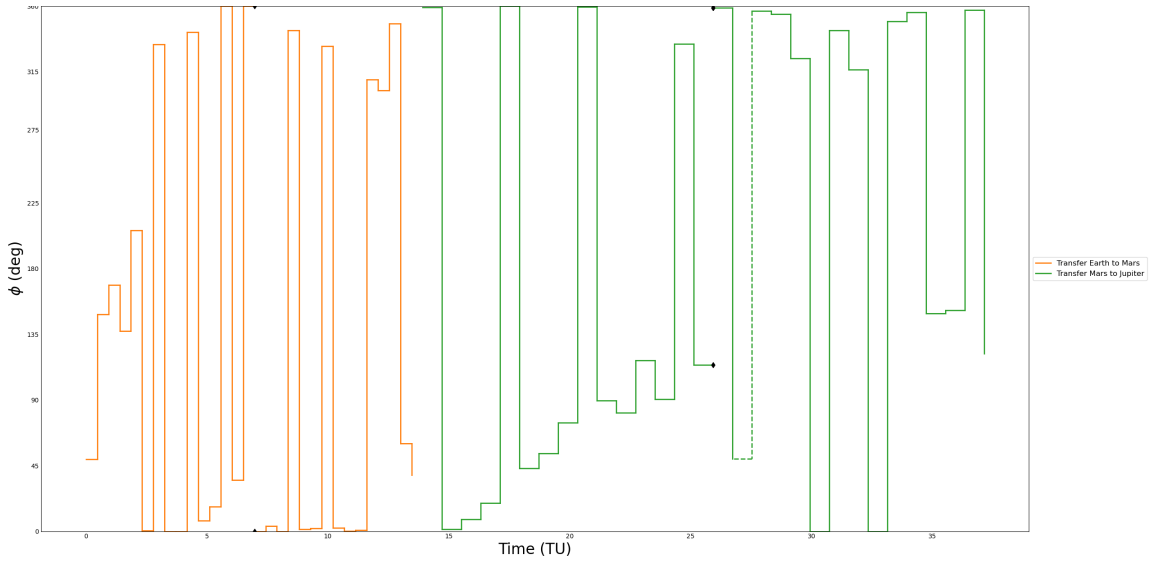

Figure B.110: Best Leg Tracking Direct Method Optimal Solution 2 Thrust

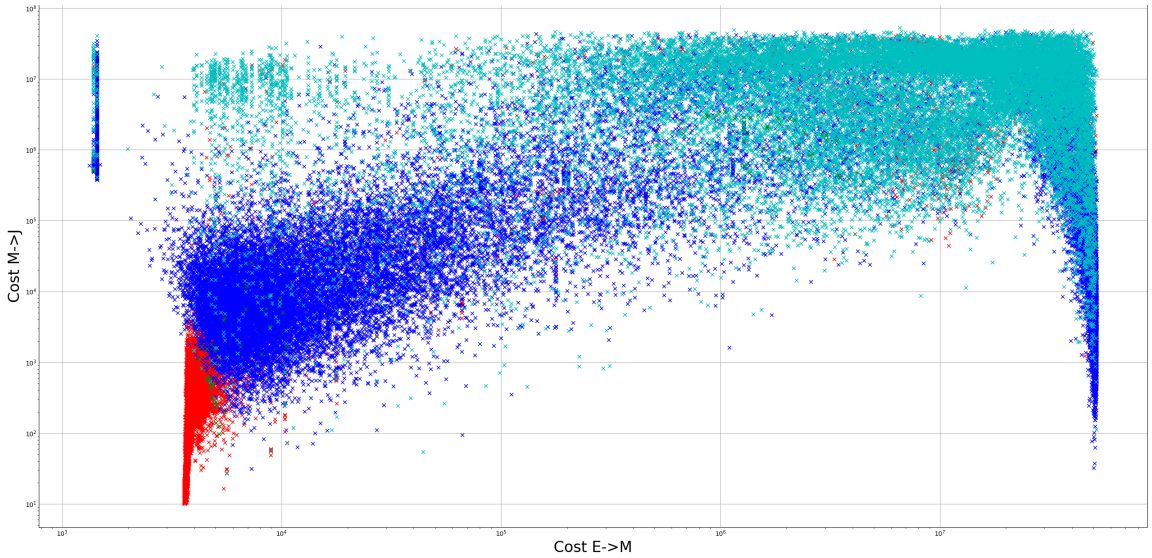

Figure B.111: Best Leg Tracking Direct Method Optimal Solution 2 Leg Tracking

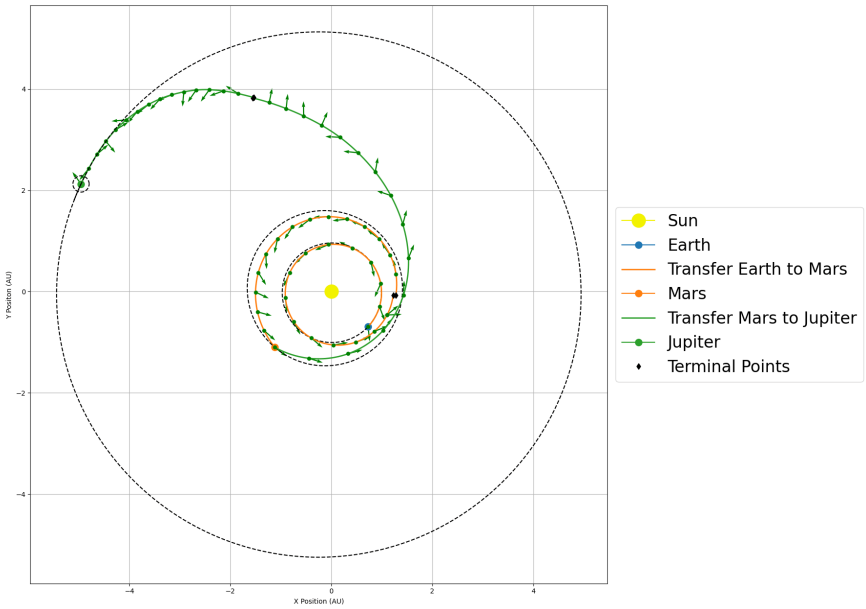

Figure B.112: Best Leg Tracking Direct Method Optimal Solution 3 Trajectory

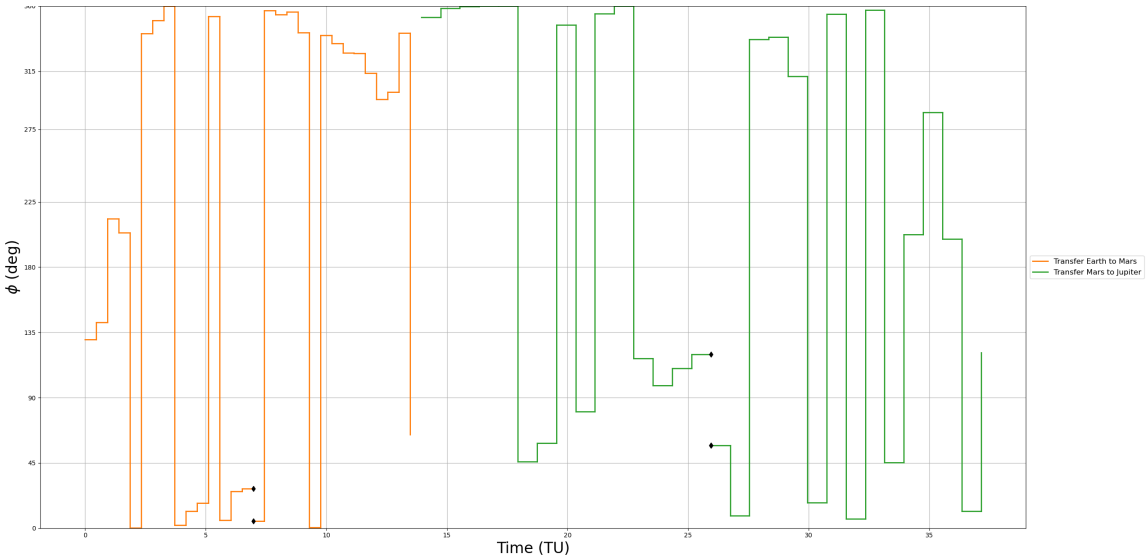

Figure B.113: Best Leg Tracking Direct Method Optimal Solution 3 Thrust

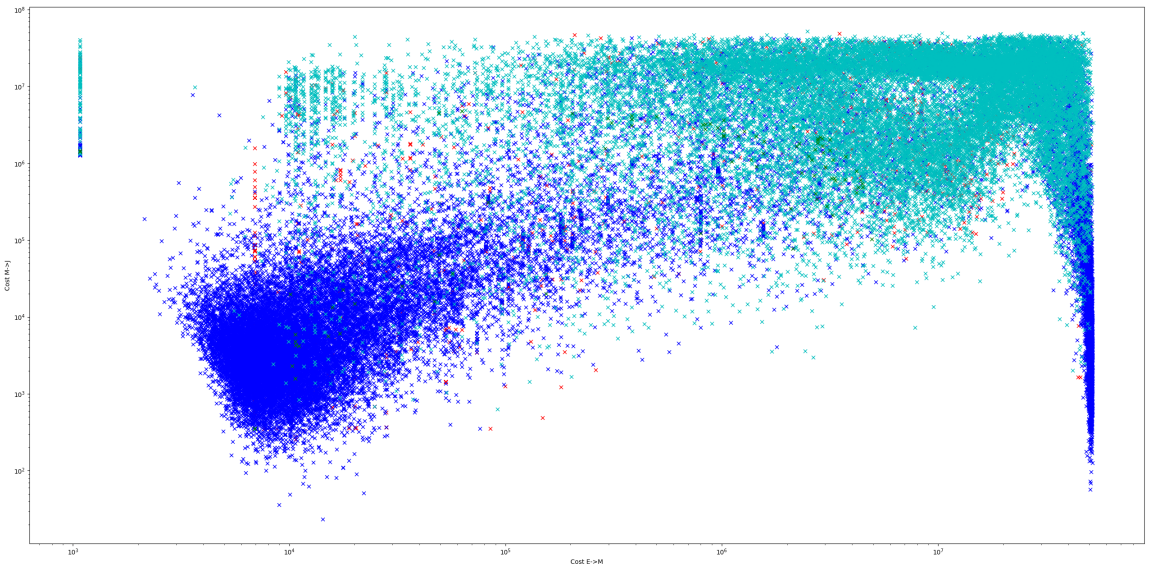

Figure B.114: Best Leg Tracking Direct Method Optimal Solution 3 Leg Tracking

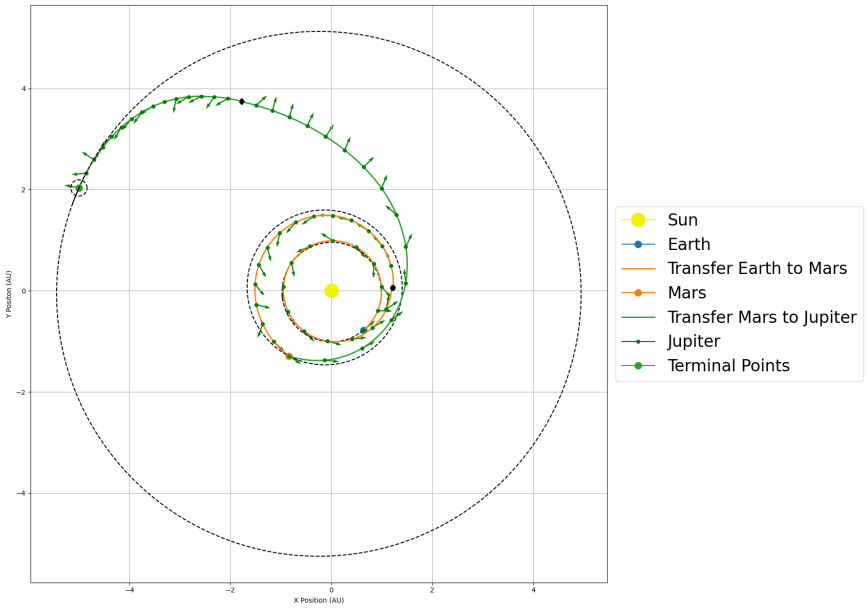

Figure B.115: Best Leg Tracking Direct Method Optimal Solution 4 Trajectory

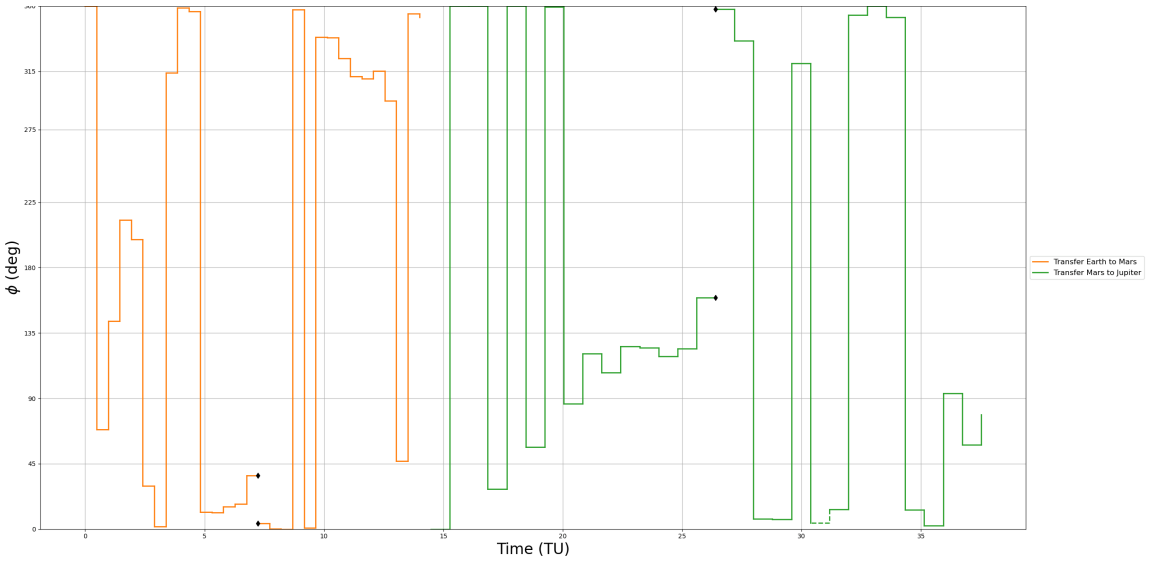

Figure B.116: Best Leg Tracking Direct Method Optimal Solution 4 Thrust

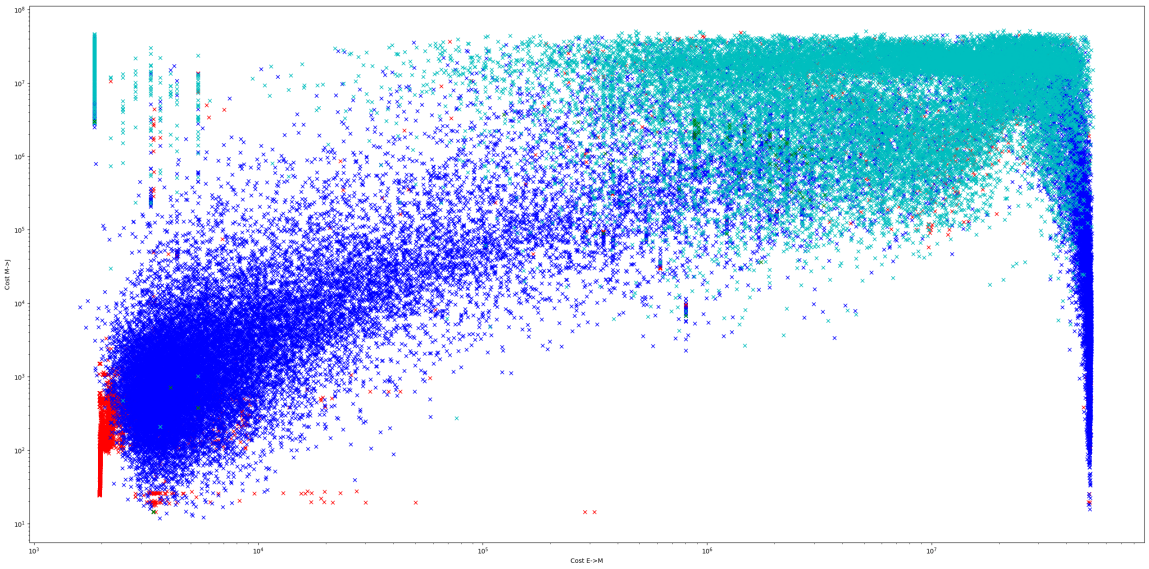

Figure B.117: Best Leg Tracking Direct Method Optimal Solution 4 Leg Tracking

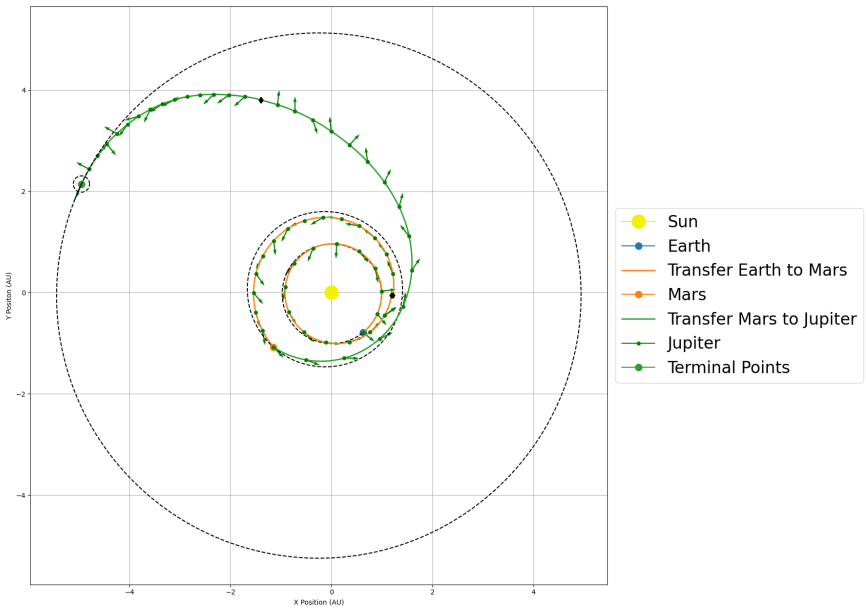

Figure B.118: Best Leg Tracking Direct Method Optimal Solution 5 Trajectory

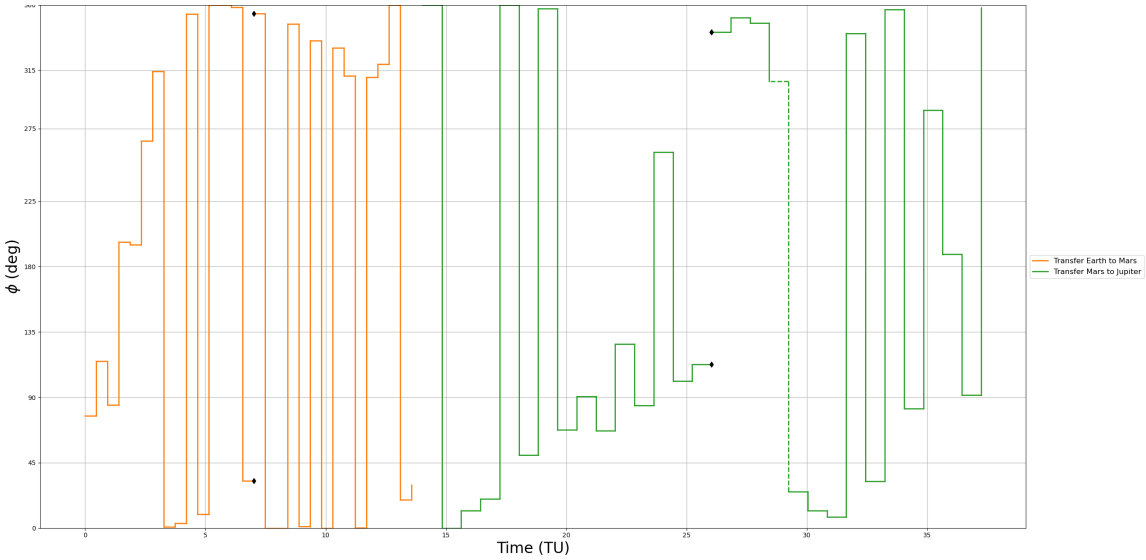

Figure B.119: Best Leg Tracking Direct Method Optimal Solution 5 Thrust

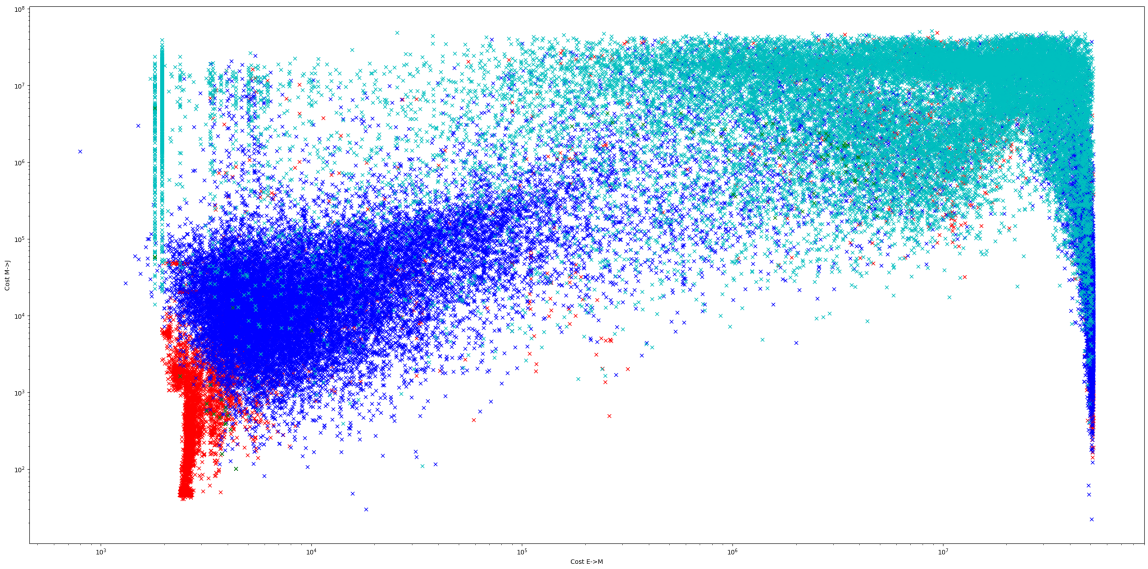

Figure B.120: Best Leg Tracking Direct Method Optimal Solution 5 Leg Tracking

B.4.1.1 Best Leg Tracking with Thruster Angle Pruning

Table B.15: MGALT-STOpS Direct Method Best Leg Tracking with Thruster Angle Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3474.11	4539.47	3519.04	3313.20	6019.37
Departure Date	Aug 24, 2021	Aug 15, 2021	Aug 18, 2021	Aug 18, 2021	Aug 13, 2021
Flyby ToF (days)	808	848	825	813	825
Arrival ToF (days)	1395	1395	1396	1396	1395
Total ToF (years)	6.03	6.14	6.08	6.05	6.08
Propellant Mass Fraction	36.55%	37.33%	36.85%	36.64%	36.83%

Table B.16: MGALT-STOpS Direct Method Best Leg Tracking with Thruster Angle Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3474.11	4539.47	3519.04	3313.20	6019.37
Earth to Mars Cost	2763.99	3747.77	3094.94	3098.87	3898.09
Mars to Jupiter Cost	710.12	791.70	424.09	214.31	2121.29

