Algorithms and Orders for Finding Noncommutative Gröbner Bases

by

Benjamin J. Keller

Dissertation submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science and Applications

APPROVED:

Lenwood S. Heath, Co-chairman	Edward L. Green, Co-chairman
Donald C.S. Allison	Daniel R. Farkas
Michael A. Keenan	Clifford A. Shaffer

March, 1997

Blacksburg, Virginia

Algorithms and Orders for Finding Noncommutative Gröbner Bases

by

Benjamin J. Keller

Committee Co-chairmen:

Lenwood S. Heath

Computer Science

and

Edward L. Green

Mathematics

(ABSTRACT)

The problem of choosing efficient algorithms and good admissible orders for computing Gröbner bases in noncommutative algebras is considered. Gröbner bases are an important tool that make many problems in polynomial algebra computationally tractable. However, the computation of Gröbner bases is expensive, and in noncommutative algebras is not guaranteed to terminate. The algorithm, together with the order used to determine the leading term of each polynomial, are known to affect the cost of the computation, and are the focus of this thesis.

A Gröbner basis is a set of polynomials computed, using Buchberger's algorithm, from another set of polynomials. The noncommutative form of Buchberger's algorithm repeatedly constructs a new polynomial from a *triple*, which is a pair of polynomials whose leading terms overlap and form a nontrivial common multiple. The algorithm leaves a number of details underspecified, and can be altered to improve its behavior. A significant improvement is the development of a dynamic dictionary matching approach that efficiently solves the pattern matching problems of noncommutative Gröbner basis computations. Three algorithmic alternatives are considered: the strategy for selecting triples (*selection*), the strategy for removing triples from consideration (*triple elimination*), and the approach to keeping the set interreduced (*set reduction*).

Experiments show that the selection strategy is generally more significant than the other techniques, with the best strategy being the one that chooses the triple with the shortest common multiple. The best triple elimination strategy ignoring resource constraints is the Gebauer-Möller

strategy. However, another strategy is defined that can perform as well as the Gebauer-Möller strategy in less space.

The experiments also show that the *admissible order* used to determine the leading term of a polynomial is more significant than the algorithm. Experiments indicate that the choice of order is dependent on the input set of polynomials, but also suggest that the length lexicographic order is a good choice for many problems. A more practical approach to chosing an order may be to develop heuristics that attempt to find an order that minimizes the number of overlaps considered during the computation.

ACKNOWLEDGEMENTS

There are several people who have made it possible for me to finish my Ph.D. First, are my advisors Dr. Ed Green who introduced me to my research problem, and Dr. Lenny Heath who has helped me explore it and otherwise had to endure my being across the hall. Second, are my committee consisting of Dr. Allison, Dr. Farkas, Dr. Keenan and Dr. Shaffer who have put up with the conditions under which I have completed my degree. Special thanks go to Drs. Allison and Shaffer for suspending their disbelief about the relevance of my research topic, and to Dr. Keenan for driving eight hours to be at my defense.

Over the years I have received financial support from the Department of Computer Science and the Systems Research Center. People that I owe for these opportunities are Dr. Verna Schuetz and Dr. Dick Nance.

What is left of my sanity, I owe to the many friends I have made during my years at Virginia Tech including Siva Challa, Susan Keenan, Mani Mukherji, Greg Lavender and Ernie Page. Special thanks to Susan for listening to me during the roughest time of my Ph.D. studies; and to Greg for the lunches at the College Inn, for the trips to Reiters, and for sharing the secrets of finishing.

TABLE OF CONTENTS

1 Introduction **1**

1.1 Research Context 2

1.1.1 Related Problems 2

1.1.2 Noncommutative Algebras 4

1.1.3 Applications 4

1.2 Overview of Results 5

1.3 Organization 6

2 An Introduction to Noncommutative Gröbner Bases **8**

2.1 Two Algebraic Problems 9

2.1.1 Subspace Membership 9

2.1.2 Ideal Membership 11

2.2 Gröbner Bases 14

2.3 Computing Commutative Gröbner Bases 16

2.4 Computing Noncommutative Gröbner Bases 21

2.5 Decidability 22

2.6 Relationship to Rewriting 23

2.7 Path Algebras 25

2.8 Summary 27

3 Computing Noncommutative Gröbner Bases **29**

3.1 The Basic Algorithm 29

3.2 Termination 30

3.3 Algorithmic Alternatives 35
3.3.1 Selection Strategy 36
3.3.2 Polynomial Reduction 36
3.3.3 Set Reduction 37
3.3.4 Triple Elimination 40
3.4 Data Structures 43
3.4.1 Polynomials 43
3.4.2 Polynomial Sets 44
3.4.3 Triple Sets 44
3.5 Algorithmic Experimentation 45
3.5.1 A Prototype Implementation 46
3.5.2 Experiments 49
3.5.3 Implications for Implementation 60

4 Pattern Matching 63

4.1 Pattern Matching in the Gröbner Basis Computation 64
4.2 Related Problems 65
4.3 Suffix Trees and Dictionary Matching 66
4.4 Pattern Matching Solution 72
4.4.1 Superword Search 73
4.4.2 Left-overlap Search 73
4.4.3 Right-overlap Search 74
4.5 Summary 75

5 Admissible Orders 76

5.1 Related Work 77
5.2 Definition 77
5.3 A Class of Orders 80
5.4 Experimentation 81
5.4.1 Algorithm 82
5.4.2 Input Problems 82
5.4.3 Execution 82

5.4.4 Results 83
5.5 Analysis 83
5.6 Alphabetic Orders 86
5.7 Summary and Directions 90

6 Admissible Orders in Path Algebras **92**
6.1 Equivalence of Orders 93
6.2 Spanning Trees and Orders 95
6.3 Future Directions 102
6.3.1 Orders and Generators 102
6.3.2 Equivalence and Choosing a Good Order 103
6.3.3 Gröbner Walks 103

7 Conclusions **105**
7.1 Contributions 105
7.1.1 Algorithms 105
7.1.2 Orders 106
7.1.3 Implementation 107
7.2 Directions 107

A Useless Triple Elimination **113**

B Suffix Tree Insertion Algorithm **119**

C Problem Instances **125**
C.1 Free Algebras 125
C.1.1 A4 through A8 125
C.1.2 Control Theory Problems 127
C.2 Other Free Instances 127
C.3 Path Algebras 127
C.3.1 CGL and Derivatives 128
C.3.2 DCYC and ICYC 130
C.3.3 P5 132

C.3.4 Binary Tree Quivers . . . 132
C.3.5 M1 and Derivatives . . . 136
C.3.6 MS, MTB, MM . . . 138
C.4 Random Instances . . . 140
C.4.1 A51E and A51H . . . 140
C.4.2 AGS . . . 141
C.4.3 GL . . . 141

D Problem Instance Generation **142**
D.1 Graph generation . . . 142
D.2 Generating Set Generation . . . 142

E Experimental Results **152**
E.1 Algorithm Experiments . . . 152
E.1.1 Counts . . . 152
E.1.2 Times . . . 180
E.2 Order Experiments . . . 207

LIST OF FIGURES

2.1 Buchberger's Algorithm. 18

2.2 Confluence of Rewrite System. 24

2.3 Quiver for Free Algebra in a, b. 25

2.4 Quiver for Simple Path Algebra. 26

2.5 Two-Node Quiver for Uniform Projection Example. 26

3.1 Buchberger's Algorithm for Noncommutative Algebras. 31

3.2 Initialization for Buchberger's Algorithm. 31

3.3 Basis Reduction. 32

3.4 Simple UPDATE. 32

3.5 Buchberger's Algorithm for the General Case. 33

3.6 Buchberger's Algorithm for the Degree Homogeneous Case. 34

3.7 Standard Selection Algorithm. 37

3.8 Tip Reduction Algorithm. 38

3.9 Total Reduction Algorithm. 38

3.10 UPDATE Using Redundant Element Deletion. 39

3.11 UPDATE Algorithm Using Element Reduction. 40

3.12 Division of Common Multiple for Buchberger's Second Criterion. 41

3.13 Selection With Triple Elimination. 42

3.14 Gebauer-Möller Elimination. 42

3.15 Structure of the Prototype. 47

3.16 Generic Quadratic Relations for Free Algebra Instances. 50

3.17 Example Graph for Mesh Algebra. 50

4.1 Dictionary Suffix Tree. 67
4.2 Dictionary Suffix Tree with Suffix Links. 68
4.3 Construction of Suffix Tree for *cc*. 69
4.4 Extension of Suffix Tree for *cc* by Inserting *cab*. 69
4.5 Extension of Suffix Tree for $\{cc, cab\}$ by Inserting *baba* (Part One). 70
4.5 Extension of Suffix Tree for $\{cc, cab\}$ by Inserting *baba* (Part Two). 71

5.1 Quiver for A51 Problem Instance. 89

6.1 Graph for Uniform Equivalence Class Example. 96
6.2 Graph with Three Paths ab, cd, ef from u to v. 98
6.3 Spanning Tree with Path ef from u to v. 98
6.4 Graph for Spanning Tree Example. 99
6.5 Example Family of Spanning Trees. 100
6.6 Graph for Inconsistency Example. 100
6.7 Inconsistent Family of Spanning Trees. 101

B.1 Suffix Tree Insertion Algorithm. 120
B.2 Algorithm for Insertion of a Single Suffix. 121
B.3 Scan Algorithm. 122
B.4 Rescan Algorithm. 124

C.1 Quiver for CGL, CGL1, and CG5. 128
C.2 Quiver for the DCYC Problem Instance. 131
C.3 Quiver for the ICYC Problem Instance. 131
C.4 Quiver for P5 Instance. 132
C.5 Quiver for BT7 instance. 133
C.6 Quiver for BT31 instance. 133
C.7 Quiver for M39 instance. 135
C.8 M1 Quiver. 136
C.9 Quiver of the MS Instance. 138
C.10 Quiver of the MTB Instance. 139
C.11 Quiver of the MM Instance. 139

C.12 Quiver for A51E Problem Instance. 140
C.13 Quiver for A51H Problem Instance. 140

D.1 Graph Generation Function. 143
D.2 Multigraph Generation Function 144
D.3 Acyclic Graph Generation 145
D.4 Coefficient Generation Function. 145
D.5 Function to Compute Set of Successors of Node. 146
D.6 Function to Compute Successors of Node in Graph. 146
D.7 Function to Generate Paths (Part One). 147
D.8 Function to Generate Paths (Part Two). 148
D.9 Function to Generate Polynomials. 149
D.10 Function to Generate Polynomial Sets (Part One). 150
D.11 Function to Generate Polynomial Sets (Part Two). 151

LIST OF TABLES

2.1 Trace for Commutative Gröbner Basis Example Computation (Part One). 19
2.1 Trace for Commutative Gröbner Basis Example Computation (Part Two). 20

3.1 Ranking of Algorithms by Counts for Problem A4 (Part One). 52
3.1 Ranking of Algorithms by Counts for Problem A4 (Part Two). 53
3.1 Ranking of Algorithms by Counts for Problem A4 (Part Three). 54
3.1 Ranking of Algorithms by Counts for Problem A4 (Part Four). 55
3.2 Average Rank of Algorithms for Free Algebra Instances. 55
3.3 Average Rank of Algorithms for Mesh Algebra Instances. 56
3.4 Average Rank of Algorithms by Counts for P5 Instance. 56
3.5 Average Rank of Algorithms by Time for Free Algebra Instances. 57
3.6 Average Rank of Algorithms by Time for Mesh Algebra Instances. 57
3.7 Average Rank of Algorithms by Time for P5 Instance. 58
3.8 Observations for Comparison of Set Reduction Techniques. 58
3.9 Counts for Comparison of Elimination Strategies for Problem BT31. 60
3.10 Counts for Comparison of Elimination Strategies for Problem BT7. 60
3.11 Counts for Comparison of Elimination Strategies For Problem DCYC. 60
3.12 Counts for Comparison of Elimination Strategies For Problem ELP. 61
3.13 Counts for Comparison of Elimination Strategies for Problem ICYC. 61
3.14 Counts for Comparison of Elimination Strategies For Problem A4. 61
3.15 Counts for Comparison of Elimination Strategies for Problem P5. 61

5.1 Ranking of Admissible Orders for Instance MM. 84

5.2 95% Confidence Intervals for Differences between Admissible Orders for Instance MM. 85
5.3 Ranking of Alphabetic Orders for Instance MM. 85
5.4 95% Confidence Intervals for Differences between Alphabetic Orders for Instance MM. 86
5.5 Ranking of Orders for Individual Free Problems. 86
5.6 Ranking of Orders for Individual Mesh Algebra Problems. 87
5.6 Ranking of Orders for Individual Mesh Algebra Problems (cont.). 87
5.7 Ranking of Orders for Individual Random Problems. 87
5.8 Ranking of Orders for Individual Path Algebra Problems. 88
5.9 Ranking of Admissible Orders. 88
5.10 Counts for Classes of Alphabetic Orders on Problem A51. 90

E.1 Counts for Problem A4 Instance (Part One). 153
E.1 Counts for Problem A4 Instance (Part Two). 154
E.1 Counts for Problem A4 Instance (Part Three). 155
E.2 Counts for Problem A5 (Part One). 156
E.2 Counts for Problem A5 (Part Two). 157
E.2 Counts for Problem A5 (Part Three). 158
E.3 Counts for Problem A6 (Part One). 159
E.3 Counts for Problem A6 (Part Two). 160
E.3 Counts for Problem A6 (Part Three). 161
E.4 Counts for Problem A7 (Part One). 162
E.4 Counts for Problem A7 (Part Two). 163
E.4 Counts for Problem A7 (Part Three). 164
E.5 Counts for Problem A8 (Part One). 165
E.5 Counts for Problem A8 (Part Two). 166
E.5 Counts for Problem A8 (Part Three). 167
E.6 Counts for Problem BT7 (Part One). 168
E.6 Counts for Problem BT7 (Part Two). 169
E.6 Counts for Problem BT7 (Part Three). 170
E.7 Counts for Problem BT31 (Part One). 171
E.7 Counts for Problem BT31 (Part Two). 172
E.7 Counts for Problem BT31 (Part Two). 173

E.8 Counts for Problem M39 (Part One). 174
E.8 Counts for Problem M39 (Part Two). 175
E.8 Counts for Problem M39 (Part Three). 176
E.9 Counts for Problem P5 (Part One). 177
E.9 Counts for Problem P5 (Part Two). 178
E.9 Counts for Problem P5 (Part Three). 179
E.10 Times for Problem A4 (Part One). 180
E.10 Times for Problem A4 (Part Two). 181
E.10 Times for Problem A4 (Part Three). 182
E.11 Times for Problem A5 (Part One). 183
E.11 Times for Problem A5 (Part Two). 184
E.11 Times for Problem A5 (Part Three). 185
E.12 Times for Problem A6 (Part One). 186
E.12 Times for Problem A6 (Part Two). 187
E.12 Times for Problem A6 (Part Three). 188
E.13 Times for Problem A7 (Part One). 189
E.13 Times for Problem A7 (Part Two). 190
E.13 Times for Problem A7 (Part Three). 191
E.14 Times for Problem A8 (Part One). 192
E.14 Times for Problem A8 (Part Two). 193
E.14 Times for Problem A8 (Part Three). 194
E.15 Times for Problem BT31 (Part One). 195
E.15 Times for Problem BT31 (Part Two). 196
E.15 Times for Problem BT31 (Part Three). 197
E.16 Times for Problem BT7 (Part One). 198
E.16 Times for Problem BT7 (Part Two). 199
E.16 Times for Problem BT7 (Part Three). 200
E.17 Times for Problem M39 (Part One). 201
E.17 Times for Problem M39 (Part Two). 202
E.17 Times for Problem M39 (Part Three). 203
E.18 Times for Problem P5 (Part One). 204

E.18 Times for Problem P5 (Part Two). 205
E.18 Times for Problem P5 (Part Three). 206
E.19 Counts for Admissible Order Comparison with Problem A4. 208
E.20 Counts for Admissible Order Comparison with Problem A5. 209
E.21 Counts for Admissible Order Comparison with Problem A51E. 210
E.22 Counts for Admissible Order Comparison with Problem A51H. 211
E.23 Counts for Admissible Order Comparison with Problem A6. 212
E.24 Counts for Admissible Order Comparison with Problem AGS. 213
E.25 Counts for Admissible Order Comparison with Problem BT7. 214
E.26 Counts for Admissible Order Comparison with Problem CG5. 215
E.27 Counts for Admissible Order Comparison with Problem CGL. 216
E.28 Counts for Admissible Order Comparison with Problem CGL1. 217
E.29 Counts for Admissible Order Comparison with Problem DCYC. 218
E.30 Counts for Admissible Order Comparison with Problem ELP. 219
E.31 Counts for Admissible Order Comparison with Problem HWEB. 220
E.32 Counts for Admissible Order Comparison with Problem HWRES. 221
E.33 Counts for Admissible Order Comparison with Problem ICYC. 222
E.34 Counts for Admissible Order Comparison with Problem M1. 223
E.35 Counts for Admissible Order Comparison with Problem MBFS. 224
E.36 Counts for Admissible Order Comparison with Problem MDFS. 225
E.37 Counts for Admissible Order Comparison with Problem MM. 226
E.38 Counts for Admissible Order Comparison with Problem MS. 227
E.39 Counts for Admissible Order Comparison with Problem MT1. 228
E.40 Counts for Admissible Order Comparison with Problem MT2. 229
E.41 Counts for Admissible Order Comparison with Problem MT3. 230
E.42 Counts for Admissible Order Comparison with Problem MT4. 231
E.43 Counts for Admissible Order Comparison with Problem MTB. 232
E.44 Counts for Admissible Order Comparison with Problem MTRI. 233
E.45 Counts for Admissible Order Comparison with Problem P4. 234
E.46 Counts for Admissible Order Comparison with Problem P5. 235
E.47 Counts for Admissible Order Comparison with Problem P6. 236

Chapter 1

Introduction

A Gröbner basis is a set of polynomials with a property that ensures that unique normal forms of other polynomials can be found as the remainder of dividing by the Gröbner basis elements. Gröbner bases are important because they make many problems in polynomial algebra computationally tractable. Unfortunately, the computation of Gröbner basis itself can be very expensive. This expense is compounded in noncommutative algebras by the fact that Gröbner bases may be infinite, and so a Gröbner basis computation may not terminate. Despite this extra difficulty, the goal of this research is to develop techniques that produce a noncommutative Gröbner basis as efficiently as possible.

We approach this task experimentally and concentrate on alternatives of two factors that affect the efficiency of Gröbner basis computations. The first factor is algorithmic. For commutative Gröbner basis computations, progress has been made toward improving the efficiency through algorithms that eliminate unnecessary work. However, for Gröbner bases in noncommutative algebras, all that is known (or speculated) is that the commutative techniques should still apply (see the survey by Mora [46]). The algorithms considered in this research are a mix of new algorithms and adaptations of algorithms used in the commutative case. We successfully identify a configuration of alternative algorithms that computes noncommutative Gröbner bases more efficiently.

The second factor is the choice of order used to determine the leading terms of polynomials. These orders are called *admissible orders* and are special well-orders on the terms of the polynomials (see Chapter 2 for the definition). Admissible orders help determine the Gröbner bases for a given input and are known to significantly affect the computation. In the commutative case, an optimal order is known, but in the noncommutative case there is little information about the effect of an order on the efficiency of computing Gröbner bases. We experimentally compare a small class of admissible orders and develop a ranking based on a limited number of problems. However, what the experiments indicate is that the choice of order is highly problem dependent and that a simple ranking is not really valid.

The remainder of this chapter describes the relationship of this research to other work in symbolic computation in Section 1.1, summarizes the results in Section 1.2, and outlines the organization of the thesis in Section 1.3. Readers not familiar with noncommutative algebras and Gröbner bases may want to read Chapter 2 before reading the remainder of the thesis.

1.1 Research Context

The focus of this research is the efficient computation of Gröbner bases in noncommutative algebras. How this research fits into existing work is briefly reviewed in this section. The first subsection describes how the noncommutative Gröbner basis computation relates to other problems in symbolic computation. The second subsection relates the noncommutative algebras considered here to other noncommutative algebras for which Gröbner bases have been studied. Finally, the third subsection describes some of the known applications of noncommutative Gröbner bases.

This section is meant to be an overview for readers who are at least vaguely familiar with Gröbner bases, noncommutative algebras, and symbolic computation. Other literature relevant to the goals of this research is discussed later as appropriate.

1.1.1 Related Problems

The problem of computing a Gröbner basis is a special case of the problem of finding a convergent term rewriting system. Term rewriting is used primarily to test equivalence of terms in a universal algebra [55] using a set of rules that defines an equivalence relation on the terms. The goal in term rewriting is to have a system of rules that can be used to rewrite any term to a unique irreducible form regardless of which rules are applied and in what order. Such a system is *convergent* (a more formal definition is given in Chapter 2). Given an arbitrary rewriting system, the Knuth-Bendix completion algorithm can be used to find an equivalent convergent rewriting system [36]. The difficulty is that for any given rewrite system the equivalent convergent system may not be finite; so, in general, a Knuth-Bendix computation might not terminate.

If we view a set of polynomials together with polynomial division as a rewrite system, then a Gröbner basis is exactly a convergent system of rules. (See the work by Bündgen [14, 15] for more details.) However, a Gröbner basis is a tool for answering a different (but analogous) question, that of membership in ideals of polynomial rings. Here, instead of having a set of rules that represents (or generates) an equivalence relation, we have a set of polynomials that represents the ideal. A Gröbner basis of an ideal allows testing whether a polynomial is an element of the ideal by testing if the normal form is zero.

Polynomials are sums of a finite number of monomials that consist of a nonzero coefficient and a term. In commutative polynomial rings, the terms are elements of an abelian monoid, which means that the order of the indeterminates in a term is not important. Hironaka [32] first defined *standard* bases, which are closely related to Gröbner bases; however, Buchberger first defined Gröbner bases (for commutative polynomial rings) and the algorithm for computing them [11]. Mishra and Yap [43] discuss the relationship between standard and Gröbner bases. The algorithm takes as input a set of polynomials and adds new polynomials to create a new set for which all polynomial reductions converge. In the commutative case, a finite Gröbner basis always exists, and Buchberger's algorithm always terminates.

In noncommutative algebra, the terms are words in a noncommutative monoid (or in our case, a semigroup). Noncommutative algebras are quotients of *free (associative) algebras*, where the terms are elements of a free monoid over the alphabet of indeterminates. Bergman [8] first defines the concept of Gröbner bases for free algebras, but the algorithm is due to Mora [45]. The algorithm for the noncommutative case is nearly identical to the one for the commutative case with the primary difference being how the new elements are formed. However, for noncommutative algebras, Gröbner bases are not guaranteed to be finite. In fact, the algorithm terminates only when there is a finite Gröbner basis for the input set and admissible order.

The noncommutative case actually subsumes the problem of completion of string rewriting systems. String rewriting is a specific case of term rewriting where the terms are words from a free monoid rather than from an arbitrary universal algebra. The rules of a string rewriting system correspond to binomials in a noncommutative polynomial ring with some restrictions on the coefficient field. Therefore, the Knuth-Bendix completion algorithm for string rewriting is basically the same as the noncommutative form of Buchberger's algorithm.

1.1.2 Noncommutative Algebras

Our problem is actually that of computing Gröbner bases for ideals in path algebras (as presented by Farkas, Feustel, and Green [20]). Path algebras are quotients of free algebras that can be conveniently described in terms of a graph. If K is a field, a path algebra $K\Gamma$ consists of K-linear combinations of finite paths in a directed multigraph Γ (called the *quiver*). To compute Gröbner bases in path algebras the input generators must be given in a particular form; otherwise, the algorithms for path algebras and free algebras are the same. Since free algebras can be presented as path algebras, all finitely generated noncommutative algebras can be obtained by forming a quotient of some path algebra. Despite the fact that the algorithms are identical, using path algebras where possible does appear to have a number of advantages (not the least being that the quotient relations need not be part of the input, since they are implicit in the graph).

Gröbner bases in other noncommutative algebras occur in the literature. However, most are "almost-commutative" polynomial rings where the relationship between products of indeterminates like ab is not ba but some other expression of a and b. Examples of these algebras are Weyl algebras [21], enveloping algebras of Lie algebras [3], algebras of solvable type [35], Grassman algebras [26, 53], and Clifford algebras [26]. Strictly speaking these algebras are noncommutative, but all have properties that imply that all ideals have finite Gröbner bases. Problem instances in these algebras probably can be solved more directly than by presenting them as quotients of path algebras. Because of this, these cases are not considered here.

The Gröbner basis theory for rings whose terms are elements of monoids presented by string rewriting systems is developed by Reinert [48]. Path algebras can be described in this setting as monoid rings with zero divisors. The rings with many objects defined by Mitchell [44] are very similar to path algebras, and the Gröbner basis theory likely extends to these algebras.

1.1.3 Applications

Noncommutative Gröbner bases were developed as a tool for algebraic research, and therefore most applications are in algebra. However, there are also other more "real-world" applications. One such application is the simplification of polynomial equations that arise in operator theory and linear control theory [29, 30, 31]. In essence, a Gröbner basis is found for a set of equations that express the basic assumptions of the theory, and so can be used to simplify other equations so that they can be solved by other means. Algebraic applications include finding more information about the quotient algebra $K\Gamma/I$ when we have a Gröbner basis for I.

Other applications are expected to develop. One area being explored is in performing computations in quantum physics where computations are for the most part done by hand, but use of Gröbner bases and other techniques may help. Noncommutativity is also common in computation, and at times abstract algebraic structures correspond to underlying computational models. Two examples are the use of categories of Hopf algebras as models of linear logic [9], and the use of polynomials to express polymorphic type systems [34]. Whether these particular areas will provide important applications for noncommutative Gröbner bases is not clear, but they do provide evidence of places where noncommutative algebras occur and Gröbner bases might serve some useful purpose.

1.2 Overview of Results

This research was done in two phases, with algorithms compared first, and orders compared second. The algorithms were compared using a family of prototype systems that implement various combinations of algorithmic alternatives. The results of the experiments with the algorithms were used to guide the construction of the Opal system which was used to compare the admissible orders.

In the algorithmic experiments, three algorithmic alternatives were considered: the strategy by which pairs of polynomials and their overlaps are selected (*selection*), the strategy by which pairs of polynomials and overlaps are removed from consideration (*triple elimination*), and the manner in which the set is kept interreduced (*set reduction*). The experiments show that the choice of selection strategy is generally more significant than the other techniques, although the triple elimination strategy is also important. The selection strategy that is generally best chooses a pair of polynomials together with an overlap that has the shortest common multiple of their leading terms among all possible overlapping polynomials. Ignoring space and time constraints, the best triple elimination strategy is the noncommutative version of the Gebauer-Möller strategy that removes pairs as soon as possible. However, a more practical algorithm combines the Gebauer-Möller strategy with one of Buchberger and can be as good for some inputs without the added space requirement. The Opal system was written to use this hybrid approach to triple elimination, but allows different selection and set reduction strategies to be used.

Both Opal and the prototype systems use a dictionary matching approach adapted to the pattern matching problems of the Gröbner basis computation. This pattern matching approach allows fast changes to the dictionary, which is significant to the efficient implementation of the basic algorithm for noncommutative Gröbner bases. The approach can also be used for completion in string rewriting where a static form of the approach has been used, but the dynamic version has yet to be applied.

The problem of choosing an *admissible order* was also considered experimentally. The Opal system was used (with shortest selection) to compare seven orders: length lexicographic, left vector lexicographic, right vector lexicographic, length left vector lexicographic, length right vector lexicographic, length reverse left vector lexicographic, and length reverse right vector lexicographic. The experiments show that which order is best is highly problem dependent, and that it really is not possible to define a general ranking. Despite this difficulty, a pragmatic ranking is given. In general, the length lexicographic order appears to be the best order to try first (which is supported by string-rewriting folklore). More practical, however, is the observation that the best order minimizes the number of overlaps during the computation. This observation should help identify heuristics that can be used to choose the best order for a given problem.

Other approaches for characterizing and finding the best choice of orders were also considered. These results are incomplete, but suggest promising directions for future work.

1.3 Organization

The remainder of the thesis is organized as follows. Chapter 2 is a tutorial that introduces noncommutative Gröbner bases and other key concepts used. In Chapter 3 we describe the algorithm, its variants and experimentation to compare the variants. Chapter 4 develops a pattern matching approach used in the implementation of the algorithm. Chapter 5 investigates admissible orders and experimentation with the orders. Chapter 6 considers the relationship of orders and the problem instances in path algebras. Chapter 7 contains conclusions and a list of open problems.

There are several appendices that provide supplementary information for the earlier chapters. Appendix A contains a proof of Buchberger's Second Criteria for noncommutative algebras and supplements Chapter 3, where the result is used. Appendix B gives details of the insertion algorithm for suffix trees described in Chapter 4. Appendix C is a list of problem instances used in the experimentation, and Appendix D describes the algorithms used to generate random problem instances.

Finally, Appendix E contains the tables of raw data from the experimentation.

Chapter 2

An Introduction to Noncommutative Gröbner Bases

This Chapter introduces noncommutative Gröbner bases and their computation. Since most readers are likely not familiar with noncommutative algebras, Gröbner bases are first developed in the more familiar setting of commutative polynomial rings, and then defined for noncommutative algebras. Along the way, an analogy is drawn between computing Gröbner bases and performing Gaussian elimination in linear algebra. Later, the relationship with term rewriting is discussed. All three computations are strongly related.

The Chapter begins in Section 2.1 by developing an analogy between a problem in linear algebra solved by Gaussian elimination and another in polynomial algebra for which Gröbner bases provide a solution. Gröbner bases are defined in Section 2.2, the algorithm to compute them is introduced in Section 2.3, and the noncommutative case is considered in Section 2.4. Decidability issues are discussed in Section 2.5, the relationship between Gröbner bases and term rewriting is discussed in Section 2.6. Finally, the Chapter ends by defining path algebras, the form of noncommutative algebra used in the remaining Chapters.

2.1 Two Algebraic Problems

To introduce Gröbner bases, we consider two analogous problems, one from linear algebra and the other from polynomial algebra. In linear algebra, the problem is that of determining whether a vector is a member of a given subspace. The solution to the corresponding problem in polynomial algebra leads to Gröbner bases. For simplicity, we fix $\mathbb{Q}$, the field of rational numbers, as the set of scalars.

2.1.1 Subspace Membership

Recall that a subspace U of a vector space V is a subset of V that is also a vector space (U is closed under addition and scalar multiplication). An example of a subspace in $\mathbb{Q}^3$ (3-space over the rational numbers) is the subset $\{(0, 0, q) : q \in \mathbb{Q}\}$. A *generating system* S for a vector space V is a subset such that all elements of V can be expressed as linear combinations of elements of S. A generating system consisting of linearly independent elements is a *basis*. For example, the set $\{(2, 3, 1), (5, 1, 7), (6, 2, 3), (9, 7, 9)\}$ is a generating system for $\mathbb{Q}^3$ (but not a basis since $(9, 7, 9) = 2 \cdot (2, 3, 1) + (5, 1, 7)$), and $\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ is a basis for $\mathbb{Q}^3$.

Consider the problem of deciding whether an arbitrary vector is in a particular subspace of $\mathbb{Q}^3$. Stated precisely, it is the Subspace Membership problem.

Problem 2.1 (Subspace Membership) *Let S be a finite subset of the vector space $\mathbb{Q}^3$, and let U be the subspace generated by S. Given a vector $\mathbf{v}$ in V, decide whether $\mathbf{v}$ is an element of U, that is, whether $\mathbf{v}$ is a linear combination of elements of S.*

Let S be the set $\{(1, 3, 0), (0, 2, 4), (1, 5, 4), (1, 1, -4)\}$, and let U be the subspace generated by S. To show that a vector $\mathbf{v} \in \mathbb{Q}^3$ is in U, we need to find a linear combination of elements in S that is equal to $\mathbf{v}$. For example, the vector $(13, 33, -12)$ is an element of U because it is equal to the linear combination

$$5 \cdot (1, 3, 0) + 3 \cdot (0, 2, 4) + (1, 5, 4) + 7 \cdot (1, 1, -4).$$

In general, a vector $\mathbf{v}$ is an element of U if there is an indexed set of scalars $\{k_{\mathbf{g}} : \mathbf{g} \in G\}$ such that $\mathbf{v} - \sum_{\mathbf{g} \in G} k_{\mathbf{g}} \cdot \mathbf{g} = \mathbf{0}$. Let A be the matrix formed by taking the vectors in S as columns. Finding the scalars $k_{\mathbf{g}}$ is equivalent to solving the equation $A\mathbf{x} = \mathbf{v}$. For example, the membership of $\mathbf{v} = (13, 33, -12)$ in U can be demonstrated by solving the system

$$\left[\begin{array}{rrrr|r} 1 & 0 & 1 & 1 & 13 \\ 3 & 2 & 5 & 1 & 33 \\ 0 & 4 & 4 & -4 & -12 \end{array}\right].$$

To solve the system, it is first reduced to row echelon form using Gaussian elimination. Then, if there is a row in the reduced form with the first 4 entries zero and the last entry nonzero, then $\mathbf{v}$ is *not* in U.

Gaussian elimination consists of a series of row reductions. Row reduction is a process analogous to dividing one row by another to eliminate the leading nonzero entry. The reduced row is replaced with the remainder. In the matrix above, the row reduction that subtracts 3 times the first row from the second row yields the new (reduced) matrix

$$\left[\begin{array}{rrrr|r} 1 & 0 & 1 & 1 & 13 \\ 0 & 2 & 2 & -2 & -6 \\ 0 & 4 & 4 & -4 & -12 \end{array}\right].$$

Since only the first row now has a nonzero first component, the next step is to reduce the third row using the second. The matrix that results from Gaussian elimination on this example is

$$\left[\begin{array}{rrrr|r} 1 & 0 & 1 & 1 & 13 \\ 0 & 1 & 1 & 1 & -3 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right].$$

According to the test described, the vector $(13, 33, -12)$ is linearly dependent on the elements of S and so is an element of the subspace U.

The test for subspace membership (or, equivalently, linear dependence) does not require exhibiting a linear combination but merely proving the existence of one. Since the example generating set S is not linearly independent, there are an infinite number of linear combinations for $(13, 33, -12)$. If a unique linear combination is required, then a basis should be used instead of an arbitrary generating system.

A basis for U is $\{(1, 3, 0), (0, 1, 2)\}$ (found by performing Gaussian elimination on S). Repeating the test for $(13, 33, -12)$, the augmented matrix using the basis instead of the generating system is

$$\left[\begin{array}{cc|c} 1 & 0 & 13 \\ 3 & 1 & 33 \\ 0 & 2 & -12 \end{array}\right].$$

When Gaussian elimination is applied to the matrix, the reduced form

$$\left[\begin{array}{cc|c} 1 & 0 & 13 \\ 0 & 1 & -6 \\ 0 & 0 & 0 \end{array}\right]$$

again shows that $(13, 33, -12)$ is an element of the subspace U. However, the reduced matrix also yields the unique linear combination $13 \cdot (1, 3, 0) - 6 \cdot (0, 1, 2)$ for $(13, 33, -12)$.

2.1.2 Ideal Membership

An analogous problem to subspace membership occurs in rings. A *ring* is a set R with two operations addition $+$ and multiplication $\cdot$, and a zero element 0 such that

1. R with addition is an Abelian group with identity 0;
2. multiplication is associative; and
3. multiplication distributes with addition [7, p.19].

In this thesis, all rings also have a *unit*, which is the identity for multiplication.

A polynomial ring has elements that are polynomials. If the order of the variables in the terms is not significant then the polynomial ring is *commutative*, and the terms are usually written so that multiple occurrences of a variable are combined as a power (so $xyxyzz$ is written $x^2y^2z^2$). A polynomial ring for which the order of the variables in the terms is significant (for example, $xyxyzz$ and $xxyyzz$ are different) is called *noncommutative*. Noncommutative rings are discussed further in Section 2.4. In what follows commutative variables are written as capital letters, and noncommutative variables are written in lower case letters.

An example of a commutative polynomial ring is $\mathbb{Q}[X, Y, Z]$, which is the set of polynomials with rational coefficients and terms that are products of powers of the variables X, Y and Z. For example, $\frac{2}{5}X^2Y^2Z^2 + \frac{9}{2}XY - 2XZ$ is a polynomial where the term $X^2Y^2Z^2$ has coefficient $2/5$, the term XY has coefficient $9/2$, and the term XZ has coefficient -2. Note that $\mathbb{Q}[X, Y, Z]$, like $\mathbb{Q}^3$, is a vector space over the rationals, and at first glance it may appear that they are essentially the same. However, $\mathbb{Q}[X, Y, Z]$ has infinite dimension, while $\mathbb{Q}^3$ has dimension 3.

A subspace in a vector space corresponds to an ideal in a ring. A *(two-sided) ideal* is a subset I of a ring $\mathbb{Q}[X_1, \ldots, X_n]$ that is closed under addition and satisfies the property that if p and q are polynomials and g is a member of I, then pgq, the product of p, g, and q, is in I. In the polynomial ring $\mathbb{Q}[X, Y, Z]$, examples of ideals include the sets $\{0\}$, $\mathbb{Q}[X, Y, Z]$ itself, and $\{fZ^2g : f, g \in \mathbb{Q}[X, Y, Z]\}$. A *generating set* of polynomials P for an ideal I is a subset of $\mathbb{Q}[X, Y, Z]$ such that all elements of I can be expressed as combinations of elements of P:

$$I = \left\{ \sum_{g \in P} p_g g q_g : p_g, q_g \in \mathbb{Q}[X, Y, Z] \right\};$$

where all but a finite number of $p_g g q_g = 0$. The ideal I generated by a subset P of $\mathbb{Q}[X, Y, Z]$ is denoted $\langle P \rangle$. While, in general, we deal with two-sided ideals, in commutative rings two-sided ideals are the same as ideals that are closed by multiplying by ring elements only on the left or on the right.

The ideals given above can be written as $\left\langle \{Z^2\} \right\rangle$ for $\{fZ^2g : f, g \in \mathbb{Q}[X, Y, Z]\}$, and $\langle \{1, X, Y, Z\} \rangle$ for $\mathbb{Q}[X, Y, Z]$ (or just $\langle \{1\} \rangle$). No direct analogue to a vector subspace basis exists for an ideal. As we shall see, Gröbner bases, generating sets with particular properties, play a similar role.

The analogue of the subspace membership problem in polynomial rings is the problem of deciding whether an element of $\mathbb{Q}[X, Y, Z]$ is in a particular ideal of $\mathbb{Q}[X, Y, Z]$.

Problem 2.2 (Ideal Membership) *Let P be a finite subset of the ring $\mathbb{Q}[X, Y, Z]$. Given a polynomial f in $\mathbb{Q}[X, Y, Z]$, decide whether $f \in \langle P \rangle$.*

Let P be the set $\left\{XY^2Z - Z, XY - 1\right\}$, and let $g_1 = XY^2Z - Z$ and $g_2 = XY - 1$. To show that a polynomial f from $\mathbb{Q}[X, Y, Z]$ is in $\langle P \rangle$, we need to find polynomials $p_{g_1}, p_{g_2} \in \mathbb{Q}[X, Y, Z]$ such that

$$f = p_{g_1}(XY^2Z - Z) + p_{g_2}(XY - 1).$$

For example, the polynomial $f = X^2Y^2Z + X^2YZ + XY^2Z - XYZ - 2XZ$ is an element of I because choosing $p_{g_1} = X + 1$ and $p_{g_2} = XZ - Z$ yields

$$f = (X + 1)(XY^2Z - Z) + (XZ - Z)(XY - 1).$$

The polynomials $p_{g_1} = X + 1$ and $p_{g_2} = XZ - Z$ constitute evidence that f is in the ideal. (In general, testing membership in a two-sided ideal requires finding multipliers on both sides, but this is not necessary in commutative rings.)

So the condition for membership in an ideal $I = \langle P \rangle$ is analogous to that of membership in a subspace: a polynomial f is an element of I if there is an indexed set of polynomials $\{p_g, q_g \in \mathbb{Q}[X,Y,Z] : g \in P\}$ such that $f - \sum_{g \in P} p_g g q_g = 0$. While this equation is reminiscent of the one for subspace membership, here the "scalars" are polynomials. Therefore, the relationship between f and the generators is nonlinear, and so the simple test using Gaussian elimination is not possible.

A more direct approach to solving $f - \sum_{g \in P} p_g g q_g = 0$ is to use an operation based on division similar to row reduction in Gaussian elimination. Consider the example of determining whether $f = X^2Y^2Z + X^2YZ + XY^2Z - XYZ - 2XZ$ is an element of I. Dividing f by $XY^2Z - Z$ gives $f - X(XY^2Z - Z) = X^2YZ + XY^2Z - XYZ - XZ$. Dividing again gives

$$(f - X(XY^2Z - Z)) - 1(XY^2Z - Z) = X^2YZ - XYZ - XZ - Z.$$

Further dividing the remainder by $XY - 1$ gives

$$((f - (X+1)(XY^2Z - Z)) - XZ(XY - 1)) + Z(XY - 1) = 0$$

or equivalently

$$f - ((X+1)(XY^2Z - Z) + (XZ - Z)(XY - 1)) = 0.$$

Here polynomial division gives the desired test for ideal membership: find the remainder r of dividing f by the generating set P; if r is zero, then $f \in \langle P \rangle$, otherwise $f \notin \langle P \rangle$.

The operation of computing the remainder of a single polynomial division is a *simple reduction*, and the operation of computing the remainder of a polynomial after dividing a polynomial by a set of polynomials is called *reduction*. The example above shows the reduction of $X^2Y^2Z + X^2YZ + XY^2Z - XYZ - 2XZ$ to 0 by $\{XY^2Z - Z, XY - 1\}$.

In a vector space, testing subspace membership using a generating system may not find a unique linear combination of generating vectors. Something similar but more serious happens in polynomial rings: there may not be a unique remainder of division by an arbitrary generating set. In the example, $X^2Y^2Z + X^2YZ + XY^2Z - XYZ - 2XZ$ reduces to 0 by first dividing by $XY^2Z - Z$ and then by $XY - 1$. However, if division is done by repeatedly using $XY - 1$ instead, then we obtain the remainder $-XZ - YZ$, which is not divisible by either element of the generating set and so is not further reducible.

Therefore, given an arbitrary generating set P for an ideal I, different reductions may yield different results. In fact, ideal membership is in general undecidable [35, 46]. However, for decidable instances (including all instances in commutative polynomial rings), if a finite generating set P is given, then it is possible to find another generating set G_P which generates the same ideal and for which reduction always yields a unique value. Such a generating set is called a Gröbner basis.

2.2 Gröbner Bases

A Gröbner basis for an ideal plays a similar role to that of a subspace basis. A basis for a subspace can be found from a generating system by using Gaussian elimination. To find the basis for the generating system $\{(1,3,0),(0,2,4),(1,5,4),(1,1,-4)\}$, Gaussian elimination is performed on the matrix

$$\begin{bmatrix} 1 & 3 & 0 \\ 0 & 2 & 4 \\ 1 & 5 & 4 \\ 1 & 1 & -4 \end{bmatrix}$$

to find the reduced row-echelon form of the matrix

$$\begin{bmatrix} 1 & 0 & -6 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

This final matrix gives the basis $\{(1,0,-6),(0,1,2)\}$.

Note that in this example, the leading (nonzero) component of each of the first two rows of the reduced matrix is distinct from that of the other rows. Therefore, the elements of the basis are independent and so do not divide one another. More importantly, for any linear combination of the elements of the generating system, the leading component is divisible by the leading component of one of the elements of the basis. This is the key property of a Gröbner basis.

The coordinate structure of a vector implies an order on the coordinates. For polynomials, an order on terms is needed to identify the *leading term* of a polynomial. In linear equations over the variables x, y, z, the lexical order on the variables is typically $z < y < x$, and so the terms with x

are the leading ones. More general polynomials have more complex terms like X^3, XY and XY^3, and these more general terms require explicit orders on terms.

An *admissible* order is a total order $<$ on the set of terms (variable strings) such that for terms s, t, v and w,

1. If $s < t$, then $vsw < vtw$; and

2. If v and w are not both empty, then $s < vsw$.

Admissible orders differ for commutative and noncommutative terms. Noncommutative terms are variable strings, but commutative terms are really equivalence classes of variable strings. Typically, a commutative term such as X^2Y^3Z is represented by a tuple $(2, 3, 1)$ where the entries correspond to the count of the variables (in the sequence defined by the lexical order). So admissible orders must be defined differently for commutative and noncommutative terms.

Most admissible orders are based on the lexicographic order. If we define a lexical order on the variables $z < y < x$, noncommutative terms can be lexicographically ordered such that the term s is less than the term t if at the first position where s and t differ the variable in s lexically precedes the variable in t (e.g. $xzy < xxy$). However, this order does not make sense for commutative terms since the order of the variables is not significant. Using the tuple representation mentioned above, the commutative terms can be lexicographically ordered such that the term s is less than the term t if at the first position where the tuples for s and t differ the count is less for s (e.g. $XYZ < X^2Y$ since $(1, 1, 1) < (2, 1, 0)$). The lexicographic order is admissible in commutative polynomial rings, but not admissible in noncommutative ones.

An example order admissible for both commutative and noncommutative terms is the length (or degree) lexicographic order. In this order, the term s is less than t ($s < t$) whenever the length of the string for s is shorter than the length of the string for t, or if they are the same length, then s is lexicographically less than t. For noncommutative terms this order gives $z < y < x < zz < zy < zx < yz < yy < yx < xz < xy < xx < zzz < \cdots$, and for commutative terms gives $Z < Y < X < Z^2 < YZ < Y^2 < XZ < XY < X^2 < Z^3 < \cdots$.

For each admissible order and for each polynomial f, we obtain a unique representation of f as a linear combination of terms written in decreasing order. For example, $f = X^2Y^2Z + X^2YZ + XY^2Z - XYZ - 2XZ$ is written in decreasing order according to the (commutative) length lexicographic order. The *tip* of a polynomial f is the maximal (leading) term in a polynomial f with respect to a particular admissible order. So for the length lexicographic order, the tip of the polynomial f above is $tip_<(f) = X^2Y^2Z$. The set $Tip_<(H)$ is the set of tips for the set of polynomials H. A Gröbner basis is defined as follows.

Definition 2.1 *Let I be an ideal of $\mathbb{Q}[x_1, \ldots, x_m]$. A generating set G of I is a* Gröbner basis for *I (and an admissible order $<$) if $\langle Tip_<(G)\rangle = \langle Tip_<(I)\rangle$.*

The choice of admissible order is significant since it determines the tip set of the ideal. So, for most ideals I, if two orders $<_1$ and $<_2$ disagree (meaning $Tip_{<_1}(I) \neq Tip_{<_2}(I)$) then the Gröbner bases of I with respect to $<_1$ and $<_2$ are different. There are ideals for which this is not true, an example is an ideal generated by a single generator such as $XY + YZ$ (in the noncommutative case the situation is slightly more complicated). A Gröbner basis always exists regardless of the admissible order (an ideal is its own trivial Gröbner basis). In commutative polynomial rings, finite Gröbner bases always exist (in fact, they are usually defined to be finite [7, p.207]), but in noncommutative polynomial rings one admissible order may yield an infinite Gröbner basis while another yields a finite Gröbner basis.

From the definition, a Gröbner basis G has the property that if t is the tip of a nonzero polynomial in $\langle G\rangle$, then the tip of some polynomial in G divides t. This property ensures that when a polynomial in an ideal is divided by a member of the corresponding Gröbner basis, the leading term of a nonzero remainder is divisible by some other member of the basis. Therefore, when testing ideal membership, reduction of a polynomial f by a Gröbner basis always converges to a remainder that is zero if and only if f is in $\langle G\rangle$. In the next section, we explain how to obtain a Gröbner basis and give an example.

2.3 Computing Commutative Gröbner Bases

The algorithm for computing Gröbner bases for ideals of commutative polynomial rings is due to Buchberger [13]. Buchberger's algorithm is based on the subtle fact that it is sufficient to complete the generating set with polynomials of a particular form. Each of these special polynomials arises as a combination of a pair of generators that is not divisible by the constituent generators because their leading terms cancel.

Consider using reduction to test whether $XY^2Z - Z$ is an element of $\langle P\rangle$ for $P = \{XY^2Z - Z, XY - 1\}$ (using the length lexicographic order). In this example, $XY^2Z - Z$ reduces to zero by itself, and reduces to $YZ - Z$ by $XY - 1$. The set P can be extended by adding $YZ - Z$ without affecting the generated ideal (since $YZ - Z$ is in the ideal). With the new generating set $P' = \{XY^2Z - Z, XY - 1, YZ - Z\}$, all possible reductions of $XY^2Z - Z$ find zero.

This new set of generators P' is still not a Gröbner basis. The polynomial $XZ - Z$ is the difference $Z(XY - 1) - X(YZ - Z)$ and so is an element of the ideal $\langle P' \rangle$. However, the tip XZ (with respect to the length lexicographic order) of $XZ - Z$ is not divisible by the tip of any of the generators (and so is not reducible to zero). The term XYZ is the least common multiple of the tips XY and YZ. Since the least common multiple occurs in both $Z(XY - 1)$ and $X(YZ - Z)$ of $Z(XY-1) - X(YZ-Z)$ the occurrences cancel each other [7, p.210]. A pair of polynomials (p, q) for which the leading terms can be canceled in this way is called a *critical pair*, and the corresponding polynomial (called the s-polynomial) is denoted $SPol(p, q)$. Formally, the s-polynomial of p and q is

$$SPol(p, q) = \frac{1}{LC(p)} s_q p - \frac{1}{LC(q)} s_p q,$$

where $s_q = lcm(tip(p), tip(q))/tip(q)$ and $s_p = lcm(tip(p), tip(q))/tip(p)$ ($LC(p)$ is the leading coefficient of p). (Note that in general, the coefficients of the leading terms must be canceled in the s-polynomial, but in our examples the leading coefficient is always one.)

Adding the nonzero reduced s-polynomials $SPol(p, q)$ for critical pairs to a basis clearly expands the set of terms divisible by some tip of a generator. Not so clear is the fact that adding the s-polynomials $SPol(p, q)$ that cannot be further reduced is sufficient to find a Gröbner basis. This fact is due to the following theorem by Buchberger.

Theorem 2.1 *If all the* s*-polynomials $SPol(p, q)$ for the critical pairs of G reduce to zero by G, then G is a Gröbner basis [7, p.211].*

This result suggests the basic forms of Buchberger's algorithm shown in Figure 2.1. The algorithm is underspecified and allows many alternative implementations. Also, many variations to the algorithm are described in the literature. These variations are the subject of Chapter 3 and are introduced there.

Note that the algorithm maintains the set of critical pairs of basis elements. The algorithm removes one of these critical pairs from the set each iteration and adds new pairs if the resulting polynomial does not reduce to zero. The algorithm terminates when the set of critical pairs is empty, at which point G is a Gröbner basis for P. Typically the algorithm removes the leading coefficient of each new basis element by multiplying by the inverse of the coefficient.

GRÖBNER(P). Buchberger's algorithm for commutative Gröbner bases.

INPUT: A set P of generators, admissible order $<$.
OUTPUT: A finite, totally reduced Gröbner basis G of $\langle P \rangle$ with respect to $<$.

```
G ← P
C ← {(f, g) critical pairs for G}
while C is not empty do
    Select and remove a critical pair (f, g) from C
    Form p = SPol(f, g)
    Reduce p by G, and let h be the result
    if h ≠ 0 then
        Add h to G
        Add all critical pairs of h with elements of G to C
return G
```

Figure 2.1: Buchberger's Algorithm.

As an example, consider the computation of a Gröbner basis for the set

$$P = \{XY^2Z - W, Y^2ZW, XY^2W - Z\}.$$

The computation is shown in Table 2.1. Initially, G is P, and C consists of the three pairs shown in the first row of Table 2.1. For each iteration, the first pair in the column for C is selected and the s-polynomial is formed. For the first iteration, the pair $(XY^2Z - W, Y^2ZW)$ is selected, and the corresponding s-polynomial is

$$\begin{aligned} p &= SPol(XY^2Z - W, Y^2ZW) \\ &= (XY^2Z - W)W - X(Y^2ZW) \\ &= -W^2. \end{aligned}$$

Removing the leading coefficient gives the new polynomial as W^2. Since W^2 is not reducible by G, it is added to G as shown in the second row of Table 2.1.

Each row of the tables shows G, C, and p at the end of an iteration. Note that pairs of monomials always result in a zero s-polynomial and may safely be ignored when adding new pairs to C. Also, when the algorithm ends

$$G = \{XY^2Z - W, Y^2ZW, XY^2W - Z, XY^2W - Z, W^2, ZW, Z^2\}.$$

Table 2.1: Trace for Commutative Gröbner Basis Example Computation (Part One).

Iteration	G	C	p
initial	XY^2Z-W, Y^2ZW, XY^2W-Z	(XY^2Z-W, Y^2ZW) (Y^2ZW, XY^2W-Z) (XY^2Z-W, XY^2W-Z)	
1st	XY^2Z-W, Y^2ZW, XY^2W-Z, W^2	(Y^2ZW, XY^2W-Z) (XY^2Z-W, XY^2W-Z) (Y^2ZW, W^2) (W^2, XY^2W-Z)	W^2
2nd	XY^2Z-W, Y^2ZW, XY^2W-Z, W^2, Z^2	(XY^2Z-W, XY^2W-Z) (Y^2ZW, W^2) (W^2, XY^2W-Z) (Z^2, XY^2Z-W) (Z^2, Y^2ZW)	Z^2
3rd	XY^2Z-W, Y^2ZW, XY^2W-Z, W^2,Z^2	(Y^2ZW, W^2) (W^2, XY^2W-Z) (Z^2, XY^2Z-W) (Z^2, Y^2ZW)	W^2-Z^2
4th	XY^2Z-W, Y^2ZW, XY^2W-Z, W^2,Z^2	(W^2, XY^2W-Z) (Z^2, XY^2Z-W) (Z^2, Y^2ZW)	0
5th	XY^2Z-W, Y^2ZW, XY^2W-Z, W^2,Z^2, ZW	(Z^2, XY^2Z-W) (Z^2, Y^2ZW) (ZW, XY^2Z-W) (ZW, Y^2ZW) (ZW, XY^2W-Z) (ZW, W^2) (ZW, Z^2)	ZW

Table 2.1: Trace for Commutative Gröbner Basis Example Computation (Part Two).

Iteration	G	C	p
6th	$XY^2Z - W$, Y^2ZW, $XY^2W - Z$, W^2,Z^2, ZW	(Z^2, Y^2ZW) $(ZW, XY^2Z - W)$ (ZW, Y^2ZW) $(ZW, XY^2W - Z)$ (ZW, W^2) (ZW, Z^2)	ZW
7th	$XY^2Z - W$, Y^2ZW, $XY^2W - Z$, W^2,Z^2, ZW	$(ZW, XY^2Z - W)$ (ZW, Y^2ZW) $(ZW, XY^2W - Z)$ (ZW, W^2) (ZW, Z^2)	0
8th	$XY^2Z - W$, Y^2ZW, $XY^2W - Z$, W^2,Z^2 ZW	(ZW, Y^2ZW) $(ZW, XY^2W - Z)$ (ZW, W^2) (ZW, Z^2)	W^2
9th	$XY^2Z - W$, Y^2ZW, $XY^2W - Z$, W^2,Z^2, ZW	$(ZW, XY^2W - Z)$ (ZW, W^2) (ZW, Z^2)	0
10th	$XY^2Z - W$, Y^2ZW, $XY^2W - Z$, W^2,Z^2, ZW	(ZW, W^2) (ZW, Z^2)	Z^2
11th	$XY^2Z - W$, Y^2ZW, $XY^2W - Z$, W^2,Z^2, ZW	(ZW, Z^2)	0
12th	$XY^2Z - W$, Y^2ZW, $XY^2W - Z$, W^2,Z^2, ZW		0

Note, however, that Y^2ZW is reducible by ZW and so can be removed from the basis giving the reduced basis

$$G_P = \{XY^2Z - W, XY^2W - Z, W^2, ZW, Z^2\}.$$

If Y^2ZW is removed when ZW is first computed (in the 5th iteration), then the computation of unnecessary critical pairs can be avoided. G_P is the Gröbner basis for the input set P.

2.4 Computing Noncommutative Gröbner Bases

The discussion so far has addressed Gröbner bases in commutative polynomial rings, but now we turn to the noncommutative case. The paradigmatic noncommutative algebra is the free associative algebra. An example is $\mathbb{Q}\langle x, y, z\rangle$, which is the ring of polynomials with rational coefficients and terms that are strings of the variables x, y, z. To be more precise, the set of terms of $\mathbb{Q}\langle x, y, z\rangle$ is the free monoid $\{x, y, z\}^* = \{\lambda, x, y, z, xx, xy, xz, yx, yy \ldots\}$.

The definition of Gröbner bases in noncommutative rings is the same as in the commutative case. However as mentioned earlier, noncommutative Gröbner bases may be infinite. As a consequence, in the computation of general noncommutative Gröbner bases, we must be concerned with termination. When we know in advance that there is a finite Gröbner basis, the noncommutative algorithm is not drastically different from the commutative algorithm.

Buchberger's algorithm for commutative polynomial rings uses least common multiples to find the s-polynomials $SPol(p, q)$ for a critical pair (p, q). However, for noncommutative polynomial rings just using least common multiples is not sufficient; all common multiples must be used instead. Consider the set

$$P = \{xyzy - w, yzyw, xywy - z, ywyz\},$$

assuming the variables do not commute (and so $yzyw$ is not the same as $ywyz$).

Buchberger's algorithm as discussed above yields the generating set

$$G = \{xyzy - w, yzyw, xywy - z, ywyz, ww, zz\},$$

which is essentially the same as the commutative Gröbner basis for P. However, G is not a Gröbner basis, since the polynomial $wzyw$ is an element of $\langle P\rangle$ (as a combination of $yzyw$ and $xyzy - w$) but is not divisible by any tip in G.

The polynomial $wzyw$ is

$$wzyw = xyz(yzyw) - (xyzy - w)zyw$$

which eliminates the common multiple $xyzyzyw$ of $xyzy$ and $yzyw$ rather than the least common multiple $xyzyw$. In general, the algorithm needs to compute polynomials for all common multiples of a pair of tips. So the algorithm cannot simply consider critical pairs but must consider *triples* which consist of a pair of polynomials and an overlap of their tips. For example, the polynomial $wzyw$ corresponds to the triple $\langle xyzy - w, yzyw, y\rangle$. Such a polynomial is called an *overlap relation* and is denoted $o(xyzy - w, yzyw, y)$. For a triple $t = \langle p, q, v\rangle$, the overlap relation is defined as

$$o(t) = \frac{1}{LC(p)} p s_q - \frac{1}{LC(q)} s_p q$$

where $tip(p) = s_p v$ and $tip(q) = v s_q$.

The set G can be completed to a Gröbner basis by adding $wzyw$ and

$$xyw(ywyz) - (xywy - z)wyz = zwyz$$

to G. The overlap relations for all other triples reduce to zero.

2.5 Decidability

The problem of determining whether a finite Gröbner basis exists for an ideal in a noncommutative polynomial ring is an undecidable problem. Undecidability is shown using a reduction of the word problem for semigroups to the ideal membership problem [35, p.25] [46, p.137].

The source of the undecidability is not directly related to finiteness. This is indicated by using polynomial reduction to solve the ideal membership problem using a finite subset of a Gröbner basis for the ideal. Deciding whether a polynomial f is in an ideal I only requires the elements of the Gröbner basis for I whose tips are less than $tip(f)$. Since every admissible order is well-founded, any choice of admissible order $<$ has the property that for all terms t, the set $\{s \text{ a term } : s < t\}$ is finite. Using this order, the partial Gröbner basis bounded by $tip_<(f)$ is finite. However, the undecidability of the ideal membership problem implies that this finite partial basis is not computable [46, pp.137–138].