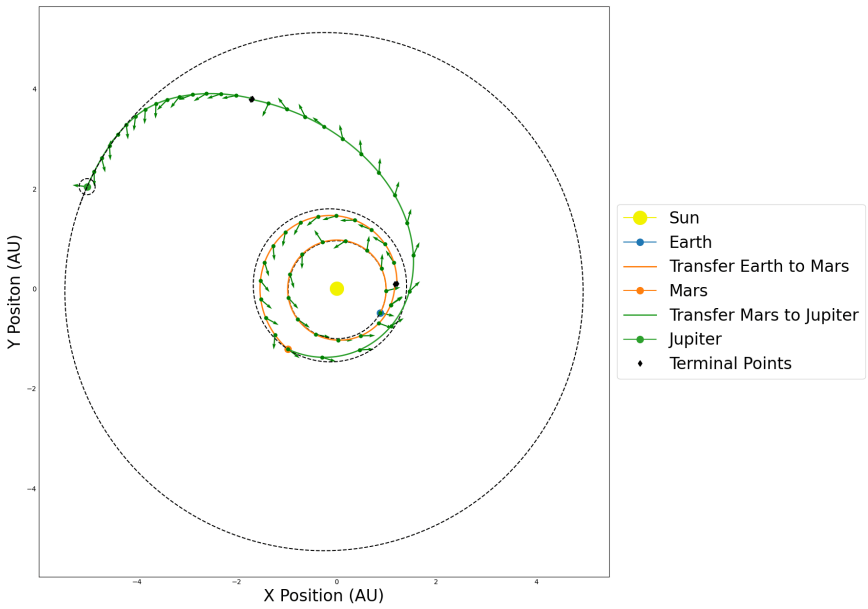

Figure B.121: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory

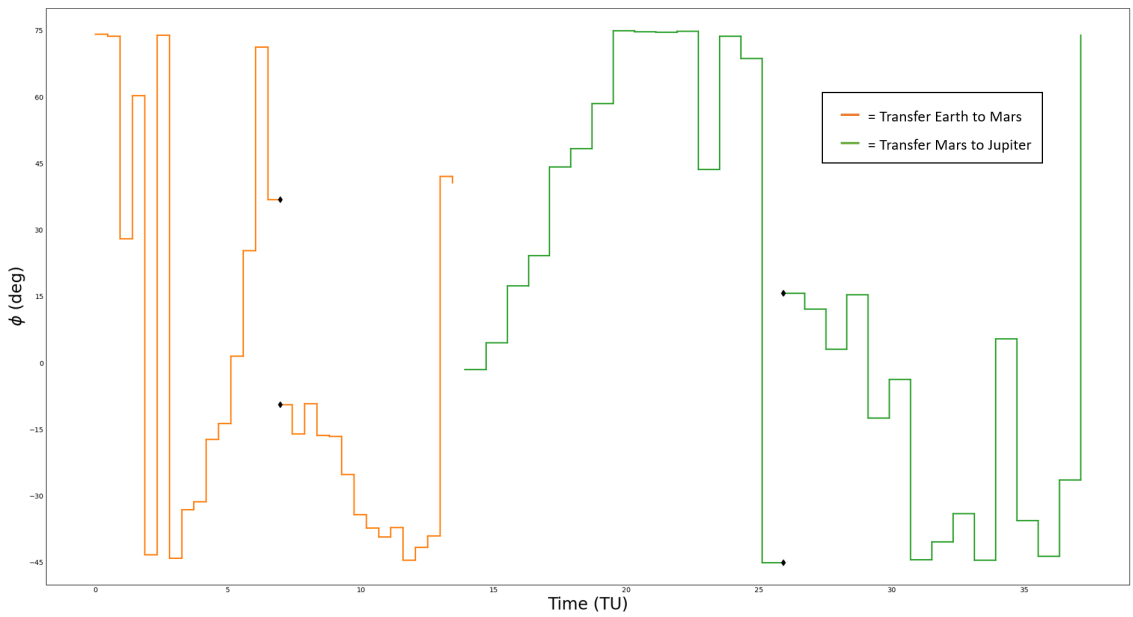

Figure B.122: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust

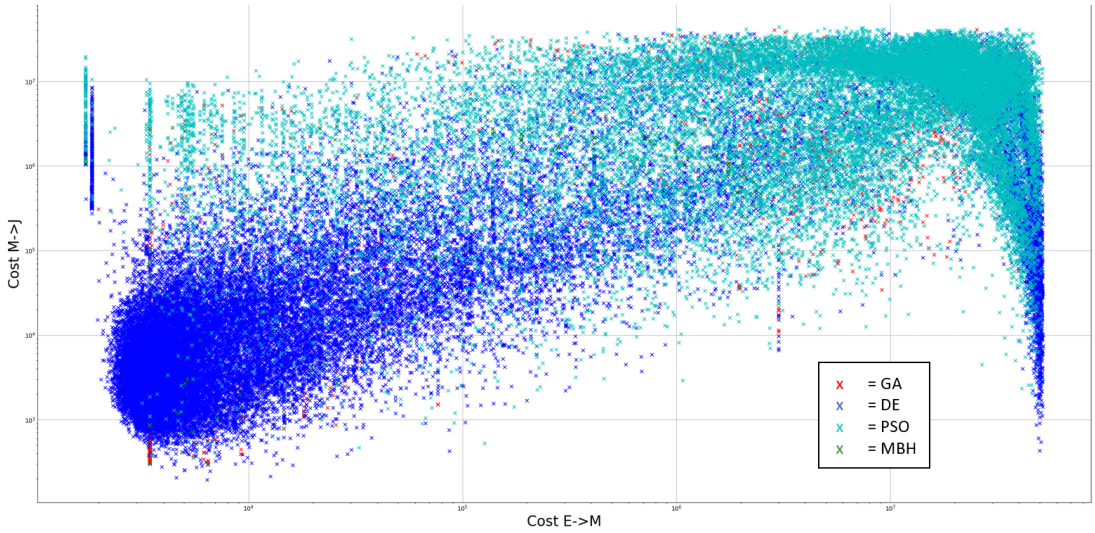

Figure B.123: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking

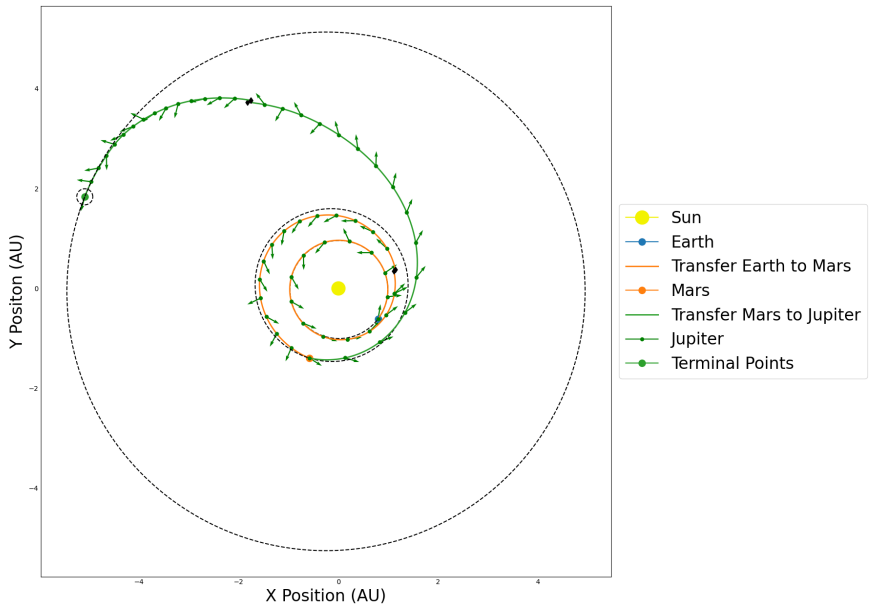

Figure B.124: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory

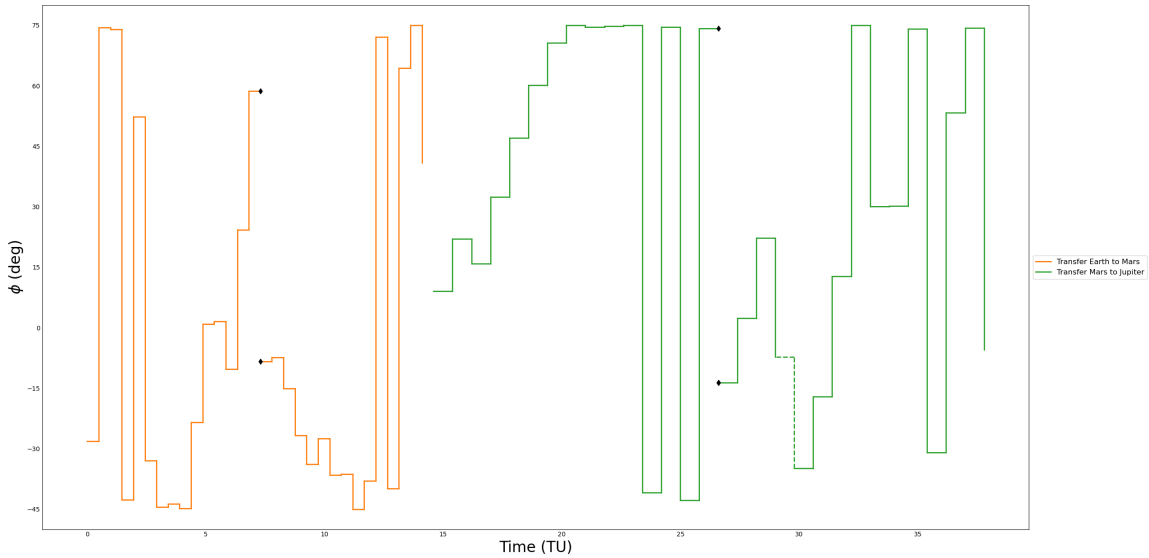

Figure B.125: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust

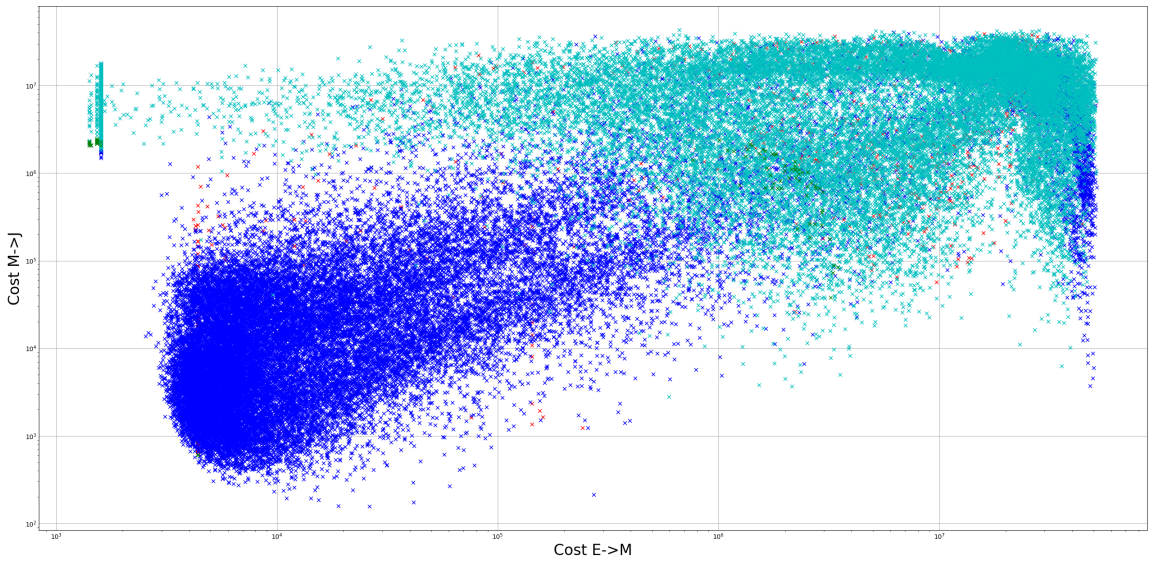

Figure B.126: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking

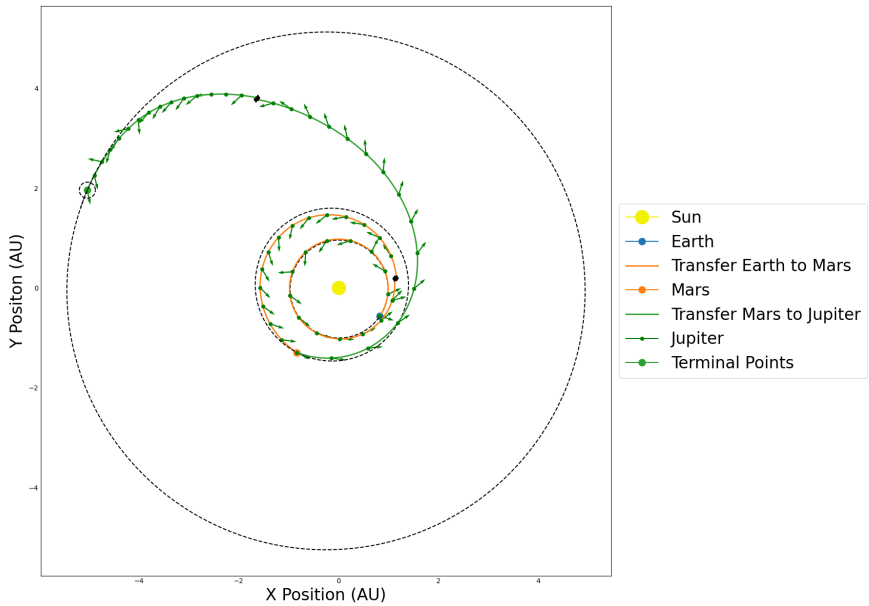

Figure B.127: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory

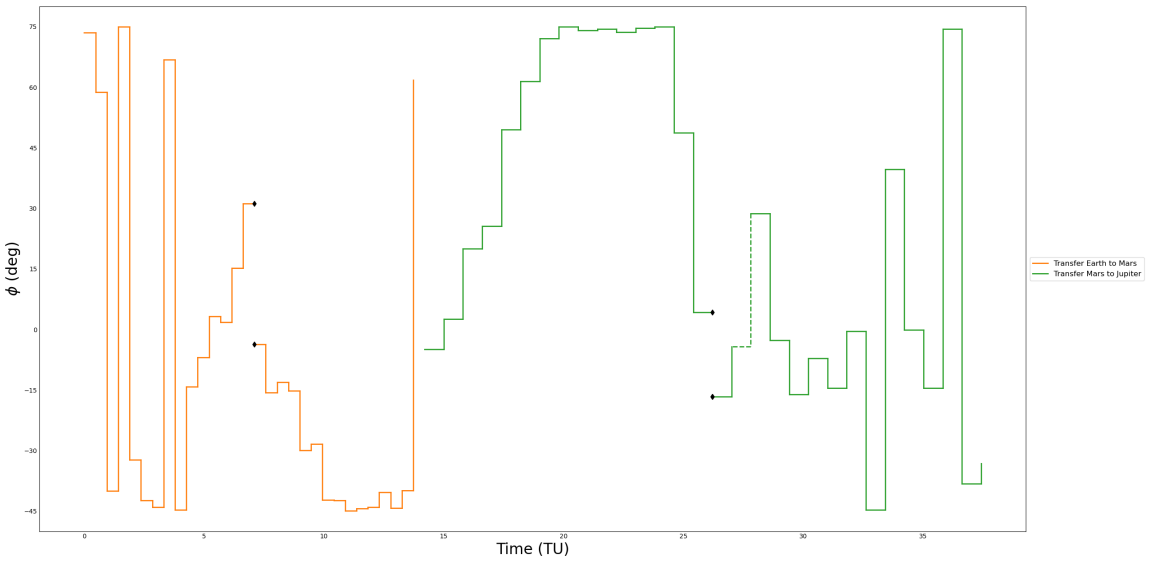

Figure B.128: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust

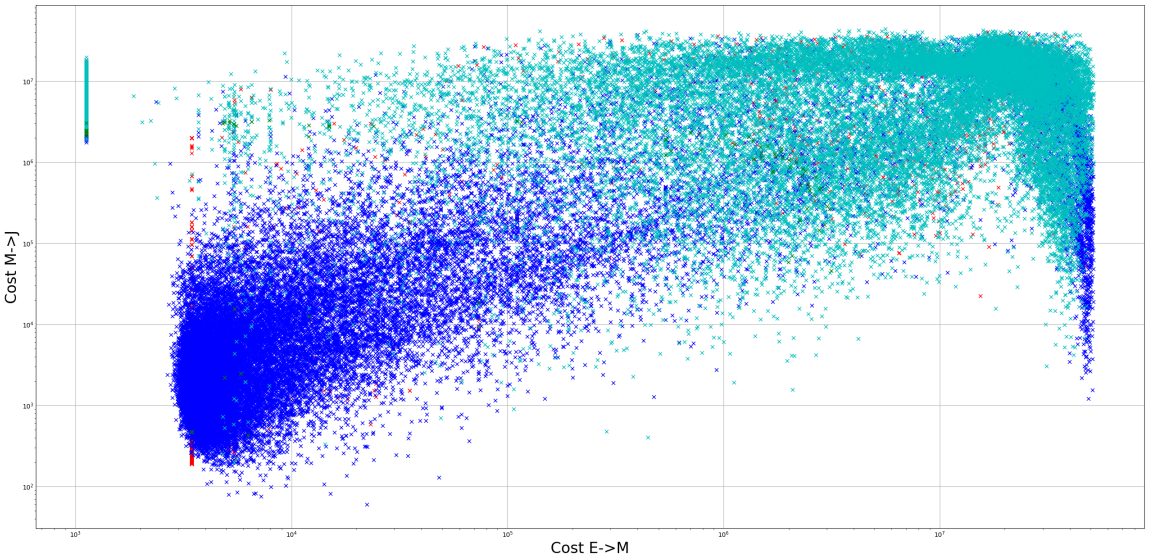

Figure B.129: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking

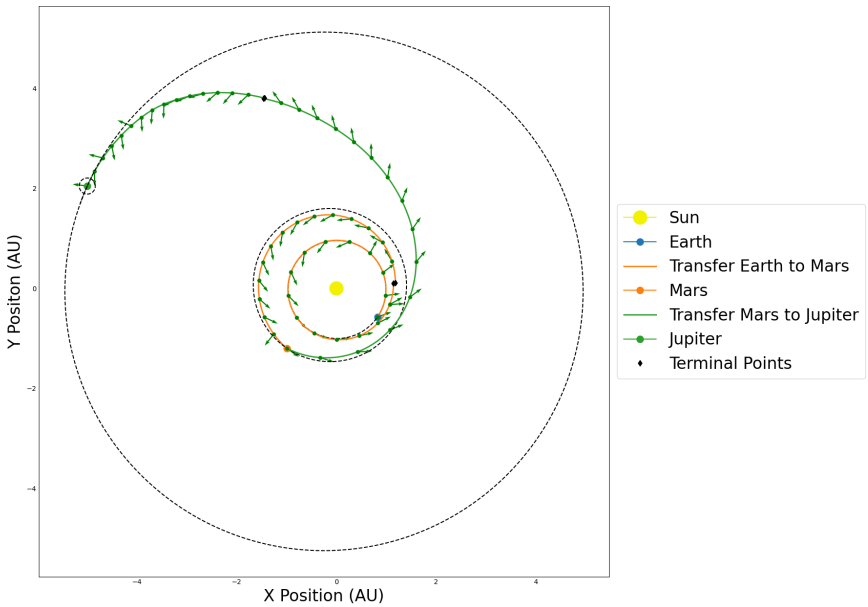

Figure B.130: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory

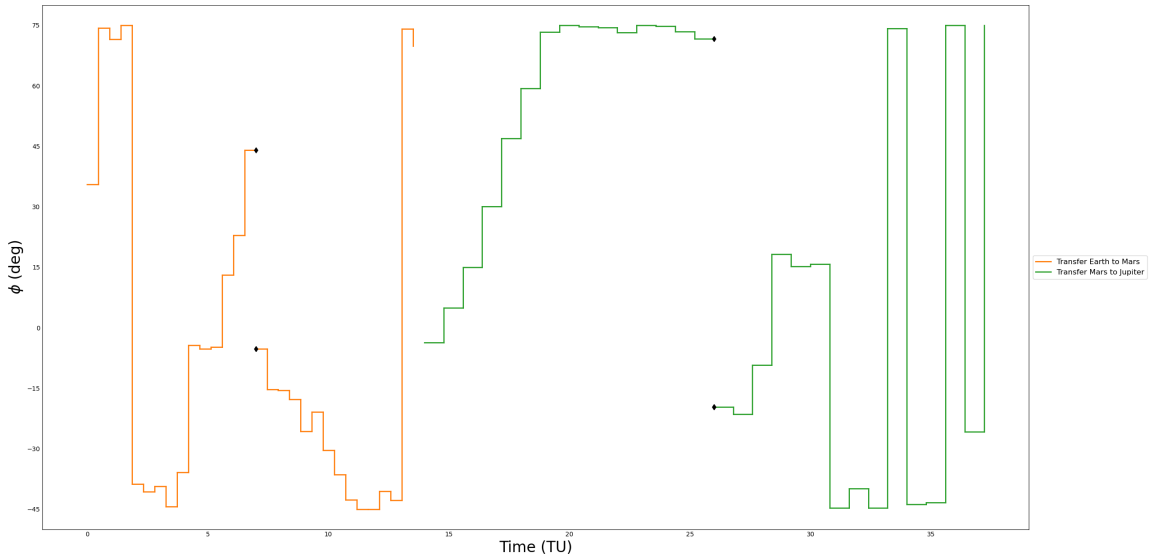

Figure B.131: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust

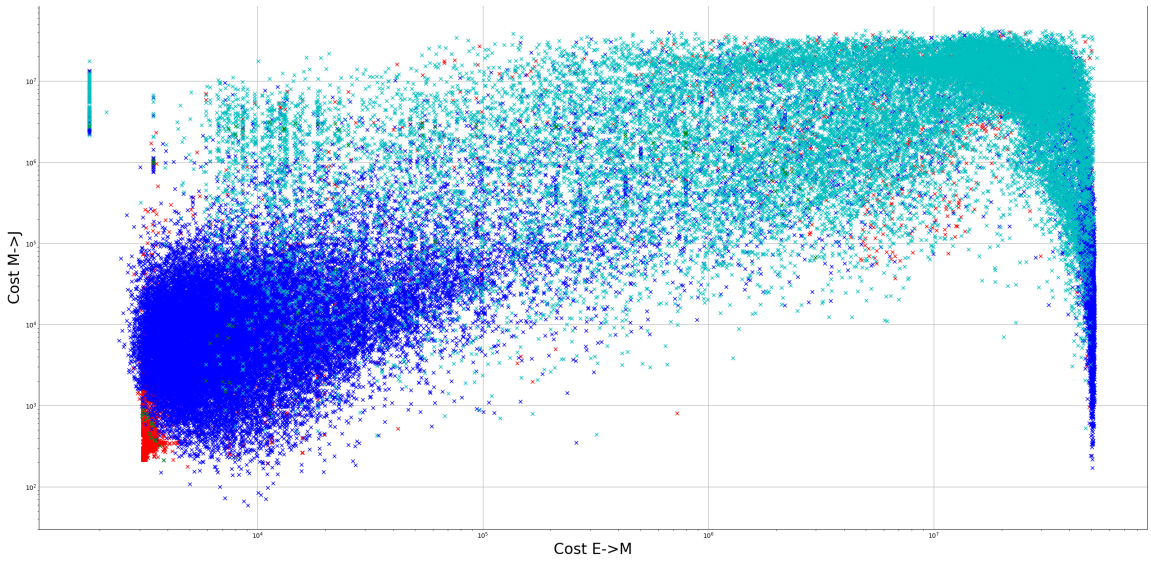

Figure B.132: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking

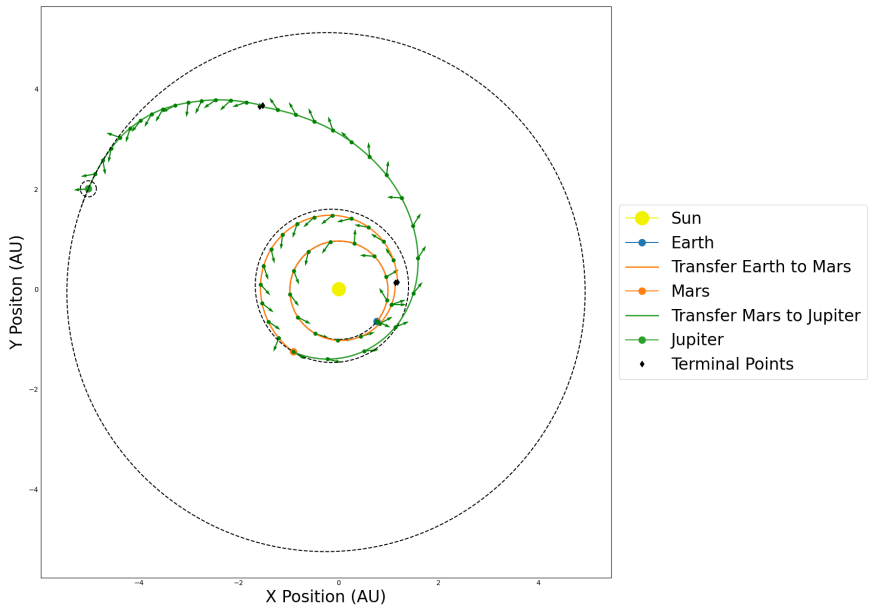

Figure B.133: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory

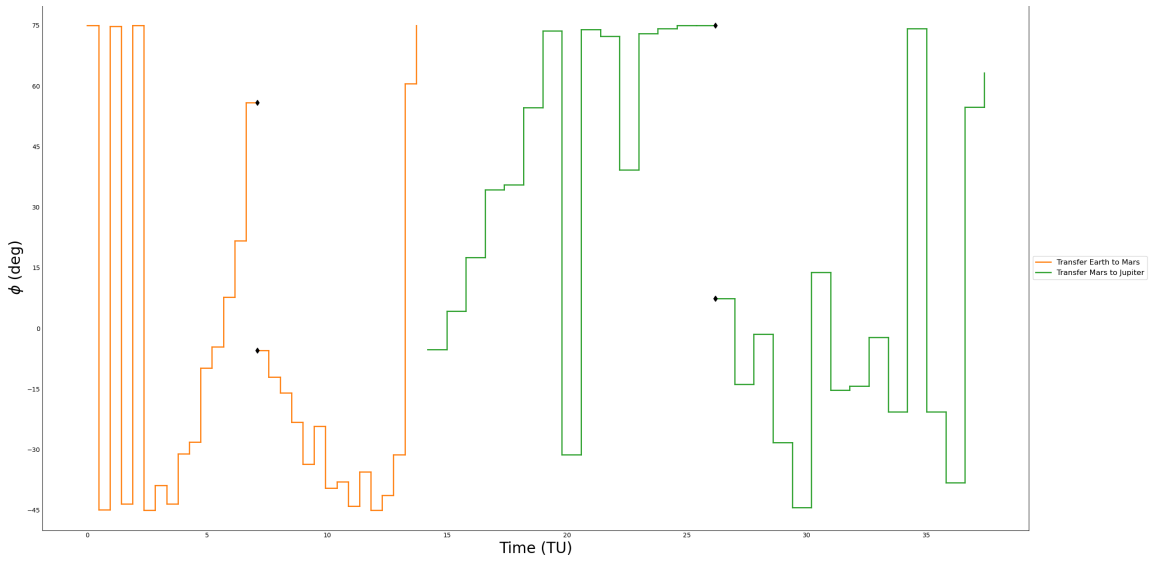

Figure B.134: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust

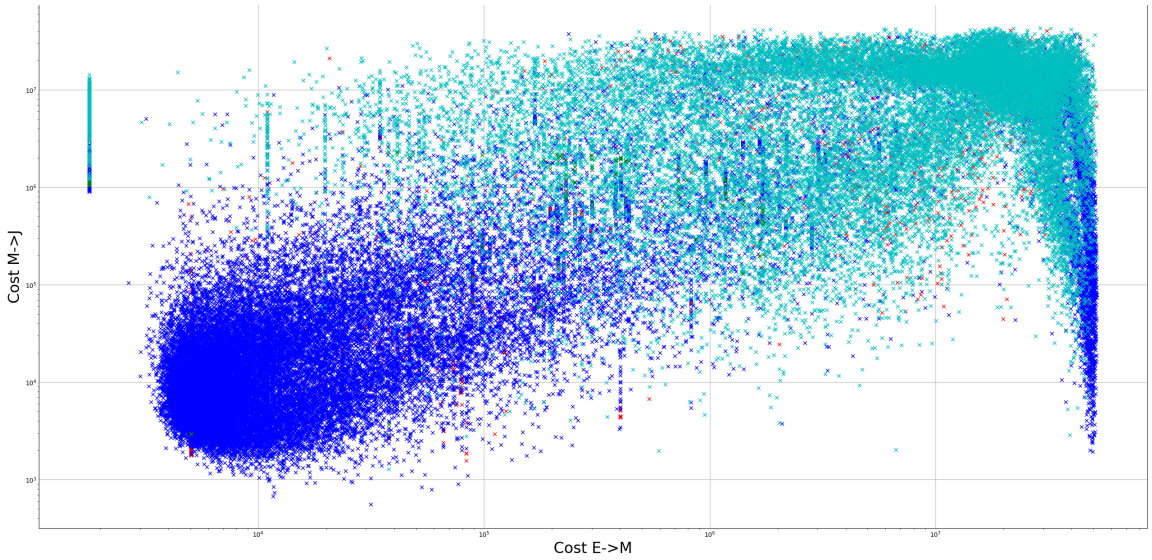

Figure B.135: Best Leg Tracking with Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking

B.4.1.2 Best Leg Tracking with Cluster Pruning

Table B.17: MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3369.39	4733.64	3150.06	2838.76	3064.07
Departure Date	Aug 1, 2021	Aug 1, 2021	Aug 5, 2021	Aug 13, 2021	Aug 22, 2021
Flyby ToF (days)	807	827.45	814	821	816
Arrival ToF (days)	1393	1395.98	1396	1396	1396
Total ToF (years)	6.02	6.09	6.05	6.07	6.06
Propellant Mass Fraction	36.50%	36.88%	36.66%	36.77%	36.70%

Table B.18: MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	3369.39	4733.64	3150.06	2838.76	3064.07
Earth to Mars Cost	3061.72	4604.84	3061.45	2817.65	2995.23
Mars to Jupiter Cost	307.66	128.79	88.61	21.10	68.83

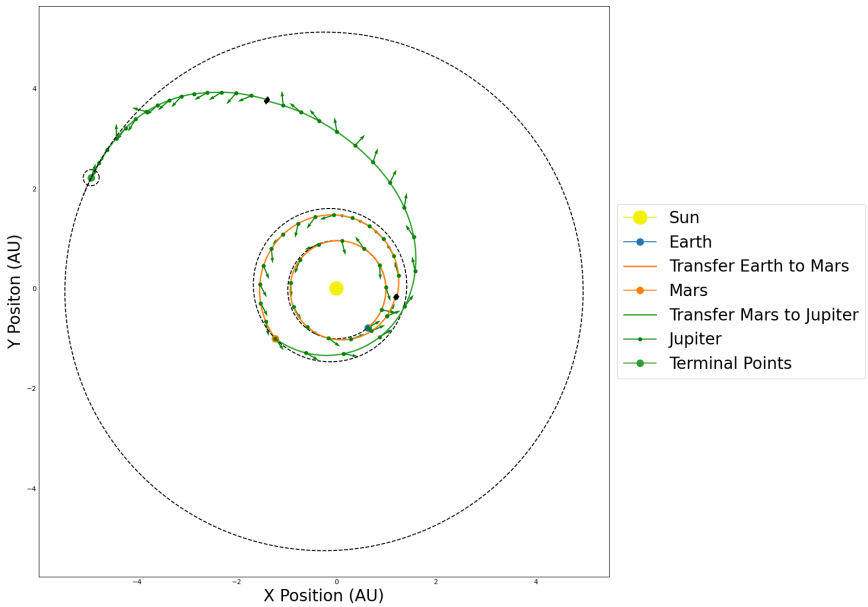

Figure B.136: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Trajectory

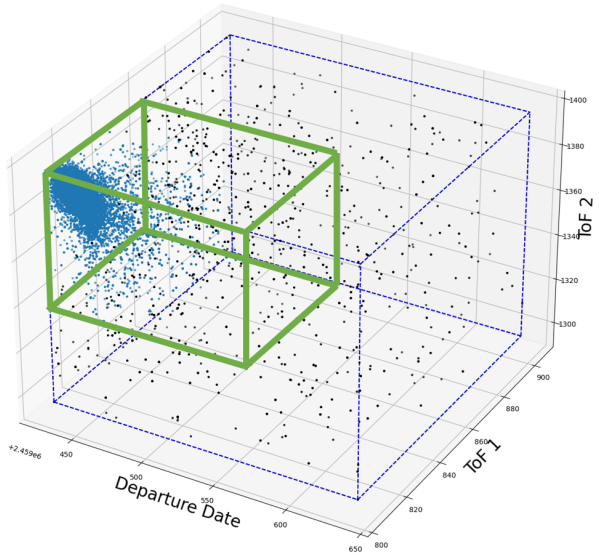

Figure B.137: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Clustering

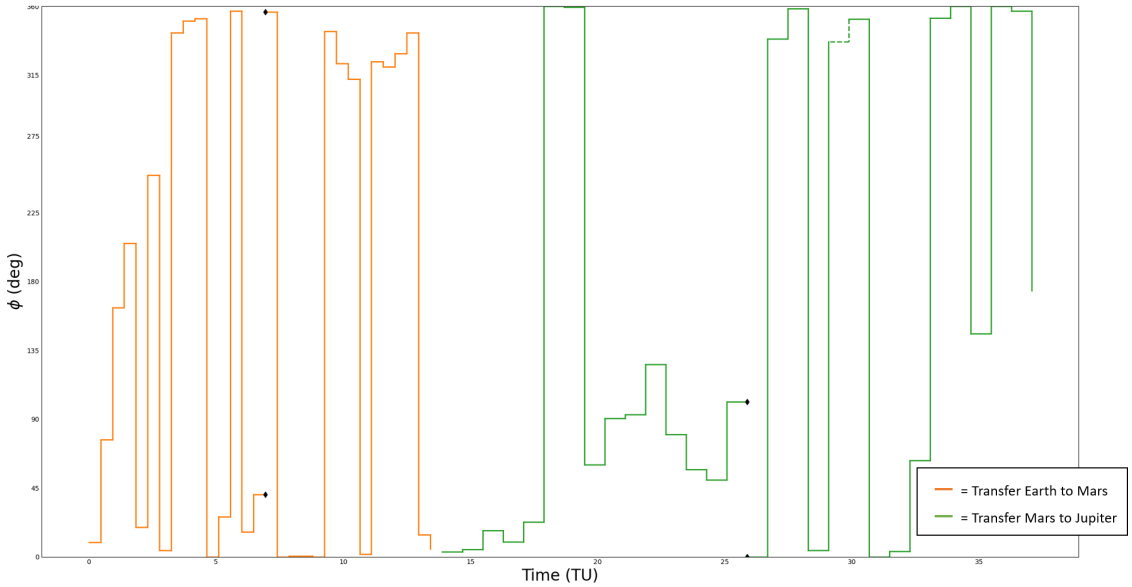

Figure B.138: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Thrust

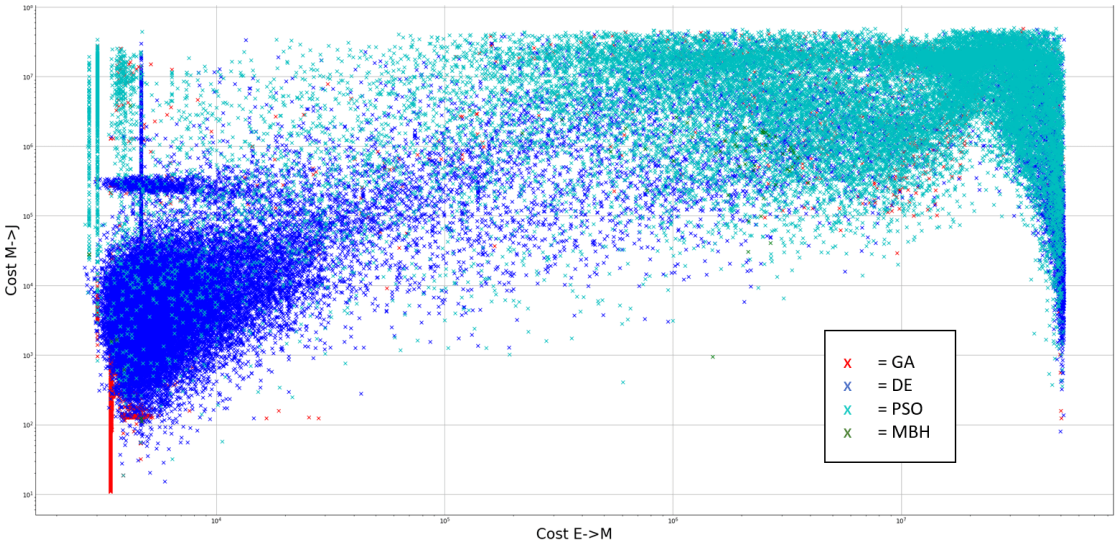

Figure B.139: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 1 Leg Tracking

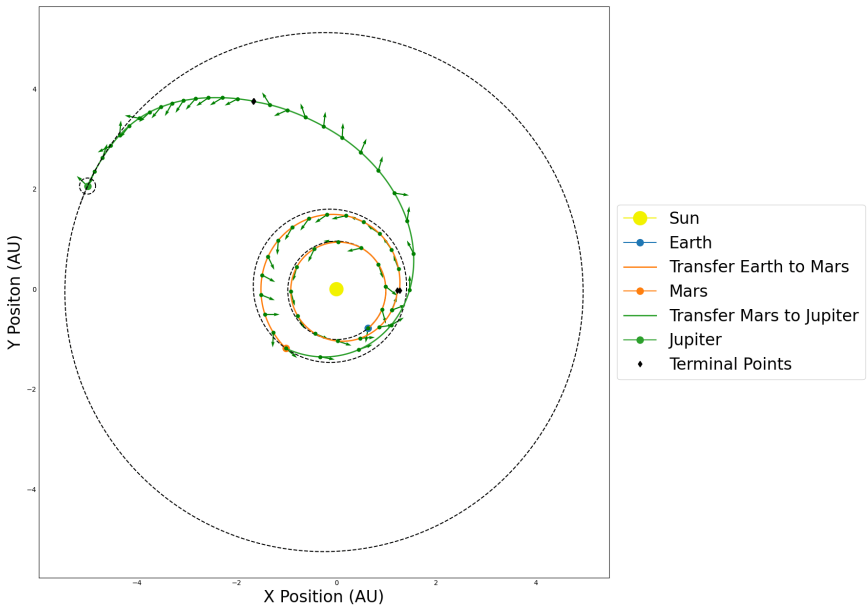

Figure B.140: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Trajectory

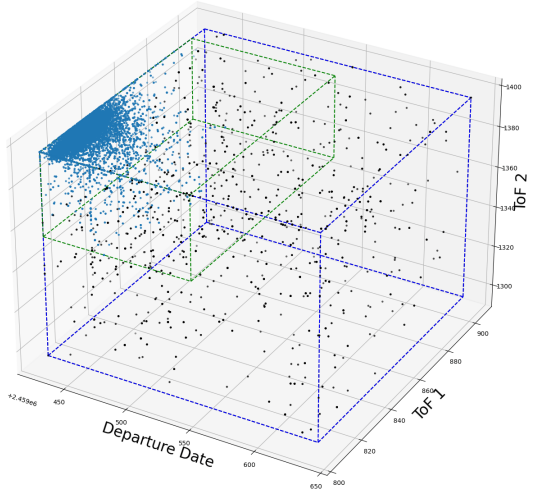

Figure B.141: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Clustering

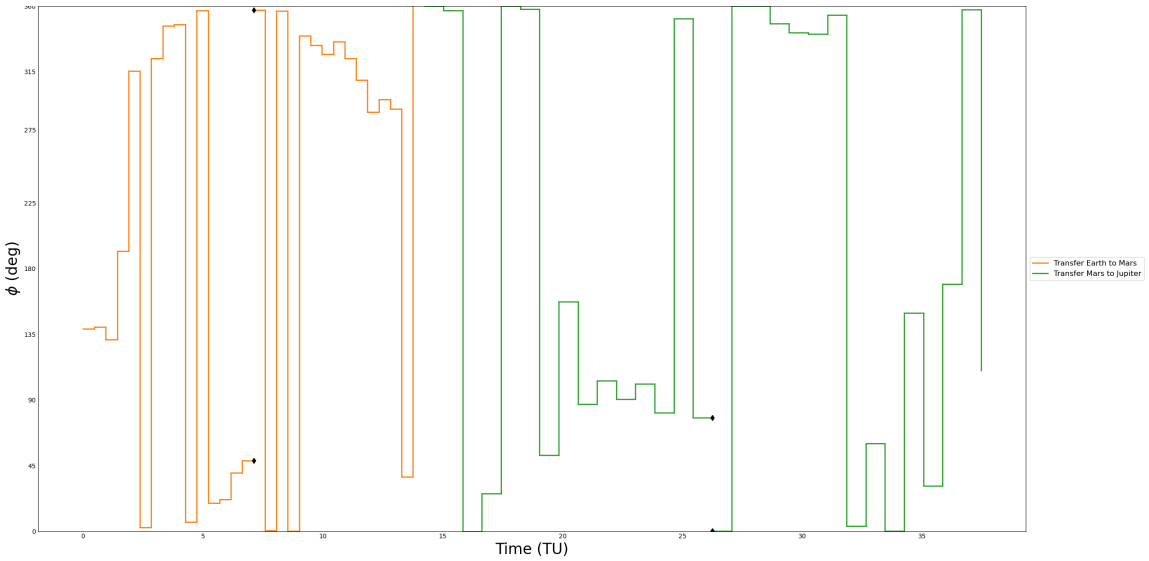

Figure B.142: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Thrust

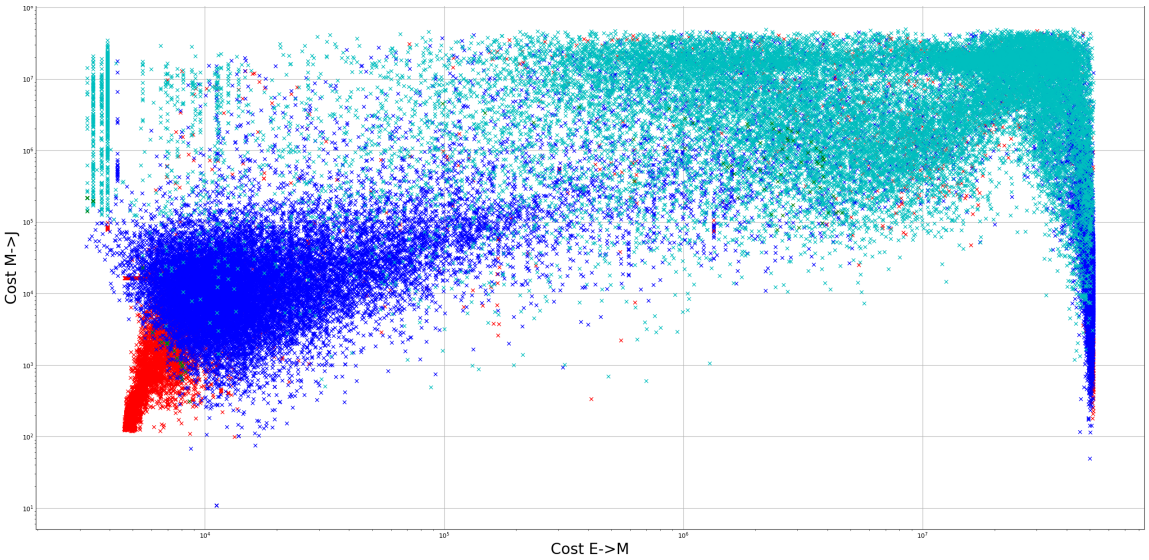

Figure B.143: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 2 Leg Tracking

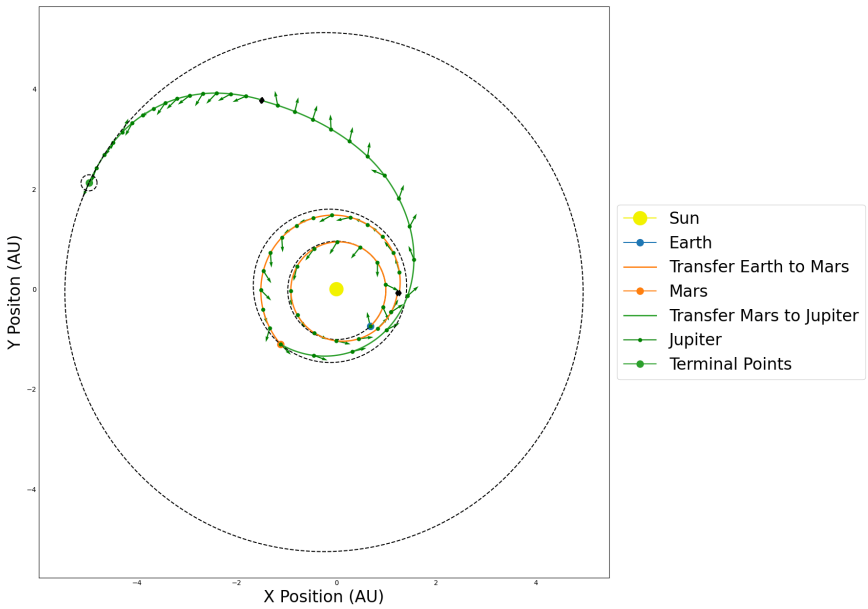

Figure B.144: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Trajectory

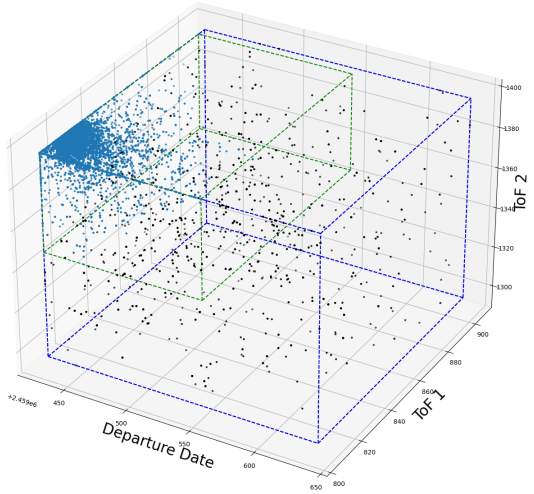

Figure B.145: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Clustering

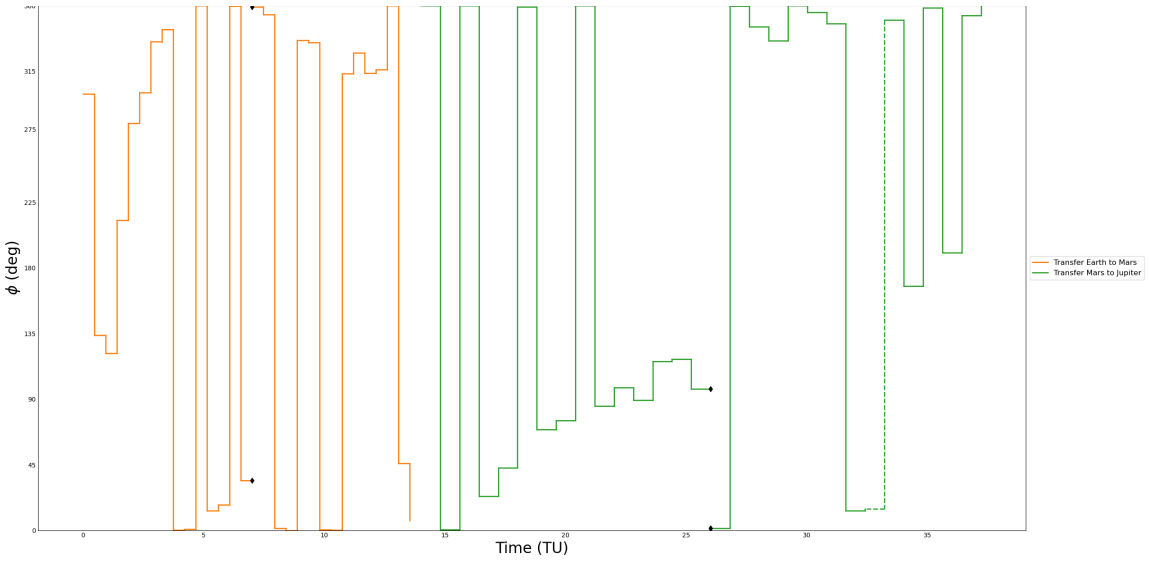

Figure B.146: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Thrust

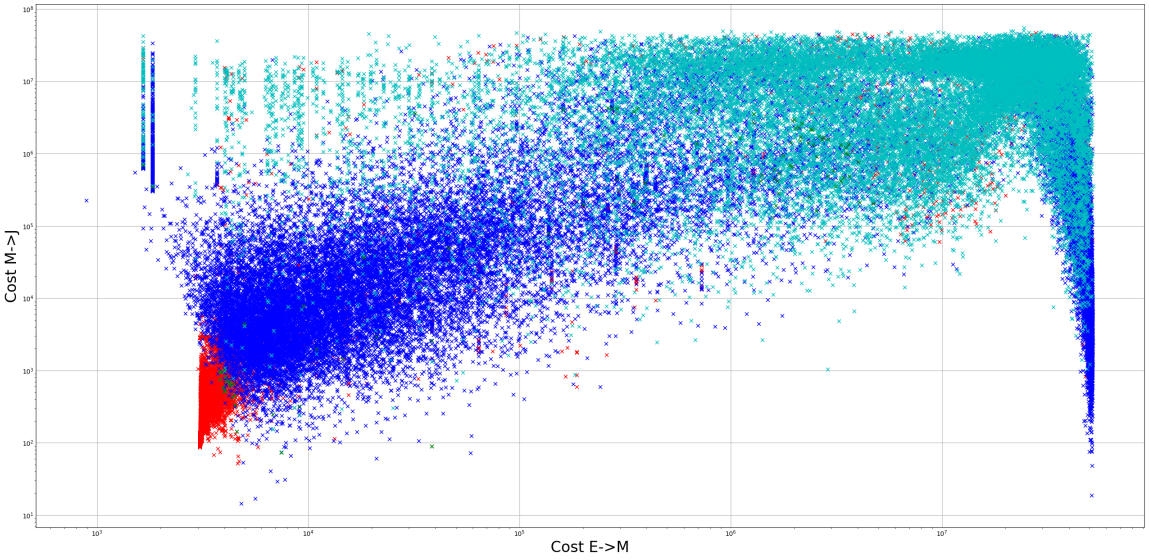

Figure B.147: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 3 Leg Tracking