There are classes of ideals for which the existence of a finite Gröbner basis is decidable. Decidable instances occur when the ideal has a finite Gröbner basis either for all admissible orders or for some order that can be chosen by some means based on the input generators (if this is at all possible). The first case occurs when the algebra is finite dimensional, and when the algebra is noetherian (meaning all ideals are finitely generated). No general conditions for the second case are known. The slightly different problem of finding a partial (bounded) basis is decidable when all elements of the ideal are (degree) homogeneous and the order is compatible with the length of terms [46, p.138].

2.6 Relationship to Rewriting

The earlier discussion showed how the Gröbner basis computation is a kind of nonlinear Gaussian elimination. In this section, the relationship of Gröbner basis to another kind of algebraic computation called term rewriting is considered. This relationship is more of an algorithmic one than algebraic.

Term rewriting is used to decide equivalence between terms in some universal algebra [18]. Terms are mathematical expressions composed of constants, variables and operators. A rewrite rule $s \longrightarrow t$ is used to rewrite (or reduce) a term w by first finding a substitution σ of terms for variable names in s such that $s\sigma$ is a subterm of w. The result of rewriting w in this way is the term formed by replacing $s\sigma$ in w by $t\sigma$. So for example, a rule $f(x, g(y)) \longrightarrow g(f(x, y))$ could be used to reduce $f(u, g(z))$ to $g(f(u, z))$ using the substitution σ for which $x \mapsto u$ and $y \mapsto z$. A sequence of zero or more rewrites $t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \cdots \longrightarrow t'$ is denoted $t \longrightarrow^* t'$. If $t \longrightarrow^* t'$ and t' is not further reducible, then t' is a *normal form* of t.

A relation $\longrightarrow$ induces another relation $\longleftrightarrow$, which is defined by $s \longleftrightarrow t$ if either $s \longrightarrow t$ or $t \longrightarrow s$. The transitive closure $\longleftrightarrow^*$ of $\longleftrightarrow$ induces an equivalence relation $\equiv$ on terms, which is defined as $t \equiv s$ whenever $t \longleftrightarrow^* s$.

A *rewrite system* is the set of terms together with the relation determined by a set of rewrite rules on the terms. A rewrite system is *confluent*, if for all terms w, $w \longrightarrow^* s$ and $w \longrightarrow^* s'$, implies that there exists a term t such that $s \longrightarrow^* t$ and $s' \longrightarrow^* t$ (see Figure 2.2). If for any term t there is no infinite sequence of rewrites, then the rewrite relation is *noetherian*. Confluence ensures that if all rewrites of a term lead to a normal form, then that normal form is unique. Having a noetherian system guarantees that all terms have normal forms, therefore the combination of the two properties is an important one for testing equivalence by rewriting. A rewrite system that is both noetherian and confluent is called *convergent* or *complete*.

$$\begin{tikzcd} & w \arrow[dl, "*"'] \arrow[dr, "*"] & \\ s \arrow[dr, "*"'] & & s' \arrow[dl, "*"] \\ & t & \end{tikzcd}$$

Figure 2.2: Confluence of Rewrite System.

Clearly, convergence is not a property of all rewrite systems. On the other hand, given some noetherian system (usually obtained by choosing a well-ordering of the terms compatible with the operation structure and orienting the rules by the order), an equivalent convergent system may be found using Knuth-Bendix completion [36]. The completion algorithm considers *critical pairs* of terms which correspond to a pair of rules whose left-hand sides interact. In particular, a critical pair is a pair of terms (t_1, t_2) such that there exists a pair of rules $l_1 \longrightarrow r_1$ and $l_2 \longrightarrow r_2$ and a term t that can be rewritten by the first rule to t_1 and by the second rule to t_2. For each critical pair, the algorithm rewrites both terms until a normal form is found for both. If the normal forms are distinct then a new rule is formed by orienting the normal forms. Knuth-Bendix completion stops when all critical pairs converge. However, since the equivalent convergent rewrite system may be infinite, the completion algorithm may not terminate.

The Gröbner basis computation is a special form of the Knuth-Bendix completion algorithm for term-rewriting. A polynomial p can be treated as a rewrite rule $l \longrightarrow r$ in which the left-hand side l is $tip(p)$ the tip of p, and the right-hand side r is the remainder, $1/\alpha p - tip(p)$ (where α is the leading coefficient of p). While polynomial reduction is basically rewriting, it is slightly different since it includes scalar multiplication and term cancellation. However, using this view of polynomials as rewrite rules, a Gröbner basis is a convergent set of rules, and Buchberger's algorithm corresponds to Knuth-Bendix completion. (More details of this relationship can be found elsewhere [14, 15].)

Noncommutative Gröbner bases include a special form of term rewriting called *string rewriting*. In string rewriting the terms are words in a free monoid, and rewriting is done by replacing occurrences of the left-hand side by the right-hand side [10]. A string rewriting rule $s \longrightarrow s'$ can be considered to be a polynomial $s - s'$ (equal to zero) that is an element of the corresponding free algebra. Therefore, completion in string rewriting corresponds to computing Gröbner bases for

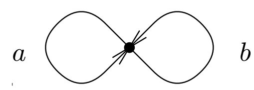

Figure 2.3: Quiver for Free Algebra in a, b.

binomial ideals (ideals generated by sums of pairs of terms).

2.7 Path Algebras

The algebras that this research deals with are path algebras. The polynomials in a path algebra are linear combinations where each term is a pair consisting of a coefficient and a path from a graph (called the *quiver* of the algebra). In general, the coefficients are elements of a field K.

Let $\Gamma = (\Gamma_0, \Gamma_1)$ be a finite directed multigraph with vertex set Γ_0 and arc set Γ_1 (Γ may have multiple arcs between a pair of vertices and may have loops at a single vertex). The set B of finite paths in Γ includes the vertices, the arcs, and all finite walks of Γ. Each path p has a *source* $src(p)$ and a *target* $tgt(p)$ that are the initial and terminal vertices of the path. B is closed under valid path compositions: if $p, q \in B$ and the target $tgt(p)$ of p is the source $src(q)$ of q, then the product $p \cdot q$ is the path pq formed by composition of paths. Note that if v is a vertex, then $v \cdot v = v$, and so the vertices are idempotents. Also, if $v = src(p)$ then $v \cdot p = p$ and so the vertices act like identities for particular elements (so B is like the arrows of the free category of Γ [5]). If we add a zero value 0 to B and extend the operation to return 0 for invalid compositions, then $B \cup \{0\}$ is closed under the composition operator. Since $p \cdot q = pq$, the operator is generally not written. If $p = aqb$ for paths p, q, a, b, we say that q *divides* p and write $q|p$. Two paths p, q are *uniform equivalent* if they begin at the same vertex and end at the same vertex ($src(p) = src(q)$ and $tgt(p) = tgt(q)$).

Two example graphs are shown in Figure 2.3 and Figure 2.4. The set of finite paths for the graph in Figure 2.3 is isomorphic to the Kleene closure $\{a, b\}^*$ of the two letter alphabet $\{a, b\}$ where the vertex corresponds to the empty string. Such a graph with one node and multiple loops corresponds to a free algebra and is referred to as a *free graph*. The set of paths for the graph in Figure 2.4 is $\{u, v, w, a, b, ab\}$.

Given a graph Γ and a field K, the path algebra $K\Gamma$ is the collection of linear combinations $\sum_{b \in B} \alpha_b b$ where α_b is a coefficient from K and only a finite number of the coefficients are nonzero.

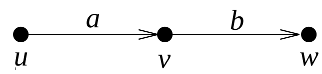

Figure 2.4: Quiver for Simple Path Algebra.

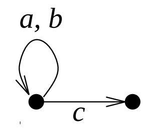

Figure 2.5: Two-Node Quiver for Uniform Projection Example.

The unit (the element that acts like 1 in multiplication) is the sum of the vertices $\sum_{v\in\Gamma_0} v$. The elements of $K\Gamma$ are called *relations* (but the term *polynomial* is also used to refer to them as formal sums of terms).

The set of paths whose coefficients are nonzero in a relation f is called the *support* of f and is denoted $supp(f)$. A polynomial p for which all elements of $supp(p)$ are uniform equivalent to each other is called *uniform*. Given an admissible order $<$ on B, the leading term or *tip* of a polynomial p is the maximum element $tip_<(p)$ in the support of p. If the graph Γ is a single vertex with several loops labeled a, b, c, then $K\Gamma$ is the free associative algebra, usually denoted $K\langle a, b, c\rangle$.

Ideals, generating sets, and Gröbner bases are all defined as before. However, computing Gröbner bases in path algebras can lead to problems if the input generators are not uniform. The problem is that during reduction the tip of the dividing polynomial is multiplied on both sides, and if there is a path with different source and target in the support, that term is canceled by path compositions (rather than by the polynomial difference). Consider the computation of the Gröbner basis for the generators $\{abc + bab + a + c\}$ from the algebra whose quiver is the graph in Figure 2.5. Using the length lexicographic order defined for $a > b > c$, the algorithm does not find any new elements. However, the generator $abc+bab+a+c$ is a sum of uniform polynomials $abc+c$ and $bab+a$. Using the generating set $\{abc+c, bab+a\}$ the algorithm computes the set $\{abc+c, bab+a, ac-bc, aab-baa\}$. The first generating set does not satisfy the Gröbner basis criterion since the element $bab + a$ is in the ideal, but is not divisible by the tip abc. So, for the nonuniform generator, the set returned is not a Gröbner basis. As this example shows, using nonuniform generators as input to the Gröbner basis computation can give incorrect results.

By definition, ideals are closed under multiplication by elements of the algebra $K\Gamma$. So, in particular, if we have an element p of an ideal I and two vertices u and v then $upv \in I$. For example, suppose that $p = t_1 + t_2 + t_3$, for paths t_1, t_2, t_3. Also let the sources and targets for these paths be the following

Path	Source	Target
t_1	s	v
t_2	s	v
t_3	u	v

where s, u, and v are distinct vertices. Now consider the products

$$\begin{aligned} spv &= st_1v + st_2v + st_3v \\ &= t_1 + t_2 + 0 \end{aligned}$$

and

$$\begin{aligned} upv &= ut_1v + ut_2v + ut_3v \\ &= 0 + 0 + t_3. \end{aligned}$$

Hence, p can be written as the sum $spv + upv$. Since both spv and upv are relations whose support elements are uniform equivalent, we call spv and upv *uniform projections* of p.

In general, it is sufficient to consider only uniform generators [20]. To see this, suppose we have a generating set P with a nonuniform element p. Let p_1 and p_2 be the uniform projections of p. Then p_1 and p_2 are in $\langle P \rangle$. But since ideals are closed under addition $p_1 + p_2 = p$ is in $\langle P \rangle$ and we can replace p by its uniform projections in P and still generate the same ideal. Precisely,

$$\langle P \rangle = \langle (P \setminus \{p\}) \cup \{p_1, p_2\} \rangle$$

Therefore, generators given as input to the noncommutative form of Buchberger's algorithm can be assumed to be uniform (or equivalently, for the purposes of the computation, elements of a free algebra).

2.8 Summary

This chapter has introduced the computation of Gröbner bases of (two-sided) ideals of noncommutative algebras. The key concepts to remember from this chapter are the basic definition of noncommutative algebras, noncommutative polynomials, ideals, admissible orders and Gröbner bases. The algorithm and orders are revisited in later chapters.

Path algebras are important in this thesis because all specific problem instances are presented in path algebras (for example, in the discussion of experiments in Chapter 3). However, the path algebra definition is most important in Chapter 5 where the problem of choosing an admissible order is addressed.

Chapter 3

Computing Noncommutative Gröbner Bases

This chapter considers the problem of finding a good combination of algorithms and data structures to implement Buchberger's algorithm for noncommutative algebras. The emphasis is on the alternative algorithms defined in the commutative Gröbner basis literature, but some new algorithms are also presented. Data structures are rarely discussed in the Gröbner basis literature, so they are considered here for the sake of thorough discussion.

The chapter begins in the first section by revisiting the basic algorithm and the different methods used for termination. Then, in the subsequent sections, the algorithmic variants and alternative data structures are considered. The fourth section deals with experiments that compare the different configurations of the algorithm.

3.1 The Basic Algorithm

Buchberger's algorithm for computing noncommutative Gröbner bases is shown in Figure 3.1. The algorithm takes a set F of generators as its argument, and finds the reduced Gröbner basis G for F. Formally, the algorithm also has an admissible order $<$ as an argument, but the order is implicit in the algorithm as shown in Figure 3.1. This algorithm terminates if and only if there is a finite Gröbner basis for the given generating set and order (modifications for termination are considered below).

The algorithm begins with the INITIALIZE procedure (Figure 3.2), which makes a reduced copy G of F and a corresponding set T of triples. A *triple* is a pair of polynomials and a (nonempty) overlap of their tips. The triples are used to keep track of the overlap relations for G (see Section 2.4 for the definition of overlap relation). The INITIALIZE function reduces each element f of F by G, and if the result g is nonzero adds it to G and forms new triples of g with elements of G (using the UPDATE procedure). The REDUCE function can do either tip- or total-reduction, which are the functions TIP_REDUCE and TOTAL_REDUCE detailed in Section 3.3.2.

Following initialization of G, the algorithm iterates the process of selecting (and removing) a triple from T (with SELECT), forming the corresponding overlap relation, reducing the overlap relation, and, if the result is nonzero, using UPDATE to add the result to G (and any new triples to T). The loop ends when there are no triples remaining to consider. The algorithm ends by totally reducing G using REDUCE_BASIS (Figure 3.3).

The UPDATE procedure in its simplest form is shown in Figure 3.4. Basically, UPDATE finds overlaps of the new element h with elements of G, and adds h to G. This UPDATE procedure does not keep G reduced. The next section shows different implementations of UPDATE that both keep the set reduced and delete "useless" triples. The function OVERLAP computes the triples of elements of G with h. Details of how OVERLAP can be implemented are given in Chapter 4.

3.2 Termination

The fact that noncommutative Gröbner bases may not be finite requires modifications to the algorithm to force it to terminate. The algorithm in Figure 3.1 only terminates if the input generators (and order) have a finite Gröbner basis. (The proof of termination when the Gröbner basis is finite is based on the "diamond lemma" of Bergman [8].) Otherwise, the algorithm enumerates an infinite set. There are two ways in which the algorithm is modified to use bounds to force termination. Both modifications will find a finite Gröbner basis if it exists within the given bound.

In the general case, the algorithm is modified to take an additional argument N that is a bound on the number of nonzero reductions of overlap relations. This algorithm is shown in Figure 3.5 (the differences are the added test on line 3, and the addition of line 14). Once N nonzero reductions have occurred the modified algorithm stops. This form of the algorithm extends the reduced form

GRÖBNER(F). Buchberger's algorithm for instances with finite Gröbner bases.

INPUT: Set F of generators.
OUTPUT: G a finite, totally reduced Gröbner basis for F.

```
(G, T) ←INITIALIZE(F);
while (T ≠ ∅) do
     begin
          ▷ select a triple
          t ← SELECT(T, G);
          ▷ form overlap relation
          h ← OVERLAP_RELATION(t);
          ▷ reduce overlap relation
          h′ ← REDUCE(h, G);
          ▷ add h′ to G if not zero
          if (h′ ≠ 0) then
               UPDATE(G, T, h′);
     end
REDUCE_BASIS(G);
return G;
```

Figure 3.1: Buchberger's Algorithm for Noncommutative Algebras.

INITIALIZE(F). Initialization function for Buchberger's algorithm.

INPUT: Set F of generators.
OUTPUT: G a self-reduced copy of F, and the set T of triples for G.

```
G ← ∅;
T ← ∅;
foreach (f ∈ F) do
     begin
          g ← REDUCE(f, G);
          if (g ≠ 0) then
               UPDATE(G, T, g);
     end
return (G, T)
```

Figure 3.2: Initialization for Buchberger's Algorithm.

REDUCE_BASIS(G)**.** Total reduction of set G.

INPUT: Tip-reduced set G of generators.
OUTPUT: G totally reduced.

```
foreach (g ∈ G) do
    begin
        G ← G \ {g};
        g′ ← TOTAL_REDUCE(g, G);
        G ← G ∪ {g′};
    end
```

Figure 3.3: Basis Reduction.

UPDATE(G, T, h)**.** Procedure to update G and T with h.

INPUT: Tip-reduced set G, triple set T, tip-reduced polynomial p.
OUTPUT: Self-reduced G with $h \in G$, triples for h in T

```
T ← T ∪ OVERLAPS(G, h);
G ← G ∪ {h};
```

Figure 3.4: Simple UPDATE.

GRÖBNER(F, N)**.** Buchberger's algorithm for instances with finite Gröbner bases.

INPUT: Set F of generators, positive integer N.
OUTPUT: G a finite, totally reduced Gröbner basis for F.

```
(G, T) ← INITIALIZE(F);
k ← 0;
while (T ≠ ∅ and k < N) do
    begin
        ▷ select a triple
        t ← SELECT(T, G);
        ▷ form overlap relation
        h ← OVERLAP_RELATION(t);
        ▷ reduce overlap relation
        h' ← REDUCE(h, G);
        ▷ add h' to G if not zero
        if (h' ≠ 0) then
            UPDATE(G, T, h');
            k ← k + 1;
    end
REDUCE_BASIS(G);
return G;
```

Figure 3.5: Buchberger's Algorithm for the General Case.

of F by at most N new elements when run with a bound of N.

Theorem 3.1 *The process of computing a (partial) Gröbner basis by bounding the maximum number of nonzero reductions of overlap relations terminates and returns a subset of the Gröbner basis.*

Proof If this algorithm does not terminate for a given input F, then there must be an infinite number of zero reductions of overlap relations computed from F. However, each overlap relation is determined by a triple in the set T of triples, and at any point during the computation the set T is finite. (Since there are only a finite number of overlaps of the elements of the current generating set, which is itself finite.) Since new triples are added only when a nonzero overlap reduction occurs, there cannot be an infinite sequence of zero reductions. Therefore, the algorithm will terminate either by emptying T, or by reaching the bound on nonzero reductions. In the first case, the result is a complete Gröbner basis, and in the second case the result is a nontrivial subset of the Gröbner basis. $\square$

GRÖBNER(F, W). Buchberger's algorithm for instances with finite Gröbner bases.

INPUT: Set F of generators, positive integer W.
OUTPUT: G a finite, totally reduced Gröbner basis for F.

```
1   (G,T) ←INITIALIZE(F);
2   while (T ≠ ∅) do
3       begin
4           ▷ select a triple
5           t ← SELECT(T, G);
6           ▷ form overlap relation
7           h ← OVERLAP_RELATION(t);
8           ▷ reduce overlap relation
9           h' ← REDUCE(h, G);
10          ▷ add h' to G if not zero
11          if (h' ≠ 0) then
12              if (|tip(h')| ≤ W) then
13                  UPDATE(G, T, h');
14      end
15  REDUCE_BASIS(G);
16  return G;
```

Figure 3.6: Buchberger's Algorithm for the Degree Homogeneous Case.

Alternatively, if the elements of F are all homogeneous, a different kind of bound can be used. All of the terms of a (*degree*) homogeneous polynomial have the same length, which is called the *degree* of the polynomial. When all generators are homogeneous, the degree of new basis elements is nondecreasing and so a bound on the degree of elements can be used. The algorithm for homogeneous generators shown in Figure 3.6 inserts a polynomial into G only if its tip is shorter than the bound W. The only change to the algorithm is the additional test on line 12 that the length of $tip(h')$ is less than or equal to W. The elements with longer tips can either be discarded, or saved in another set.

Theorem 3.2 *The algorithm for degree homogeneous generators shown in Figure 3.6, terminates and returns all elements of a Gröbner basis having degree less than W.*

Proof The termination of this algorithm is based on the fact that, given a finite alphabet, there is a finite number of words of length at most W. This means that there can only be a finite number of elements of a Gröbner basis whose tips are of length at most W. Therefore, we need to show that the algorithm computes all elements of the bounded basis and no more.

First, notice that if we reduce a degree homogeneous polynomial p by another degree homogeneous polynomial q, then the result is degree homogeneous and of the same degree as p. Therefore, the reduced form h' of an overlap relation $h = o(g_1, g_2, v)$ has the same degree as h. Also, note that the degree of h is greater than both the degree of g_1 and the degree of g_2. The implication is that the algorithm cannot find a new basis element whose tip is shorter than every other element of the basis.

Suppose that there is some g with degree at most W that is an element of the Gröbner basis, but that g is not found by the algorithm. Then either g cannot be computed from the input generating set, or requires computing elements whose degree is larger than W (since the algorithm ignores these elements, g would never be found). However, if g cannot be computed from the generators, g cannot be in the Gröbner basis; and, as discussed above, g could not possibly be computed from elements of larger degree. Therefore, g must be found by the algorithm, and so the algorithm must find all elements of the Gröbner basis of degree at most W. As a consequence, if every element of the Gröbner basis has degree less than W, the result will be the complete set.

Now suppose that the algorithm does not terminate. Then the algorithm must make an infinite selection of triples whose overlap relations are of degree greater than W. However, since the algorithm makes no triples for such elements, all such triples must correspond to overlaps of elements of shorter degree. But since this set is finite, there is only a finite number of overlaps, and only a finite number of possible triples to select. Therefore, the triple set must eventually be empty, and the algorithm must terminate. $\square$

3.3 Algorithmic Alternatives

As has been discovered in the setting of commutative polynomial rings, there are several variations of the basic Buchberger algorithm. Variations occur in the selection strategy (the Select function in Figure 3.1), triple elimination (the Update function), basis reduction (in both Update and Reduce_Basis), and polynomial reduction (in function Reduce). Our goal is to determine what combination of these variations is the most efficient.

3.3.1 Selection Strategy

A *selection strategy* is a method for selecting a triple from the triple set so that its overlap relation can be used in the computation. The role of a selection strategy is to choose triples in such a way that the Gröbner basis is found as directly as possible. Usually, this means finding new elements that can be used to eliminate other triples (see the discussion on triple elimination below).

In general, a selection strategy need only satisfy a fairness property that the selection of a particular triple is not postponed indefinitely (see [46]). One form of selection strategy is to choose the triple with the smallest common multiple with respect to some well-order (that need not be the same as the admissible order on the polynomials). If the well-order used for selection is also the admissible used to order the polynomials, this strategy is called the *normal* strategy. Fairness is ensured by the property of well-orders (there is not an infinite sequence of triples less than any particular triple). Traverso and Donati [54] describe other forms of selection. One example that performs well in their experiments (in the commutative case) is ordering the triples by the leading term of the corresponding overlap relation.

In Figure 3.1, the Select function performs selection. The selection strategy used here (Figure 3.7) selects the triple t with minimal common multiple $cm(t)$ with respect to some order ($\min_< T$ is the minimal such triple in T with respect to $<$). Two such strategies are considered in our experiments. The first is the normal strategy, and the second uses a length lexicographic order to compare the common multiples and is called the *shortest* strategy.

Assuming that triples are computed as new elements are added, it is most efficient to store the triples in a priority queue data structure where the next triple to be selected can be extracted in logarithmic time. One such data structure is a heap, although the prototype on which the experiments are run uses a sorted list. A heap is used in the Opal system.

3.3.2 Polynomial Reduction

Polynomial reduction is an operation performed repeatedly during the Gröbner basis computation. The two forms of reduction, tip- and total reduction introduced in Chapter 2, are the alternative algorithms for reduction. The algorithm for tip-reduction is shown in Figure 3.8 and the algorithm for total reduction is shown in Figure 3.9. Clearly, tip reduction is a simpler algorithm, but it is not clear which is the better choice. For the commutative case, Traverso and Donati [54, p.196]

SELECT(T). Triple selection.

INPUT: Set T of triples.
OUTPUT: Triple t with $cm(t)$ minimal with respect to a given admissible order $<$.

$t \leftarrow \min_{<} T$;
$T \leftarrow T \setminus \{t\}$;
return t;

Figure 3.7: Standard Selection Algorithm.

state that total reduction is a better choice than tip reduction (but the experiments from which this conclusion is drawn are not described).

Neither reduction algorithm specifies how the divisor should be chosen for each simple reduction (e.g., line 6 in Figure 3.9 and line 3 in Figure 3.8). The choice of divisor can lead to coefficient explosion if the coefficient field is not finite (e.g., is the rational numbers, or a rational function field). Traverso and Donati [54] discuss experiments in the noncommutative case that imply a good choice is the divisor with the fewest terms; however, they also note that the choice of divisor is less significant than the selection strategy.

Coefficient explosion is also common in other normal form computations such as finding canonical forms of matrices, and the heuristics [27, 28] used in these situations may extend well to Gröbner basis computations.

Nothing is done in this research to compare the two reduction strategies in the noncommutative case, nor is anything done to consider the selection of a divisor. In the prototype, the "first" divisor found is used; and in Opal, the one with the fewest number of terms is used.

3.3.3 Set Reduction

If the goal of the Gröbner basis computation is to find the minimal reduced Gröbner basis, then keeping the working set reduced (or at least, tip-reduced) during the computation is important. Otherwise, unnecessary overlap relations are computed and reduced. If during the computation, some element p of the working set P is tip-reducible by the leading term of the newest reduced overlap relation h, then p needs to be reduced. Gebauer and Möller call such a tip-reducible element

TIP_REDUCE(f, P). Polynomial Tip Reduction.

INPUT: Polynomial f, set P of polynomials.
OUTPUT: Remainder of dividing f by elements of P at the tip.

while $(\exists p \in P$ such that $tip(p) | tip(f))$ **do**
    begin
        Choose $p \in P$ and $a, b \in B$ such that $tip(f) = a(tip(p))b$;
        $f \leftarrow f - a \cdot p \cdot b$;
    end
return f;

Figure 3.8: Tip Reduction Algorithm.

REDUCE(f, P). Polynomial Reduction.

INPUT: Polynomial f, set P of polynomials.
OUTPUT: Remainder of dividing f by elements of P.

$f' \leftarrow 0$;
while $(f \neq 0)$ **do**
    begin
        if $(\exists p \in P$ such that $tip(p) | tip(f))$ **then**
            begin
                Choose $p \in P$ and $a, b \in B$ such that $tip(f) = a(tip(p))b$;
                $f \leftarrow f - a \cdot p \cdot b$;
            end
        else
            $f' \leftarrow f' + tip(f)$;
            $f \leftarrow f - tip(f)$;
    end
return f';

Figure 3.9: Total Reduction Algorithm.

$\textsc{Update}(G, T, h)$**.** Add h to G and new triples to T using redundant element deletion.

INPUT: Tip-reduced set G, triple set T, tip-reduced polynomial p.
OUTPUT: Self-reduced G with $h \in G$, triples for h in T

$T \leftarrow T \cup \textsc{Overlaps}(G, h)$;
$D \leftarrow \{g \in G : \mathit{tip}(h) | \mathit{tip}(g)\}$;
$G \leftarrow (G \setminus D) \cup \{h\}$;

Figure 3.10: UPDATE Using Redundant Element Deletion.

redundant [22].

Tip-reducible elements of P can be dealt with in two ways. The first technique, due to Gebauer and Möller, is to simply delete the reducible elements. This approach, called *redundant element deletion*, requires that the triples involving reducible elements be kept. Otherwise, the result may not be complete. The second approach is to (tip-) reduce the reducible elements and add them back to the set. This approach, called *(redundant) element reduction*, deletes all triples involving reducible elements, and then finds triples for the reduced form.

In the commutative case, reductions of redundant elements can be done by the ordinary computation and reduction of s-polynomials. Specifically, if p is tip-reducible by q, then p and q form a critical pair for which the s-polynomial $SPol(p, q)$ is the simple reduction of p by q. Traverso and Donati [54] consider a selection strategy that chooses reduction of redundant elements over other pairs that is analogous to element reduction. In the noncommutative case, however, overlap relations cannot correspond to reductions of redundant elements, and so redundant element reduction must be done explicitly.

Set reduction, the process of removing redundant elements, is done in Figure 3.1 by the UPDATE procedure. The two forms of set reduction give two forms of the UPDATE procedure. One using redundant element deletion is given in Figure 3.10, and the other using element reduction is given in Figure 3.11. (The notation left(t) and right(t) refers to the polynomials occurring in the left and right respectively of a triple t.) The element reduction form of UPDATE calls itself for each reduced element added back into the set. Both forms of set reduction are compared in the experimentation described in Section 3.5.

UPDATE(G, T, h)**.** Procedure to update G and T with h using element reduction.

INPUT: Tip-reduced set G, triple set T, tip-reduced polynomial p.
OUTPUT: Self-reduced G with $h \in G$, triples for h in T

$D \leftarrow \{g \in G : tip(h) | tip(g)\}$;
$G \leftarrow (G \setminus D)$;
$T' \leftarrow \{t \in T : \text{left}(t) \in D \text{ or right}(t) \in D\}$;
$T \leftarrow T \setminus T'$;
$G \leftarrow G \cup \{h\}$;
$T \leftarrow T \cup$ OVERLAPS(G, h);
foreach $g \in D$ **do**
  begin
    $g' \leftarrow$ REDUCE(g, G);
    if $(g' \neq 0)$ **then**
      UPDATE(G, T, g');
  end

Figure 3.11: UPDATE Algorithm Using Element Reduction.

3.3.4 Triple Elimination

The elimination of *useless pairs* (pairs for which the s-polynomial reduces to zero) is first considered by Buchberger [12]. Buchberger defines several criteria that can be used to determine whether a critical pair can be eliminated, and later Gebauer and Möller [22] derive similar criteria using syzygies. Mora [46] shows (using syzygies) that these criteria can be applied in computing noncommutative Gröbner bases. Another proof is given in Appendix A that uses an approach like that of Becker and Weispfenning [7].

What is proved in the appendix is known as Buchberger's second criterion. The other well-known Buchberger criterion is Buchberger's first, which says that pairs of polynomials with disjoint leading terms (terms with no common divisor) need not be considered. In the noncommutative case, this criterion is: the overlap relation of two polynomials whose tips do not overlap need not be considered. Since triples are only computed for polynomials whose tips overlap, this criterion is vacuous. (In the commutative case, the criterion is not vacuous since the definition of s-polynomials allows disjoint leading terms.)