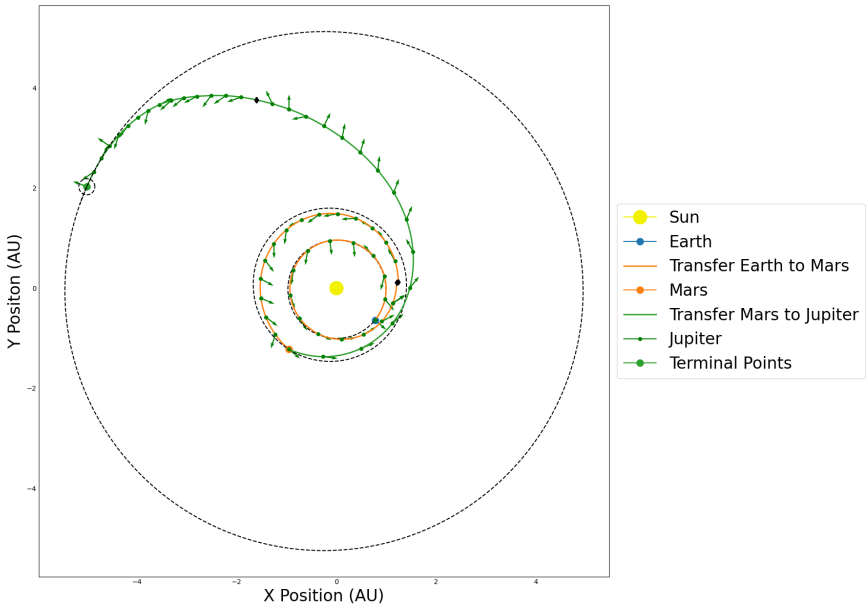

Figure B.148: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Trajectory

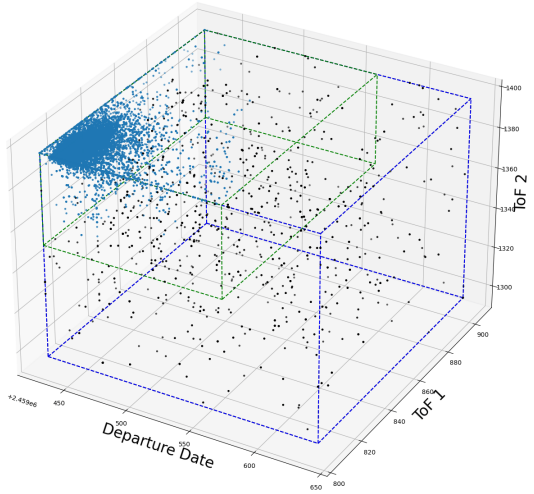

Figure B.149: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Clustering

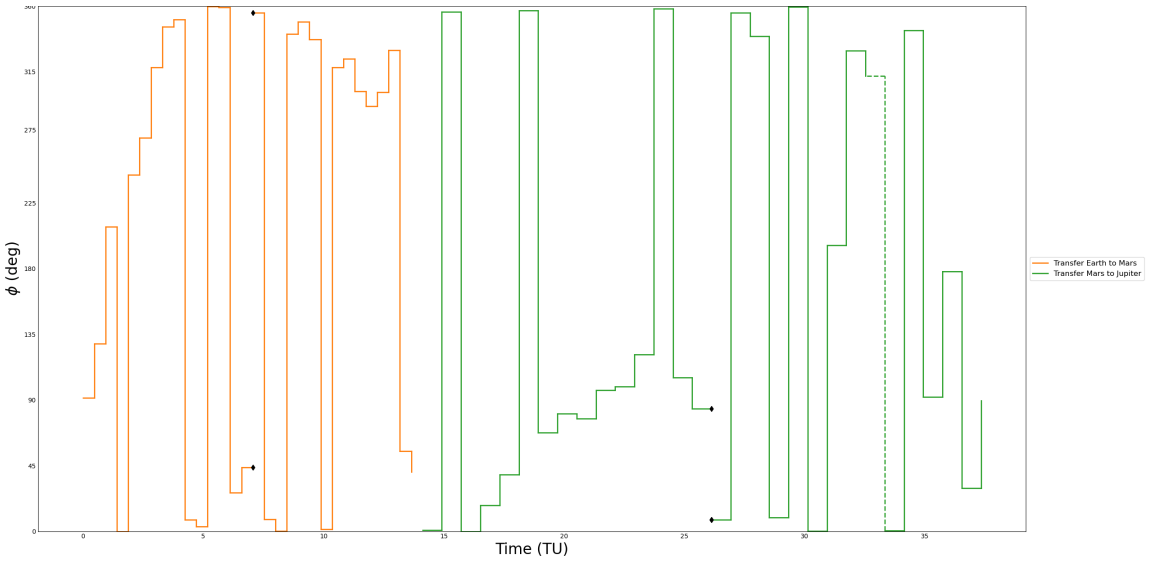

Figure B.150: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Thrust

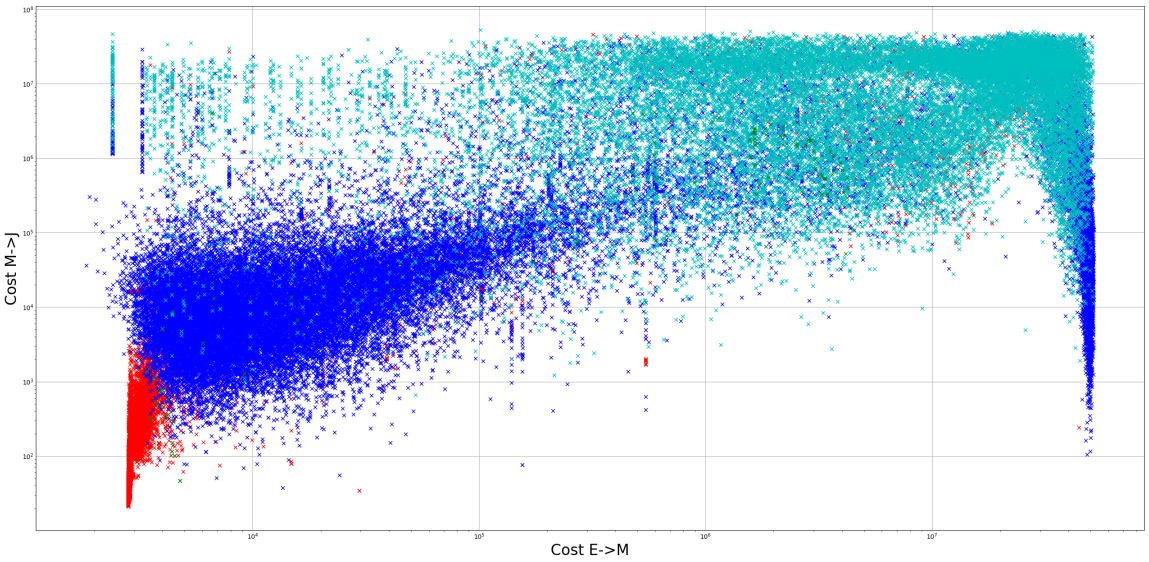

Figure B.151: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 4 Leg Tracking

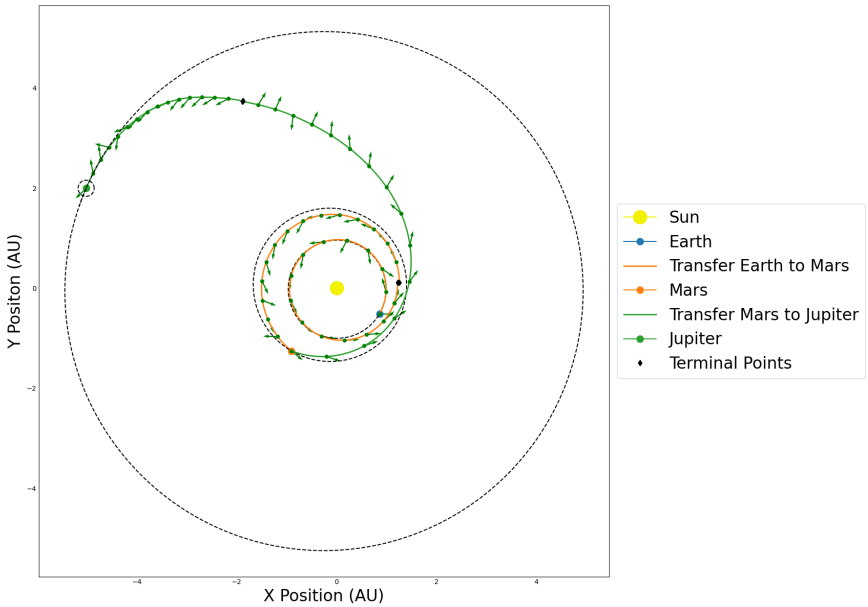

Figure B.152: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Trajectory

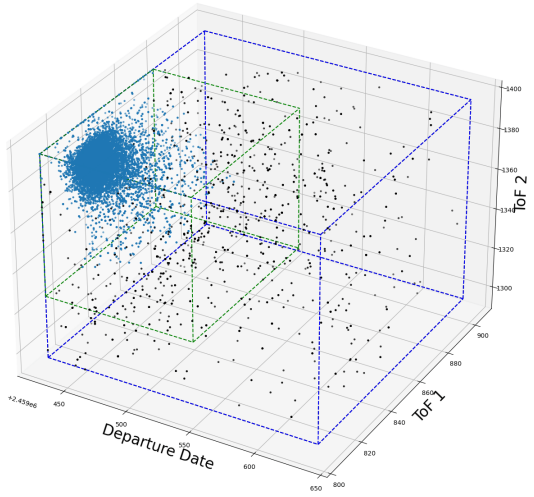

Figure B.153: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Clustering

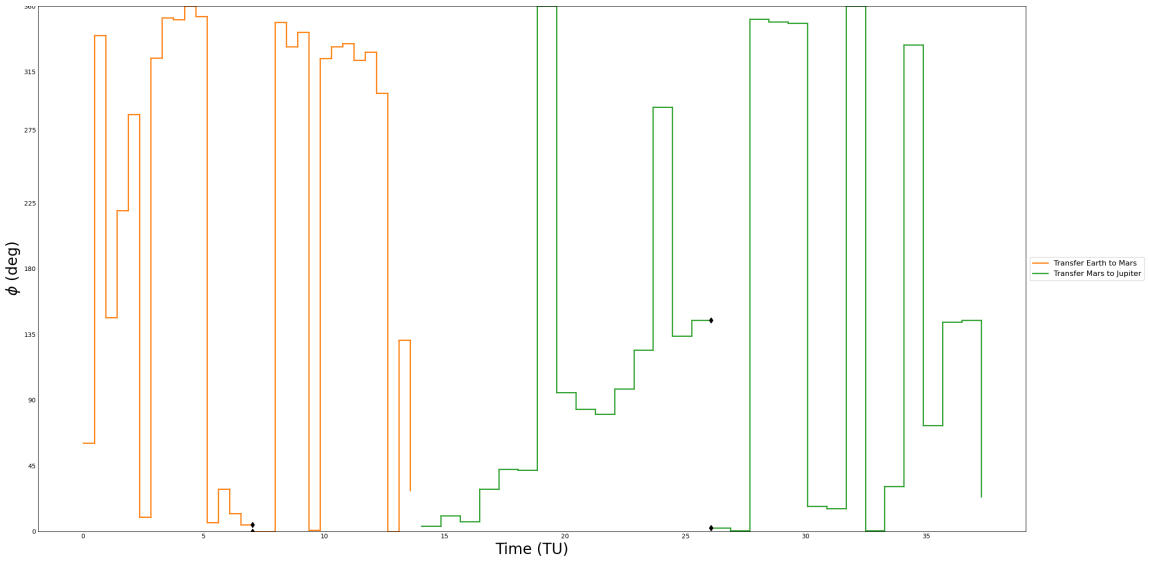

Figure B.154: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Thrust

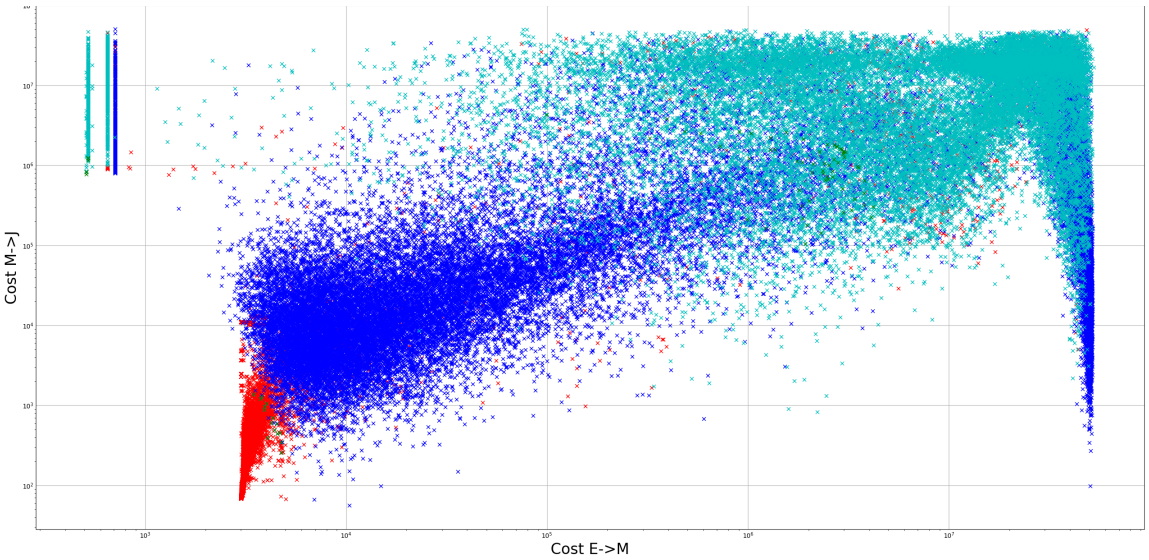

Figure B.155: Best Leg Tracking with Cluster Pruning Direct Method Optimal Solution 5 Leg Tracking

B.4.1.3 Best Leg Tracking with Thruster Angle Pruning and Cluster Pruning

Table B.19: MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	4797.14	5634.42	5479.25	3176.08	3733.69
Departure Date	Aug 26, 2021	Aug 13, 2021	Sep 7, 2021	Aug 15, 2021	Aug 23, 2021
Flyby ToF (days)	813	840	805	810	829
Arrival ToF (days)	1395	1396	1396	1396	1396
Total ToF (years)	6.05	6.12	6.03	6.04	6.09
Propellant Mass Fraction	35.87%	37.09%	35.73%	36.59%	36.90%

Table B.20: MGALT-STOpS Direct Method Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	4797.14	5634.42	5479.25	3176.08	3733.69
Earth to Mars Cost	3644.11	4592.25	4106.43	2819.78	2969.29
Mars to Jupiter Cost	1153.03	1042.17	1372.83	356.30	764.40

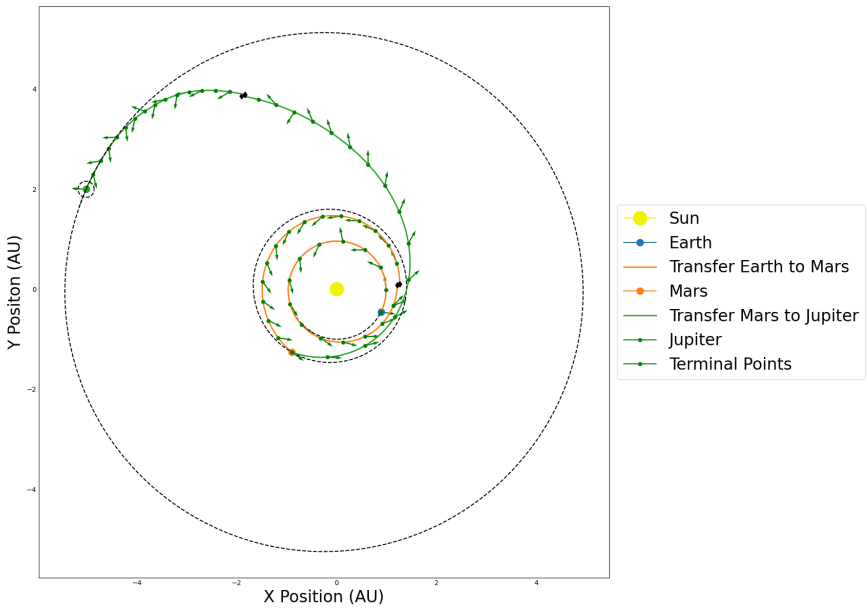

Figure B.156: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Trajectory

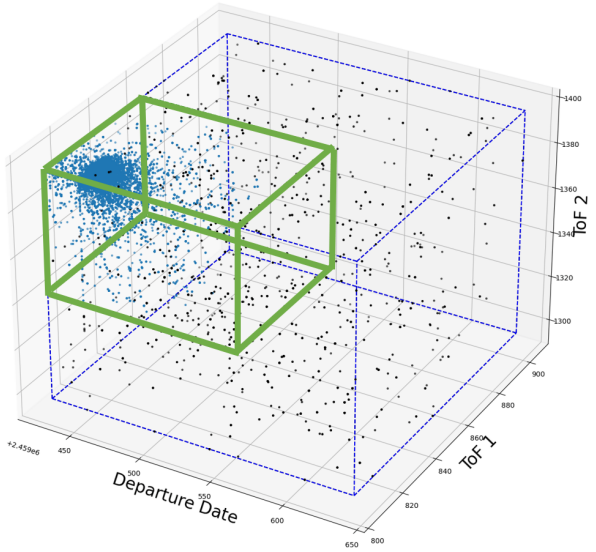

Figure B.157: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Clustering

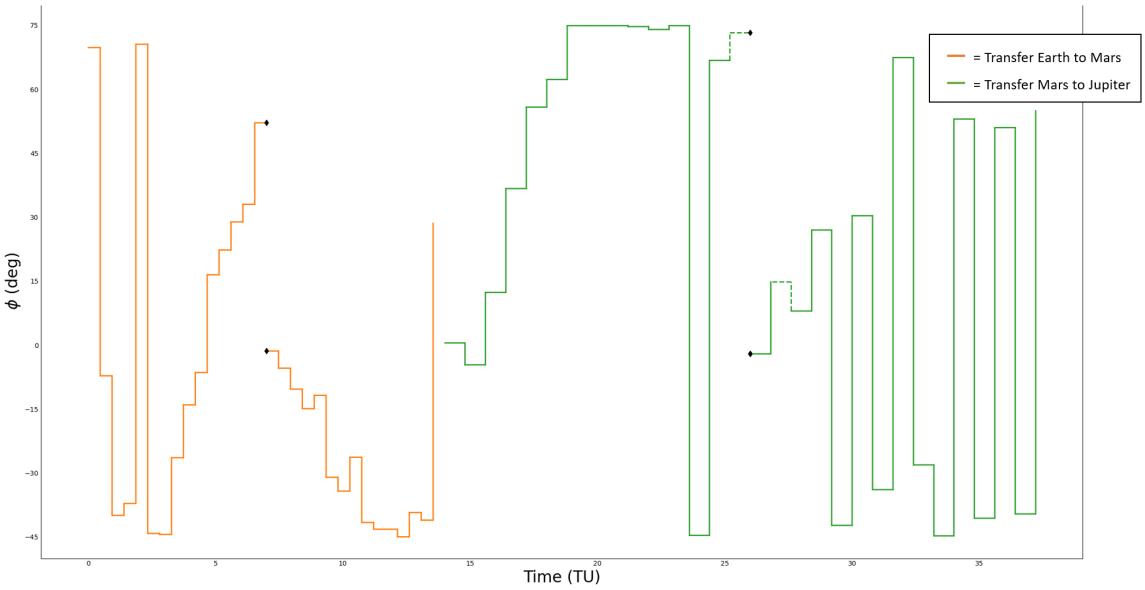

Figure B.158: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Thrust

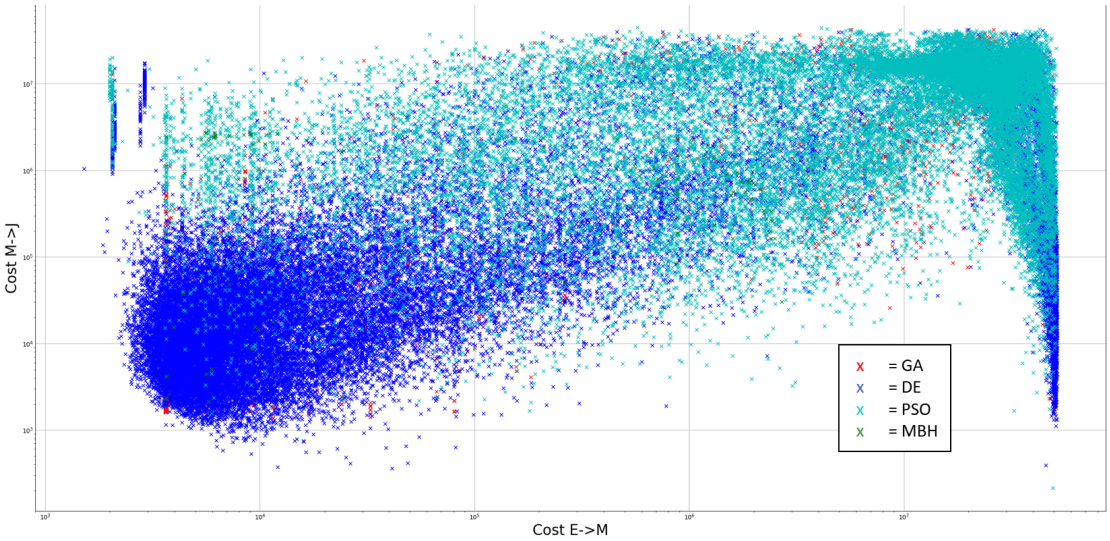

Figure B.159: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 1 Leg Tracking