The noncommutative form of Buchberger's second criterion states something roughly like the following (the exact statement appears in Appendix A). Let F be a generating set, and let g_1 and

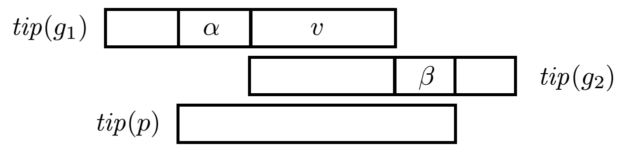

Figure 3.12: Division of Common Multiple for Buchberger's Second Criterion.

g_2 be elements of F. Suppose that $tip(g_1)$ and $tip(g_2)$ have an overlap v, and that the common multiple $cm_v(tip(g_1), tip(g_2))$ is properly divisible by the tip of an element $p \in F$. In this situation, $tip(p)$ overlaps with $tip(g_1)$ on the left, and with $tip(g_2)$ on the right (see Figure 3.12). Call these overlaps w_l and w_r. If $o(g_1, p, w_l)$ and $o(p, g_2, w_r)$ reduce to zero by F, then $o(g_1, g_2, v)$ reduces to zero by F.

The implication is that the overlap relation $o(g_1, g_2, v)$ is superfluous once the other two relations have been added to F (or are likewise expressible by another pair of relations). The second criterion translates to the following algorithmic test: if the common multiple $cm_v(tip(g_1), tip(g_2))$ is properly divisible by $tip(p)$ for some $p \in F$, then the triple $\langle g_1, g_2, v\rangle$ can be discarded. Note that p could be either g_1 or g_2 so long as the tip of p divides somewhere other than as a prefix (if $p = g_1$) or as a suffix (if $p = g_2$) of the common multiple. To be more precise, let $tip(p)$ divide $cm_v(tip(g_1), tip(g_2))$ such that $tip(p) = \alpha v \beta$ as in Figure 3.12. Then, in the case that $p = g_1$, the elimination can only be performed if β is a nonempty word (e.g., it is not the case that $tip(p) = \alpha v$). Similarly, when $p = g_2$, α must be nonempty for the elimination to be valid.

In general, we will assume that the generating set F is kept tip-reduced during the computation. So, in particular, the tip of one element will never divide the tip of another. Notice that, in this case, if $tip(p)|cm_v(tip(g_1), tip(g_2))$, then the overlap v must divide $tip(p)$. However, it is not necessary to check this condition.

In the literature, two strategies for using this result are described. The first is due to Buchberger and the second to Gebauer and Möller. Although, in the commutative case these strategies are significantly different (or at least appear so), in the noncommutative case the test for elimination is the same. The main difference is when the elimination is done.

Buchberger's strategy tests the common multiple for divisibility for each triple selected, and if the common multiple is divisible, then selects another triple (Figure 3.13). The Gebauer-Möller

SELECT(T). Triple selection with Buchberger triple elimination.

INPUT: Set T of triples.
OUTPUT: Triple t with $cm(t)$ minimal with respect to a given admissible order $<$.

repeat
    $t \leftarrow \min_< T$;
    $T \leftarrow T \setminus \{t\}$;
until (there is no $g \in G$ such that $tip(g) | cm(t)$);
return t;

Figure 3.13: Selection With Triple Elimination.

EAGER(G, T, h). Gebauer-Möller triple elimination.

INPUT: G tip-reduced set of generators, T set of triples for G, h polynomial.
OUTPUT: T' triple set containing nondivisible elements of T and triples of h.

$T_O \leftarrow \{t \in T : tip(h) \not| \; cm(t)\}$;
$T_N \leftarrow \{t \in \text{OVERLAPS}(G, h) : \exists g \in G \;\; tip(g) \not| \; cm(t), g \neq h\}$;
return $(T_O \cup T_N)$;

Figure 3.14: Gebauer-Möller Elimination.

strategy is to perform eliminations as soon as possible, which is when a new element is introduced to the Gröbner basis during a computation. This requires testing common multiples for overlaps of the newly inserted polynomial with elements already in the set (e.g., testing divisibility by "old" tips), and testing existing common multiples for divisibility by the tip of the new element (Figure 3.14). The Gebauer-Möller strategy is applied in the UPDATE procedure (replacing line 1 of Figure 3.10 or line 7 of Figure 3.11 with $T \leftarrow \text{EAGER}(G, T, h)$).

Again, the only real difference between these two approaches is when they are applied. Since the Gebauer-Möller strategy is to apply the test as soon as possible, and Buchberger's strategy is to apply the test as late as possible, we refer to them as the *eager* and *lazy* elimination strategies.

A third approach is a hybrid of the eager and lazy strategies. In this approach, new triples (those involving h and the elements of G) are eagerly eliminated in the UPDATE algorithm, and old triples (those involving just current elements of G) are eliminated lazily in SELECT. We call this approach the *hybrid* elimination strategy. All three elimination strategies are considered in the experiments.

Note that, in general, *elimination* also refers to an application of Gröbner bases (see Becker and Weispfenning [7, p.256], or Helton and Stankus [29]), so some care is needed in using the term. In this thesis, *triple elimination* and *elimination* always mean the same thing.

3.4 Data Structures

The main data structures required for the algorithm are polynomial sets for storing basis elements (and facilitating pattern matching) and triple sets. The representations of paths and polynomials are also important, but neither provides much opportunity for finding an exceptionally efficient data structure (most operations can be done in linear time in the number of terms and there is not much chance of improving this dramatically).

3.4.1 Polynomials

Polynomials are typically stored as a list of coefficient-term pairs, sorted with respect to an admissible order on the terms. This data structure allows the tip, which is the head of the list, to be found in constant time. Addition and subtraction are basically merge sorts with combination or cancellation of like terms.

The most costly operation on polynomials is reduction. A simple reduction of p by q using lists takes time on the order of $n + k$ where n and k are the number of terms in p and q respectively. A reduction of p by a set P takes time on the order of $O(n^2)$.

An alternative we have considered is to impose an indexing structure on the list that makes the task of finding the first position for merging more efficient. Two possible structures are a binary tree that would hold pointers to monomials in the list, and a bucket structure that partitions the monomials. When reducing p by P, the index structure is first added to the list for p, and then each simple reduction $p - a \cdot q \cdot b$ by some $q \in P$ uses the index to merge the list for $a \cdot q \cdot b$ into the list for p. The indexing is discarded once the reduction is complete. The only advantage of this approach is when p has a large number of terms, and the sequential search for the beginning of the merge is too inefficient. Yan [56] describes another data structure to store subterms during reduction as a list of sorted lists of geometrically increasing length. Merging is then done after the reduction is complete.

3.4.2 Polynomial Sets

Other than simply storing polynomials, the polynomial set data structure must also support polynomial reduction, basis reduction, and finding overlaps (for forming triples). Reduction of a polynomial p by a set S requires searching for an element q of S such that $tip(q)$ divides a term of p. In basis reduction, the objective is to find all the basis elements g whose tips are divisible by the tip of a polynomial h not in the basis. Overlaps are found when a new element h is added to the basis. A new overlap occurs between $tip(h)$ and the tip of some element currently in the polynomial set. An overlap of two paths α and β is determined by a common subpath that is a suffix of α and a prefix of β. Therefore, the overlaps formed by a polynomial h with a polynomial set G can be viewed as two types: overlaps where the basis elements are on the left (left-overlaps), and overlaps where the the basis elements are on the right (right-overlaps). All of these operations can be thought of as pattern matching operations with a set of patterns, the tip set of the current generating set. Chapter 4 discusses the pattern matching problems in detail and gives a solution based on the dynamic dictionary matching approach of Amir *et al.* [2] in which all searches take time linear in the size of the search string.

3.4.3 Triple Sets

One key to efficient Gröbner basis computation is the efficient handling of triples. This follows from the frequency of operations on triples (selection, formation of overlap relations, and elimination), and the large number of triples (which can be at least exponential in the number of initial generators).

The triple set data structure needs to support the addition and deletion of elements as well as selection of the next triple in whatever selection strategy is used. Selection needs to be as cheap as possible since it is the most frequent operation. Keeping the triples in some kind of priority queue sorted in the order of the selection strategy means that selection can be done in (at least amortized) logarithmic time. In this case, the complexity of insertions depends on the data structure chosen for the priority queue.

Deletions from the triple set are only done in conjunction with redundant element reduction. Deletions occur when a polynomial is removed from the polynomial set, and all triples involving the tip must be removed. This requires some form of index into the priority queue structure that allows the deleted elements to be removed without significant searching. Two indices are needed, one that maps polynomials to triples in which they occur as the left polynomial, and the other that maps polynomials to triples in which they occur on the right. For each polynomial, the indices hold a list of (references to) triples.

With this structure, deletion by polynomial p first finds the list of left occurrences of p, deletes each occurrence from the priority queue, and then does the same for the right occurrences of p. The order of deleting left and right occurrences is not important, but the algorithm must take care not to attempt the deletion of triples corresponding to self-overlaps of $tip(p)$ twice.

In the prototype system described below, the priority queue is implemented as a sorted linked list with pointers to the front and rear of the list. The entries of the list are not triples but lists of overlap lengths for each pair of polynomials sorted from shortest overlap to longest overlap. Although this approach of storing the triples saves some space and allows quicker deletion, it means that selecting a triple may require more time since the triple for the next overlap in the front list entry may not correspond to the smallest common multiple for the set. So the entry may need to be pushed back into the list. The best choice of data structure for implementing the priority queue is a heap with each node corresponding to a triple.

The efficiency of the eager triple elimination strategy also depends on the heap structure. The second part of eager elimination is the test that the common multiples of any existing triples are divisible by the tip of the newly inserted basis element. This test requires that *every* triple in the set be tested (which is what the prototype does).

Clearly, searching the whole triple set is not time efficient, so another approach is to include a pattern matching dictionary of the kind used for the polynomial sets and described in Chapter 4. Instead of holding the tips of the polynomial set, the dictionary for the triple set holds the common multiples for all of the triples. Only the subword and superword searches would be necessary to test for elimination. This approach to implementing eager elimination is not implemented in the prototype, and so is not considered in the experiments described in the next section. Opal uses the hybrid approach to triple elimination which does not require the extra dictionary.

3.5 Algorithmic Experimentation

The algorithmic alternatives of triple elimination, set reduction, and selection strategies are compared experimentally. The experiments use the prototype system described in the first subsection to compare different configurations of these three algorithmic variations. The experiments discussed in the second subsection first do a general comparison, and then do separate comparisons of the set reduction strategies, and the triple elimination strategies. The implications for implementation of Gröbner basis systems, and, in particular, the Opal system are discussed in the third subsection.

3.5.1 A Prototype Implementation

A prototype system written in Standard ML is used to study the interactions between the algorithmic variations described above. The system is actually a family of different prototypes that implement seven different configurations of the algorithmic variations (each configuration is built by compilation). The seven configurations are combinations of the two approaches to basis reduction (element reduction, redundant element deletion) and four approaches to triple elimination (none, lazy strategy, eager strategy, and the hybrid strategy). (The hybrid elimination strategy cannot be combined with redundant element deletion.)

The system is built using the SML module facilities of signatures, structures, and functors (signatures define the module interface, structures give the module implementation, and functors are parameterized modules). The relationships between the primary functors and structures of the system are shown in Figure 3.15. (The system consists of many other structures, some of which are implementations of data structures from the SML/NJ library, version 0.2 [4].) In the figure, the square boxes represent the functors and the rounded boxes represent the structures created using the functors. The arrows into the functor boxes indicate which structures are arguments to the functors.

Each different configuration of the system is primarily determined by a different "Buchberger" functor (there are seven files containing the different versions of the functor). Most of the differences are localized to the function that implements the UPDATE procedure discussed above. The only other structure that varies for the different configurations is the polynomial set structure, which is different for the redundant element deletion approach to set reduction.

Triple sets are implemented as doubly linked lists (using SML references) with the nodes sorted in increasing order using the admissible order used by the selection strategy. Each list node holds all the triples for a pair of polynomials as a tuple consisting of references to the two polynomials and a list of overlap lengths (sorted longest to shortest). The order on the nodes compares the common multiples for the first overlap in each node using the selection order.

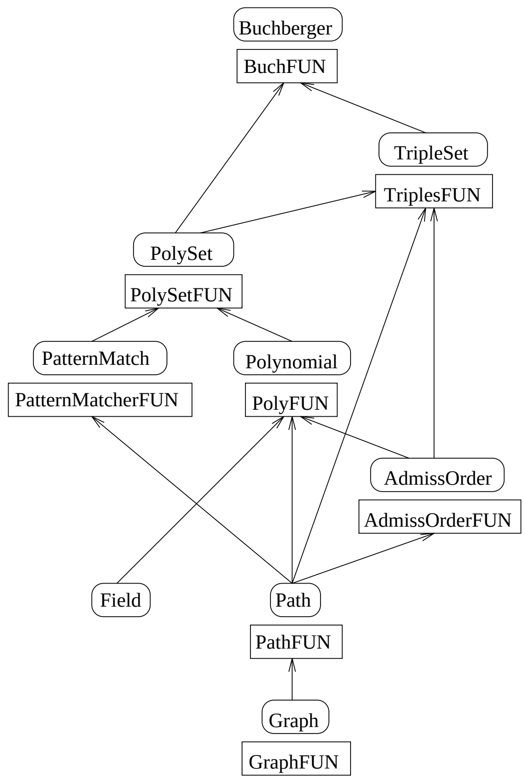

Figure 3.15: Structure of the Prototype.

When an element is selected, the first overlap in the tuple at the head is used to form a triple, and the tuple consisting of the remaining overlaps is reinserted into the list. A pair of hash tables for looking up triples for particular polynomials is also included (the tables correspond to polynomials occurring on the right and left of some triple). The hash tables are used for deleting triples when a polynomial is reduced during redundant element reduction.

Polynomial sets are implemented as a dynamic array (from the SML/NJ library) of polynomials and a dynamic dictionary pattern matcher. For redundant element deletion, the deleted elements are kept in the array, but their leading terms are removed from the pattern matcher. So, the deleted elements are not included in new triples and are not accessible except by operations on triples in which they already occur.

Polynomials are implemented as SML lists of coefficient-path pairs, with the empty list representing the zero polynomial. The polynomials are kept sorted by the admissible order. Paths are also implemented as an SML data type consisting of a zero value and tuples describing the path. The tuples hold the source and target vertices, the list of arcs in the path, and the weight of the path. A path which is a single vertex is a tuple with the vertex as both its source and target and the arc list empty. The coefficients are from the field structure, which represents the Integers modulo a prime (which for the experiments is fixed to be 32117).

The admissible order structure provides higher order functions that can be used to build up orders from an alphabetic order (constructed using the graph structure). The functions to build orders include one for the lexicographic order that takes an alphabetic order as an argument, and seven other functions that take orders on paths as arguments and produce orders on paths. The order producing functions are length, weight (for integer weightings), reversal, vector lexicographic (identical to the commutative left lexicographic ordering), and inverse vector lexicographic (commutative right lexicographic). To build an order function, a lexicographic order is built first and then one of the other orders is used to add other tests as prefixes to the order. (Not all compositions of these functions produce admissible orders, but the prototype does not check admissibility.)

In the prototype, the graph structure plays the role of a symbol table for checking whether the symbols in an input polynomial are valid, and whether two arcs compose. The graph is implemented using the binary tree dictionary structure from the SML/NJ library to store a dictionary of arcs and vertices.

In addition to the structures shown in Figure 3.15, the prototype includes an instrumentation structure used to collect information about the execution of the system. Information collected includes execution time, counts of reductions, and maximum cardinality of the generating set (or basis) and the triple set. Information about coefficient size is not gathered since the size of the representation of finite field elements is bounded and is small in our case (for the field of rationals or rational function fields, gathering this information can be important to understand how coefficient explosion affects the computation).

3.5.2 Experiments

Three basic experiments have been done using the prototype. The first experiment is a comparison of the different strategies for a small set of 'real' problems (provided by Dr. Green). The second experiment compares the set reduction strategies using a randomly generated instance for which the Gröbner basis is (effectively) infinite. The third experiment compares triple elimination strategies for problems that are specifically designed to produce unnecessary overlaps.

For all of the experiments, each of the prototype configurations was built using the SML of New Jersey compiler (version 0.93) on an IBM RS/6000 model 530H running AIX 3.2. Each problem instance was run over the network with no other users on the system. No repetitions were done since other experiments showed that the variance of observed times is very small for repetitions under these conditions.

3.5.2.1 General Comparison

The following experiments are a comparison of most of the combinations of strategies for selection, triple elimination, and set reduction.

Input Instances The selection strategies considered are shortest and normal, and both forms of set reduction are considered. For elimination, the eager, lazy, and hybrid strategies are compared, as is the algorithm with no elimination. The combination of these strategies is not complete, since the hybrid strategy was not run with redundant element deletion, and the algorithm with no elimination was only run using the normal selection strategy.

Each algorithm was run with problem instances from nine problems. (More kinds of problems were considered initially, but the computations for some are too trivial to distinguish between the different algorithms.) The problems are divided between five related problems in a free algebra,

$$
\begin{aligned}
aa + 5ab + 7ac + 11ba + 2bb + 31bc + 19ca + 13cb + 23cc,\\
ab + 5ac + 7ba + 11bb + 2bc + 31ca + 19cb + 13cc,\\
ac + 5ba + 7bb + 11bc + 2ca + 31cb + 19cc,\\
ba + 5bb + 7bc + 11ca + 2cb + 31cc,\\
bb + 5bc + 7ca + 11cb + 2cc,\\
bc + 5ca + 7cb + 11cc,\\
ca + 5cb + 7cc,\\
cb + 5cc
\end{aligned}
$$

Figure 3.16: Generic Quadratic Relations for Free Algebra Instances.

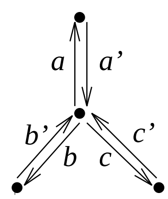

Figure 3.17: Example Graph for Mesh Algebra.

three instances from mesh algebras, and one other problem. These problems are not representative of a wide range of problems, but can, for some orders, be difficult to compute.

The free algebra problems are size k subsets (taken as prefixes of the list, for $k = 4, \ldots, 8$) of the generic quadratic relations over three variables $\{a, b, c\}$ as shown in Figure 3.16. These free algebras problems are identified as Ak where k is the size of the generating set.

Generators for mesh algebras are determined by the graph. The graph must have pairs of inverse arrows between adjacent vertices (if an arc exists, so must its inverse). There is one generator for each vertex in the graph. This generator is the sum of all length 2 paths from the vertex to itself (so each path is an arc and its inverse). As an example, the generators for the graph in Figure 3.17 are $\{aa' + bb' + cc', a'a, b'b, c'c\}$.

The mesh algebra instances used in the experiment are based on binary trees. The graph for instance BT7 is a binary tree with seven nodes, and the graph for BT31 is a binary tree with thirty-one nodes. Instance M39 is a modification of BT31, where the graph has an additional eight vertices used to create cycles by adding arcs to the eight pairs of leaves.

The last input instance is P5, which is a Froebenius algebra [50]. The exact problem instance is given in Appendix C.

Each of these individual problems is combined with each of eight orders to form an input instance. The orders considered are length lexicographic (l), length reverse lexicographic (lr), length vector lexicographic (lv), length reverse vector lexicographic (li), length reverse right vector lexicographic (lri), vector lexicographic (v), and right vector lexicographic (i). The total number of cases for the experiment is 864.

The free and mesh algebra problems are all homogeneous and are run with a degree bound of six. The P5 instance has a finite Gröbner basis and so is run without a bound.

Results The prototype provides both count and timing information from each run. The results from the experiments is given in Appendix E in individual tables for each problem. The count results are given in Tables E.1–E.9 and the timing results are given in Tables E.10–E.18. The count results only include the number of reductions, the number of reductions to zero, and the maximum cardinalities of the triple set and basis. The other counts are not common to every algorithm and so are not included.

Analysis A simple ranking method is used to summarize the results. Each table is first partitioned into groups of instances that have the same values for each of the four observed counts. These groups are sorted lexicographically by the number of reductions, the number of zero reductions, the triple set cardinality and then the basis cardinality. The groups are then assigned a rank based on their order in the sequence of groups, which gives each combination of algorithms a rank for each input. Finally, the ranks are averaged for each combination of algorithms. Table 3.1 shows the ranking and average ranks for problem A4.

The average ranks for the free algebra instances are given in Table 3.2, the average ranks for the mesh algebra instances are given in Table 3.3, and the average ranks for the P5 instance are given in Table 3.4. The selection strategy divides each table into two parts, with shortest being best. Also, the average rank values are equal for the different forms of elimination when shortest selection is used, but not for normal selection.

The next obvious partition is how the three elimination strategies further divide the two groups. In both groups, they occur ranked eager first, hybrid second, lazy third, and no elimination last. (Note also that this ranking mirrors the pattern between these strategies in Table E.9 which suggest the ranking method isn't masking any major effects.) The relationship between set reduction strategies is not as clear, although for most problems element reduction is ranked better than element

Table 3.1: Ranking of Algorithms by Counts for Problem A4 (Part One).

Configuration	Order	Rank	Avg Rank
Eager, Deletion, Normal	l	1	7
	lr	1	
	li	4	
	lv	4	
	lri	7	
	lrv	7	
	v	12	
	i	13	
Eager, Deletion, Shortest	l	1	4
	lr	1	
	i	4	
	li	4	
	lv	4	
	v	4	
	lri	7	
	lrv	7	
Eager, Reduction, Normal	l	1	8.5
	lr	1	
	li	4	
	lv	4	
	lri	7	
	lrv	7	
	v	14	
	i	16	
Eager, Reduction, Shortest	l	1	4
	lr	1	
	i	4	
	li	4	
	lv	4	
	v	4	
	lri	7	
	lrv	7	

Table 3.1: Ranking of Algorithms by Counts for Problem A4 (Part Two).

Configuration	Order	Rank	Avg Rank
Hybrid, Reduction, Normal	l	2	10
	lr	2	
	li	5	
	lv	5	
	lri	8	
	lrv	8	
	v	15	
	i	18	
Hybrid, Reduction, Shortest	l	2	5
	lr	2	
	i	5	
	li	5	
	lv	5	
	v	5	
	lri	8	
	lrv	8	

deletion.

The time data reveals a slightly different ranking. Using the same method, the average rank of the algorithms determined by the times is given in Table 3.5, Table 3.6 and Table 3.7. Here the hybrid approach to elimination wins over both eager and lazy elimination. The likely cause of this is the way that eager elimination is implemented in the prototype; the count behavior of the eager and hybrid strategies is considered again below.

Looking at the tables for this experiment in the appendix and Table 3.1, there is a strong relationship between the groups and the orders. The ranks assigned to individual combinations of algorithms and orders are grouped primarily by order rather than algorithm. Note that the length orders, in particular, have the highest rank in all cases in Table 3.1. The choice of order is considered in Chapter 5, but it is important to note here how the count results are uniform with respect to the order. This suggests that orders have a stronger influence over the count behavior than the algorithm (this is consistent with the folklore for commutative Gröbner bases).

3.5.2.2 Set Reduction

To address the question of which form of set reduction is better, we consider one larger problem. The problem instance is randomly generated using the algorithms described in Appendix C. The

Table 3.1: Ranking of Algorithms by Counts for Problem A4 (Part Three).

Configuration	Order	Rank	Avg Rank
Lazy, Deletion, Normal	l	3	12
	lr	3	
	li	6	
	lv	6	
	lri	9	
	lrv	9	
	v	20	
	i	21	
Lazy, Deletion, Shortest	l	3	6
	lr	3	
	i	6	
	li	6	
	lv	6	
	v	6	
	lri	9	
	lrv	9	
Lazy, Reduction, Normal	l	3	11
	lr	3	
	li	6	
	lv	6	
	lri	9	
	lrv	9	
	v	17	
	i	19	
Lazy, Reduction, Shortest	l	3	6
	lr	3	
	i	6	
	li	6	
	lv	6	
	v	6	
	lri	9	
	lrv	9	

Table 3.1: Ranking of Algorithms by Counts for Problem A4 (Part Four).

Configuration	Order	Rank	Avg Rank
None, Deletion, Normal	l	10	17.5
	li	10	
	lr	10	
	lv	10	
	lri	11	
	lrv	11	
	i	22	
	v	25	
None, Reduction, Normal	l	10	17
	li	10	
	lr	10	
	lv	10	
	lri	11	
	lrv	11	
	v	23	
	i	24	

Table 3.2: Average Rank of Algorithms for Free Algebra Instances.

Elimination	Set Reduction	Selection	A4	A5	A6	A7	A8
Eager	Deletion	Shortest	4	4	2.5	1	1
Eager	Reduction	Shortest	4	4	2.5	1	1
Hybrid	Reduction	Shortest	5	5	3	1	1
Lazy	Deletion	Shortest	6	6	4	2	2
Lazy	Reduction	Shortest	6	6	4	2	2
Eager	Deletion	Normal	7	11	6.5	2	1
Eager	Reduction	Normal	8.5	9.5	4.5	2	1
Hybrid	Reduction	Normal	10	11	4.5	2	1
Lazy	Reduction	Normal	11	12.5	5.5	3	2
Lazy	Deletion	Normal	12	13.5	7.5	3.5	2
None	Reduction	Normal	17	17	8.5	6.5	3
None	Deletion	Normal	17.5	17.5	9.5	7	3

Table 3.3: Average Rank of Algorithms for Mesh Algebra Instances.

Elimination	Set Reduction	Selection	BT7	BT31	M39
Eager	Deletion	Shortest	13	17	7
Eager	Reduction	Shortest	13	17	7
Hybrid	Reduction	Shortest	14.5	19	8
Lazy	Deletion	Shortest	15	20.5	8.5
Lazy	Reduction	Shortest	15	20.5	8.5
Eager	Deletion	Normal	16.5	16.5	6.5
Eager	Reduction	Normal	16.5	16.5	6.5
Hybrid	Reduction	Normal	17	18.5	7.5
Lazy	Reduction	Normal	17.5	21.5	9
Lazy	Deletion	Normal	18	21.5	9
None	Reduction	Normal	21	13.5	10.5
None	Deletion	Normal	22.5	13.5	10.5

Table 3.4: Average Rank of Algorithms by Counts for P5 Instance.

Elimination	Set Reduction	Selection	P5
Eager	Deletion	Shortest	10
Eager	Reduction	Shortest	10
Hybrid	Reduction	Shortest	12
Lazy	Deletion	Shortest	14
Lazy	Reduction	Shortest	14
Eager	Deletion	Normal	15.5
Eager	Reduction	Normal	16
Hybrid	Reduction	Normal	17
Lazy	Reduction	Normal	18.5
Lazy	Deletion	Normal	19
None	Reduction	Normal	36
None	Deletion	Normal	36.5

Table 3.5: Average Rank of Algorithms by Time for Free Algebra Instances.

Elimination	Set Reduction	Selection	A4	A5	A6	A7	A8
Hybrid	Reduction	Shortest	2.5	1.5	1.5	1	1
Lazy	Deletion	Shortest	3	1.5	1.5	1	1
Lazy	Reduction	Shortest	3	1.5	1.5	1	1
Eager	Deletion	Shortest	4	1.5	1.5	1	1
Eager	Reduction	Shortest	4.5	1.5	2	1	1
None	Reduction	Normal	9	6	3	1	1
None	Deletion	Normal	10.5	9	5	1	1
Eager	Deletion	Normal	12.5	6.5	5	1	1
Lazy	Deletion	Normal	13	7.5	3.5	1	1
Eager	Reduction	Normal	13	7.5	5.5	1	1
Hybrid	Reduction	Normal	13.5	4.5	4	1	1
Lazy	Reduction	Normal	13.5	5.5	3.5	1	1

Table 3.6: Average Rank of Algorithms by Time for Mesh Algebra Instances.

Elimination	Set Reduction	Selection	BT31	BT7	M39
Lazy	Reduction	Normal	22.5	2	11
Lazy	Reduction	Shortest	23	2	12.5
Hybrid	Reduction	Normal	23.5	2	13.5
Hybrid	Reduction	Shortest	24	2	12
Lazy	Deletion	Normal	24.5	2	14
Lazy	Deletion	Shortest	25	2	14
None	Deletion	Normal	26	8	15
None	Reduction	Normal	28	9	16
Eager	Deletion	Shortest	30	2.5	10.5
Eager	Reduction	Shortest	31	2.5	11
Eager	Deletion	Normal	32	3	11.5
Eager	Reduction	Normal	33	3	13

Table 3.7: Average Rank of Algorithms by Time for P5 Instance.

Elimination	Set Reduction	Selection	P5
Hybrid	Reduction	Shortest	4
Hybrid	Reduction	Normal	6
Eager	Deletion	Shortest	11
Eager	Reduction	Shortest	11
Eager	Deletion	Normal	14
Eager	Reduction	Normal	14.5
Lazy	Deletion	Shortest	27.5
Lazy	Reduction	Shortest	27.5
None	Reduction	Normal	28
None	Deletion	Normal	29.5
Lazy	Reduction	Normal	31
Lazy	Deletion	Normal	32

Table 3.8: Observations for Comparison of Set Reduction Techniques.

Algorithm	Total	Zero	Triple Set	Basis	Time	Complete
Element Deletion	624	224	260	2413	30 hours	No
Element Reduction	726	475	822	904	36 hours	Yes

problem is instance GL in Appendix D (see the Appendix for the parameters for generation). The graph has 30 nodes and 457 arcs, and the generating set has 1940 inhomogeneous elements (the input file requires 1.5 Megabytes of disk space). Only the right vector lexicographic order is used, along with the shortest selection strategy. The problem was run using eager elimination combined with both redundant element deletion and element reduction. The problem was run with a bound of 400 nonzero reductions.

The results are given in Table 3.8, which includes reduction counts, elimination counts, the maximum cardinalities and approximate computation time. The table also shows whether the result is the entire Gröbner basis.

This experiment illustrates a phenomenon where using redundant element deletion fails to find a finite Gröbner basis that is found by using redundant element reduction. The difference between the number of reductions and the number of zero reductions is exactly 400 for the redundant element deletion observation, and less than 400 for the redundant element reduction observation. This suggests that the reduction of triples for redundant elements may not necessarily go to zero. Also, note that the maximum cardinality of the working generating set for redundant element deletion is more than twice the size of that for redundant element reduction.