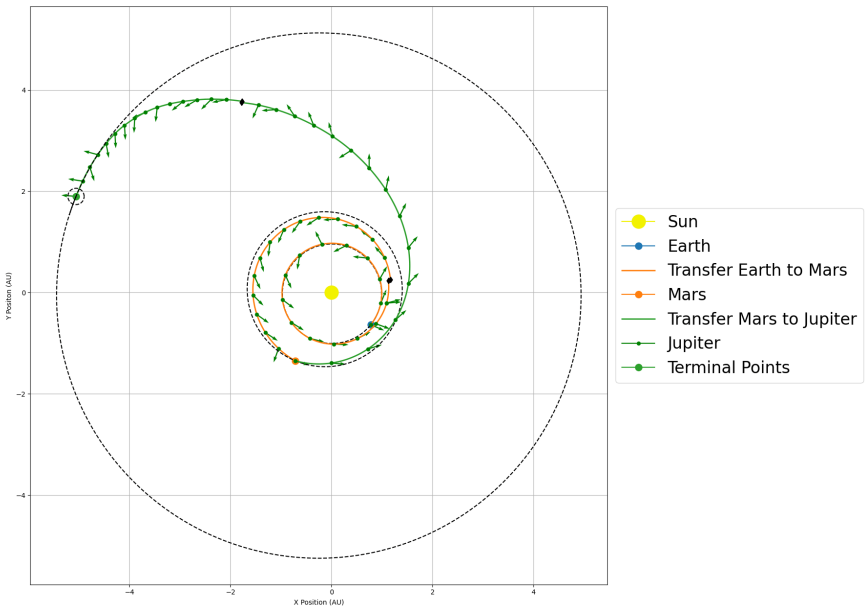

Figure B.160: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Trajectory

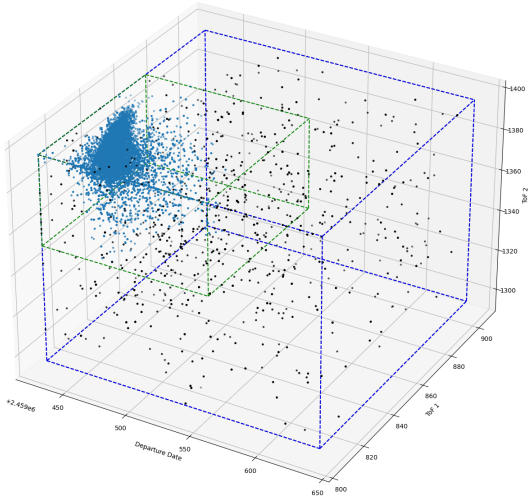

Figure B.161: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Clustering

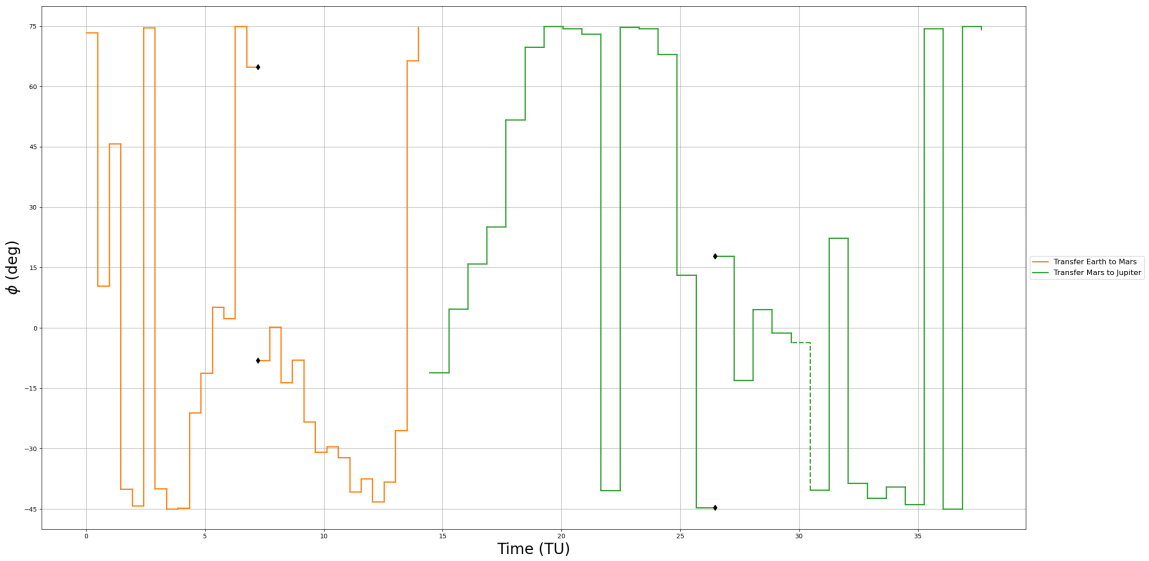

Figure B.162: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Thrust

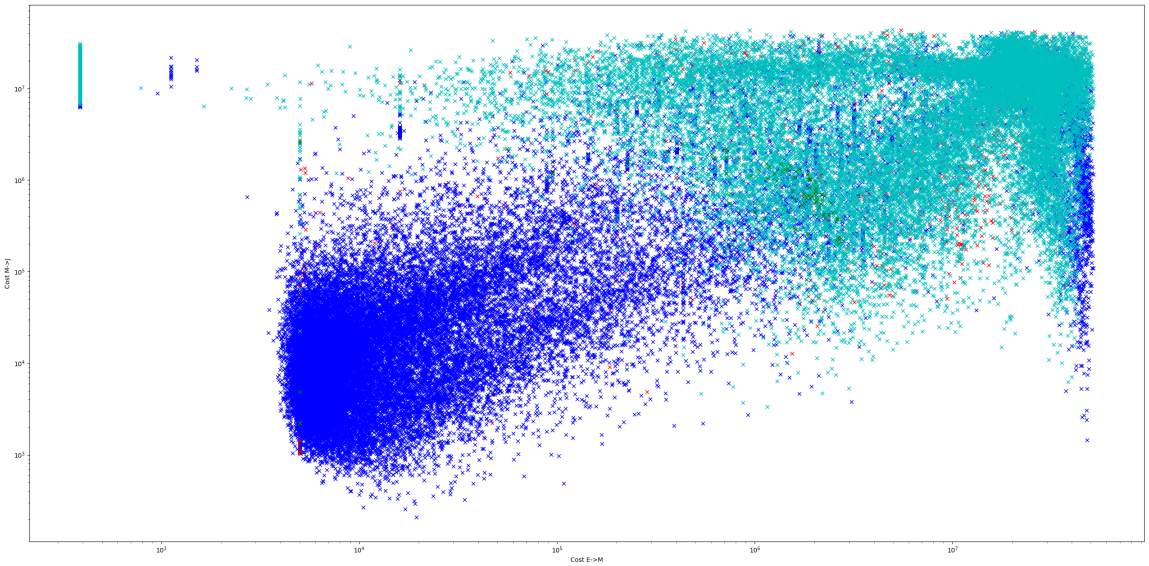

Figure B.163: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 2 Leg Tracking

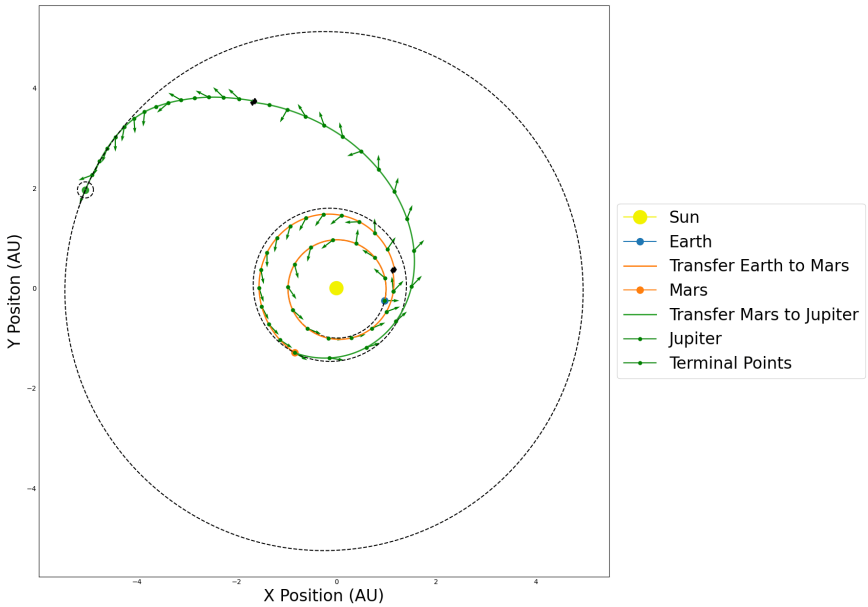

Figure B.164: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Trajectory

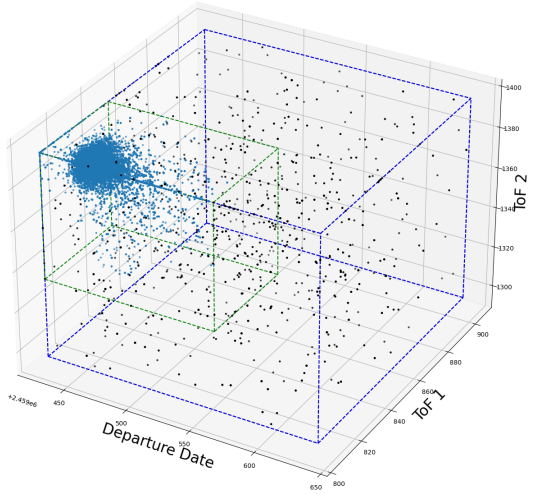

Figure B.165: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Clustering

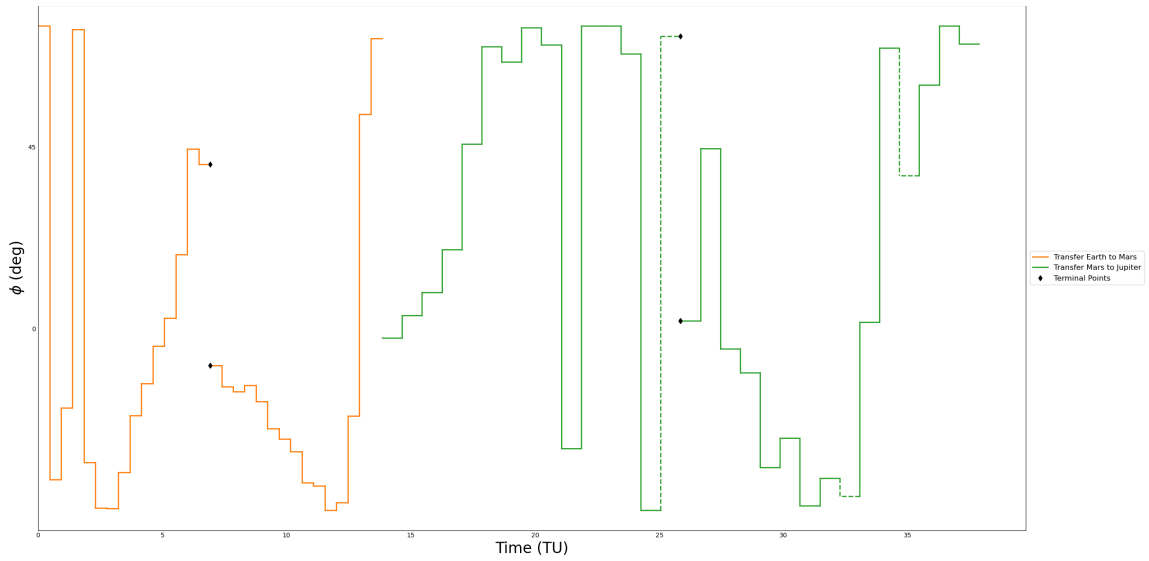

Figure B.166: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Thrust

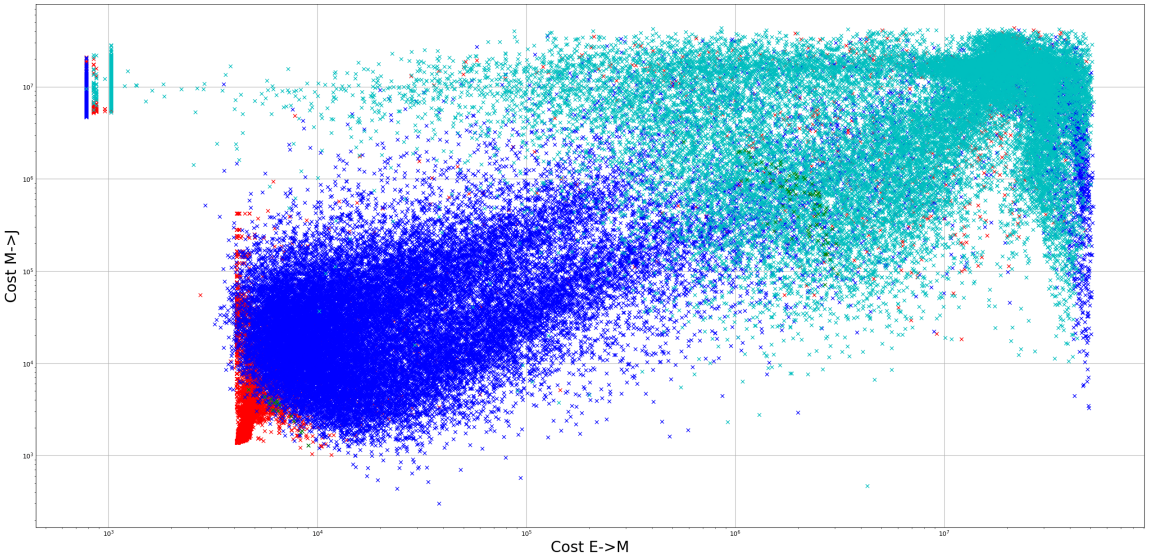

Figure B.167: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 3 Leg Tracking

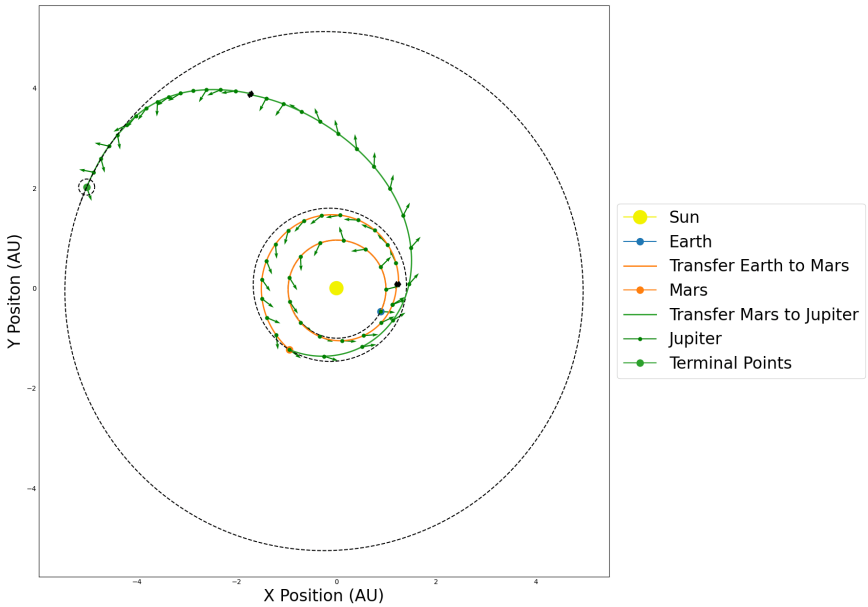

Figure B.168: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Trajectory

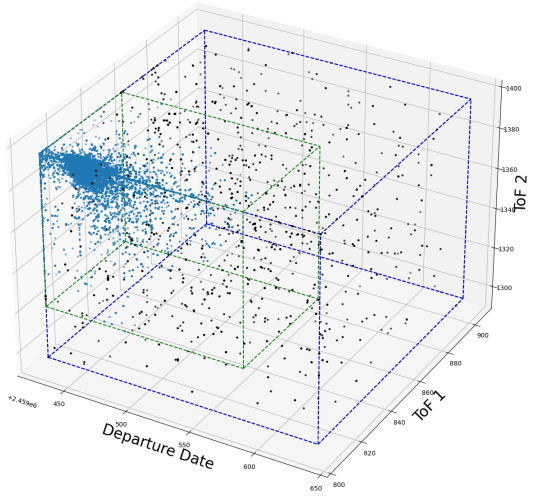

Figure B.169: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Clustering

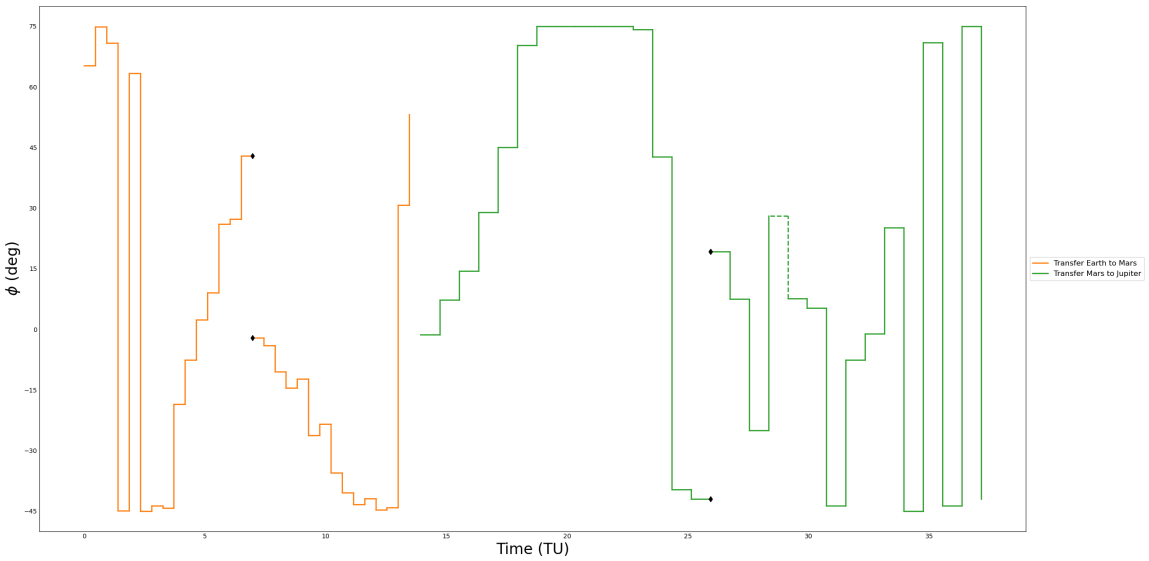

Figure B.170: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Thrust

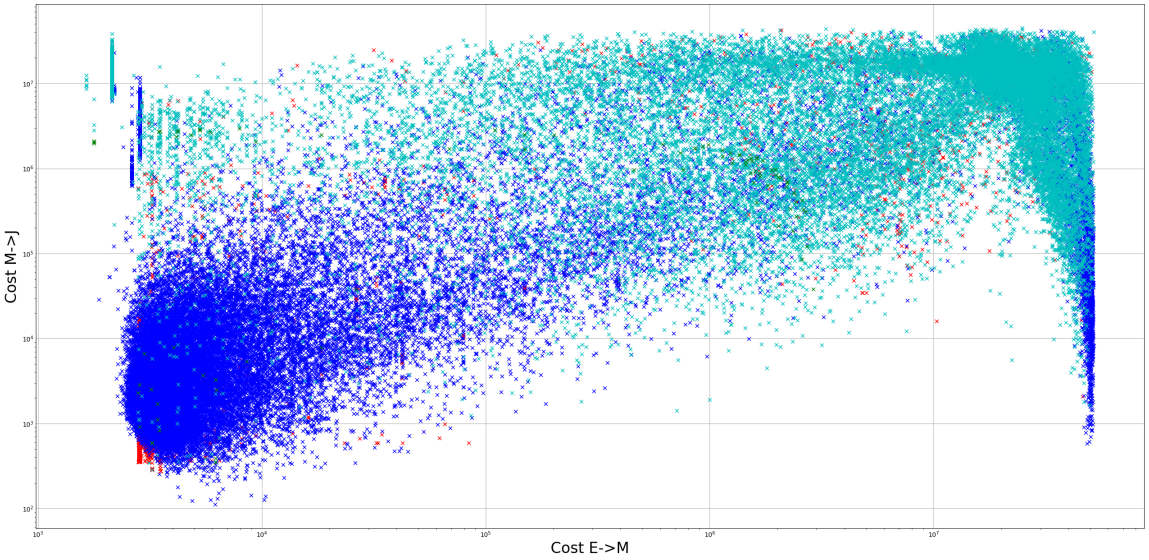

Figure B.171: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 4 Leg Tracking

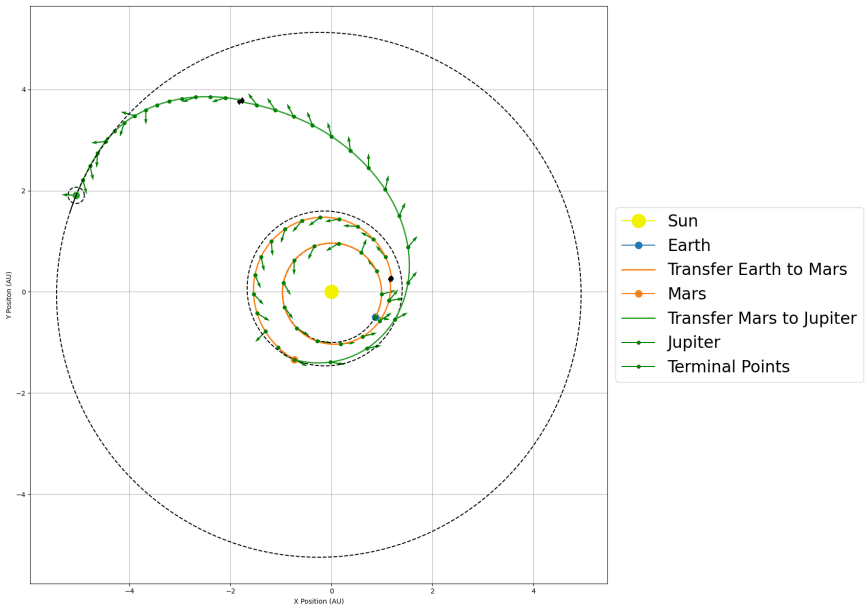

Figure B.172: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Trajectory

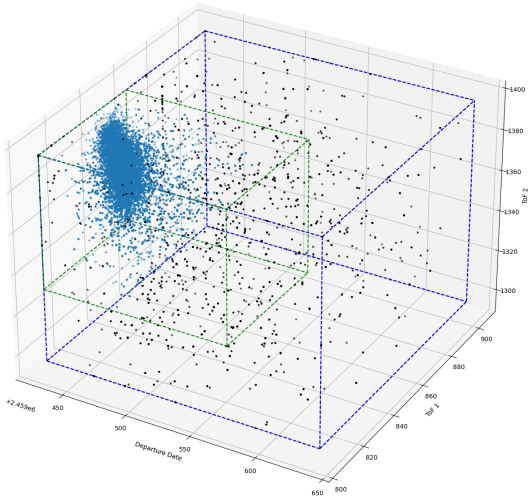

Figure B.173: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Clustering

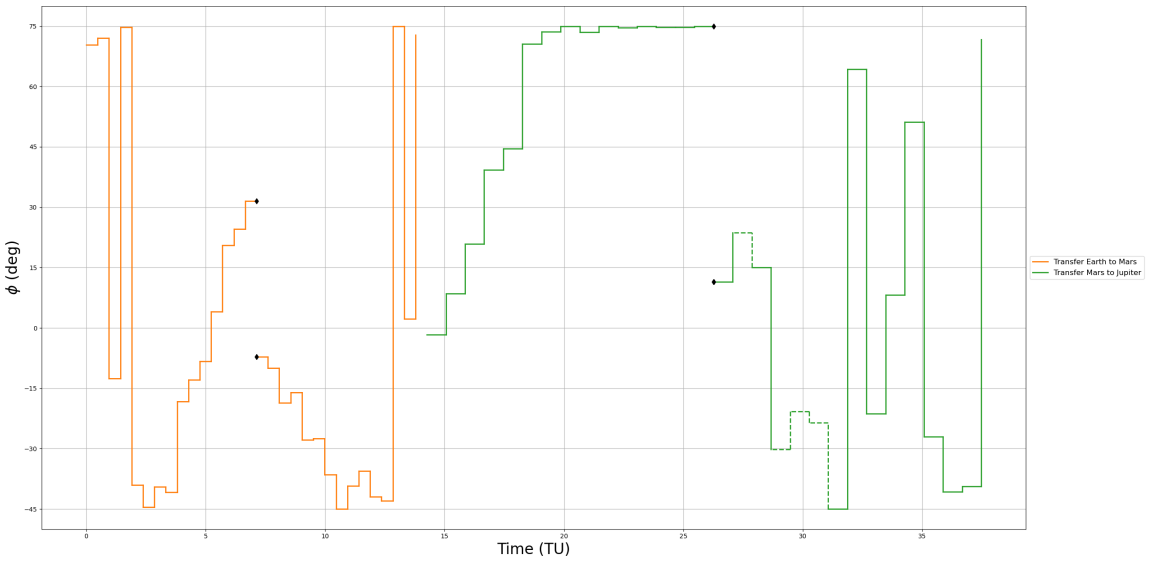

Figure B.174: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Thrust

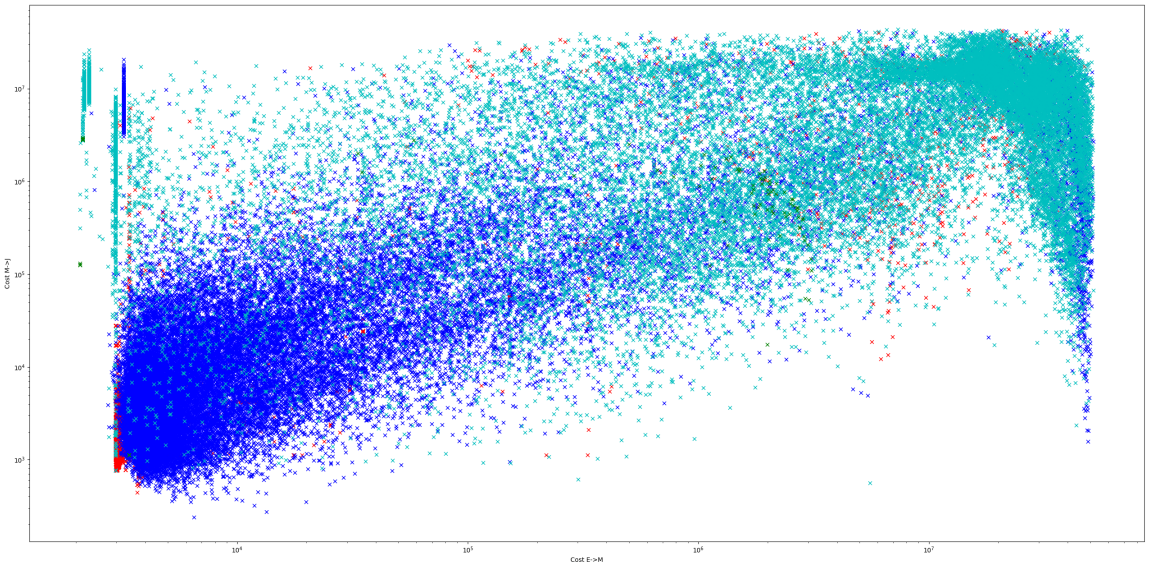

Figure B.175: Best Leg Tracking with Cluster Pruning and Thruster Angle Pruning Direct Method Optimal Solution 5 Leg Tracking

B.4.2 Indirect Method Results

Table B.21: MGALT-STOpS Indirect Method Best Leg Tracking Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	281.90	105.46	421.26	473.33	210.81
Departure Date	Oct 17, 2021	Nov 11, 2021	Nov 29, 2021	Nov 20, 2021	Apr 24, 2021
Flyby ToF (days)	709	691	660	674	852
Arrival ToF (days)	1713	1748	1734	1744	1746
Total ToF (years)	6.63	6.68	6.55	6.62	7.11
Propellant Mass Fraction	40.18%	40.46%	39.71%	40.11	43.10%

Table B.22: MGALT-STOpS Indirect Method Best Leg Tracking Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	281.90	105.46	421.26	473.33	210.81
Earth to Mars Cost	173.54	53.55	288.18	305.62	12.05
Mars to Jupiter Cost	108.37	51.91	133.09	167.72	198.76

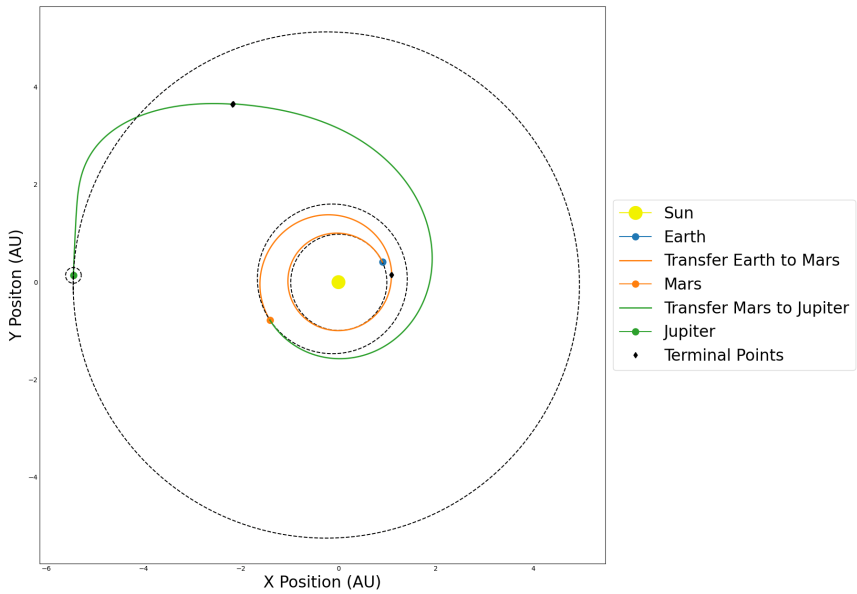

Figure B.176: Best Leg Tracking Indirect Method Optimal Solution 1 Trajectory

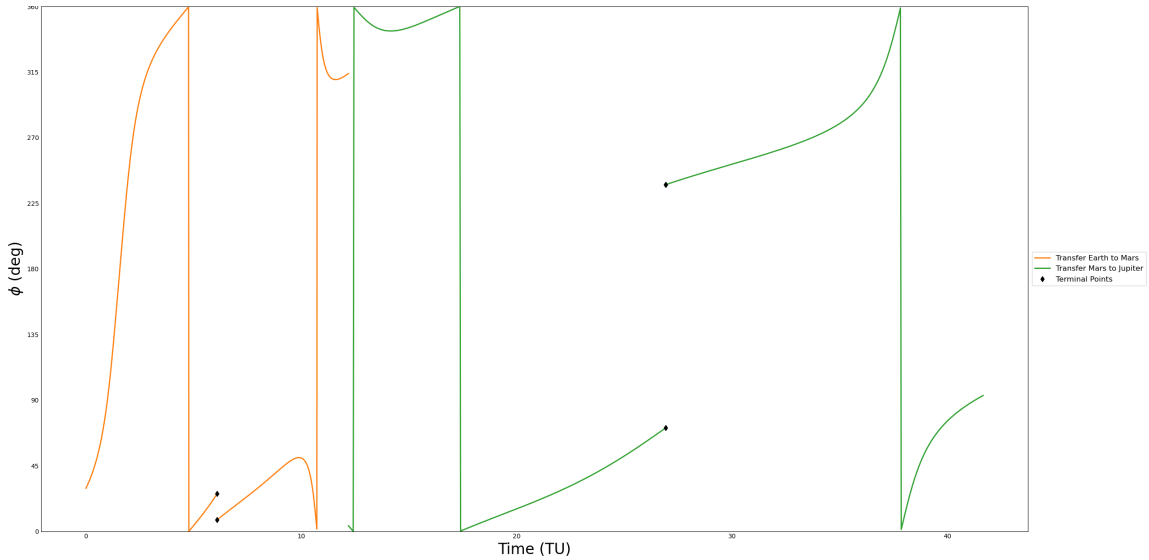

Figure B.177: Best Leg Tracking Indirect Method Optimal Solution 1 Thrust

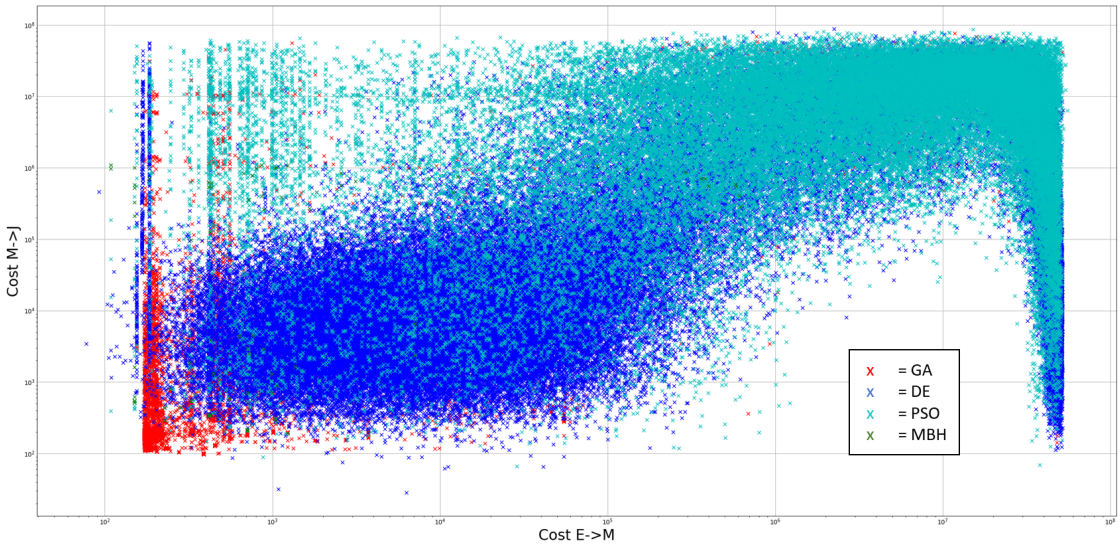

Figure B.178: Best Leg Tracking Indirect Method Optimal Solution 1 Leg Tracking

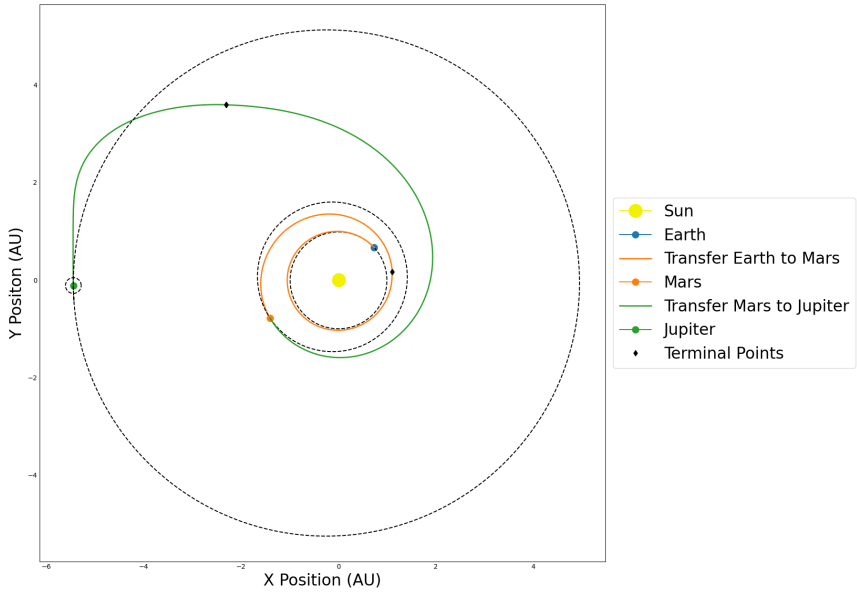

Figure B.179: Best Leg Tracking Indirect Method Optimal Solution 2 Trajectory

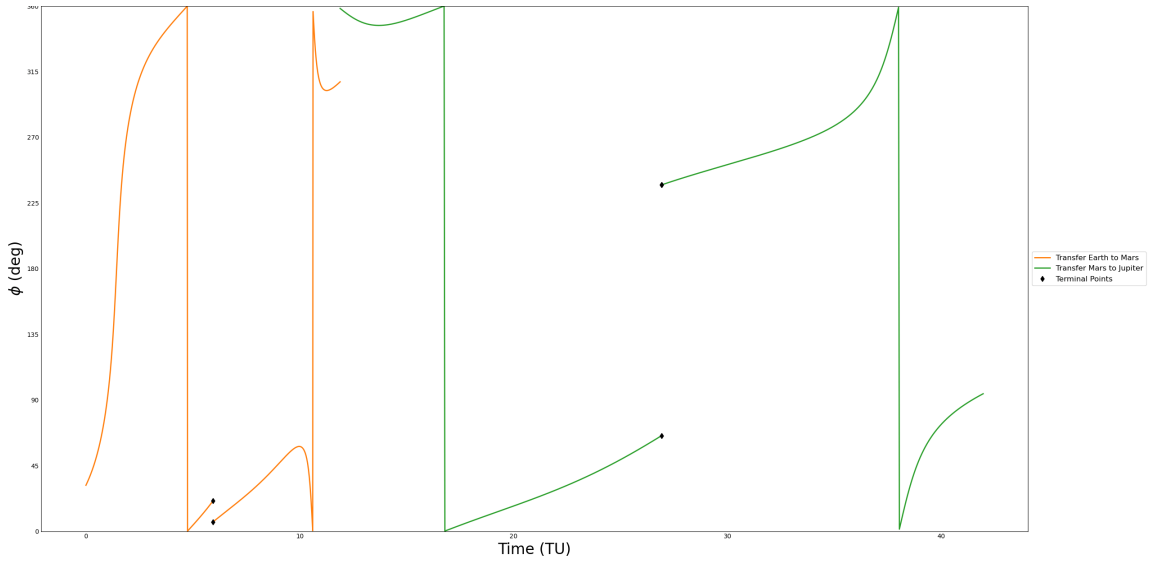

Figure B.180: Best Leg Tracking Indirect Method Optimal Solution 2 Thrust

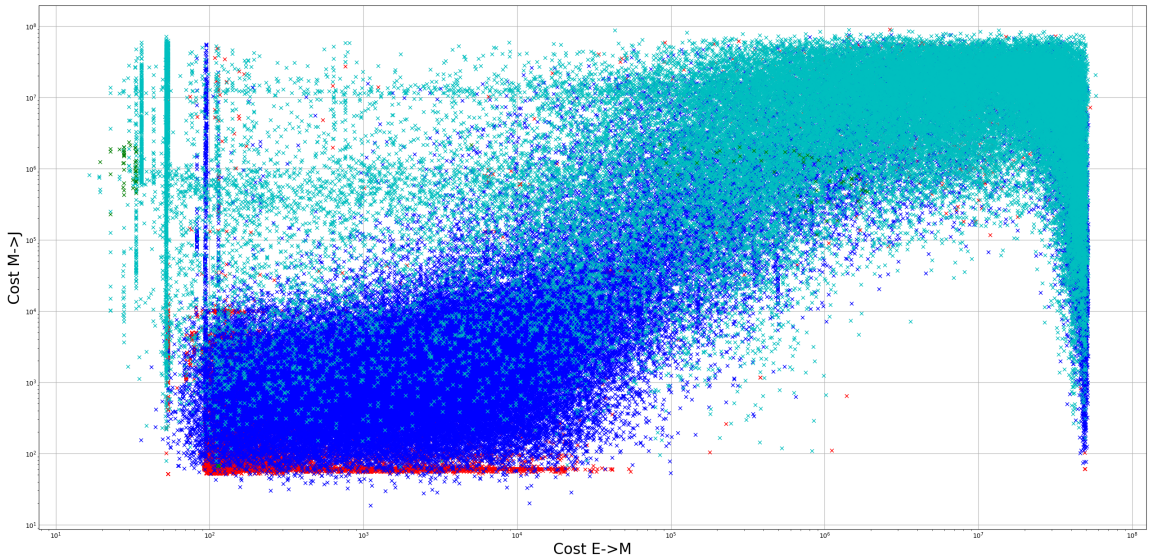

Figure B.181: Best Leg Tracking Indirect Method Optimal Solution 2 Leg Tracking

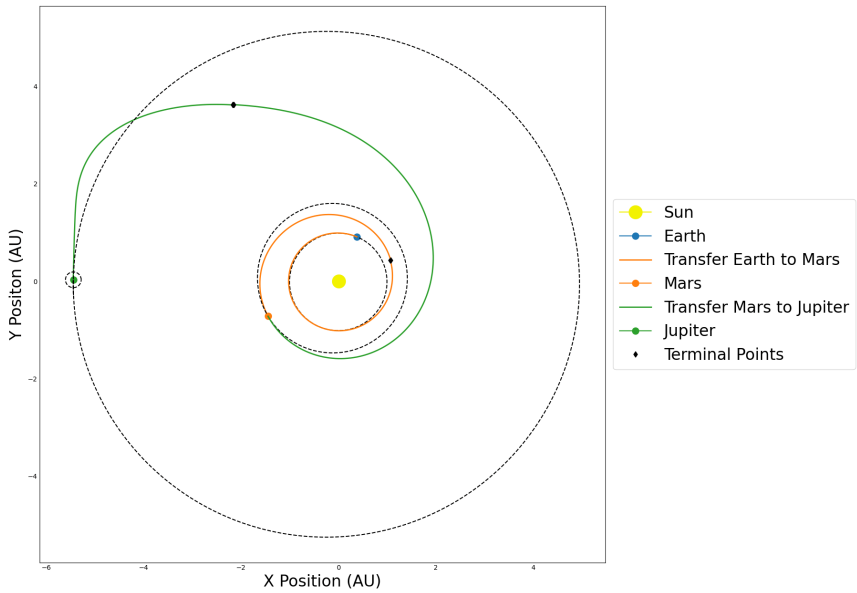

Figure B.182: Best Leg Tracking Indirect Method Optimal Solution 3 Trajectory

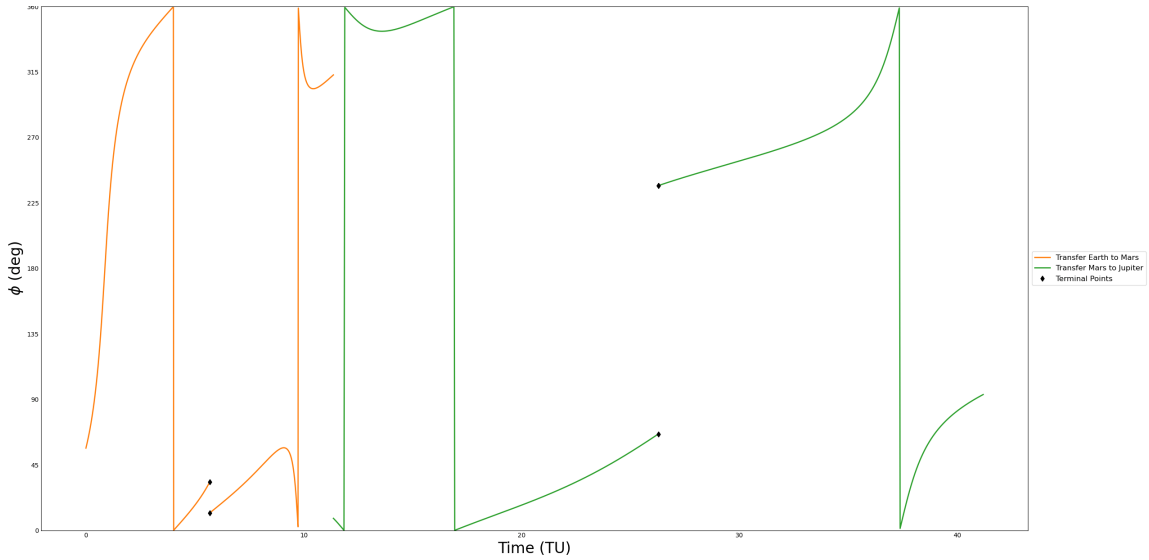

Figure B.183: Best Leg Tracking Indirect Method Optimal Solution 3 Thrust

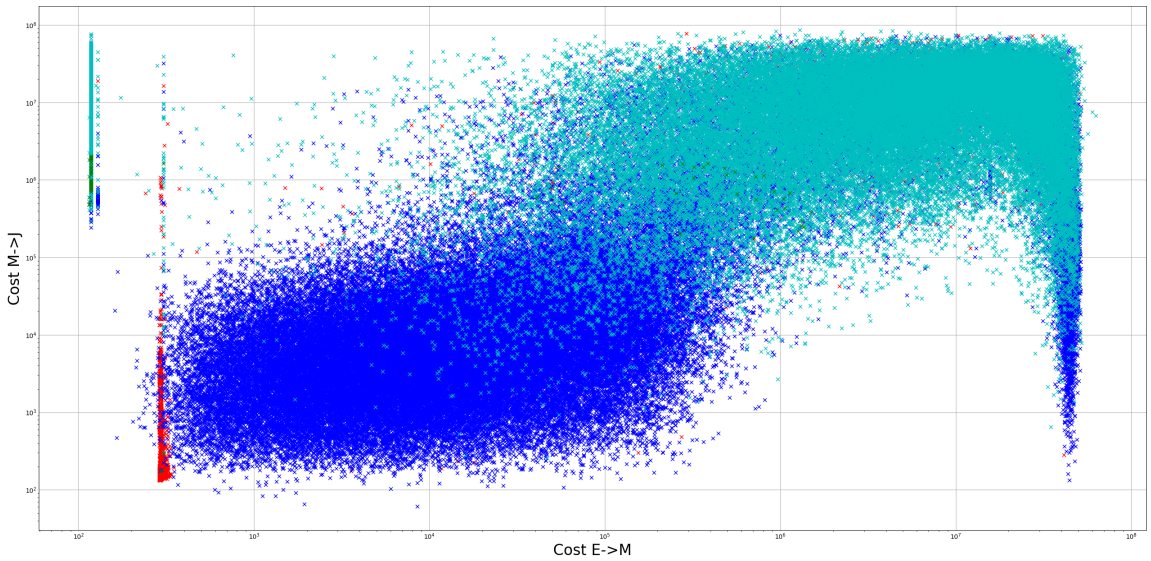

Figure B.184: Best Leg Tracking Indirect Method Optimal Solution 3 Leg Tracking

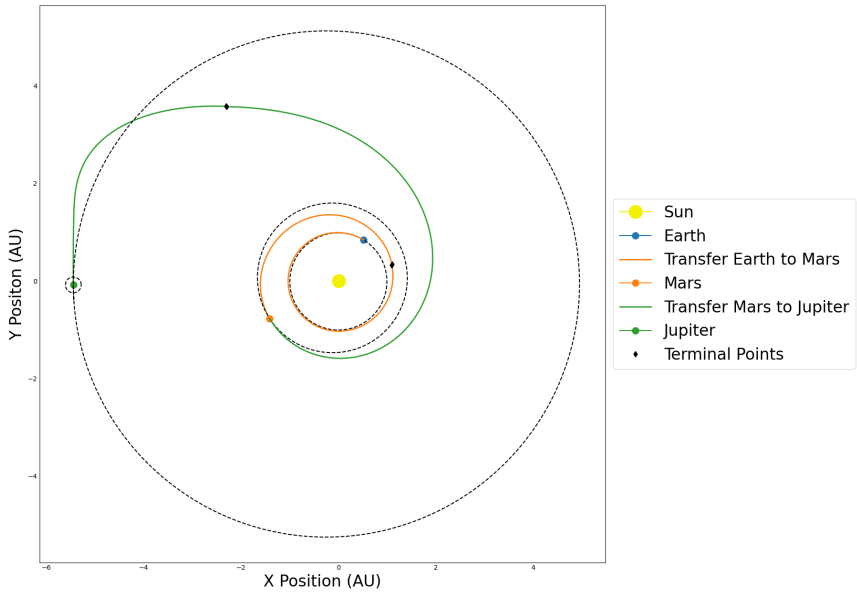

Figure B.185: Best Leg Tracking Indirect Method Optimal Solution 4 Trajectory

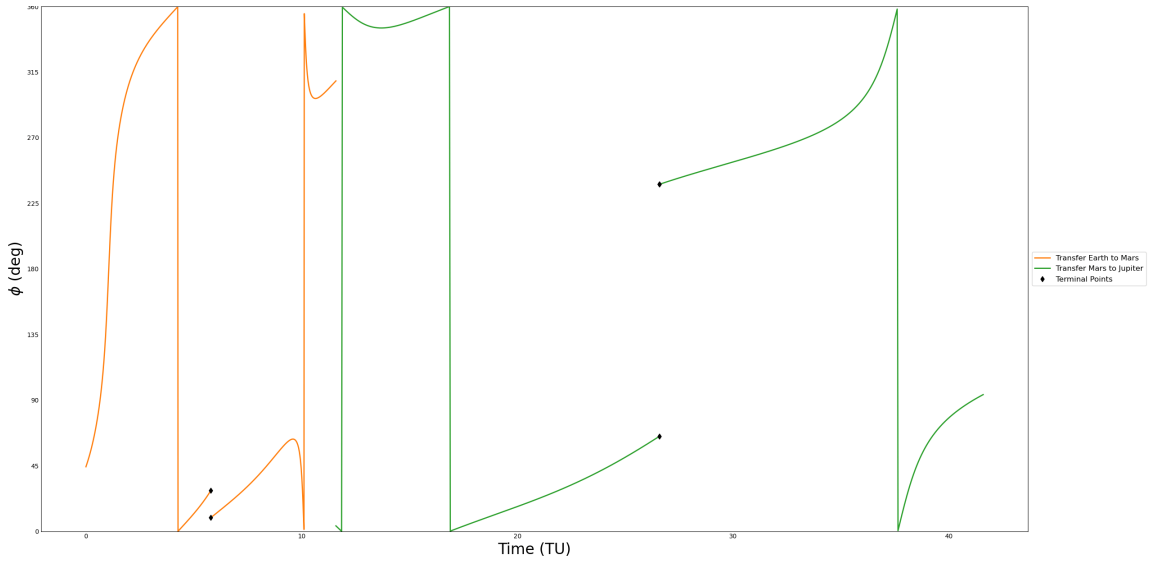

Figure B.186: Best Leg Tracking Indirect Method Optimal Solution 4 Thrust

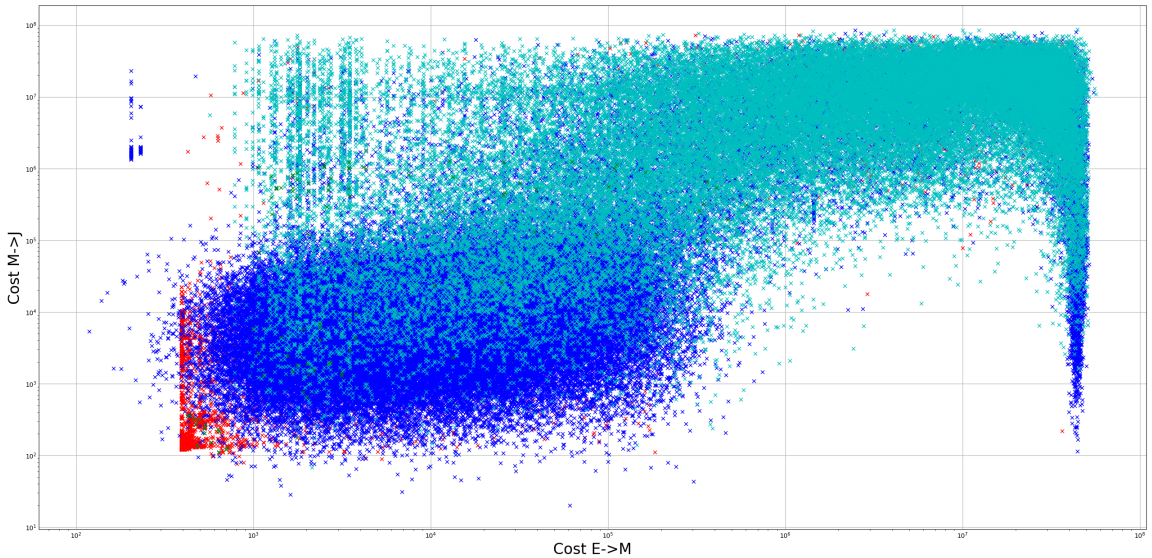

Figure B.187: Best Leg Tracking Indirect Method Optimal Solution 4 Leg Tracking