This result shows only that redundant element deletion is not suitable to be used in conjunction with the bounded algorithm. It is not clear that element reduction is better in general. However, the fact that the overlap set for element deletion is more than three times the size of the set for element reduction is a good indication of the space overhead involved. More experimentation is required to give a definitive answer, but the first set of experiments indicate that other factors have more influence on the computation.

3.5.2.3 Elimination Strategy

Finally, the three triple elimination strategies are compared again. The real question is how to choose between the eager and hybrid strategies.

For this experiment, seven problem instances were run. The problems are BT7, BT31, A4 and P5 from before, and three new problems DCYC, ELP and ICYC that are designed to cause eliminations to occur. The DCYC, ELP, and ICYC instances are given in Appendix D. They all include a set of elements whose tips overlap and subsequent terms are chosen to form new overlaps and divide others.

All seven problems are run using the length lexicographic order defined in terms of the default alphabetic order determined by their specification, and also using a randomly chosen alphabetic order. Only shortest selection was used. The count results are given in Tables 3.9–3.15.

Note that for most problems, there seems to be no (or little) difference between the count observations for eager and hybrid elimination. To determine whether the difference between the results for the different techniques is truly significant, we compute the 95% confidence intervals of the differences between count values for the three algorithms and for each problem. In particular, we consider the maximum triple set cardinality and the total number of triple eliminations (both during initialization and the computation). For the count of triples eliminated, results for five of the seven problems show that the difference between the eager and hybrid is not significant. However, for problems for which eager and hybrid are not significantly different, none of the approaches are distinguished. For the maximum triple set cardinality, three of the seven problems show that eager and hybrid are not significantly different.

These results suggest that eager elimination does perform better than the hybrid approach in

Table 3.9: Counts for Comparison of Elimination Strategies for Problem BT31.

Algo.	Order	Total	Zero	Elims	Max Triple	Max Basis
Eager	Default	368	212	192	82	147
Lazy	Default	368	212	192	209	147
Hybrid	Default	368	212	192	82	147
Eager	Random	368	212	192	82	147
Lazy	Random	368	212	192	209	147
Hybrid	Random	368	212	192	82	147

Table 3.10: Counts for Comparison of Elimination Strategies for Problem BT7.

Algo.	Order	Total	Zero	Elims	Max Triple	Max Basis
Eager	Default	10	5	0	5	12
Lazy	Default	10	5	0	5	12
Hybrid	Default	10	5	0	5	12
Eager	Random	82	50	24	30	27
Lazy	Random	82	50	24	47	27
Hybrid	Random	82	50	24	30	27

general. However, only for the DCYC instance is the average difference for these counts over 10. So, from a practical perspective, the difference between the approaches may not be enough to justify the extra space overhead required for an efficient implementation of eager elimination.

3.5.3 Implications for Implementation

Although the experimentation discussed above is not exhaustive, it does suffice to make some choices for implementation. The best configuration of algorithms appears to be shortest selection, hybrid elimination, and redundant element reduction.

Table 3.11: Counts for Comparison of Elimination Strategies For Problem DCYC.

Algo.	Order	Total	Zero	Elims	Max Triple	Max Basis
Eager	Default	198	125	479	87	75
Lazy	Default	198	125	262	516	75
Hybrid	Default	198	125	460	105	75
Lazy	Random	208	140	279	390	75
Eager	Random	218	146	540	94	79
Hybrid	Random	218	146	514	125	79

Table 3.12: Counts for Comparison of Elimination Strategies For Problem ELP.

Algo.	Order	Total	Zero	Elims	Max Triple	Max Basis
Eager	Default	45	38	6	28	13
Lazy	Default	45	38	10	73	13
Hybrid	Default	45	38	6	30	13
Eager	Random	93	79	25	55	20
Lazy	Random	93	79	32	132	20
Hybrid	Random	93	79	27	56	20

Table 3.13: Counts for Comparison of Elimination Strategies for Problem ICYC.

Algo.	Order	Total	Zero	Elims	Max Triple	Max Basis
Eager	Default	29	20	17	13	16
Lazy	Default	29	20	21	32	16
Hybrid	Default	29	20	18	15	16
Eager	Random	54	38	42	25	21
Lazy	Random	54	38	47	59	21
Hybrid	Random	54	38	43	28	21

Table 3.14: Counts for Comparison of Elimination Strategies For Problem A4.

Algo.	Order	Total	Zero	Elims	Max Triple	Max Basis
Eager	Default	48	38	39	38	14
Lazy	Default	48	38	39	80	14
Hybrid	Default	48	38	39	45	14
Eager	Random	48	38	39	38	14
Lazy	Random	48	38	39	80	14
Hybrid	Random	48	38	39	45	14

Table 3.15: Counts for Comparison of Elimination Strategies for Problem P5.

Algo.	Order	Total	Zero	Elims	Max Triple	Max Basis
Eager	Default	69	43	47	22	41
Lazy	Default	69	43	47	301	41
Hybrid	Default	69	43	47	27	41
Eager	Random	65	41	59	27	39
Lazy	Random	65	41	59	284	39
Hybrid	Random	65	41	59	28	39

As discussed above, the choice between hybrid and eager elimination is not one of choosing the optimal approach. Hybrid elimination misses some possible eliminations, but eager elimination requires more space for efficient implementation. Since, the difference between the two strategies is generally not large, hybrid elimination appears to be the best choice.

The choice of redundant element reduction is based on the one problem instance for which redundant element deletion performs so badly. For finite and homogeneous input, the choice is not so clear. However, it does appear that element reduction does require less overhead in general.

The Opal system is implemented using this configuration, but allows the user to choose selection strategy (the choice is between normal, shortest, and minimum weight using a weight order).

Chapter 4

Pattern Matching

Pattern matching problems arise naturally in the computation of Gröbner bases in noncommutative algebras. The algorithm repeatedly tests whether one polynomial term divides another, or if two terms have a nontrivial common multiple. Because terms of noncommutative polynomials are words in a free semigroup, these tests are pattern matching searches with a dictionary of strings. The dictionary D is the set of tips of the generating set. In this chapter, the dynamic dictionary matching approach of Amir *et al.* [2] is extended to solve the pattern matching problems involved in the noncommutative Gröbner basis computation.

A similar situation occurs in the closely related computation of Knuth-Bendix completion for string rewriting (see [10]). There the dictionary is the set of left-hand sides of rewrite rules, but the pattern matching problems are identical. Although a static form of dictionary matching is used in string rewriting, the dynamic technique developed here has not been used previously in string rewriting or for similar applications. The approach described is significant because it dramatically improves the time required for the pattern matching involved in the Gröbner basis and string completion computations.

The chapter is organized as follows. Section 4.1 identifies the pattern matching problems involved in the computation of Gröbner bases. Then Section 4.2 discusses the similar problems of matching for Knuth-Bendix completion for string rewriting and dynamic dictionary matching. Section 4.3 presents the suffix tree data structure used for the solution, and the searches are presented in 4.4. Finally, section 4.5 is a brief summary.

4.1 Pattern Matching in the Gröbner Basis Computation

The Gröbner basis computation is dominated by operations that use pattern matching. These operations are overlap computations to form triples, polynomial reduction, set reduction, and triple elimination. Both reduction and overlap computation occur at every iteration of the algorithm, and it is important to make sure that these operations are as fast as possible.

Recall (from Section 2.2) that the tip of a polynomial p is denoted $tip(p)$, and the set of tips of a set of polynomials P is denoted $Tip(P)$. Also, a triple $t = \langle p, q, v \rangle$ is a pair of polynomials p, q together with an overlap v of their tips. The overlap determines a common multiple of the tips denoted $cm(t)$.

The operations in the Gröbner basis computation that use pattern matching are the following.

Computing overlaps. Overlaps are computed when a new nonzero, reduced overlap relation h is added to G. An overlap of a word p with a word q is a nonempty suffix of p that is also a prefix of q. The overlaps to be computed are all those of $tip(h)$ with $Tip(G \cup \{h\})$.

Polynomial reduction. Polynomial reduction of a polynomial q by a set of polynomials P requires finding all $p \in P$ such that $tip(p)|m$ for some term m of q. For tip-reduction, $m = tip(q)$.

Set reduction. Set reduction is performed when a new nonzero reduced overlap relation h is added to G. Since set reduction removes all elements of G that are tip-reducible by h, the goal is to find all $g \in G$ such that $tip(h)|tip(g)$.

Triple elimination. Triple elimination is performed when the triples formed by a new element h' and G are added to T. For each new triple s, first test that there is no $t \in T$ such that $cm(t)|cm(s)$; if not, remove all $t \in T$ such that $cm(s)|cm(t)$.

Both the set $Tip(G)$ of tips of G and the set of common multiples C for the triples can be viewed as dictionaries of words on which searches must be performed. Only three types of pattern matching searches are needed for the four operations described. Overlap searches are (obviously) used to compute overlaps; subword searches are used for polynomial reduction and triple elimination; and superword searches are used for both set reduction and triple elimination. In addition to fast algorithms for these searches, a pattern matching mechanism must allow fast insertions to and deletions from the dictionary. Our extension of dynamic dictionary matching satisfies these requirements.

Note that G is tip-reduced and so has the property that no tip of an element of G is a subword of another tip. Therefore, it is safe to assume that the dictionary of words D has this property. In particular, for all $d \in D$, there is no d' such that $d'|d$. This assumption allows a significant simplification in the data structures of Amir *et al.* [2].

If w is a string not in D, the subword and superword searches for w are the following:

1. *Subword search* – find all (d, i) where $d \in D$ and d is a subword of w beginning at the ith symbol of w.

2. *Superword search* – find all (d, i) where $d \in D$ and w is a subword of d beginning at the ith symbol of d.

The overlap searches are split into two kinds. Again, w is not a subword of any element of D.

1. *Left-overlap search* – find all (d, s) where $d \in D$ and s is a suffix of d and a prefix of w.

2. *Right-overlap search* – find all (d, p) where $d \in D \cup \{w\}$ and p is a prefix of d and a suffix of w.

The suffix-tree insertion algorithm used makes it convenient to have the search of overlaps of w with itself to be a right-overlap search. If another insertion algorithm is used, then it may be better to have the search for self-overlaps be a left-overlap search.

4.2 Related Problems

As noted in Chapter 2, the computation of Gröbner bases is similar to Knuth-Bendix completion for term-rewriting and string rewriting. In term rewriting the corresponding matching problems are to find left-hand sides of rules that unify with terms to be reduced and finding critical pairs for pairs of rules. Completion in term rewriting uses discrimination trees to match terms and to find critical pairs [23].

Other (simpler) structures are used for matching in string rewriting. Sims [52] defines *index automata* that can be used for the pattern matching problems in completion of string rewriting systems. These matching problems are the same as for the Gröbner basis computation; however, Sims suggests using the dictionary matching approach of Aho and Corasick [1]. The dictionary matching problem is to find all occurrences of all words in a given dictionary in a search string (this is the same as the subword problem described in the previous section). Aho and Corasick show how to compute an automaton for the dictionary that can be used for the search. This data structure cannot be modified and so the addition and deletion of elements of the dictionary is linear in the size of the dictionary. Therefore, the approach of Aho and Corasick is not suited for use in the Gröbner basis computation (or Knuth-Bendix completion). The result is that index automata are too expensive when implemented this way. In the string rewriting system KBMAG the index automata are used for testing confluence, but not for general completion since they are too expensive to rebuild when new words are added to the dictionary [33].

Amir *et al.* [2] introduce the dynamic dictionary matching approach as a dynamic alternative to Aho and Corasick's static approach. Instead of using a static data structure, Amir *et al.* implement the search automaton using a suffix tree. The suffix tree allows additions to and deletions from the dictionary in time linear in the size of the added word.

In the general situation considered by Amir *et al.*, one dictionary word may be a subword of another. In this case, the longer word is hidden by the shorter word, and so an additional data structure is required to keep track of the subword relationships. However, this situation does not occur for the Gröbner basis computation as long as the set of generators is tip-reduced. As a consequence, the additional data structure is not needed in our application.

4.3 Suffix Trees and Dictionary Matching

Dynamic dictionary matching is implemented using suffix trees. A suffix tree is a compacted trie (an edge may be labeled by an arbitrary length word) for the suffixes of a word. Suffix trees can be defined as follows.

Definition 4.1 *Let s be a string of length m terminated by a unique symbol that appears nowhere else in s. The* suffix tree *for s is the rooted tree with a leaf for each nonempty suffix and at most $m-1$ internal nodes and satisfying:*

1. *each edge of the tree is labeled by a substring of s, and each node corresponds to a position in s;*
2. *no two sibling edges have labels with a common nonempty prefix; and*

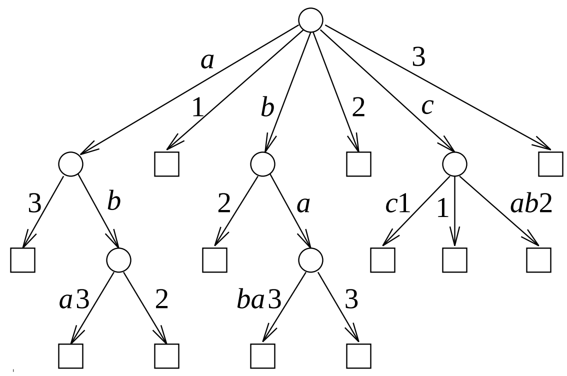

Figure 4.1: Dictionary Suffix Tree.

3. *each leaf corresponds to a particular suffix of s given by the concatenation of the labels from the root to the leaf [25, pp.182–183].*

Note that each node v in a suffix tree corresponds to a unique string $l(v)$ formed by concatenating the labels of the edges from the root to v. For each v, the label $l(v)$ is a prefix of the suffixes represented by all leaves of the subtree rooted at v. Given a word s such that $s = l(v)$ for some node v of a suffix tree, then v is called the *locus* of s [42]. Note that for a word of length m, there are m leaves of the suffix tree, and the size of the tree is bounded by $2m - 1$.

For dictionary matching, the suffix tree holds the suffixes for the individual dictionary words. If the dictionary is $\{w_1, \ldots, w_n\}$, the suffix tree is isomorphic to the suffix tree for the word formed by concatenating the words w_i separated by special symbols outside of the alphabet. Numeric symbols are sufficient here, so the tree for a dictionary of n words corresponds to the tree for the string $w_1 1 w_2 2 \cdots n-1 w_n n$. For example, the tree for the dictionary $\{cc, cab, baba\}$ shown in Figure 4.1 corresponds to the suffix tree for $cc1cab2baba3$ (where any part of the suffix past the first digit has been deleted). The tree is actually built by inserting each dictionary word in sequence.

The insertion algorithm and data structure used for the suffix tree are based on that of McCreight [42]. McCreight's algorithm inserts the suffixes from the longest to the shortest, and uses suffix links in the tree so that insertion requires only linear time. A *suffix link* is an extra arrow from a node v to the node labeled by the suffix of $l(v)$ formed by removing the first symbol. (Only internal nodes have suffix links.) The tree for the dictionary $\{cc, cab, baba\}$ with suffix links added is shown in Figure 4.2.

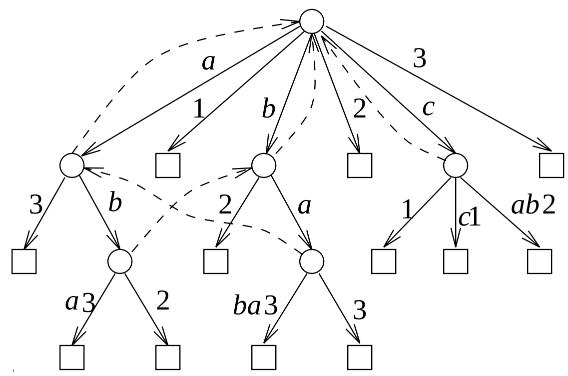

Figure 4.2: Dictionary Suffix Tree with Suffix Links.

McCreight's algorithm is based on the fact that prefixes of subsequent suffixes are related. Let w be a word and w_k be the length k suffix of w. Define $head(w_k) = s$, where s is the longest prefix of w_k that is also a prefix of w_l for some l satisfying $|w| > l > k$. McCreight [42] shows that if $head(w_{k+1}) = a\alpha$ for a symbol a and a possibly empty word α then α is a prefix of $head(w_k)$. The implication for the algorithm is that if the locus of $head(w_{k+1})$ is given, then the common prefix with w_k does not have to be scanned.

The algorithm for inserting each suffix is given by Amir *et al.* [2] (they call it procedure *STI*). The arguments to the algorithm are the locus v of the previous head $head(w_{k+1})$ and the current suffix w_k. The locus v is used to determine whether the previous head is empty or not, which determines how to insert the suffix. Basically the algorithm has the goals of inserting each suffix and updating the suffix link for the head of each suffix. The insertion algorithm is given in Appendix B.

Each step of the construction of the suffix tree for the dictionary $\{cc, cab, baba\}$ is shown in Figures 4.3–4.5. Figure 4.3 shows the suffix tree before the insertions and after the insertion of each suffix of *cc*. In each tree, a leaf (the black node) is added, a suffix link for the previous head is added, and possibly new interior nodes are added. Figure 4.4 shows the insertion of *cab* into the tree for *cc*, and Figure 4.5 and Figure 4.5 show the insertion of *baba* into the tree for $\{cc, cab\}$. These figures do not illustrate the use of suffix links, but do show how later suffixes are folded into the tree, possibly introducing new interior nodes. Deletion is done in a similar fashion by removing suffixes from longest to shortest. Deletion may remove nodes and merge arcs when a child node is deleted.

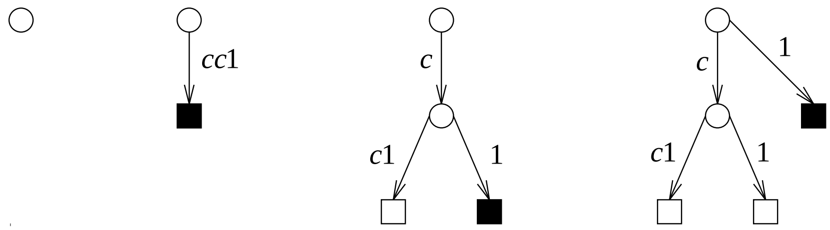

Figure 4.3: Construction of Suffix Tree for *cc*.

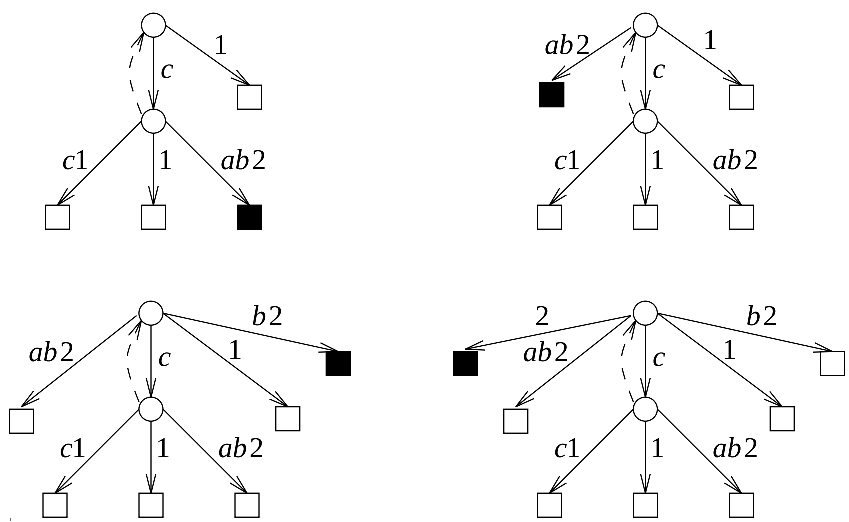

Figure 4.4: Extension of Suffix Tree for *cc* by Inserting *cab*.

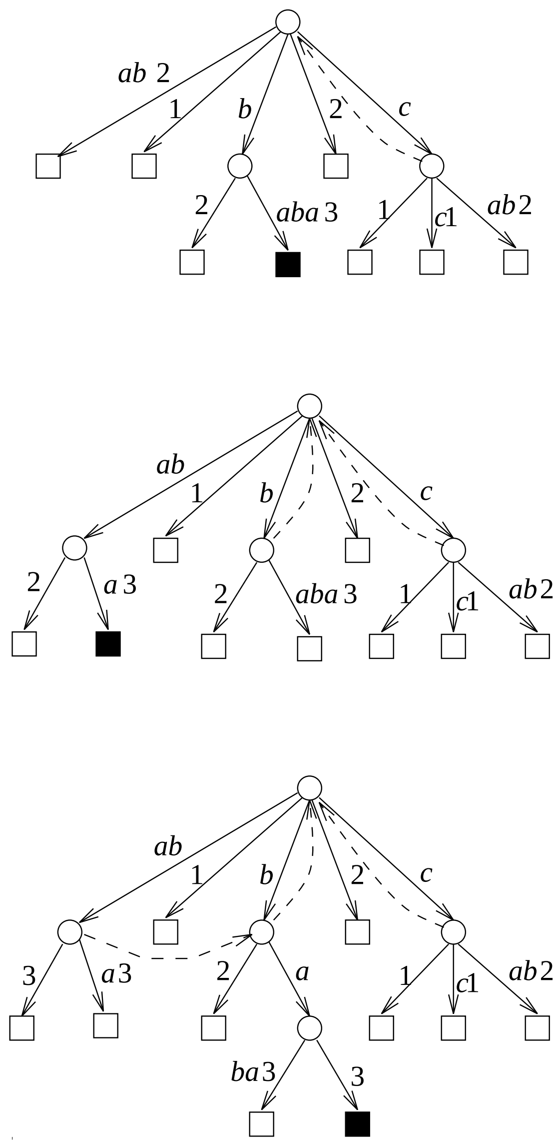

Figure 4.5: Extension of Suffix Tree for $\{cc, cab\}$ by Inserting $baba$ (Part One).

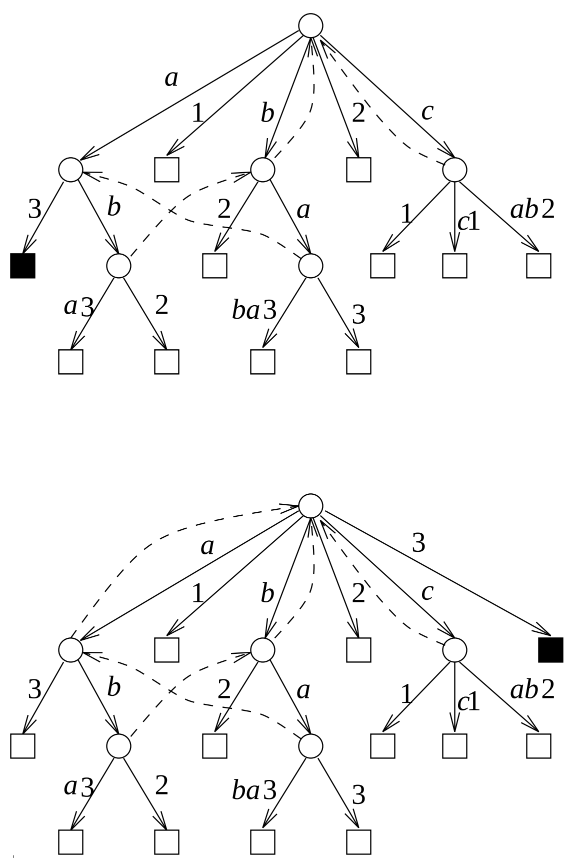

Figure 4.5: Extension of Suffix Tree for $\{cc, cab\}$ by Inserting *baba* (Part Two).

Two modifications are made to the suffix tree. The first is to distinguish between nontrivial suffixes and full dictionary words. To do this, define *pattern* leaves to correspond to full dictionary words, and *suffix* leaves to correspond to any other suffix. This modification is mostly conceptual since in practice each leaf is labeled to identify the suffix and the word to which it corresponds.

The second modification to the suffix tree is to the nodes of the tree. A counter is added to each node v that indicates the number of pattern leaves in the subtree rooted at v. This counter helps to restrict the superword search as discussed in Section 4.4.1. The insertion operation is changed so that when a full pattern word w is inserted, the counter for each existing node along the path to the leaf for w is incremented as the scan passes it. Also, when a new interior node is created by splitting an arc, the counter of the new node is set to the value of its children. The deletion operation is changed to decrement the counters during the search for the leaf of the full word. The modified insertion and deletion operations are still accomplished in linear time. Another modification of the insertion algorithm is needed for computing overlaps and is described below.

4.4 Pattern Matching Solution

The dynamic dictionary matching approach of Amir *et al.* [2] supports linear time modifications to the dictionary, but only implements the subword search. In particular, they show how to use a suffix tree to implement the automata for the subword search. We describe how the suffix tree structure also supports the superword and overlap searches.

Each search *scans* its input with the suffix tree. Scanning is the operation of comparing substrings of the word to labels of the edges. For example, to scan the word *cab* by the tree in Figure 4.2, begin at the root, match c with the arc labeled c and then match ab to the arc labeled ab. Scans fail if none of the labels for the child arcs of the current node match a prefix of the remaining input word. As an example, a scan of the word *bac* by the tree in Figure 4.2 fails because there is no arc from the locus of ba that is labeled by c. Each step of the scan requires finding the edge from the current node whose label matches the remaining substring. The cost of this operation depends on the implementation of the suffix tree, but in the worst case takes time on the order of the number of letters in the alphabet (which is constant in practice because the unique termination symbols are not really needed). So, regardless of implementation, scanning a word w takes time $O(|w|)$.

The time complexity of each search is of the form $O(n+m)$, where n is the size of the input and

m is the size of the output. The input size is $n = |w|$ in all searches, but the size of the output m depends on the search. In particular, the result by Amir *et al.* for the subword search [2] implies the following. Let $\#(d, w)$ is the number of times d occurs in w.

Theorem 4.1 *The subword search using the suffix tree takes time $O(n + m)$ for which $n = |w|$, and $m = \sum_{d \in D} \#(d, w)$.*

4.4.1 Superword Search

The superword search takes as input a word w that is not an element of D, and returns as output the set of pairs (d, i) where $d \in D$ and w divides d beginning at position i. This problem is equivalent to finding all suffixes of $d \in D$ that have w as a prefix. The length of the suffix indicates the position where w divides d, and therefore determines the pair.

The algorithm then is to scan w with the suffix tree for D to find the locus of w. If the scan fails, then w is not a subword of any dictionary word. Otherwise, the locus v of w exists in the tree. In this case, return all leaves of the subtree rooted at v.

Theorem 4.2 *The superword search of D by w takes time $O\left(|w| + \sum_{d \in D} \#(w, d)\right)$.*

Proof Scanning w takes time $O(|w|)$, and the search of the subtree takes time proportional to the size of the subtree. The size of a suffix tree is on the order of the number of leaves, which for the subtree is the size of the output $\sum_{d \in D} \#(w, d)$. $\square$

4.4.2 Left-overlap Search

The left-overlap search takes as input a word w, where w is not in D and is not a subword of any d in D, and returns the set of pairs (d, k) where k is the length of a suffix of d that is a proper prefix of w. Viewed the other way, this problem is to find any proper prefix of w that is also a suffix of some d in D.

The algorithm is to scan w with the suffix tree for D. If during the scan a node is reached that has a suffix leaf as a child, then the corresponding suffix should be returned. Stop if the scan fails, or the $(|w| - 1)$th symbol of w is reached.

Theorem 4.3 *The search for left-overlaps of w with D takes time $O(|w| + m)$ where the size of the output is $m = \sum_{d \in D} \sum_{k=1}^{\min(|d|,|w|)} p(d_k, w)$ in which d_k is the length k suffix of d, and $p(d_k, w)$ is a function which returns one if d_k is a prefix of w and zero otherwise.*

Proof Scanning w with the tree takes time $O(|w|)$, and the number of suffix leaves visited is the size of the output. The size of the output is the number of suffixes of words in D that are prefixes of w, or

$$m = \sum_{d \in D} \sum_{k}^{\min(|d|,|w|)} p(d_k, w).$$

□

4.4.3 Right-overlap Search

The right-overlap search takes as input a word w, where w is not in D and is not a subword of any d in D, and returns the set of pairs (d, k) where k is the length of a prefix of d ($d \in D \cup \{w\}$) that is a proper suffix of w. This search is combined with the insertion algorithm.

The insertion algorithm inserts the suffixes of w into the tree from longest to shortest. This means that w is in the tree before any other suffix of w is inserted. Therefore, when a proper suffix w_k of w is inserted into the tree, any d from $D \cup \{w\}$ with w_k as a prefix is a pattern leaf in the subtree rooted at the locus of w_k.

The algorithm uses this fact, and after each insertion of a proper suffix w_k searches the subtree rooted at the locus of w_k for all pattern leaves. Recall that each node is marked with the number of pattern leaves in its subtree. The algorithm checks each child and searches a subtree only if its root has a nonzero number of pattern leaves. The only modification required to the insertion algorithm is that it return the locus of the suffix inserted so that the subtree can be searched.

Theorem 4.4 *The insertion algorithm together with the right-overlap search takes time $O(|w|+m)$ where m is the sum of the lengths of the words that overlap with w.*

Proof The insertion algorithm alone takes $O(|w|)$ time with each suffix insertion taking amortized constant time [2]. The addition of the subtree search does not affect the asymptotic time for inserting each suffix. During the search for each insertion, the restriction of the search to subtrees with pattern leaves takes time bounded by $(c-1)|d|$ for each dictionary word d corresponding to a pattern leaf in the subtree where c is the number of symbols in the alphabet. Since c is a constant, the total time required for searching is $O(m)$ where m is the sum of the lengths of the dictionary words in the answer. □

4.5 Summary

The extended form of dynamic dictionary matching is a perfect solution to the pattern matching problems in the Gröbner basis computation. This approach provides both fast searches (linear in the size of the input and output) and fast modifications to the dictionary. The ability to quickly modify the dictionary is crucial to Gröbner basis computations where the dictionary of patterns changes frequently.

Although our dictionary matching approach is similar to the static approach to string rewriting defined by Sims, ours is the first approach to dictionary matching that can be used in the Gröbner basis algorithm (or Knuth-Bendix completion). Previously, only standard string-matching algorithms such as Knuth-Morris-Pratt or Boyer-Moore have been used. Using these approaches the matching problems each have time-complexity dependent on the size of the dictionary, and so could take much longer than the dictionary matching approach employed here. The suffix tree data structure is implemented in both the prototype described in Chapter 3 and the Opal system.