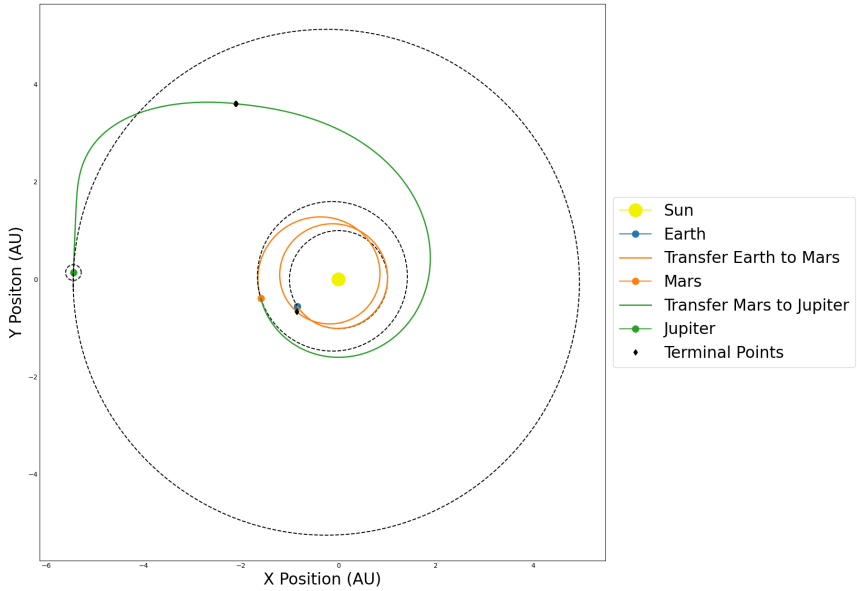

Figure B.188: Best Leg Tracking Indirect Method Optimal Solution 5 Trajectory

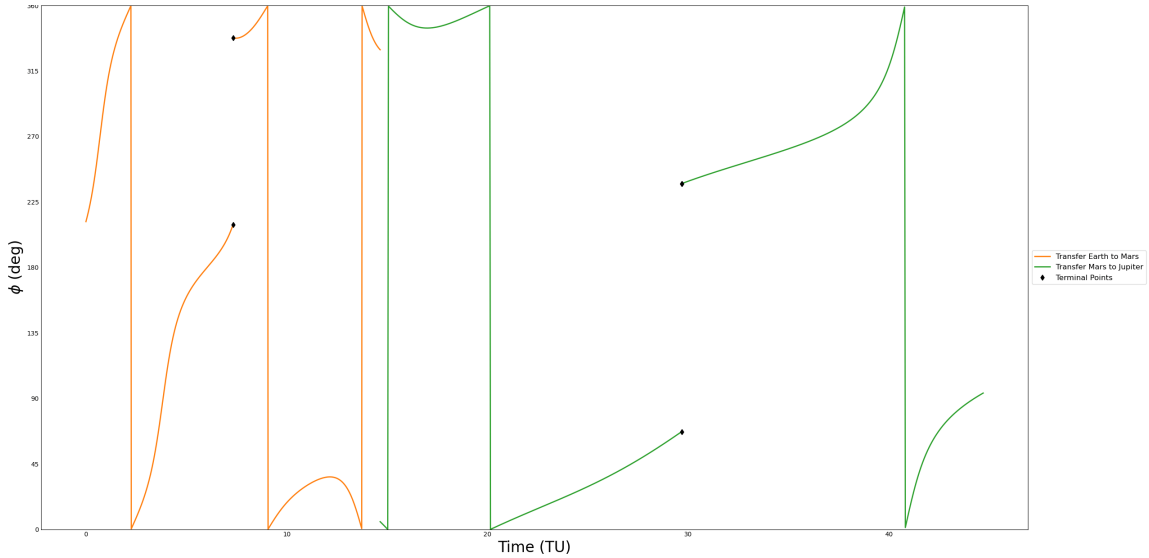

Figure B.189: Best Leg Tracking Indirect Method Optimal Solution 5 Thrust

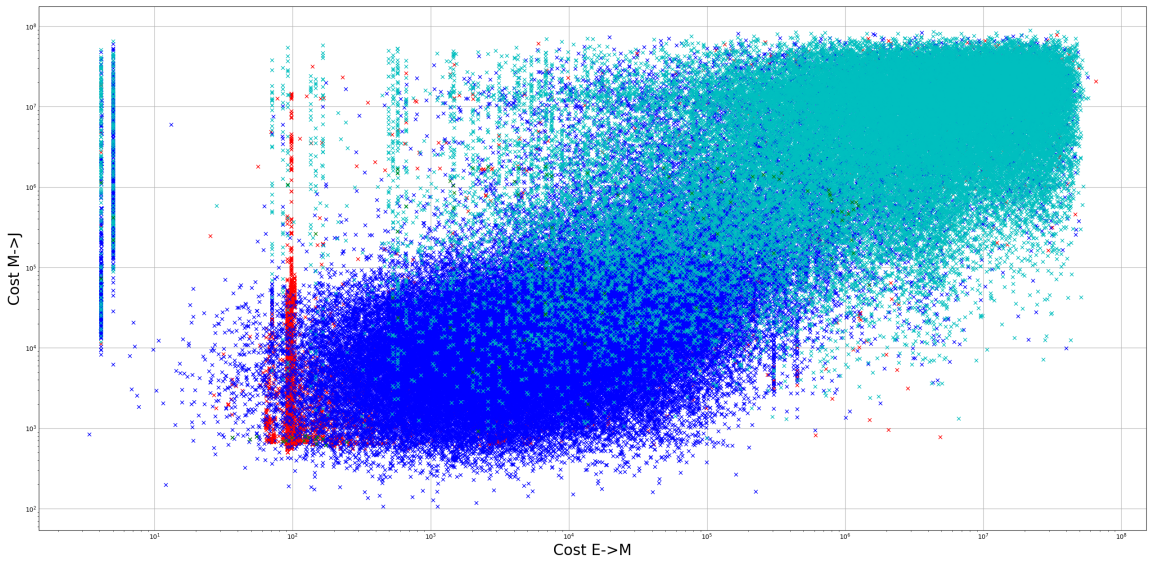

Figure B.190: Best Leg Tracking Indirect Method Optimal Solution 5 Leg Tracking

B.4.2.1 Best Leg Tracking with Cluster Pruning

Table B.23: MGALT-STOpS Indirect Method Best Leg Tracking with Cluster Pruning Results

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	292.81	321.58	1143.20	627.14	302.62
Departure Date	Aug 26, 2021	May 24, 2021	Jul 19, 2021	Dec 5, 2021	Jun 21, 2021
Flyby ToF (days)	733	852	796	689	847
Arrival ToF (days)	1388	1640	1563	1748	1564
Total ToF (years)	5.81	6.82	6.46	6.67	6.60
Propellant Mass Fraction	35.18%	41.33%	39.14%	40.42%	40.00%

Table B.24: MGALT-STOpS Indirect Method Best Leg Tracking with Cluster Pruning Cost Breakdown

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**
Total Cost	292.81	321.58	1143.20	627.14	302.62
Earth to Mars Cost	282.77	100.03	234.51	540.17	66.72
Mars to Jupiter Cost	10.05	221.55	908.69	86.96	235.90

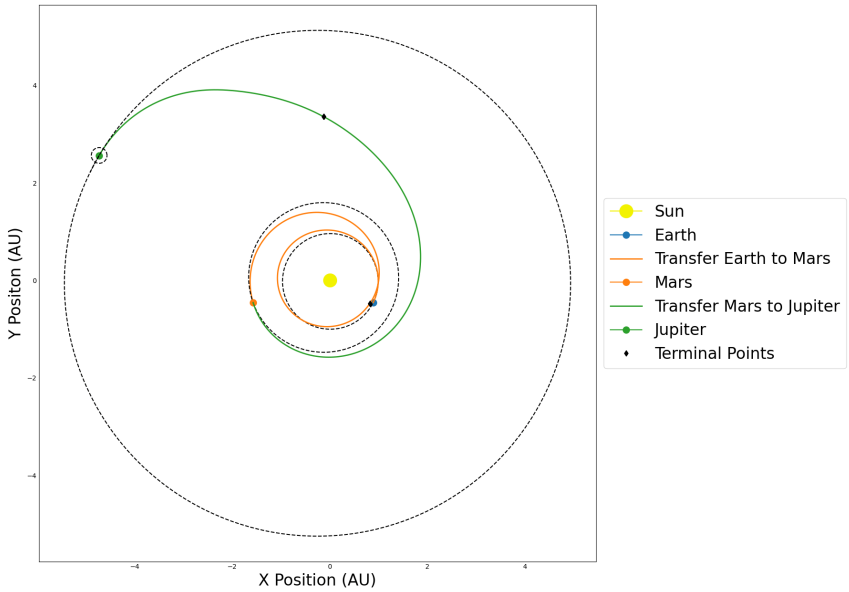

Figure B.191: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Trajectory

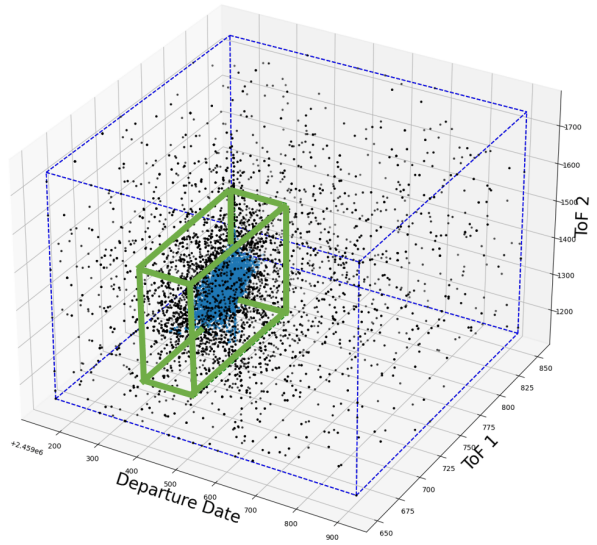

Figure B.192: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Clustering

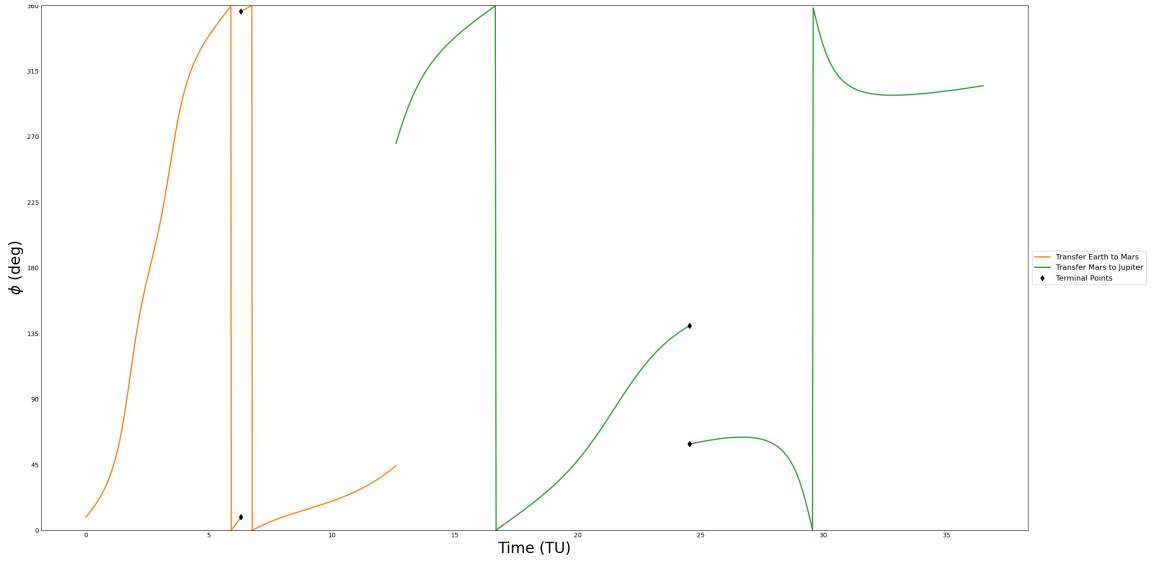

Figure B.193: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Thrust

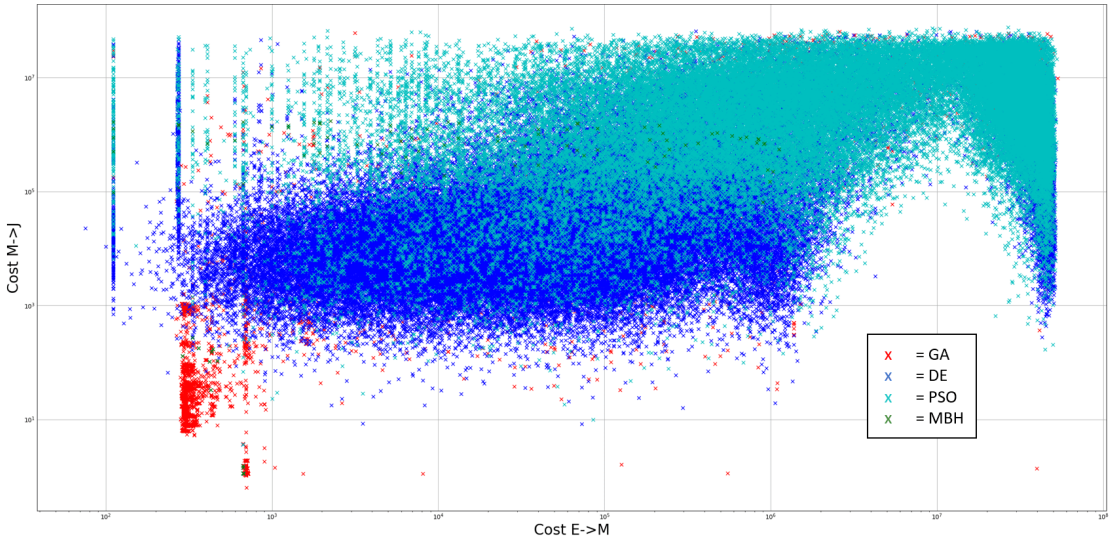

Figure B.194: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 1 Leg Tracking

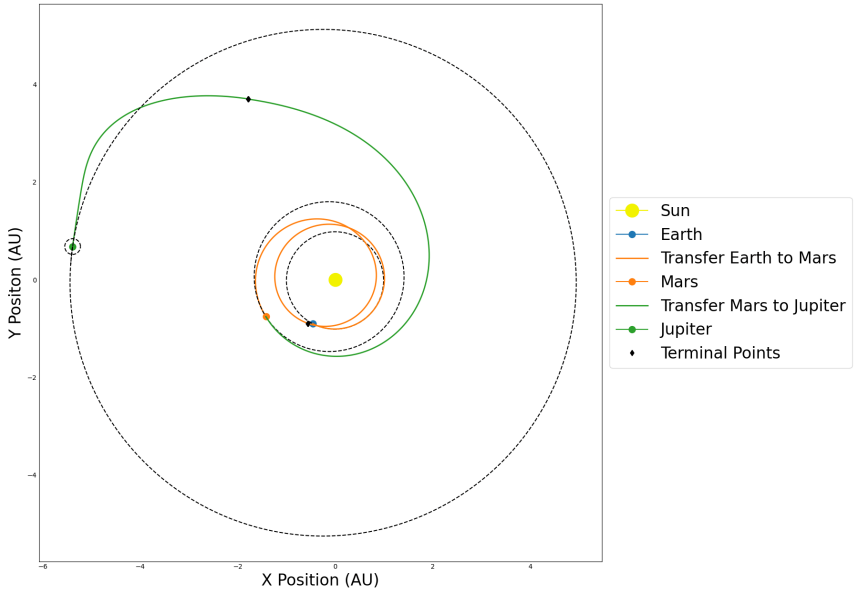

Figure B.195: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Trajectory

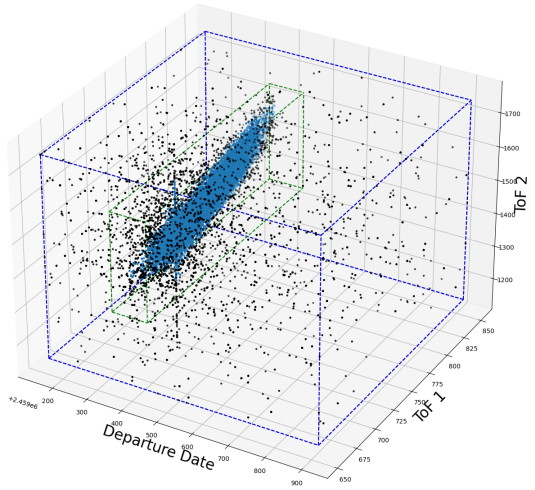

Figure B.196: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Clustering

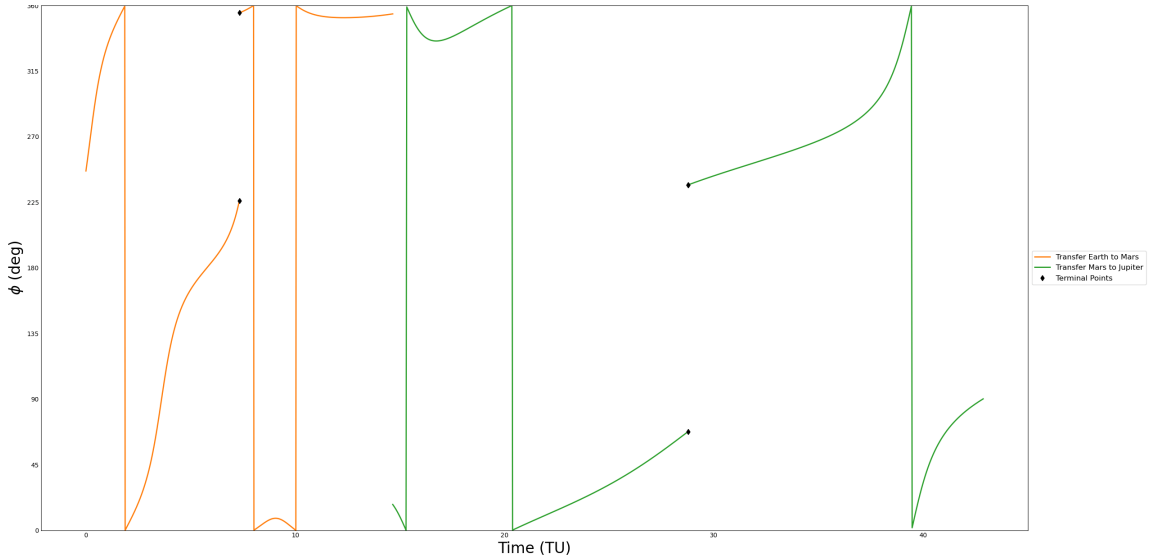

Figure B.197: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Thrust

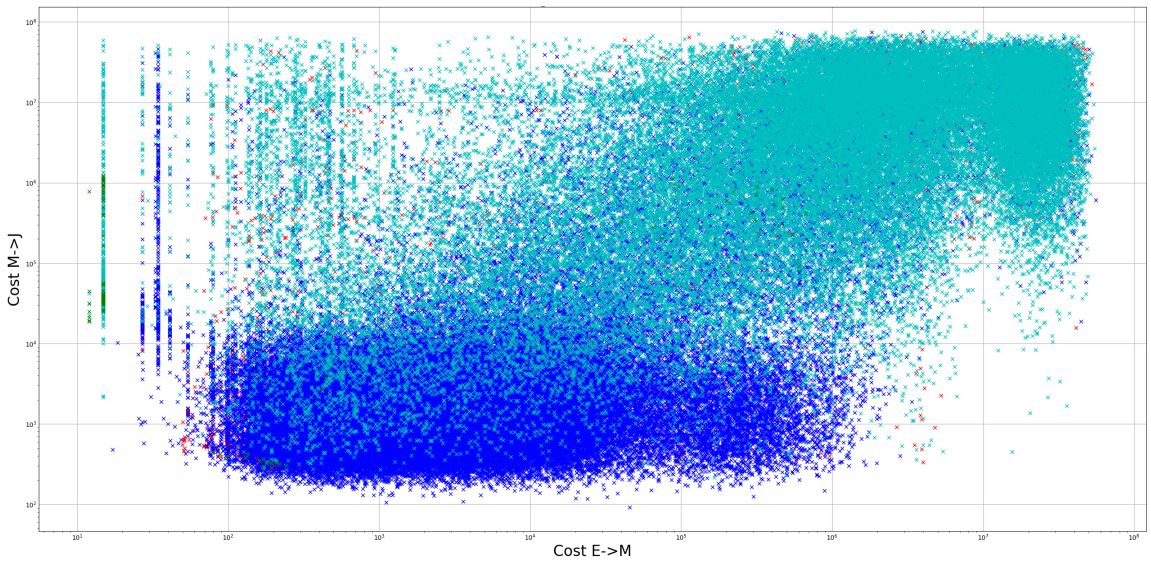

Figure B.198: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 2 Leg Tracking