A possible extension to the described approach would be to combine insertion with deletion of words that are superwords of the inserted word. This would be useful in set reduction, assuming that the operation is faster than the sequence of operations that must currently be used: scanning for superwords, removing each superword individually and inserting the new pattern.

Chapter 5

Admissible Orders

As has been seen earlier, the choice of admissible order is very significant to the computation of Gröbner bases. Not only can the order determine whether the Gröbner basis of a problem instance is finite, but it can also affect the time required to compute a finite basis if it exists. Therefore, it is very important to choose the right order for a given problem instance. This chapter addresses the problem of choosing an admissible order by experimentally comparing a small class of orders over a reasonably sized selection of problem instances. The goal is to develop a ranking that can be used as a guide for the selection of orders for similar problems.

Unfortunately, such a ranking does not appear to exist. The reason is that what order is best for a problem instance is strongly dependent on the characteristics of the instance, and the relationship between the orders does not always imply a simple ranking. Despite this fact, we give a ranking that ignores some of the statistical data that implies that the difference between some orders may not be significant. We also give some guidance for choosing an order based on experience with the experimentation.

The chapter begins with a review of the definition of admissible orders. The second section discusses the experiments conducted to study orders, and the third section analyzes the results. The next chapter (Chapter 6) explores the relationship between problem instances and admissible orders.

5.1 Related Work

Guidance for choosing admissible orders for noncommutative Gröbner basis computations is not available in the literature. For commutative Gröbner bases, an order called *(degree) reverse lexicographic* has been shown to be optimal in the sense that the maximum degree of any tip generated during a computation using the order is minimal among all possible orders [6]. Reeves [47] demonstrates that for some problem instances other orders can do as well as the degree reverse lexicographic order.

The degree reverse lexicographic order corresponds to an order on noncommutative instances (here it is called "length right vector lexicographic"). However, the proof of optimality uses algebraic geometry and does not extend to the noncommutative case. Regardless, it is not clear whether bounding the maximum size of the tip of any element generated during the computation always results in the most efficient computation. Also, this notion of optimality is insufficient for computations of partial Gröbner bases from instances for which the whole Gröbner basis is infinite or is so large as to be effectively infinite.

Another related problem is the choice of order for string rewriting. Experience in string rewriting suggests that the length lexicographic order generally results in good performance [39]. However, for particular classes of problems other orders are better; an example is the recursive path (or wreath product) order for polycyclic groups [52]. Most results about orders on strings prove general properties [38] or classification results [51], but there is no comparable result to the result regarding degree reverse lexicographic for commutative Gröbner bases.

5.2 Definition

Admissible orders in path algebras are not that different from ones in free algebras, but they must take into account the structure of the graph. As before, Γ is a finite directed multigraph (Γ_0, Γ_1) where Γ_0 is the set of vertices, and Γ_1 is the set of arcs. The admissible orders are defined on the set B of finite paths of Γ. Recall that the set B includes all vertices (as length zero paths), all arcs, and all finite walks of Γ. A special element 0 is added to B to represent invalid path compositions. The set $B \cup \{0\}$ is a semigroup with zero where the vertices are idempotents (for v a vertex, $v \cdot v = v$).

The standard kinds of orders are defined as follows. A *partial order* $\leq$ on B is a reflexive, transitive and antisymmetric relation [17]. A *pre-order* $\leq$ on B is a reflexive and transitive relation.

A *total* order on B is a partial order such that all $p, q \in B$ are comparable, meaning $p < q$, $p > q$ or $p = q$. For any partial (total or pre-) order $\leq$, $p < q$ means that $p \leq q$ but $p \neq q$. A partial (total or pre-) order $\leq$ on B is a *well-order* if there no infinite descending chains of elements $p_0 > p_1 > p_2 > \ldots$ from B.

In this chapter, we define admissibility as a property of well-orders.

Definition 5.1 (Admissible Well-Order) *An* admissible well-order $\leq$ *for the set of paths B is a well-order of B satisfying these properties*

1. *If $v \in \Gamma_0$ and $p \in B \setminus \Gamma_0$, then $v < p$.*
2. *If $p, q, r, s \in B$, $p < q$, and $rps \neq 0 \neq rqs$, then $rps < rqs$.*

These properties together imply that if $p|q$ and $p \neq q$ then $p < q$.

If a total order $\leq$ is an admissible well-order, we call $\leq$ an *admissible order*. Pre- and partial orders that are admissible well-orders are called *admissible pre-orders* and *admissible partial orders*. Notice that since a free monoid can be represented by a graph with a single vertex and a loop for each generator, the admissible orders of string rewriting are included in our definition.

All of the admissible orders dealt with in this chapter can be found by the following construction. Let $(M, \leq_m)$ be a monoid M with a total, well-order $\leq_m$ such that the operation of M is monotonic with respect to $\leq_m$ and the unit is minimal (e.g., the order satisfies the analogues of the admissibility conditions). To be precise, the fact that the operation of M is monotonic with respect to $\leq_m$ means that if $a < b$ then $ac < bc$ and $ca < cb$. (The fact that $ac = bc$ and $ca = cb$, when $a = b$ follows from the well-definedness of the operation.) This property implies that M cannot be cyclic, and so must be infinite. In most cases, M will be finitely generated.

Given M, choose a map f of $\Gamma_0 \cup \Gamma_1$ into M such that vertices are mapped to the unit, and arcs are mapped to any non-unit element. Then there is an induced map f^* of B into M given by $f^*(pq) = f^*(p)f^*(q)$. We call f^* an *order embedding* and f an *order map*. Given an order map f on the arcs and vertices, a pre-order $\leq_f$ can be defined on B as $p \leq_f q$ if and only if $f^*(p) \leq_m f^*(q)$. Note that in such pre-orders all pairs of paths are comparable because $\leq_m$ is a total order. Also, note that $f^*(p) = f^*(q)$ does not imply that $p = q$, for a pair of paths p, q.

Lemma 5.1 *A pre-order $\leq_f$ derived from an order map $f : (\Gamma_0 \cup \Gamma_1) \to M$ is admissible.*

Proof We must show that $\leq_f$ is a well-order, and that $\leq_f$ satisfies the admissibility properties.

To see that $\leq_f$ must be a well-order, suppose that there is an infinite descending chain $p_0 >_f p_1 >_f p_2 >_f \ldots$ in B. By the definition of $\leq_f$, this implies that there is an infinite descending chain $f^*(p_0) >_m f^*(p_1) >_m f^*(p_2)$ in M. But this contradicts the fact that $\leq_m$ is a well-order, and so $\leq_f$ must also be a well-order.

To prove the first admissibility property, let $v \in \Gamma_0$ and $p \in B \setminus \Gamma_0$. Then $f^*(v) = 1$ and $f^*(p) = m$ for some $m \in M \setminus \{1\}$. Since, 1 is minimal with respect to $\leq_m$, $f^*(v) <_m f^*(p)$ and so $v <_f p$.

To show the second admissibility property, assume that $r, p, q, s \in B$ satisfy $rps \neq 0 \neq rqs$ and that $p <_f q$. Then by the definition of $\leq_f$, $f^*(p) <_m f^*(q)$. By the monotonicity of the monoid operation with $\leq_m$, $f^*(r)f^*(p)f^*(s) <_m f^*(r)f^*(q)f^*(s)$. Therefore, $rps <_f rqs$ by the definition of $\leq_f$. □

This construction gives an admissible pre-order, but to have an admissible order the order must be total (antisymmetric, in particular). To build admissible orders we combine the pre-orders with a lexicographic order to break the ties. Lexicographic orders on paths are determined by a total ordering of $\Gamma_0 \cup \Gamma_1$ such that the vertices are smaller than the arcs. Such an order is called an *alphabetic ordering*. Given an alphabetic ordering $\leq_\alpha$ on $\Gamma_0 \cup \Gamma_1$, the *left lexicographic order* $\leq_{ll}$ on B is defined as $p \leq_{ll} q$ if $p = q$ or $p = wap'$, $q = wbq'$, and $a <_\alpha b$ for some path w, and arcs a and b. In particular, $p <_{ll} q$ if p is a prefix of q. Given an alphabetic ordering $\leq_\alpha$ on $\Gamma_0 \cup \Gamma_l$, the *right lexicographic order* $\leq_{rl}$ is defined as $p \leq_{rl} q$ if $p = q$ or $p = p'as$, $q = q'bs$, and $a <_\alpha b$ for some path s, and arcs a, b. In particular, $p <_{rl} q$ if p is a suffix of q. Both lexicographic orders are total and satisfy the admissibility properties, but are not admissible orders because neither is a well-order.

We can now build admissible orders using an admissible pre-order followed by a lexicographic order, or by prefixing an admissible order by an admissible pre-order. The following shows that the result of such a construction is also admissible.

Theorem 5.1 *Suppose that $\leq_f$ is an admissible pre-order on B defined from an order map $f : (\Gamma_0 \cup \Gamma_1) \to M$, and that $\leq'$ is either an admissible order or a lexicographic order on B. Define the order $\leq$ by $p \leq q$ if $p <_f q$ or if $f^*(p) = f^*(q)$ then $p \leq' q$. Then $\leq$ is admissible.*

Proof We first show that $\leq$ is a well-order. Suppose that $\leq$ is not a well-order. Then there is an infinite descending chain $p_0 > p_1 > p_2 \ldots$ in B. But since $\leq_f$ is a well-order, for this to happen there must be an infinite chain of equalities $f^*(p_i) = f^*(p_{i+1}) = f^*(p_{i+2}) = \cdots$ for some $i \geq 0$, and $\leq'$ must be a lexicographic order. Assume without loss of generality that this chain consists of the shortest paths equivalent under $\leq_f$. Then removing the first arc of each path results in a set of paths that are strictly ordered by $\leq_f$. Since there are an infinite set of paths, there must be an infinite descending chain of paths among this set of shortened paths. However, no such chain can exist since $\leq_m$ is a well-order. Therefore, $\leq$ is a well-order.

To see that $\leq$ is a total order, note that all paths $p, q \in B$ are comparable by $\leq_f$, but there are some pairs for which $f^*(p) = f^*(q)$. So, $\leq_f$ is not a partial order. In this situation, $p \leq q$ is determined by $\leq'$ which is total. So, $\leq$ is antisymmetric and therefore total. Finally, the admissibility properties hold for $\leq$ since both of the component orders respect the admissibility properties. Therefore, $\leq$ is an admissible order. $\square$

5.3 A Class of Orders

All of the specific orders considered in this thesis can be defined in terms of admissible pre-orders and lexicographic orders. These orders are the length lexicographic, weight lexicographic, and vector lexicographic orders. For the length and weight orders, the monoid is the positive integers under addition, and the total order on the monoid is the numeric order on the integers. For a length order, the pre-order on paths compares the length of the paths. For a weight order, the pre-order is defined in terms of a map w that assigns weights to the vertices and arcs. The vertices are all assigned weight zero, and the arcs are assigned positive weights. The weight pre-order then is determined by the sum of the weights of the paths. The length order is a special case of a weight order when the arcs are all given weight one.

For the vector orders, the monoid is the set of positive integer vectors $\mathbb{Z}^n$ ($n = |\Gamma_1|$) with component-wise addition. Two lexicographic orders can be defined on the vectors. The left lexicographic order compares the entries from the left to right, and the vector with the first entry that is least is less than the order. The right vector lexicographic order compares from the right to the left and looks for the last entry that is different. The pre-orders are defined by mapping the vertices to the zero vector, and mapping each arc to a vector with a 1 in a unique entry. The pre-order defined by the left lexicographic on the vectors is called the *(left) vector* order, and the pre-order defined by the right lexicographic order is called the *right vector* order. The vector orders have the same definition as the lexicographic orders for commutative monoids.

This class of orders is in no way complete. There are many other orders on strings that could be defined on the paths of a graph. Martin [39] considers a wider class of orders on strings.

In the experiments that follow, we exclude the weight orders because there are simply too many of them, and choosing the "best" weight order for the experiments is not possible. The choice of orders attempts to keep the group of orders considered small and also to avoid equivalent orders. In addition to the weight orders, the right lexicographic orders are not considered, thus halving the number of orders compared.

To avoid equivalences, none of the orders end with the reverse lexicographic order because equivalences between orders can occur by using dual alphabetic orders (orders which are the reverse of each other). An example is the length reverse lexicographic order which is equivalent to length lexicographic with a dual alphabetic order. Other more subtle equivalences may exist, but are not as easily identified.

The vector orders considered are somewhat restricted by the way in which they are implemented in Opal. The alphabetic order is used to determine the sequence in which the count vectors are built. If the alphabetic order is $a > b > c$ for the three letter alphabet $\{a, b, c\}$, then the count vectors are 3-tuples where the first entry is the number of a's, the second entry is the number of b's, and the third entry is the number of c's. So, the words aac, aca, and caa all correspond to the vector $(2, 0, 1)$. Opal does not allow the definition of vector orders that use a sequence different from the alphabetic order. Therefore, some vector lexicographic orders cannot be defined. However, these undefinable orders only differ in the order of words that are permutations of each other (since these are determined by the lexicographic order). An example is the left vector lexicographic order defined by using the alphabetic order $a > b > c$, followed by the lexicographic order defined by the alphabetic order $b > c > a$. This order indicates that $aabc > bacb > abbc$, but the order using $a > b > c$ for both the vector and lexicographic orders indicates that $aabc > abbc > bacb$.

5.4 Experimentation

In our experiments the Opal system was used to compare the seven orders: length lexicographic, (left) vector lexicographic, right vector lexicographic, length (left) vector lexicographic, length right vector lexicographic, length reverse (left) vector lexicographic, length reverse right vector lexicographic. The goal of the experiments was to find a ranking of these orders that could be used as a practical guide to the choice of order. This section describes the experiments.

5.4.1 Algorithm

The Opal system was used for the experiments. The algorithm used does shortest selection, tip-reduction, hybrid triple elimination, and eager set reduction. The three forms of termination were used where appropriate.

5.4.2 Input Problems

Twenty-nine problems were selected for the experiments. These problems can be divided into problems from mesh algebras, problems from free algebras, randomly generated problems, and problems in other path algebras. The following problems used in the algorithms experiments are also used here: A4, A5, A6, BT7, DCYC, ELP, ICYC, and P5. The following problems are new A51E, A51H, AGS, CG5, CGL, CGL1, HWEB, HWRES, MBFS, MDFS, M1, MM, MS, MT1, MT2, MT3, MT4, MTB, MTRI, P4, and P6. All of these problems are described in Appendix C.

For each of these problems, four random permutations of the arcs were computed in addition to the "natural" presentation of the arcs (how the problem instance was first stated). These permutations were used as the alphabetic orders on which the experiments were run. The goal being to neutralize the effect of the alphabetic order on the choice of admissible order (prior experience suggests that the alphabetic order has a strong effect). So, for each problem, five instances were used to compare each order.

5.4.3 Execution

Each problem instance was run noninteractively, with the algorithm using the appropriate form of termination depending on the problem. Because of the number of problem instances, a time limit of fifteen minutes was set for each. In practice, this limit was only enforced if it was not practical for all instances to be run to termination. With the exception of problems HWEB and A51E for which there are only two observations, all five observations were made for all of the other problems listed above. (Originally, there were 32 problems, but three were eliminated because they did not consistently terminate for particular alphabetic orders.)

5.4.4 Results

For each problem instance the following information was collected: the computation time, the total number of overlap relation reductions, the number of overlap relation reductions to zero, the number of simple reductions, and whether the result was a finite Gröbner basis. The detailed results for the experiments are given in Appendix E.

5.5 Analysis

The goal of these experiments is to rank the orders in a practical sense. This requires that we have some way to compare orders and determine which is best. For some applications, such as elimination theory, which is the best order is determined by properties that are needed of the Gröbner basis. But for most algebraic problems, it is sufficient to find any result. Therefore, we consider a general notion of best based on getting a finite answer as quickly as possible.

When comparing two orders $\leq_1$ and $\leq_2$ for a problem instance and a bound on the computation, order $\leq_1$ is better than order $\leq_2$ provided one of the following conditions is true.

1. If either the results found for both orders are finite Gröbner bases or both are not Gröbner bases, then the computation using $\leq_1$ takes *less work*.

2. If the result for $\leq_1$ is a finite Gröbner basis, and the result for $\leq_2$ is not.

We analyze the results based on this criterion. The quantity of *work* is determined by the number of simple reductions, the number of nonzero reductions and the time required for the computation.

For each problem and each admissible order we compute the averages of the percentage of nonzero reductions, the total computation time, the number of simple reductions, as well as the percentage of finite results. Then for each problem the orders were ranked by first sorting them in decreasing order of the percentage of finite results, and then sorting them by the averages for the number of simple reductions, total computation time, and the percentage of nonzero overlap reductions. The exact ranking was determined by comparing the 95% confidence intervals for the differences between the observations for consecutive pairs of admissible orders in the sorted list. If the confidence interval for the difference between two consecutive orders (by the sort) does not contain zero then they are ranked differently. An example for the MM instance is shown in Table 5.1 with the confidence intervals for the difference between the observations shown in Table 5.2. (In the tables, the following

Table 5.1: Ranking of Admissible Orders for Instance MM.

Admissible Order	Average % Nonzero Reductions	Average Time	Average Simple Reductions	% Finite
lrv	0.6071	1.424	32	0.8
l	0.7151	1.488	38.8	0.8
lri	0.5904	2.262	59	0.8
i	0.6203	1.968	39.8	0.6
li	0.6203	1.956	39.8	0.6
lv	0.5404	2.626	65	0.4
v	0.5404	2.648	65	0.4

notation is used to identify the orders: ‘l’ stands for length, ‘v’ stands for (left) vector, ‘i’ stands for right vector, and ‘r’ stands for reverse.)

This analysis technique finds a ranking, but has some drawbacks. Since we compare only consecutive pairs of orders, some differences are ignored even though the confidence intervals suggest that orders ranked differently, should be ranked the same. This is evident in Table 5.1 where the length reverse vector lexicographic (lrv) order has a higher rank than the length reverse right vector lexicographic (lri) order even though they have the same percentage of finite results, and the confidence intervals show that the difference between the number of simple reductions for each is not significant. What this suggests is that the variation due to alphabetic orders makes such a ranking meaningless, since the choice of alphabetic order can determine which admissible order is best for a particular problem.

By looking at these same numbers analyzed to compare alphabetic orders instead of admissible orders, it is clear that the differences between the alphabetic orders is stronger. Table 5.3 shows the ranking of alphabetic orders for instance MM, and Table 5.4 shows the 95% confidence intervals for the differences. The clearest indication of the significance of the alphabetic orders for this problem is that for the first alphabetic order, finite results were found for all admissible orders where nothing similar can be said for the admissible orders.

Despite this shortcoming in the analysis we proceed to develop a general ranking of the orders. The ranking of the orders for individual problems is shown in Tables 5.5–5.8 (the problems are divided into separate table for instances in free algebras, mesh algebras, other path algebras, and randomly generated instances). The general ranking is developed by weighting occurrences of each rank. Rank one has weight seven, rank two has weight six, etc. Then the weighted average for each

Table 5.2: 95% Confidence Intervals for Differences between Admissible Orders for Instance MM.

Difference	Nonzero Reductions		Time		Simple Reductions	
	Low	High	Low	High	Low	High
i-l	0.0676	0.3705	0.5700	2.9500	12.7293	86.0707
i-li	0	0	0.0093	0.0307	0	0
i-lri	-0.0134	0.2207	0.2663	3.8497	-8.4354	95.6354
i-lrv	0.0198	0.1136	-0.2729	2.0169	-4.7562	35.5562
i-lv	-0.0095	0.1903	-0.2261	2.7181	-8.0316	80.0316
i-v	-0.0095	0.1903	-0.2411	2.7931	-8.0316	80.0316
l-li	0.0676	0.3705	0.5708	2.9332	12.7293	86.0707
l-lri	-0.0180	0.2680	0.0581	1.8419	0.0174	45.9826
l-lrv	0.0916	0.3274	0.1076	2.3884	-0.6254	77.4254
l-lv	0.0358	0.3381	0.4215	1.8545	13.2812	49.5188
l-v	0.0358	0.3381	0.4537	1.8663	13.2812	49.5188
li-lri	-0.0134	0.2207	0.2725	3.8435	-8.4354	95.6354
li-lrv	0.0198	0.1136	-0.2482	2.0002	-4.7562	35.5562
li-lv	-0.0095	0.1903	-0.2341	2.7181	-8.0316	80.0316
li-v	-0.0095	0.1903	-0.2491	2.7931	-8.0316	80.0316
lri-lrv	0.0083	0.1824	-0.6313	3.0913	-16.7482	86.7482
lri-lv	-0.0070	0.1388	0.6420	1.9420	13.5557	47.2443
lri-v	-0.0070	0.1388	0.6135	1.9265	13.5557	47.2443
lrv-lv	-0.0033	0.1612	0.3170	3.0870	5.2969	85.5031
lrv-v	-0.0033	0.1612	0.3090	3.1470	5.2969	85.5031
lv-v	0	0	-0.0083	0.0763	0	0

Table 5.3: Ranking of Alphabetic Orders for Instance MM.

Alphabetic Order	Average % Nonzero Reductions	Average Time	Average Simple Reductions	% Finite
1	0.5000	2.2200	64.2857	1.0000
4	0.5491	1.6814	41.4286	0.8571
2	0.5983	1.5971	27.7143	0.4286
3	0.6785	2.0257	36.2857	0.4286
5	0.6983	2.7414	72.7143	0.4286

Table 5.4: 95% Confidence Intervals for Differences between Alphabetic Orders for Instance MM.

Difference	Nonzero Reductions		Time		Simple Reductions	
	Low	High	Low	High	Low	High
1-2	0.0389	0.1577	0.2933	0.9524	21.2466	51.8963
1-3	0.0880	0.2691	0.7979	1.5278	3.9688	54.3169
1-4	-0.0385	0.1724	0.5576	1.2395	6.2627	40.5945
2-3	0.0528	0.1596	0.7299	1.8930	13.0414	44.6729
2-4	0.0521	0.1557	0.3206	1.0565	13.6392	36.6465
1-5	0.0633	0.3061	1.4032	2.6025	44.8433	70.0138
2-5	0.0742	0.2370	1.1494	2.9591	21.8430	90.1570
3-4	0.0608	0.1947	1.1064	2.2879	10.7091	54.4337
3-5	0.0989	0.2702	1.3689	3.7626	32.2012	100.9416
4-5	0.1319	0.3518	1.9379	3.1307	57.2758	93.8670

Table 5.5: Ranking of Orders for Individual Free Problems.

Order	A4	A5	A6	ELP	HWEB	HWRES	P4	P6
l	2	4	1	1	2	1	1	1
i	1	2	1	4	3	1	2	5
v	1	1	1	1	1	1	2	4
li	1	2	1	4	5	1	1	5
lv	1	1	1	1	4	1	1	4
lri	1	3	1	3	4	1	1	3
lrv	1	3	1	2	6	1	1	2

was found and used to rank the orders as shown in Table 5.9. What the table shows is that, for these problems and instances, the length order is generally best, with the left vector order next.

5.6 Alphabetic Orders

We saw in the previous section that the alphabetic order seems to have a strong influence on the choice of admissible order. This should actually be fairly obvious given the definition of the vector orders in Opal and the definition of the lexicographic orders. In this light, it appears to have been too naive to expect to be able to rank the admissible orders while ignoring the alphabetic order. In this section, the problem of choosing an alphabetic order is briefly considered.

In Chapter 6, a theorem is proved that essentially says two admissible orders are equivalent if they order the uniform equivalence classes consistently (recall that these classes are the sets of paths

Table 5.6: Ranking of Orders for Individual Mesh Algebra Problems.

Order	BT7	CG5	M1	MBFS	MDFS	MM	MS
l	1	1	1	1	1	1	2
i	5	3	4	4	4	3	4
v	3	2	4	5	5	4	3
li	5	3	4	4	4	3	4
lv	3	2	4	5	5	4	3
lri	2	5	3	3	3	2	5
lrv	4	4	2	2	2	1	1

Table 5.6: Ranking of Orders for Individual Mesh Algebra Problems (cont.).

Order	MT1	MT2	MT3	MT4	MTB	MTRI
l	1	1	1	1	2	1
i	5	5	5	4	4	4
v	3	2	3	5	3	2
li	5	5	5	4	4	4
lv	3	2	3	5	3	2
lri	2	3	2	2	5	4
lrv	4	4	4	3	1	3

Table 5.7: Ranking of Orders for Individual Random Problems.

Order	A51E	A51H	AGS
l	1	1	4
i	1	2	1
v	1	–	2
li	1	2	3
lv	1	1	5
lri	2	1	7
lrv	2	2	6

Table 5.8: Ranking of Orders for Individual Path Algebra Problems.

Order	CGL	CGL1	DCYC	ICYC	P5
l	3	3	3	3	1
i	1	1	1	2	4
v	2	2	2	1	5
li	7	7	5	5	4
lv	4	4	6	3	5
lri	5	5	4	4	3
lrv	6	6	5	5	2

Table 5.9: Ranking of Admissible Orders.

Admissible Order	Average Weighted Rank
Length (l)	6.38
Left vector (v)	5.43
Right vector (i)	5.03
Length left vector (lv)	5.00
Length reverse left vector (lrv)	4.93
Length reverse right vector (lri)	4.90
Length right vector (li)	4.28

$$\begin{tikzcd}
(1) \arrow[r, "a"] & (2) \arrow[d, "b"] \\
(4) \arrow[u, "d"] \arrow[ur, "e"] & (3) \arrow[l, "c"]
\end{tikzcd}$$

Figure 5.1: Quiver for A51 Problem Instance.

with the same origin and terminus). What this implies is that when dealing with admissible orders built with the left lexicographic, how the alphabetic order sequences the out-arcs of each vertex is all that is important (at least to determining the Gröbner basis for a generating set). Similarly, if the admissible order uses the right lexicographic order, all that matters is how the alphabetic order sequences the in-arcs of each arc. To test this hypothesis, a simple problem instance was chosen and run for all possible permutations of the arcs.

The graph for this instance is shown in Figure 5.1. The graph has six arcs and so 720 possible alphabetic orderings (modulo the ordering on vertices). Two orders were used for the test: the length lexicographic order, and a weight order that makes $da < e$.

The results divided into four classes depending on the relationship between d and e and the relationship between a and c. Table 5.10 shows the results for each class (all numbers were the same for all alphabetic orders in each class, and every instance resulted in a finite Gröbner basis). The hypothesis would suggest that the relationship between d and e would be significant, but the relationship between a and c is unexpected.

One explanation of the importance of the order between a and c is that it is significant to the selection strategy for this particular problem. If this is true, it would mostly explain why the alphabetic order has such a strong influence on the computation. That is, since the selection strategy appears to have the most significant impact on the time required for the computation, the alphabetic order determines which common multiple is chosen (among those of the same length). Further experiments are needed to test how the alphabetic order influences selection and the rest of the computation.

Table 5.10: Counts for Classes of Alphabetic Orders on Problem A51.

Constraints	Order	% Nonzero Reductions	Maximum Overlaps	Maximum Cardinality
$d > e$, $a > c$	Length	79.41	848	163
	Weight	92.50	1729	256
$e > d$, $a > c$	Length	78.94	380	54
	Weight	90	1598	127
$d > e$, $c > a$	Length	80.30	3100	369
	Weight	89.78	8707	691
$e > d$, $c > a$	Length	83.33	516	66
	Weight	91.30	2229	322

5.7 Summary and Directions

In Section 5.5 a ranking of seven orders considered is given. This ranking indicates that the first order to try is the length order. In fact, this is what string rewriting folklore and our experience over the course of this research suggests. However, the ranking is not really valid because the experimental results do not imply a total ordering of the admissible orders. In fact, the results seem to imply that which admissible order is best for a particular problem really depends on the alphabetic order.

Although some experiments with alphabetic orders have been done, there is no obvious result that indicates how to choose the best alphabetic order. In fact, the results in the previous section can be interpreted as showing the influence of the alphabetic order over the selection strategy. Perhaps using an implementation that allows the use of different alphabetic orders for the selection strategy and the polynomial order would be helpful in experiments to understand the choice of alphabetic order.

Another approach that might be more practical than a simple ranking of orders, is to develop heuristics for choosing an order for a particular problem. If we reconsider the criterion for ranking the orders defined in the previous section, the best orders for a generating set P are those for which P is a Gröbner basis. This means that either there are no overlap relations for P, or they all reduce to zero. In general, the criterion translates to the best order being the one for which there are the least number of overlap relations throughout the computation. A heuristic based on this idea is to find an order that minimizes the number of overlaps of the generators. Refinements of this heuristic might try to minimize overlaps that would occur in later stages of the algorithm.

An approach of this sort might prove to be more practical, and certainly simpler to find than a characterization of which orders are the best for which problems. The following chapter explores the relationship between orders and problems in path algebras as an alternative approach to the problem of choosing an order.

Chapter 6

Admissible Orders in Path Algebras

This chapter explores the relationship between admissible orders and problem instances. The experiments in Chapter 5 suggest that the choice of order is highly dependent on the problem instance. For free algebras, the problem instance is determined by the relations, but in path algebras with more complex quivers, the quiver determines what words appear and so also affects the choice of order. This chapter explores the relationship between a graph and the orders on its paths. While the experiments in Chapter 5 also indicated that the relations have more of an effect on the choice of order, this relationship is not considered here.

In particular, this chapter looks at equivalence classes of orders on paths induced by the structure of the corresponding graph. The goal is to identify representatives of equivalences classes of orders. The equivalence we consider is whether two orders induce the same Gröbner basis (or at least the same tip set) for a given generating set and is defined in Section 6.1. Section 6.2 begins the development of an approach to the computation of representative orders for the equivalence classes for very special graphs. This approach uses a relationship of spanning trees and admissible orders observed by Green [24]. Once this approach is complete, it might be possible to combine it with an approach using information on the generating relations to make the choice of order. This and other future research directions are discussed in Section 6.3.

The work by Martin [41] on determining orders that induce termination of string rewriting systems is related and could be useful. The result on spanning trees given below is an observation of Green.

6.1 Equivalence of Orders

Two orders are considered to be equivalent for a particular problem instance if the resulting Gröbner bases are the same. Theorem 6.1 given below says that it is sufficient to consider only how two orders sequence the uniform equivalence classes.

First we define some notation. Fix a finite directed multigraph $\Gamma = (\Gamma_0, \Gamma_1)$. Suppose that $u, v \in \Gamma_0$ are vertices, and $<$ is an admissible order on B the set of paths of Γ. Denote by $E^{<}_{u,v}$ the ordered sequence of uniform paths between u and v, and (in a slight abuse of notation) let $E^{<_1}_{u,v} = E^{<_2}_{u,v}$ denote the fact that the ordered sequences determined by $<_1$ and $<_2$ are identical. If P is a set of polynomials, then $T^{<}_{u,v}(P)$ is the set of uniform tips of the Gröbner basis (with respect to $<$) for $\langle P \rangle$ from u to v and $N^{<}_{u,v}(P)$ is the set of uniform nontips from u to v. Specifically,

$$T^{<}_{u,v}(P) = \{t : t \in \mathit{Tip}(\langle P \rangle), \mathit{src}(t) = u, \mathit{tgt}(t) = v\}\,.$$

Theorem 6.1 *Let $\Gamma = (\Gamma_0, \Gamma_1)$ be a finite directed multigraph, B the set of finite directed paths of Γ, and K a field. If $<_1$ and $<_2$ are two admissible orders on B, then the following are equivalent:*

1. *For all $u, v \in \Gamma_0$, the sequences $E^{<_1}_{u,v}$ and $E^{<_2}_{u,v}$ are identical.*

2. *For every set of polynomials $P \in K\Gamma$, and for all vertices $u, v \in \Gamma_0$, the nontip sets are identical:*
$$N^{<_1}_{u,v}(P) = N^{<_2}_{u,v}(P).$$

3. *For every set of polynomials $P \in K\Gamma$, and for all vertices $u, v \in \Gamma_0$, the tip sets are identical:*
$$T^{<_1}_{u,v}(P) = T^{<_2}_{u,v}(P).$$

Proof That condition 1 implies the other two conditions is straightforward. We first show that condition 2 implies condition 1, and then show the equivalence of conditions 2 and 3.

To show that condition 2 implies condition 1 we actually prove that if condition 1 is false, then condition 2 is also false. Assume there exists p_1 and p_2 which are uniform such that $p_2 <_1 p_1$ and $p_1 <_2 p_2$, and let $P = \{p_1 + p_2\}$. We need to show p_1 is in $\mathit{Nontips}_{<_2}(\langle P\rangle)$ and p_2 is in $\mathit{Nontips}_{<_1}(\langle P\rangle)$.

Suppose that p_1 is not a nontip of $\langle P\rangle$ with respect to $<_2$. Then it must be that some path p in $\mathit{Tip}(\langle P\rangle)$ divides p_1. But since p_2 is the tip of the single generator, $p_2|p$, and so $p_2|p_1$. So, by admissibility of $<_2$ it must be that $p_2 <_2 p_1$ which is a contradiction. Therefore, p_1 must be in $\mathit{Nontips}_{<_2}(\langle P\rangle)$. By a similar argument, $p_2 \in \mathit{Nontips}_{<_1}(\langle P\rangle)$. Since it is not the case that $p_2 \in \mathit{Nontips}_{<_2}(\langle P\rangle)$ and $p_1 \in \mathit{Nontips}_{<_1}(\langle P\rangle)$, it follows that $N^{<_1}_{u,v}(P) \neq N^{<_2}_{u,v}(P)$. Therefore, condition 2 implies condition 1.

The equivalence of conditions 2 and 3 follows from the definition of a nontip set. Since the set of tips and the set of nontips are complements, $N^{<_1}_{u,v}(P) = N^{<_2}_{u,v}(P)$ if and only if $T^{<_1}_{u,v}(P) = T^{<_2}_{u,v}(P)$ for any set of polynomials P. Therefore, the second and third conditions are equivalent. □

This theorem is a specific form of a folk theorem from string rewriting that says equivalence of two orders for a rewriting system is determined by how the two orders sequence the equivalence classes of terms defined by the rewrite rules [40].

Theorem 6.1 says that how two orders behave on uniform equivalence classes determines equivalence, at least, with respect to determining a Gröbner basis. This implies that the problem of building an order that is a representative for an equivalence class of orders can be reduced to choosing how it sequences the uniform equivalence classes. For this to be feasible, we need to find a (minimal) set of paths that generate the uniform equivalence classes of the graph, and then order these generating paths in such a way that the order can be extended to the whole uniform equivalence class.

This construction is not feasible for a graph with only one vertex. In this case, the uniform equivalence class is the whole set of paths and the minimal set of generators is just the vertex and the arcs. The problem is that simply ordering the arcs does not completely determine an order on the paths, the process of forcing a total order can be infinite. An example of this is the graph with two loops a and b. Simply setting $a < b$ does not completely determine the admissible order, since the relative order of ab and ba is not implied. Even if we set $ab < ba$, this does not determine the relative order of bab and aba. Continuing by setting $aba < bab$ again is not enough since the relative order of $abba$ and $baab$ is not known. This process of forcing the order to be total does not terminate in this case. So, it is not possible to deal effectively with more than one loop. The following section explores an alternative approach that works in special cases where loops do not occur.

6.2 Spanning Trees and Orders

In this section, we use the relationship between admissible orders and families of spanning trees on graphs to build equivalence class representatives on a limited class of graphs (without loops and having some restrictions on cycles). We begin with the observation that the minimal paths in a graph Γ with respect to some admissible order $<$ form spanning trees of Γ. Since Γ is directed, the spanning trees are *rooted* at some vertex v with arcs in the tree oriented to form paths from v to all other vertices of Γ. The relevant result is the following.

Theorem 6.2 *If Γ is a strongly connected finite directed multigraph and $<$ an admissible order on the finite paths B of Γ, then for each vertex $v \in \Gamma_0$ the set of minimal paths from v to all vertices determines a spanning tree of Γ rooted at v.*

Proof For all $u, v \in \Gamma_0$, let $P_<(v, u)$ be the set of arcs on the unique minimal path (with respect to $<$) from v to u in Γ. Let $N_<(v)$ be the union of these sets from v:

$$N_<(v) = \bigcup_{u \in \Gamma_0} P_<(v, u).$$

We show that $N = (\Gamma_0, N_<(v))$ is a spanning tree. The fact that N spans Γ comes from its definition and the fact that Γ is strongly connected. To obtain a contradiction, suppose that N is not a tree. Then there exists some $u \in \Gamma_0$ such that there are two paths p and q from v to u in N. Then there exists some $u \in \Gamma_0$, $u \neq v$, such that there are two paths p and q from v to u in N. In particular, we can choose u, p and q to be such that there are no vertices w along either p or q for which there are two paths from v to w. (So, u is the "nearest" such vertex to v.) Assume that p is the minimal path whose arcs constitute $P_<(v, u)$. Then q must be the prefix o some minimal path from v to a vertex x. So, in particular, $r = qs$ where s is a path from u to x. Since p is the minimal path from v to u, $p < q$ and so $ps < qs$. So, r cannot be minimal, thus contradicting the assumption. Therefore, N is a tree. □

These spanning trees are called $<$-spanning trees, and each admissible order determines a family of trees indexed by the vertices of the graph. If the graph is not strongly connected, then instead of spanning trees, the result is a family of trees, each of which spans some subgraph of Γ.

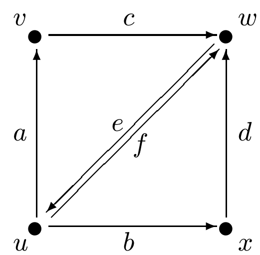

Figure 6.1: Graph for Uniform Equivalence Class Example.

The paths that form these trees are the nontips (or normal forms) for the ideal for a special class of relations on Γ. Given a graph $\Gamma = (\Gamma_0, \Gamma_1)$ the *commutativity relations* for Γ are differences of the simple paths in each uniform equivalence class, and differences of simple cycles with the corresponding vertices. For the uniform equivalence class $E_{u,v}$ from u to v such that $u \neq v$, the commutativity relations are all differences $p - q$ where paths $p, q \in E_{u,v}$ are simple. For each vertex v and each simple cycle c from v to v, $c - v$ is a commutativity relation.

As an example, consider the graph in Figure 6.1. The uniform equivalence classes for this graph are the following.

$$
\begin{aligned}
[x,w] &= \{d, def, deac, debd, \ldots\} \\
[x,u] &= \{de, debde, defe, debac, \ldots\} \\
[x,v] &= \{dea, defea, deacea, debdea, \ldots\} \\
[w,u] &= \{e, efe, eace, ebde, \ldots\} \\
[w,v] &= \{ea, eacea, ebdea, efea, \ldots\} \\
[w,x] &= \{eb, eaceb, ebdebd, efeb, \ldots\} \\
[u,v] &= \{a, fea, acea, bdea, \ldots\} \\
[u,w] &= \{ac, bd, f\} \\
[u,x] &= \{b, feb, bdeb, aceb, \ldots\} \\
[v,w] &= \{c, ceac, cebd, cef, \ldots\} \\
[v,u] &= \{ce, ceace, cefe, cebde, \ldots\}
\end{aligned}
$$

$$[v,x] \quad = \quad \{ceb, cebdeb, cebfeb, cebdeaceb, \ldots\}$$

The cycles for the graph are

$$\begin{aligned}
[u,u] &= \{u, ace, fe, bde, feace, febde, acebde, \ldots\} \\
[v,v] &= \{v, cea, cefea, cebdea, \ldots\} \\
[w,w] &= \{w, eac, ef, ebd, efeac, efebd, eacebd, \ldots\} \\
[x,x] &= \{x, deb, defeb, deaceb, \ldots\}
\end{aligned}$$

The only uniform equivalence class with more than one simple path is $[u,w]$. The commutativity relations for this class are $ac - bd$, $bd - f$, and $ac - f$. The relations for the cycles are $ace - u$, $fe - u$, $bde - u$, $cea - v$, $eac - w$, and $deb - x$. The Gröbner basis of the ideal for the commutativity relations allows the uniform equivalence of two paths p, q to be decided by testing whether $p - q$ reduces to zero. The nontips for this ideal are the minimal elements of the equivalence class and so give the $<$-spanning trees.

The existence of the $<$-spanning trees for each admissible order $<$ suggests a way to classify the admissible orders on Γ. That is to use all valid families of spanning trees to define the orders: if two orders yield the same spanning tree then they are equivalent. Unfortunately, this is not the equivalence we are after. A family of spanning trees indicates only how the orders behave on the nontips for a very special class of generators, but by Theorem 6.1 the orders are equivalent only if the nontips are all the same for every generating set of every ideal in the algebra.

In fact, it is easy to find a graph for which the spanning tree does not imply the desired equivalence of orders. Consider the graph Γ in Figure 6.2 in which there are three paths ab, cd, ef from u to v. If we pick a spanning tree rooted at u, then one of ab, cd or ef, say ef, will be in the tree (see Figure 6.3). Then any two orders $<_1$ and $<_2$ that satisfy $ef <_1 ab <_1 cd$ and $ef <_2 cd <_2 ab$, determine this tree. Clearly, if we choose the generating set $\{ab + cd\}$ the nontips and the tips determined using these two orders are different. The only case in which the spanning trees are useful for (directly) determining the orders is when the generators are binomials (e.g., the generators are a subset of the commutativity relations for the graph).

However, the spanning trees can be used to find admissible pre-orders for the graph if the family of trees corresponds to some admissible order. Assume that we have such a family of spanning trees, then this family determines a set of constraints on the paths. Given a vertex u the constraints are

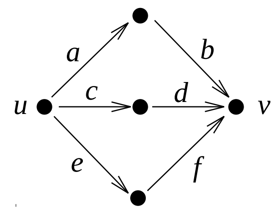

Figure 6.2: Graph with Three Paths ab, cd, ef from u to v.

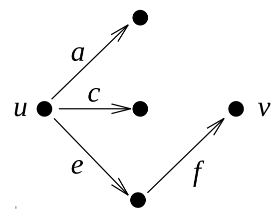

Figure 6.3: Spanning Tree with Path ef from u to v.

determined as follows. For each vertex $v \neq u$, if p is a path from u to v in T_u, then define a constraint $p < q$ for each path $q \neq p$ from u to v.

More formally, an *order constraint* on the paths of the graph Γ is an ordered pair (p, q) of paths $p, q \in B$ that is interpreted to mean $p < q$. Let C be a set of constraints on the paths of Γ, and denote by C^* the smallest set of constraints containing C, and satisfying the following properties:

1. if $(p, q) \in C^*$ and $(q, r) \in C^*$ then $(p, r) \in C^*$; and
2. if $(p, q) \in C^*$ and $r, s \in B$ such that $rps \neq 0 \neq rqs$ then $(rps, rqs) \in C^*$.

C^* is called the *closure* of C. We say that C^* is *consistent* if there is no pair $(p, q) \in C^*$ such that $(q, p) \in C^*$.

We assume that C is determined by a family of spanning trees as described above. So, C contains no pairs (p, p) for some path p, and for every pair $(p, q) \in C^*$, p and q are uniform equivalent. As an example, consider the family of trees Figure 6.5 for the graph in Figure 6.4. Using this family of trees we can find the set of constraints

$$\begin{array}{lll} c < dg, & d < cei, & ce < dhl, \\ bj < ad, & bkl < ace, & \\ g < fac, & hl < ge, & h < fbk, \\ k < jh, & j < kli, & jl < jge. \end{array}$$

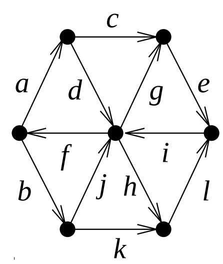

Figure 6.4: Graph for Spanning Tree Example.

Note that other constraints can be found, such as $bjhl < adge$, but they are implied by the constraints given.

This example suggests the following proposition, which has not been proven.

Conjecture 6.1 *Suppose that C is a set of order constraints on the paths B of a graph Γ derived from a family of spanning trees of Γ. Then the set*

$$C' = \{(p,q) \in C : p, q \textit{ are simple}\} \setminus \{(p,q) \in C : \exists (r,s) \in C, p|r \textit{ and } q|r\}$$

has the same closure as C, $C^* = (C')^*$.

Unfortunately, there are families of spanning trees that induce inconsistent constraints on the paths and so do not correspond to admissible orders. An example is given by the graph in Figure 6.6 and the family of trees in Figure 6.7 (there are only three trees in the figure, but the other two are determined by the three given). These trees induce the constraints $ga < he$, $hf < gb$, $bd < ac$ and $ec < fd$, and using these constraints it is possible to derive the inconsistent relation $gac < gac$ as

$$gac < hec < hfd < gbd < gac.$$

Therefore, we only want to deal with families of trees that give rise to consistent constraints.

A family F of spanning trees for a graph $\Gamma = (\Gamma_0, \Gamma_1)$, indexed by Γ_0, is called *consistent* if the constraints induced by the elements of F are consistent (there is no pair of paths p and q such that the constraints imply $p < q$ and $q < p$). The trees of Figure 6.5 are an example of a consistent family of trees, and those of Figure 6.7 are an example of an inconsistent family of trees. The inconsistency in the example of Figure 6.7 appears to be related to the fact that there are disjoint cycles gac and hfd in the graph that are not constrained by the trees. This suggests that consistency cannot be

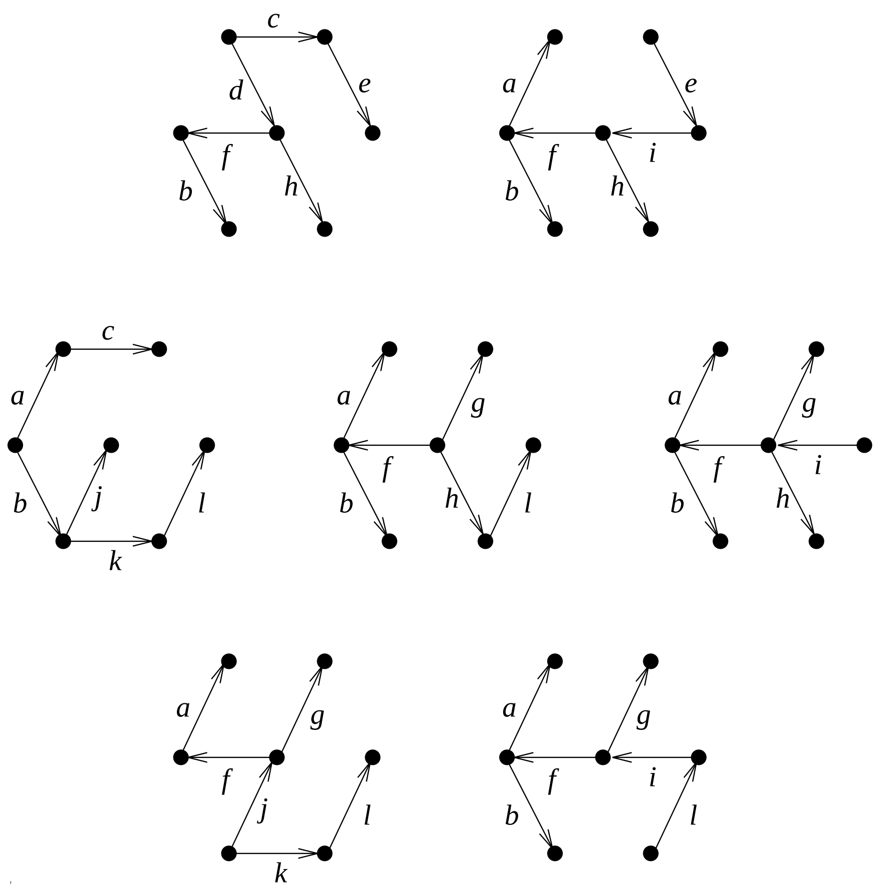

Figure 6.5: Example Family of Spanning Trees.

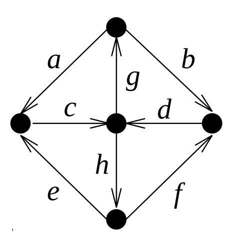

Figure 6.6: Graph for Inconsistency Example.

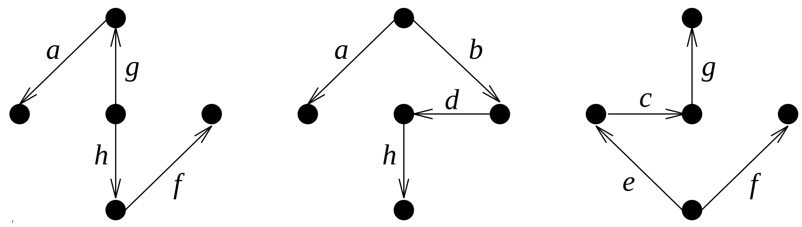

Figure 6.7: Inconsistent Family of Spanning Trees.

expressed solely in terms of properties on the spanning trees. Further work needs to be done to determine how to compute consistent families of trees for a graph.

What we want to do is use the constraints defined from a consistent family of spanning trees to define a weight pre-order on the paths by assigning weights to the arcs in a manner consistent with the constraints. The constraints determine constraints on the weight function for a weight order. For the example in Figure 6.5, it must be that $w(c) < w(d) + w(g)$ and $w(b) + w(j) < w(a) + w(d)$. For this example, any weighting that gives a and g a greater weight than the other arcs satisfies these constraints. For example, the weighting that assigns a and g weight 2 and all other arcs weight 1 satisfies these constraints. (In fact, any length lexicographic order using an alphabetic order such that $h < g$ and $b < a$ also satisfies the constraints given above.)

It is not clear that constructing a weight pre-order in this way is always possible. However, note that the closure of a set of constraints is nearly a pre-order.

Lemma 6.1 *If C is a set of constraints derived from a consistent family of trees, then C^* is an irreflexive, transitive relation.*

Proof Irreflexivity follows from the fact that no path is related to itself in C, and by the consistency of C^*. Transitivity is immediate from the definition of C^*. □

Therefore, if we assume that can define a weight function $w : (\Gamma_0 \cup \Gamma_1) \to \mathbb{N}$ consistent with a set of constraints C, the derived pre-order $<_w$ corresponds to C^*. The weight pre-order may relate more paths than C^* (in particular, $<_w$ is reflexive, and also some unequal paths may be given the same weight). We do not consider the problem of proving such a weight function exists, but the existence of such functions should follow from the consistency of the constraints.

The constraints induced by a consistent family of spanning trees only determine a pre-order on the uniform equivalent paths of the graph. Further work is needed to characterize when such pre-orders can be easily extended to a total order on the uniform equivalence classes. Note that loops and disjoint cycles (cycles that do not share any edge with the rest of the graph and so are like multiple edged "loops") are not ordered at all by the constraints. So the approach only really works for graphs without multiple loops or disjoint cycles at any vertex (single loops and disjoint cycles can be dealt with).

Clearly, the work in this section is not complete. First, the construction of orders from spanning trees is not entirely understood. In particular, it is not known how to compute or recognize a consistent family of trees, short of finding the induced constraints and checking for inconsistencies. It may be helpful to determine when inconsistencies occur. It seems that they only occur when there are cycles that share a single vertex but no edges, but this needs to be proved. If this can be shown it would be possible to check for families of trees that will give rise to inconsistencies.

6.3 Future Directions

The previous sections suggest several possible directions that future research might take.

6.3.1 Orders and Generators

Similar work is done by Martin [41], but instead of using the underlying structure of the language, she uses string rewriting rules to determine constraints on the orders. Clearly, for a rewrite rule $p \longrightarrow q$ either $p < q$ or $p > q$ and so a set of rules determines a set of constraints that are not necessarily consistent. After finding a consistent set of constraints it is possible to build weight pre-orders as was done from spanning trees in Section 6.2. Admissible orders that satisfy these constraints are equivalent in the sense given in Section 6.1 [41].

It is certainly possible (and necessary) to combine constraints determined by the quiver of the path algebra, and the relations of the generating set. However, the constraints are not so easily defined on relations, other than the fact that the term selected as the tip must be bigger than the other terms in the support. This is something that needs to be explored. (The equivalence relation determined by both the graph and relations will be a subrelation of the one determined just by the graph.)

Note that it is possible to use the structure of the words in the relations to determine a graph structure even if the words are from a free algebra. In some generating sets, certain arcs may occur only in a particular sequence and so can be treated as a path in a graph. This imposed graph structure might be used like the quiver of the algebra as discussed above to augment the order constraints defined using the relations directly. This may prove useful in developing representative admissible orders on free algebra instances.

6.3.2 Equivalence and Choosing a Good Order

Whether the equivalence relation on orders is helpful in finding a good order is not yet clear. The question that arises is whether all orders in an equivalence class behave the same computationally. They certainly give the same Gröbner bases, but perhaps the computation is not as efficient as for others. Depending on the answer to this question, we can ask either if it is possible to find the best class of orders (if members of an equivalence class all behave the same), or if it is possible to find the best representative for each equivalence class (if the members of an equivalence class do not behave the same).

6.3.3 Gröbner Walks

In the commutative theory of Gröbner bases, there is the concept of a "Gröbner walk" that is used to convert between Gröbner bases for different orders [16]. The algorithm uses the fact that admissible orders on commutative terms correspond to projective cones and transform the cone from the initial order to the cone for the final order. This algorithm depends heavily on the ability to characterize all admissible orders by weight vectors (see Robbiano [49] or Dube, Mishra and Yap [19]). While there is no such characterization for admissible orders on strings, the spanning trees might present an opportunity for a similar algorithm for path algebras. Also, most common admissible orders can be defined as weight orders, making it reasonable to deal with only this set of orders.

Assuming the graph has a reasonable structure, the problem is to convert a (finite) Gröbner basis G with respect to an order $<_i$ to a Gröbner basis for an order $<_f$. (Ignore for a moment the fact that the target Gröbner basis may not be finite.) As discussed above, the orders determine families of spanning trees on the graph. The idea is to incrementally mutate the family of trees for $<_i$ toward the family of trees for $<_f$ and change G as the orders change. Spanning trees are all related by edge swaps (see Lovász [37]), and most algorithms that generate all spanning trees of a graph use this fact. It might be possible to use a similar approach to "walk" from one family of trees to another, and then use an approach similar to the commutative Gröbner walk to change the basis. An obvious problem with noncommutative Gröbner walks is stepping into infinite potholes when an order for which the basis is infinite is found. An interesting question is whether such a pothole exists on some walk between two orders that both induce finite Gröbner bases.

The importance of the Gröbner walk is that it is often quicker to compute a Gröbner basis with one order and then "walk" it to the desired order, rather than computing with the desired order directly. This approach is used in the Gröbner basis function of the Magma computer algebra system. Situations where this works for the noncommutative case would be quite interesting.

Chapter 7

Conclusions

The objective of this research is to improve the computation of noncommutative Gröbner bases by finding the best choice of algorithm and admissible order. Progress was made in both areas. The primary contributions of the research are in finding new algorithms and noncommutative forms of existing Gröbner basis algorithms, but the research has also developed a better understanding of the choice of order, and also has led to the development of two systems that can be used for experimentation and algebraic research. This chapter surveys these contributions (in the first section), and future research directions and open questions (in the second section).

7.1 Contributions

The contributions of the research are in the areas of algorithms, orders and implementations. Each area is considered separately.

7.1.1 Algorithms

The algorithmic contributions are two-fold. The first contribution is the development of new algorithms and conversion of algorithms from the commutative case as alternatives for configuring the noncommutative form of Buchberger's algorithm. In particular, algorithms for triple selection, triple elimination, and set reduction are defined. The experimental comparison of these alternative configurations has helped identify an algorithm configuration that minimizes time and space requirements (at least for the experimental cases considered).

The second algorithmic contribution is the development of the dynamic dictionary matching approach for solving the pattern matching problems involved in computing noncommutative Gröbner bases. In this approach, all pattern matching searches take time linear in the size of the search word and the output resulting from the search. Along with the fact that insertions and deletions of words from the dictionary are also fast, the fast search times make this approach a significant improvement over the other algorithms that might be used (single-word matching algorithms, and the extension of static dictionary matching defined by Sims [52]).

Both of these algorithmic contributions dramatically improve the understanding of the computation of noncommutative Gröbner bases. They both should also extend to string rewriting where the pattern matching approach alone is a major improvement over the approaches used in existing implementations.

7.1.2 Orders

The contribution with respect to orders is mostly one of improved understanding, but also several promising research directions have been identified. The primary point observed is that the choice of order is much more problem dependent than we had understood previously. The choice of order appears to depend on the language of the terms (the paths), and how the terms are combined in the generators. Also, the admissible order is also a factor in determining which admissible order is best for a particular problem.

In Chapter 5 it was observed that a simple ranking of the orders is not directly implied by the experimental results. However, by sequencing the orders according to the observed values, and by comparing only the differences between consecutive orders in the list it is possible to develop a ranking. This ranking suggests that the length lexicographic order is generally the best (for the kind of problems considered). In general, probably the best approach is to find heuristics that help choose orders that minimize the number of overlaps considered during the computation.

Perhaps the biggest contribution with respect to orders are the possible research directions identified for exploring the choice of order. These are listed in the next section.

7.1.3 Implementation

This research has contributed two implementations of systems to compute noncommutative Gröbner bases. The first is the ngrb prototype family, which can be used to build alternative configurations of algorithms and could be relatively easily extended to study other algorithms. The second is the Opal system, which is designed to be used in algebraic research and can be used to experiment with admissible orders. Opal is being extended to include a broader range of fields and orders which should make it a good choice for algebraists who wish to test conjectures and experiment with algebraic problems.

7.2 Directions

The following is a list of open problems and research directions identified throughout the thesis. The appropriate chapter is given for each major group.

Algorithms (Chapter 4):

1. The experiments in Chapter 4 did not treat all possible algorithms. In particular, selection strategies not based on admissible orders, and reduction strategies are not tested. Probably the most significant omission is the reduction strategies.

2. The time-space trade-off between eager and hybrid triple elimination alluded to in Chapter 4 needs to be tested.

Orders (Chapters 5 and 6):

1. Development of heuristic approaches to the choice of order. Specific ideas to experiment with are the heuristic of choosing the order that minimizes the initial overlaps, and modifying the algorithm to dynamically adjust the order during the computation to minimize overlaps.

2. Combine order constraints determined by graphs and relations to define equivalence classes of orders. Determine whether it is always possible to compute a representative order of each equivalence class?

3. Determine whether there is a relationship between the equivalence classes of orders for a problem and the choice of order.

4. Determine what the relationship is between the choice of alphabetic order and problem instance. (To do this correctly, it is necessary to have an implementation that decouples the alphabetic order used for polynomials from the one used for selection.)

5. Determine whether there is a relationship between the admissible order on polynomials and the admissible order used for selection. (In Opal, the alphabetic order used for both is the same. Is this good?)

6. Explore application-dependent choices of order. For example, what elimination-order is best?

Other Algorithms:

1. The relationship between spanning trees and orders may provide a way to compute Gröbner walks (see Chapter 6). The question is whether this "walk" is always possible, or if the algorithm would only terminate in ideals for which all minimal Gröbner bases are finite?

2. Development of an algorithm for computing universal Gröbner bases (a generating set that is a Gröbner basis regardless of order).

REFERENCES

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[2] A. Amir, M. Farach, Z. Galil, R. Giancarlo, and K. Park. Dynamic dictionary matching. *Journal of Computer and Systems Sciences*, 49(2):208–222, 1994.

[3] J. Apel and W. Lassner. An extension of Buchberger's algorithm and calculations in enveloping fields of Lie algebras. In L. Robbiano, editor, *Computational Aspects of Commutative Algebra*, pages 227–236. Academic Press, 1989.

[4] AT&T Bell Laboratories. *The Standard ML of New Jersey Library Reference Manual (Release 0.1)*, 1993. Also distributed with Release 0.2.

[5] M. Barr and C. Wells. *Category Theory and Computing Science*. Prentice Hall, 1990.

[6] D. Bayer and M. Stillman. A criterion for detecting m-regularity. *Inventiones Mathematicae*, 87:1–11, 1987.

[7] T. Becker and V. Weispfenning. *Gröbner Basis: A Computational Approach to Commutative Algebra*. Springer-Verlag, New York, 1993.