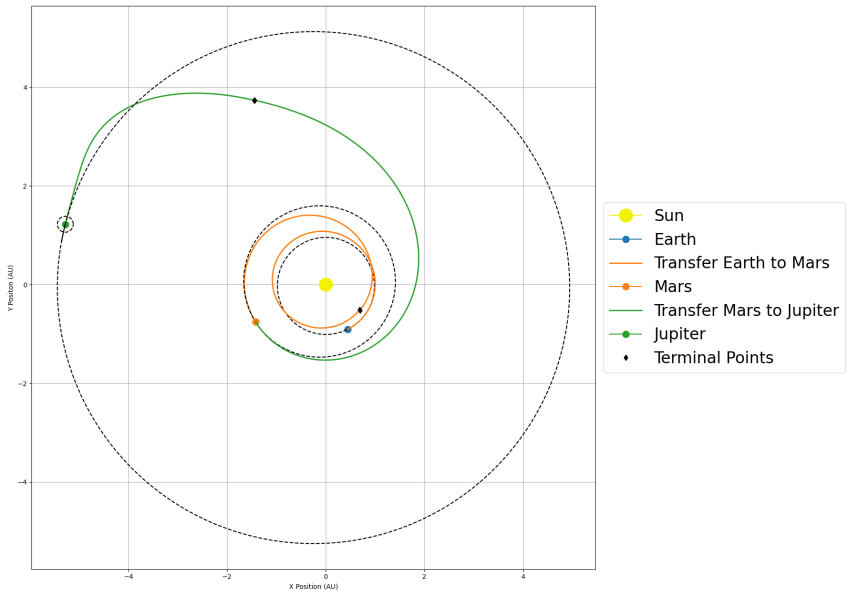

Figure B.199: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Trajectory

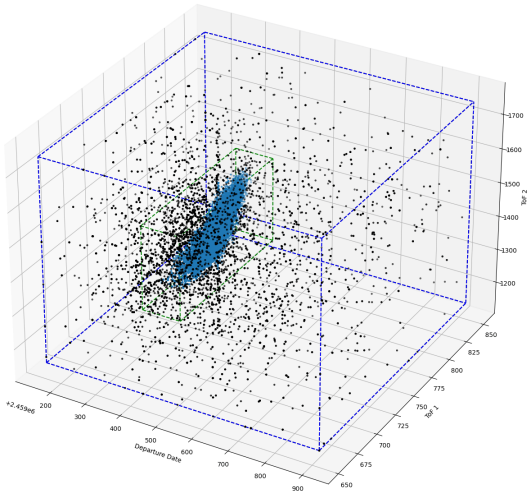

Figure B.200: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Clustering

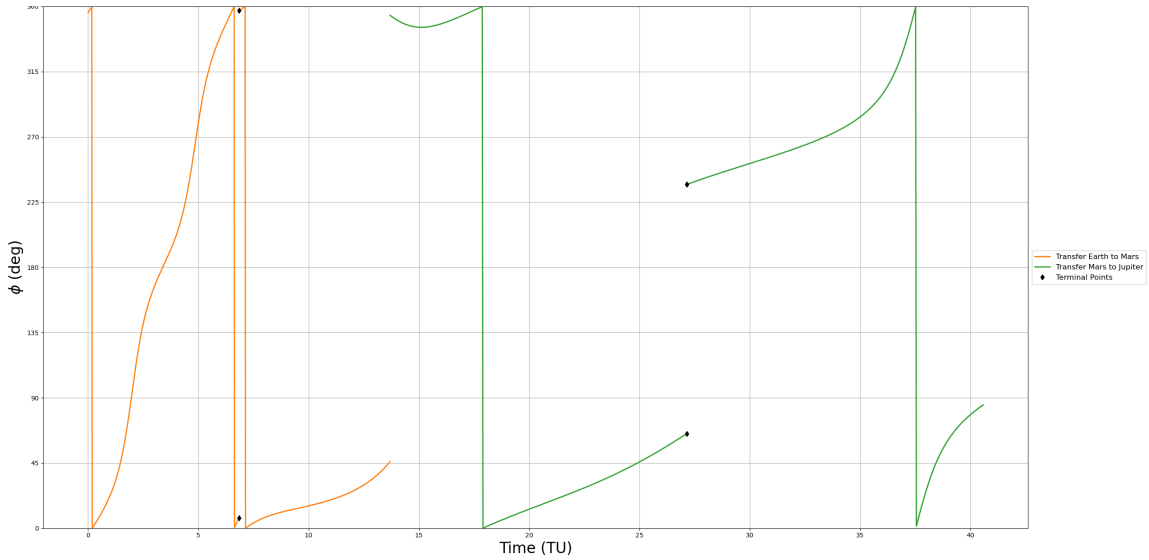

Figure B.201: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Thrust

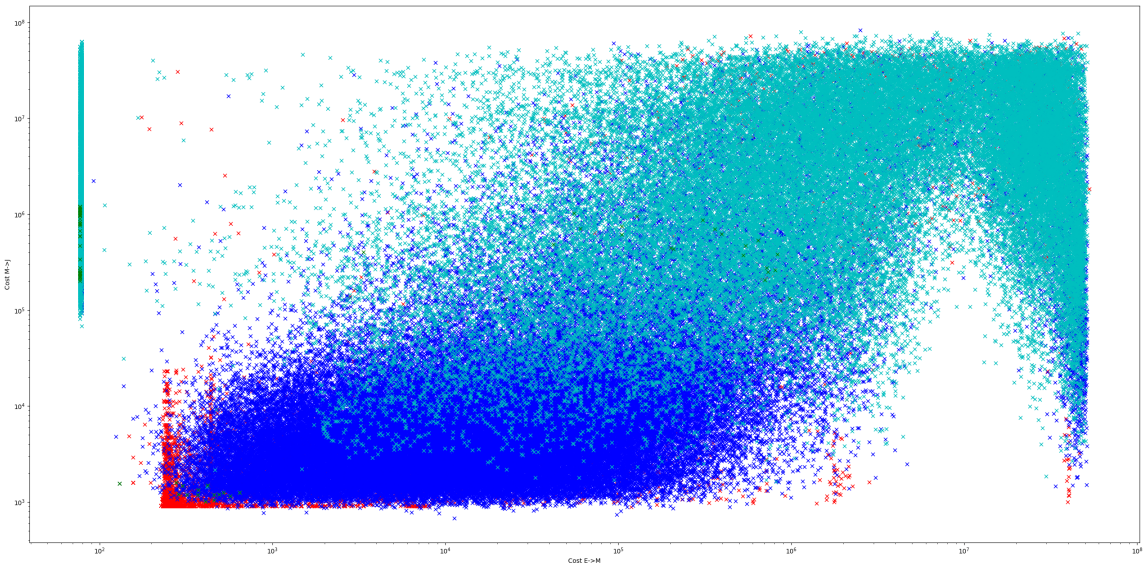

Figure B.202: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 3 Leg Tracking

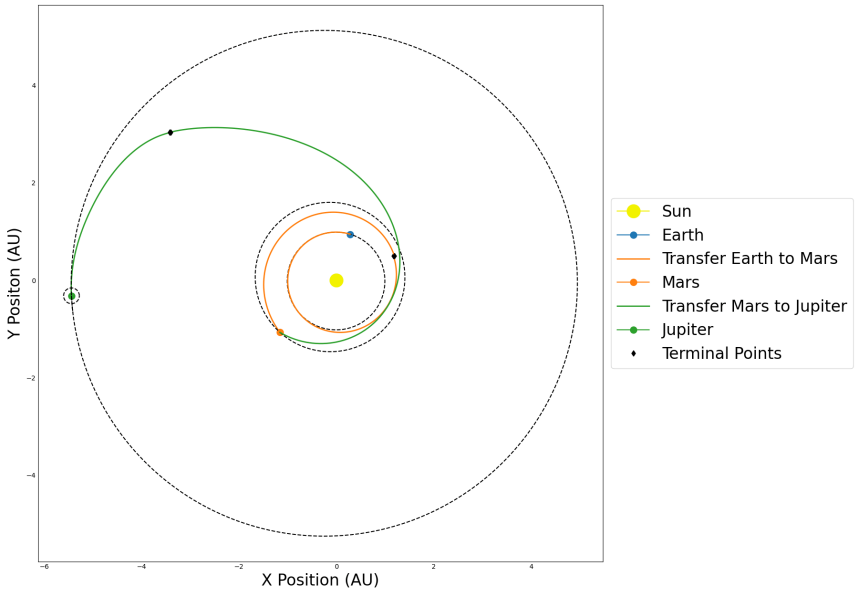

Figure B.203: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Trajectory

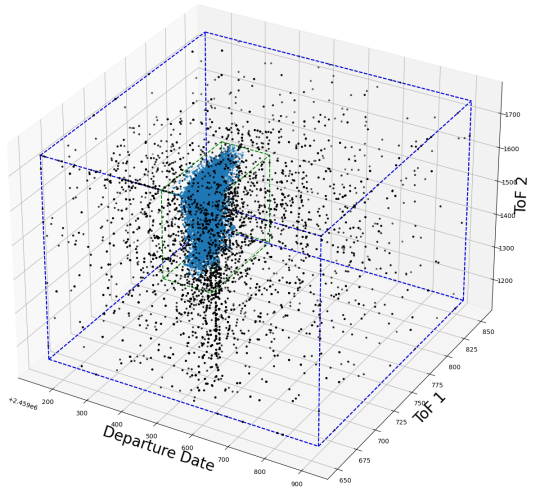

Figure B.204: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Clustering

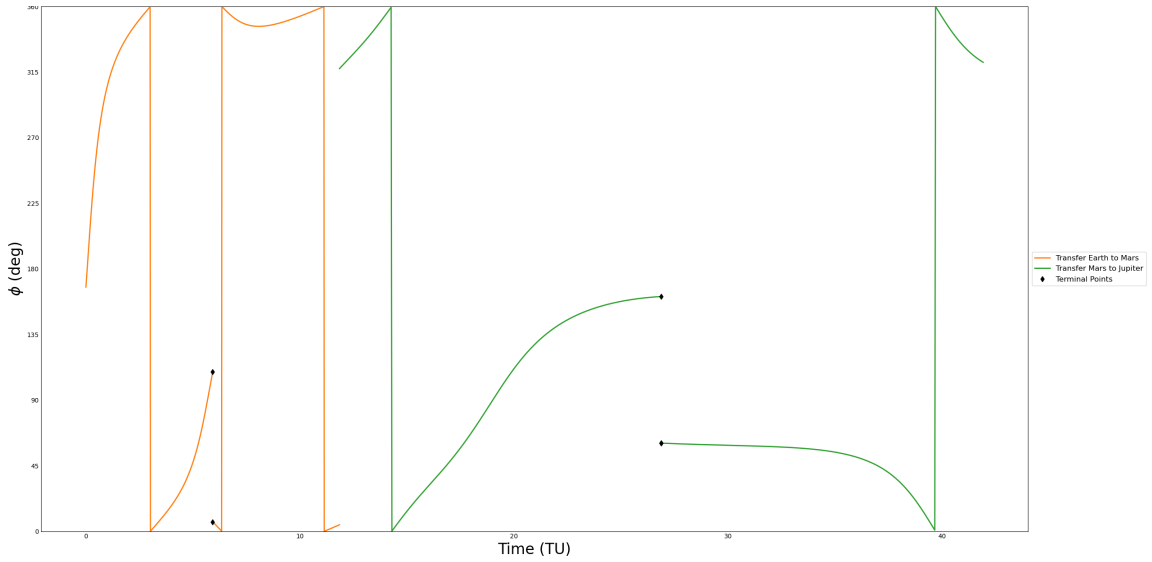

Figure B.205: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Thrust

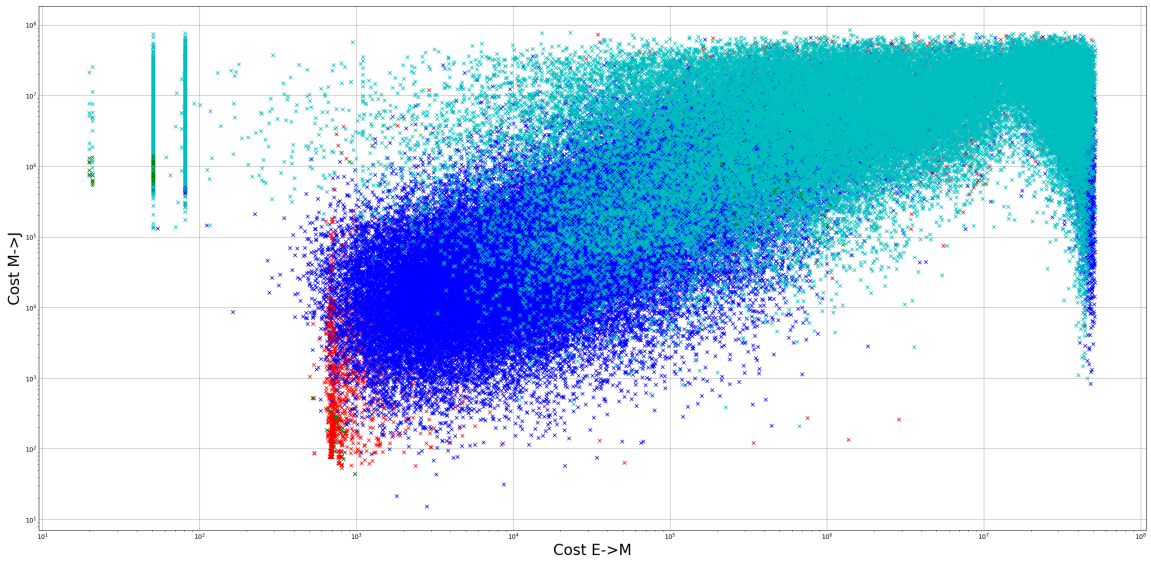

Figure B.206: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 4 Leg Tracking

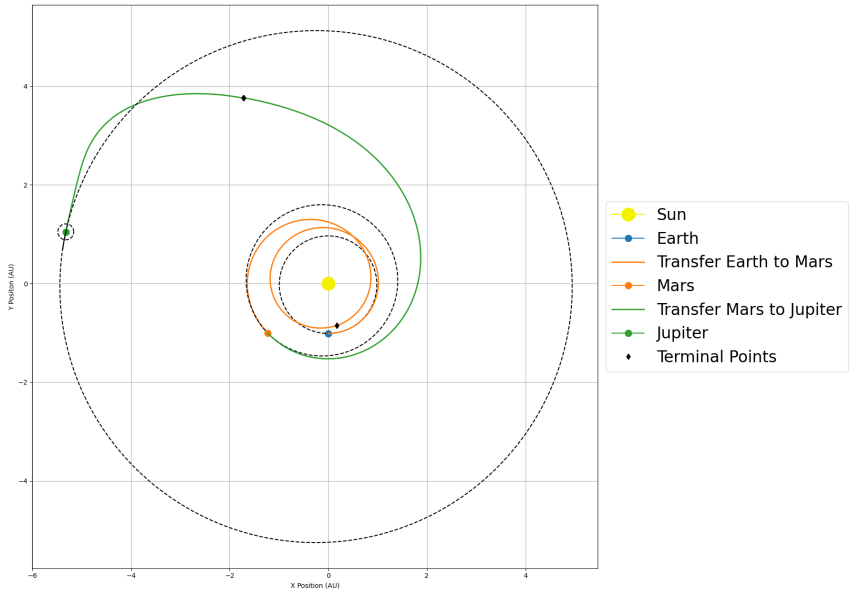

Figure B.207: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Trajectory

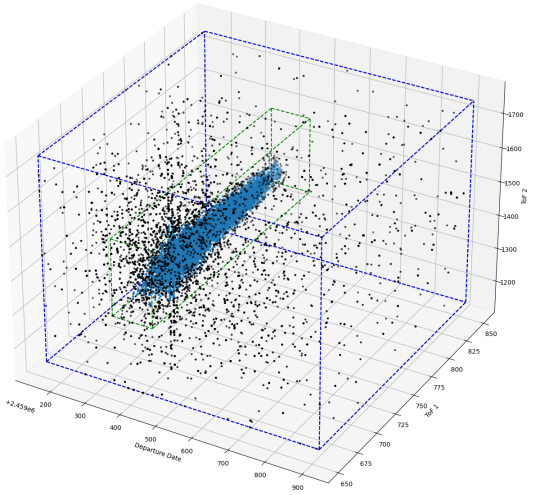

Figure B.208: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Clustering

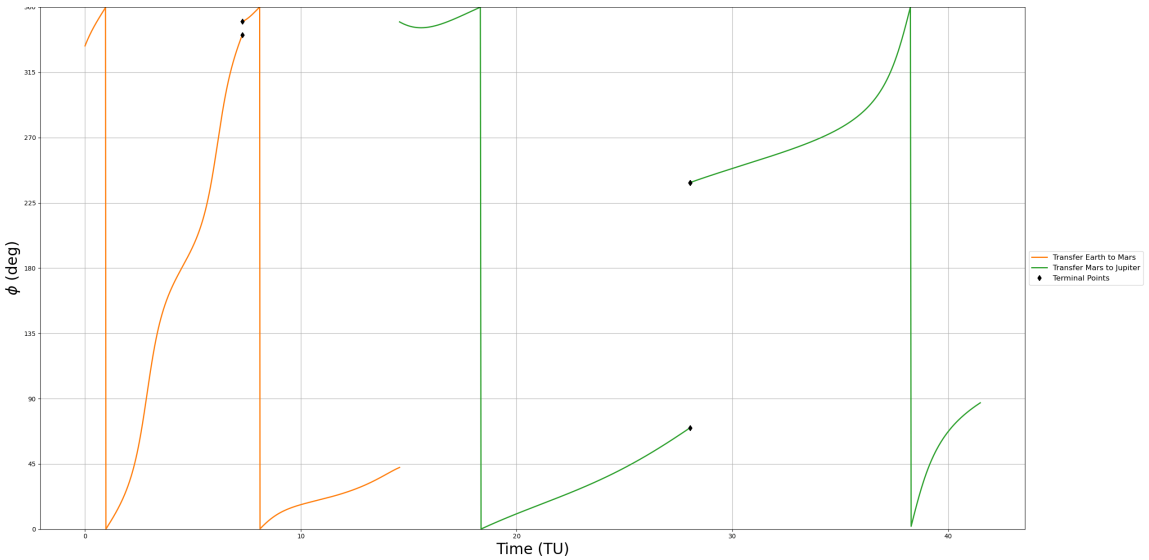

Figure B.209: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Thrust

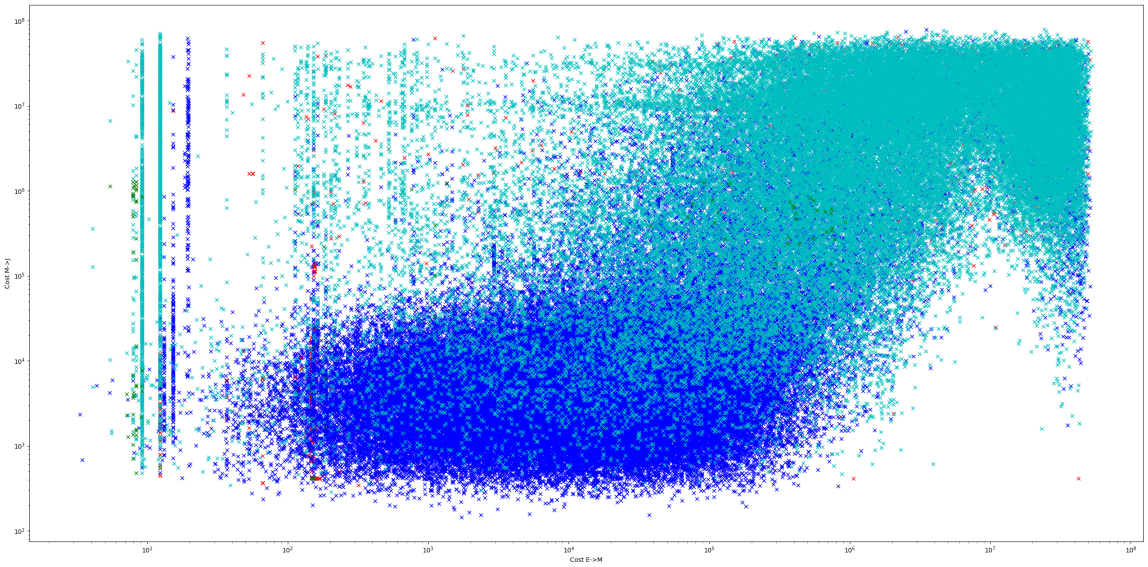

Figure B.210: Best Leg Tracking with Cluster Pruning Indirect Method Optimal Solution 5 Leg Tracking

Appendix C

MGALT-STOPS DEMO

This Appendix covers the demo mission for MGALT-STOpS in Python. To access the demo, visit 'STOpS_Demo.py'. In it, the user will see the mission described in the User Guide (Appendix A) detailed with the information from the 'README.txt' file. The user is free to change whatever parameters they desire, but they should first run the demo as-is until they are familiar with interacting with the suite.

In order to run the demo, open 'main.py' and set the variable 'miss' in the function call equal to 'demo'. Upon running the main script, MGALT-STOpS will run the demo. As a note, if the default parameters are kept, the suite will likely take multiple hours to run depending on the machine. For the machine used to write this work, it took approximately 3 hours.

The demo will produce 2 plots for the user, but they will not be saved to a file. It is up to the user to decide where and how they want to save the figures. Examples of what the plots might look like are shown in Figures C.1 and C.2.

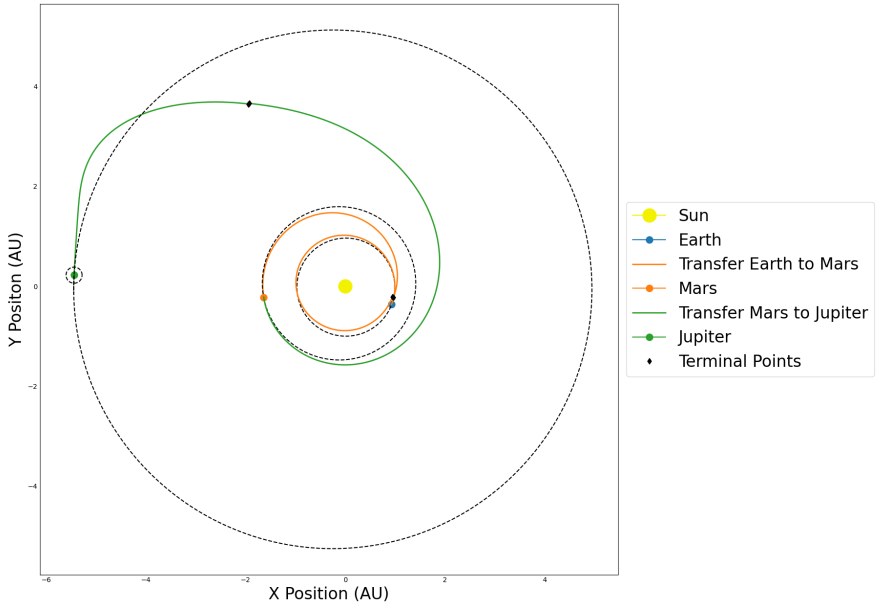

Figure C.1: Demo: Sample Trajectory with Indirect Method

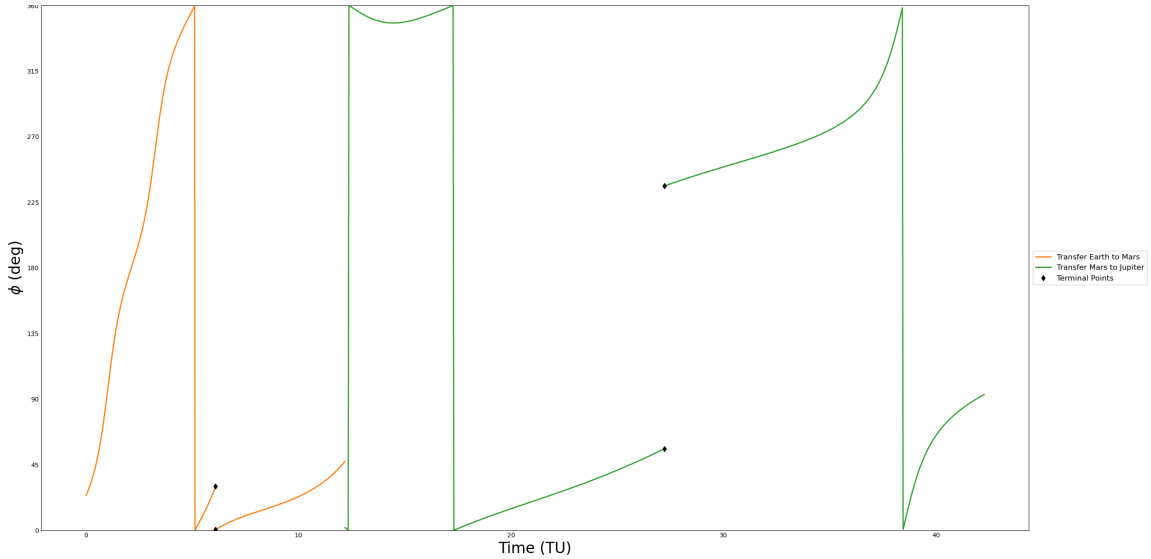

Figure C.2: Demo: Sample Thruster Profile with Indirect Method