[8] G. Bergman. The diamond lemma in ring theory. *Advances in Mathematics*, 29:178–218, 1978.

[9] R. Blute. Hopf algebras and linear logic. *Mathematical Structures in Computer Science*, 6(2):189–217, 1996.

[10] R. V. Book and F. Otto. *String-Rewriting Systems*. Springer-Verlag, 1993.

[11] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomialideal*. PhD thesis, Innsbruck, 1965.

[12] B. Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner-bases. In E. W. Ng, editor, *EUROSAM '79*, LNCS# 72, pages 3–21. Springer-Verlag, Berlin, 1979.

[13] B. Buchberger. Gröbner bases: an algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, Mathematics and its Applications, pages 184–232. D. Reidel Publishing Company, Dordrecht, Holland, 1985.

[14] R. Bündgen. Simulating Buchberger's algorithm by Knuth-Bendix completion. In R. V. Book, editor, *RTA '91*, LNCS# 488, pages 386–397. Springer-Verlag, Berlin, 1991.

[15] R. Bündgen. Buchberger's algorithm: the term rewriter's point of view. In W. Kuich, editor, *ICALP '92*, LNCS# 623, pages 380–391. Springer-Verlag, Berlin, 1992.

[16] S. Collart, M. Kalkbrener, and D. Mall. The Gröbner walk. Technical report, Department of Mathematics, Swiss Federal Institute of Technology, Zurich, 1993.

[17] B. A. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[18] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 245–320. MIT Press, 1990.

[19] T. Dube, B. Mishra, and C. K. Yap. Admissible orderings and bounds for Gröbner bases normal form algorithms. Technical Report 258, Department of Computer Science, New York University, 1986.

[20] D. R. Farkas, C. D. Feustel, and E. L. Green. Synergy in the theories of Gröbner bases and path algebras. *Canadian Journal of Mathematics*, 45(4):727–739, 1993.

[21] A. Galligo. Some algorithmic questions on ideals of differential operators. In *EUROCAL '85*, LNCS# 204, pages 413–421, 1985.

[22] R. Gebauer and H. M. Möller. On an installation of Buchberger's algorithm. In L. Robbiano, editor, *Computational Aspects of Commutative Algebra*, pages 141–152. Academic Press, 1989.

[23] P. Graf. *Term Indexing*. PhD thesis, Universität des Saarlandes, 1995.

[24] E. L. Green, 1995. Personal Communication.

[25] D. Gusfield, G. M. Landau, and B. Schieber. An efficient algorithm for the All Pairs Suffix-Prefix problem. *Information Processing Letters*, 41:181–185, 1992.

[26] D. Hartley and P. Tuckey. Gröbner bases in Clifford and Grassman algebras. *Journal of Symbolic Computation*, 20:197–205, 1995.

[27] G. Havas, D. F. Holt, and S. Rees. Recognizing badly presented z-modules. *Linear algebra and its applications*, 192:137–164, 1993.

[28] G. Havas and B. S. Majewski. Integer matrix diagonalization. *Journal of Symbolic Computation*, (to appear).

[29] J. W. Helton and M. Stankus. Computer assistance for "discovering" formulas in system engineering and operator theory. Technical report, Department of Mathematics, University of California, San Diego, California, Feb. 1996.

[30] J. W. Helton, M. Stankus, and J. Wavrik. Computer simplification of engineering formulas. *IEEE Transactions on Automatic Control*, (submitted), 1995.

[31] J. W. Helton and J. J. Wavrik. Rules for computer simplification of the formulas in operator model theory and linear systems. *Operator Theory: Advances and Applications*, 73:325–354, 1994.

[32] H. Hironaka. Resolution of singularities of an algebraic variety over a field of characteristic 0. *Annals of Mathematics*, 79:109–326, 1964.

[33] D. Holt, 1996. Personal Communication.

[34] C. Jay. Polynomial polymorphism. In R. Kotagiri, editor, *Proceedings of the Eighteenth Australasian Computer Science Conference*, volume 17, pages 237–243. A.C.S. Communications, 1995.

[35] A. Kandri-Rody and V. Weispfenning. Non-commutative Gröbner bases in algebras of solvable type. *Journal of Symbolic Computation*, 9:1–26, 1990.

[36] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297, 1970.

[37] L. Lovász. A homology theory for spanning trees of a graph. *Acta Mathematica Academiae Scientiarum Hungaricae*, 30(3–4):241–251, 1977.

[38] U. Martin. A note on division orderings on strings. *Information Processing Letters*, 36:237–240, 1990.

[39] U. Martin. On the diversity of orderings on strings. *Fundimenta Informaticae*, 24(1/2):25–46, 1995.

[40] U. Martin, 1996. Personal Communication.

[41] U. Martin. Invariants of string rewriting systems: examples and questions. In *Symbolic Rewriting Techniques*, 1996. (submitted).

[42] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[43] B. Mishra and C. Yap. Notes on Gröbner bases. *Information Sciences*, 48:219–252, 1989.

[44] B. Mitchell. Rings with several objects. *Advances in Mathematics*, 8:1–161, 1972.

[45] F. Mora. Groebner bases for noncommutative polynomial rings. In J. Calmet, editor, *Algebraic Algorithms and Error-Correcting Codes*, LNCS# 229, pages 353–362. Springer-Verlag, Berlin, 1986.

[46] T. Mora. An introduction to commutative and non-commutative Gröbner bases. *Theoretical Computer Science*, 134:131–173, 1994.

[47] A. A. Reeves. The worst order is not always the lexicographic order. *SIGSAM Bulletin*, 25(4):18–19, 1992. Erratum vol. 26, no. 1, p.13.

[48] B. Reinert. *On Gröbner Bases in Monoid and Group Rings.* PhD thesis, FB Informatik, University of Kaiserslautern, 1995.

[49] L. Robbiano. Term orderings on the polynomial ring. In B. F. Caviness, editor, *EUROCAL '85*, LNCS# 204, pages 513–517. Springer-Verlag, Berlin, 1985.

[50] J. J. Rotman. *An introduction to homological algebra.* Academic Press, 1979.

[51] E. A. Scott. Weights for total division orderings on strings. *Theoretical Computer Science*, 135:345–359, 1994.

[52] C. C. Sims. *Computation with Finitely Presented Groups.* Cambridge University Press, New York, 1994.

[53] T. Stokes. Gröbner bases in exterior algebras. *Journal of Automated Reasoning*, 6:233–250, 1990.

[54] C. Traverso and L. Donati. Experimenting the Gröbner basis algorithm with the AlP*I*. In *ISSAC '89*, pages 192–198, 1989.

[55] W. Wechler. *Universal Algebra for Computer Scientists.* Springer-Verlag, Berlin, 1992.

[56] T. Yan. The geobucket data structure for polynomials. (paper in preparation), 1996.

Appendix A

Useless Triple Elimination

In the following, the noncommutative version of Buchberger's second criterion is proved. This result shows that the triple elimination techniques discussed in Chapter 3 are valid. The proof follows that given by Becker and Weispfenning [7] for the commutative case. (In this setting, we consider the case where the leading coefficient may be something other than one. When needed the leading coefficient of a polynomial p is denoted $LC(p)$.)

We begin with another characterization of Gröbner bases. In the following let Γ be an arbitrary directed multigraph, and B be the set of finite paths in Γ.

Definition A.1 (t-representation) *Let P be a finite subset of $K\Gamma$ such that every $p \in P$ is uniform. Let f be a non-zero, uniform element of $K\Gamma$, and $t \in B$. Then a representation*

$$f = \sum_{i \in I} \alpha_i u_i p_i v_i,$$

where $a_i \in K$, u_i and v_i are paths in B, I is a finite index set, and the $p_i \in P$ are not necessarily distinct, is a t-representation *of f with respect to P if*

$$\max_{i \in I} \{tip(u_i p_i v_i)\} \leq t.$$

A t-representation of f where $t = tip(f)$ is a *standard representation* of f with respect to P. Note that a standard representation of f with respect to P is essentially a reduction of f to zero. This fact links standard representations and Gröbner bases, as shown in the following Lemma.

Lemma A.1 *Let $P \subseteq K\Gamma$ be a generating set. Then P is a Gröbner basis for $\langle P \rangle$ if and only if every nonzero element of $\langle P \rangle$ has a standard representation with respect to P.*

Proof First, suppose that P has the property that every nonzero element of $\langle P \rangle$ has a standard representation with respect to P. This means that for every nonzero element $p \in \langle P \rangle$, there is some $q \in P$ such that $tip(q)|tip(p)$. Therefore, P is a Gröbner basis of $\langle P \rangle$.

Conversely, suppose that P is a Gröbner basis of $\langle P \rangle$. Then, every $p \in \langle P \rangle$ can be reduced to zero by elements of P. For each nonzero p, this reduction by P finds a standard representation of p. Therefore, every nonzero element of $\langle P \rangle$ has a standard representation with respect to P. □

This relationship between a Gröbner basis and standard representations is key to the elimination of triples that reduce to zero at some point during the computation. We now use this fact to link t-representations to Gröbner bases. Recall that $cm_v(w_1, w_2)$ is the common multiple of word w_1 and word w_2 with overlap v (v is a suffix of w_1 and a prefix of w_2). First, we prove a lemma needed in the following theorem.

Lemma A.2 *Let G be a Gröbner basis in $K\Gamma$. Let $p, q \in K\Gamma$ and let $w \in B$ be such that $tip(p)\, w\, tip(q) \neq 0$. Then $o(pw, wq, w)$ has a t-representation with respect to $\{p, q\}$ where $t < cm_w(tip(pw), tip(qw))$.*

Proof Let $p = \sum_{i=0}^{n} \alpha_i p_i$ and $q = \sum_{j=0}^{m} \beta_j q_j$ where $\alpha_i, \beta_j \in K$ and $p_i, q_j \in B$. Assume, without loss of generality, that $\alpha_0 = \beta_0 = 1$. Also, assume that $p_i > p_j$ for i, j such that $0 \leq i < j \leq n$ and $q_i > q_j$ for i, j such that $0 \leq i < j \leq m$.

Then the overlap relation of pw with wq and overlap w is

$$\begin{aligned} o(pw, wq, w) &= p \cdot w\, tip(q) - tip(p) w \cdot q \\ &= \left(\sum_{i=1}^{n} \alpha_i p_i \right) \cdot w q_0 - p_0 w \cdot \left(\sum_{j=1}^{m} \beta_j q_j \right). \end{aligned}$$

Using the last equation, the overlap relation can be seen to have a representation

$$\sum_{k \in \mathcal{K}} \gamma_k u_k g_k v_k$$

where $\mathcal{K} = \{(i,0) : 1 \leq i \leq m\} \cup \{(0,j) : 1 \leq j \leq n\}$, $\gamma_{(i,0)} = \alpha_i \beta_0$, $\gamma_{(0,j)} = -\alpha_0 \beta_j$, $u_{(i,0)} = src(p_i)$, $u_{(0,j)} = p_0 w$, $v_{(i,0)} = w q_0$, $v_{(0,j)} = tgt(q_j)$, $g_{(i,0)} = p_i$, and $g_{(0,j)} = q_j$. This representation is a t-representation where $t = \max_{k \in \mathcal{K}} \{tip(u_k g_k v_k)\}$. By the ordering on terms of polynomials, t is either $p_1 w q_0$ or $p_0 w q_1$. Therefore, $t < p_0 w q_0 = cm_w(tip(pw), tip(qw))$, giving the desired result. □

What this lemma shows is that trivial overlaps formed by appending a word to two polynomials can always be reduced by those polynomials (i.e., have a t-representation). This situation arises in the proof of the following theorem, which shows the connection between t-representations and Gröbner bases.

Theorem A.1 *Let G be a finite subset of uniform elements of $K\Gamma$ such that $0 \notin G$, and G is tip-reduced. Suppose that for all $g_1, g_2 \in G$ such that $tip(g_1)$ and $tip(g_2)$ overlap by v, the overlap relation $o(g_1, g_2, v)$ either equals zero or has a t-representation for some $t < cm_v(tip(g_1), tip(g_2))$. Then G is a Gröbner basis.*

Proof Assume that G is a finite subset of $K\Gamma$ such that $0 \notin G$ and every $g \in G$ is uniform. Also, suppose that every overlap relation $o(g_1, g_2, v)$ is either zero or has a t-representation for each pair of $g_1, g_2 \in G$ whose tips overlap by some $v \in B$.

Since K is a field we may assume, for convenience, that all elements of G are monic. The goal is to show that every nonzero $f \in \langle G \rangle$ has a standard representation with respect to G. Since every overlap relation r is in $\langle G \rangle$, it will follow that r also has a standard representation, proving the theorem.

We first show that every element of $\langle G \rangle$ has a t-representation for some t. Let $f \in \langle G \rangle$ be nonzero and uniform. Then since G generates $\langle G \rangle$, f can be written in terms of elements of G as

$$f = \sum_{g \in G} q_g g p_g$$

where $q_g, p_q \in K\Gamma$ are uniform for all $g \in G$. Multiplying through the pairs of terms in each q_g, p_g, this representation can be rewritten as

$$f = \sum_{i \in I} \alpha_i u_i g_i v_i$$

where $\alpha_i \in K$, $u_i, v_i \in B$, I is a finite index set, and the $g_i \in G$ are not necessarily distinct. If we let $s = \max_{i \in I} \{tip(u_i p_i v_i)\}$, then f has an s-representation.

Now choose among all such possible representations of f the one with minimal s. This minimal s must be $tip(f)$, if f has a standard representation. To obtain a contradiction, we assume that $tip(f) < s$.

We induct on the number n_s of summands in the s-representation of f such that $tip(u_i g_i v_i) = s$. Since s does not occur in f, it must cancel in the representation. Hence, $n_s > 1$. The inductive hypothesis is that for all $n < n_s$ there is an s'-representation where $s' < s$.

Suppose that $n_s = 2$. Rewrite the representation of f as

$$f = \alpha_1 u_1 g_1 v_1 + \alpha_2 u_2 g_2 v_2 + \sum_{j \in I \setminus \{1,2\}} \alpha_j u_j g_j v_j$$

where $tip(u_1 g_1 v_1) = tip(u_2 g_2 v_2) = s$.

Since s does not occur in the third term of this representation, these tips cancel in

$$\begin{aligned} g &= \alpha_1 u_1 g_1 v_1 + \alpha_2 u_2 g_2 v_2 \\ &= (\alpha_1 u_1 g_1 v_1 - \alpha_1 LC(g_1) tip(u_1 g_1 v_1)) \\ &\quad + (\alpha_2 u_2 g_2 v_2 - \alpha_2 LC(g_2) tip(u_2 g_2 v_2)). \end{aligned}$$

Therefore, $tip(g) < s$. Also, the maximum path of $\sum_{j \in I \setminus \{1,2\}} \alpha_j u_j g_j v_j$ must be less than s by the assumption $n_s = 2$.

Since s is canceled in g, it must be that $\alpha_1 LC(g_1) tip(u_1 g_1 v_1) = -\alpha_2 LC(g_2) tip(u_2 g_2 v_2)$.

To reach the contradiction, we must show that g is either zero or has a t-representation for $t < s$. By the assumption $n_s = 2$, $tip(u_1 g_1 v_1) = tip(u_2 g_2 v_2) = s$, or equivalently $u_1 tip(g_1) v_1 = u_2 tip(g_2) v_2$. Because G is tip-reduced, there are three cases that satisfy this property. The first case is that $tip(g_1)$ and $tip(g_2)$ are disjoint, the second case is that $tip(g_1)$ overlaps with $tip(g_2)$, and the third case is that $tip(g_2)$ overlaps with $tip(g_1)$. The second and the third case are symmetric, so we need only consider the first and second cases.

For the first case, assume that $tip(g_1)$ and $tip(g_2)$ are disjoint. Hence we may write $s = u_1 tip(g_1) w tip(g_2) v_2$ for some $w \in B$. Define $p_1 = g_1 w$ and $p_2 = w g_2$. Since $\alpha = \alpha_1 LC(g_1) = -\alpha_2 LC(g_2)$, it is easy to see that $g = \alpha u_1 o(p_1, p_2, w) v_2$. Lemma A.2 implies that $o(p_1, p_2, w)$ has a t-representation where $t < cm_w(tip(p_1), tip(p_2)) = tip(g_1) w tip(g_2)$. Therefore, g has a $u_1 t v_2$-representation where $u_1 t v_2 < u_1 tip(g_1) w tip(g_2) u_2 = s$, which gives the needed contradiction.

For the second case, suppose that $tip(g_1)$ overlaps $tip(g_2)$ by w. This implies

$$s = u_1 cm_w(tip(g_1), tip(g_2)) v_2.$$

Let $\alpha = \alpha_1 = -\alpha_2$. It follows that

$$g = \alpha u_1 o(g_1, g_2, w) v_2.$$

By assumption, $o(g_1, g_2, w)$ is either zero or has a t-representation

$$o(g_1, g_2, w) = \sum_{k \in \mathcal{K}} u'_k g'_k v'_k$$

for some $t < cm_w(tip(g_1), tip(g_2))$. Assuming the overlap relation is not zero (in which case g is zero), g has a u_1tv_2-representation where $u_1tv_2 < u_1cm_w(tip(g_1), tip(g_2))v_2 = s$. As in the first case, this contradicts the minimality of s.

For $n_s > 2$, we assume that for all $n < n_s$ there is an s'-representation where $s' < s$. To complete the proof, assume that $tip(u_1g_1v_1) = tip(u_2g_2v_2) = s$. Then we can modify the representation of f to

$$f = \alpha_1 u_1 g_1 v_1 - \frac{\alpha_1}{\alpha_2} u_2 g_2 v_2 + \left(\frac{\alpha_1}{\alpha_2} + 1\right) u_2 g_2 v_2 + \sum_{j \in I \setminus \{1,2\}} \alpha_j u_j g_j v_j.$$

Consider this representation as two pieces, one consisting of the first two summands together, and other consisting of the last two summands together. Let f_1 be the first part and f_2 be the second. Then each part has less than n_s occurrences of s, and so, by the inductive hypothesis, has a s'-representation for a possibly unique $s' < s$. In particular, assume f_1 has an s_1-representation and f_2 has a s_2-representation. Letting s'' be the maximum of s_2 and s_2, then f has a s''-representation. Since, $s'' < s$ this contradicts the minimality of s.

We have shown that $s = tip(f)$. Therefore, every element f of $\langle G \rangle$ has a standard representation with respect to G. By Lemma A.1 this means that G is a Gröbner basis. □

Theorem A.2 (Buchberger's Second Criterion) *Let F be a finite subset of $K\Gamma$ such that every $f \in F$ is uniform, and let g_1, g_2 and p be uniform elements of $K\Gamma$ such that $tip(g_1)$ and $tip(g_2)$ overlap by v, and $tip(p)|cm_v(tip(g_1), tip(g_2))$. Let w_1 and w_2 be the corresponding common multiples of $tip(p)$ with $tip(g_1)$ and $tip(g_2)$ respectively. In addition, assume the following conditions hold*

1. *The overlap relation of g_1 with p has a t_1-representation with respect to F for $t_1 < w_1$.*

2. *The overlap relation of p with g_2 has a t_2-representation with respect to F for $t_2 < w_2$.*

Then there exists a t-representation for the overlap relation $o(g_1, g_2, v)$ of g_1 with g_2 where $t < cm_v(tip(g_1), tip(g_2))$.

Proof Assume that

$$\begin{aligned} cm_v(tip(g_1), tip(g_2)) &= tip(g_1)\beta \\ &= \alpha tip(g_2) \\ &= l tip(p) r \end{aligned}$$

for $\alpha, \beta, l, r \in B$. Also, let s_1 be the longest common suffix of r and β, and s_2 be the longest common prefix of l and α, then $\beta = \beta' s_1$, $r = r' s_1$, $\alpha = s_2 \alpha'$, and $l = s_2 l'$ for some $\beta', \alpha', r', l' \in B$.

Then the overlap relation of g_1 with p is $g_1 \beta' - lpr'$, and the overlap relation of p with g_2 is $l'pr - \alpha' g_2$.

The overlap relation of g_1, g_2 with overlap v is

$$\begin{aligned} o(g_1, g_2, v) &= g_1 \beta - \alpha g_2 \\ &= g_1 \beta - lpr + lpr - \alpha g_2 \\ &= (g_1 \beta' - lpr') s_1 + s_2 (l'pr - \alpha' g_2). \end{aligned}$$

By condition 2, $(g_1 \beta' - lpr')$ has a t_1-representation. Therefore, the relation $(g_1 \beta' - lpr') s_1$ has a $t_1 s_1$-representation for $t_1 s_1 < cm_v(tip(g_1), tip(g_2))$. By condition 2, $s_2(l'pr - \alpha' g_2)$ has an $s_2 t_2$-representation for $s_2 t_2 < cm_v(tip(g_1), tip(g_2))$. Choosing $t = \max\{t_1 s_1, s_2 t_2\}$, we have that $o(g_1, g_2, v)$ has a t-representation where $t < cm_v(tip(g_1), tip(g_2))$. $\square$

Given Theorem A.1, Buchberger's second criterion tells us which overlap relations do not need to be computed directly. Assuming G is tip-reduced, any overlap relation $o(p, q, v)$, formed from $p, q \in G$, whose corresponding common multiple $cm_v(p, q)$ is divisible by the tip of some $g \in G$ ($g \neq p$ and $g \neq q$) can safely be ignored. (The restriction on g can actually be loosened to the condition that if $g = p$ then it cannot be a prefix of $cm_v(p, q)$, and if $g = q$ then it cannot be a suffix of $cm_v(p, q)$.) The reason that the computation of $o(p, q, v)$ can be avoided is that the division by $tip(g)$ implies that $o(p, q, v)$ can be built from the overlap relations $o(p, g, w_1)$ and $o(g, q, w_2)$ or their reduced form. The algorithms that use Buchberger's second criterion are defined in Chapter 3.

Appendix B

Suffix Tree Insertion Algorithm

The algorithm for inserting a word into a suffix tree is described. This algorithm is an adaptation of McCreight's algorithm as described by Amir *et al* [2]. The primary change is that the algorithm also finds right-overlaps (see Chapter 4) for the input word with words already in the tree.

The insertion algorithm (Figure B.1) takes a word w as an argument and inserts it into the tree (rooted by *root*). The result of the algorithm is that the suffixes of w are inserted into the tree, and a set R of right-overlaps is found. We assume that w is terminated by a special symbol '#' that does not match itself. In an implementation, either be a unique symbol is used for each input word, or no terminating symbols are used.

For each suffix s of w, the insert algorithm inserts the suffix using ST_Insert, and then finds the right-overlaps for the previous suffix using Pattern_Leaves. ST_Insert returns a node t that is the root of the subtree containing all patterns prefixed by the inserted suffix. Pattern_Leaves just searches this subtree and returns the identifiers for the corresponding words.

The variables v and p_v refer to the locus of the head of the previous suffix inserted and its parent. Both are initially given the root as their value. Once a suffix has been inserted their values may be updated to other nodes by ST_Insert. ST_Insert handles assigning the suffix link of v for each insertion but the final one. Therefore, if the previous suffix inserted was length one and v is not the root, Insert assigns the suffix link to be the root. The next suffix of w is found by suffix(s), which simply removes the first symbol of s.

The ST_Insert algorithm (Figure B.2) is the algorithm described by Amir *et al* [2]. As before, INSERT(w). Modified McCreight insertion algorithm for suffix-tree.

INPUT: A word w to be inserted.
OUTPUT: A set R of right-overlaps.

```
R ← ∅;
s ← w;
v ← root;
p_v ← root;
while (s ≠ λ) do
    begin
        (v, p_v, t) ← ST_INSERT(v, p_v, s);
        R ← R ∪ PATTERN_LEAVES(t);
        if (v ≠ root ∧ |s| = 1) then
            sufflink(v) ← root;
        s ← suffix(s)
    end
return R;
```

Figure B.1: Suffix Tree Insertion Algorithm.

the argument v is the head of the previous suffix, p_v is the parent of v (if v is the root then $p_v = v$), and s is the suffix to be inserted. The results are the head v' for s, the parent of v' and a set of nodes for finding right-overlaps (the set is either a singleton node, or is empty).

The algorithm uses the fact that head(w_{k+1}) and head(w_k) have the following relationship: if head(w_{k+1}) $= a\alpha$ for some symbol a and some possibly empty word α, then α is a prefix of head(w_k). So, if we have the locus of head(w_{k+1}) we don't have to scan the common prefix with the tree.

ST_INSERT decides how to handle the insertion. If v is the root, then the previous head was empty. Therefore, the whole suffix must be scanned to find its locus in the tree. To do this, the function SCAN is called. Otherwise, there is prior information that can be used. If p_v is the root, then the arc from p_v to v is the previous head. To find the next head, the first symbol must be removed before calling the function RESCAN. In the case where p_v is not the root, the RESCAN of the previous head is begun from the locus of the first suffix of $l(p_v)$ found by following the suffix link of p_v.

The SCAN algorithm (Figure B.3) is used when no information about prior suffixes is available (or has been "used up" by the RESCAN algorithm). SCAN matches the suffix with the labels of the arcs in the tree until the scan stops by either reaching a locus for the suffix, or reaches a node for

ST_INSERT(v, p_v, s). McCreight algorithm for inserting a suffix into tree.

INPUT: Locus v of previous head, its parent p_v, and suffix s.
OUTPUT: Locus v' of head of s, parent of v' and set of nodes.

```
if (v = root) then
    return SCAN(v, v, s);
else
    if (p_v = root) then
        β ← suffixlink(label(v));
        return RESCAN(v, p_v, β, s);
    else
        β ← label(v);
        return RESCAN(v, suffixlink(p_v), β, s);
```

Figure B.2: Algorithm for Insertion of a Single Suffix.

which no out-arc has a label matching the suffix.

The input to SCAN is a node y of the tree and a suffix s. The algorithm returns the locus of the head of s and a set containing the locus of s (minus the special tag character added to the end). In addition to finding the locus of s, SCAN also increments the pattern child counts of each node encountered. If s is the full word w, then the counter for each node encountered is incremented, otherwise the counter is left alone.

The scan of s is done by finding an arc from y to a child f whose label has a prefix α that is also a prefix of s. If α is the label of the arc from y to f, then the prefix α is removed from s and the process is repeated from f. Otherwise, if α is empty, there is no matching child and so a new child v of y is added and the arc from y to v is labeled by s. If α is nonempty then the arc to f is split by adding a child p of y (with arc labeled α) with children f and a new node v as a locus for s. Note that SCAN only returns a non-empty set containing the parent of the locus of s when the label of the arc to the leaf for s is the special unique terminating symbol.

The RESCAN algorithm (Figure B.4) is used when the previous head is non-empty. Aside from inserting the suffix s, RESCAN has the goal of setting the suffix link for the locus v of the previous head. The inputs are the node v and the node x (found in ST_INSERT as the node pointed to by the suffix link of the parent of v). The other two arguments are the words β and s, where β is a

SCAN(y, s). Construct locus of *head*(s).

INPUT: A node y, and a suffix s.
OUTPUT: A node v locus of *head*(s), and set of nodes.

```
if ( s is full pattern ) then
    c ← 1;
else
    c ← 0;
count(y) ← count(y) + c;
(f, α) ← FindMatch(y, s);
while (α = label(f)) do begin
    y ← f;
    count(y) ← count(y) + c;
    s ← s \ α;
    (f, α) ← FindMatch(y, s);
end
if (α = #) then
    child(y) ← child(y) ∪ {v};
    label(v) ← s;
    if (s = #) then
        return (v, {y});
    else
        return (v, {});
else
    child(y) ← child(y) ∪ {p};
    child(p) ← child(y) ∪ {f, v};
    count(p) ← count(f);
    label(p) ← α;
    label(v) ← s \ α;
    return (v, {});
```

Figure B.3: Scan Algorithm.

prefix of a path from x and s is the suffix being inserted.

The function MATCHING_CHILD finds a child f of x whose arc matches β (since f is guaranteed to exists, only the first symbols of β and arc label are needed to make the match). The first loop of RESCAN uses MATCHING_CHILD to traverse the tree to find a path from x labeled by β. If the loop ends with β equal to the matching arc label, then the corresponding node f is the target of the suffix link of v, and scan is called to finish the insertion of s. Otherwise, the locus of β must be constructed as in SCAN.

RESCAN(v, x, β, s). Construct suffix link of $head(w_{k+1})$ where $s = w_k$.

INPUT: Locus v of previous head, x, β, s
OUTPUT: Locus v' of head of s, parent of v' and set of nodes.

```
f ← MatchingChild(x, β);
α ← label(f);
while (|α| < |β|) do begin
      β ← β \ α;
      x ← f;
      f ← MatchingChild(x, β);
      α ← label(f);
end
if (|α| = |β|) then
      suffix(v) ← f;
      return Scan(f, x, s \ l(x));
else begin
      label(f) ← α \ β;
      s ← s \ l(x);
      label(d) ← β;
      count(d) ← count(f);
      child(x) ← child(x) \ {f};
      child(x) ← child(x) ∪ {d};
      child(d) ← child(x) ∪ {f, w};
      label(w) ← s \ l(d);
      suffix(v) ← d;
      if (label(w) = #) then
            return (d, {f});
      else
            return (d, {});
end
```

Figure B.4: Rescan Algorithm.

Appendix C

Problem Instances

C.1 Free Algebras

The problem instances in this section all come from free algebras. Some problems may include the monomial 1 which is not valid in a path algebra. For the experiments, in the instances where 1 occurs, 1 was replaced by the single vertex (1).

C.1.1 A4 through A8

All five of these problems are from the free algebra $\mathbb{Z}_p \langle a, b, c \rangle$, where for the experiments $p = 32117$. The generators each problem are given separately below.

A4

$$\begin{aligned} aa + 5ab + 7ac + 11ba + 2bb + 31bc + 19ca + 13cb + 23cc, \\ ab + 5ac + 7ba + 11bb + 2bc + 31ca + 19cb + 13cc, \\ ac + 5ba + 7bb + 11bc + 2ca + 31cb + 19cc, \\ ba + 5bb + 7bc + 11ca + 2cb + 31cc \end{aligned}$$

A5

$$aa + 5ab + 7ac + 11ba + 2bb + 31bc + 19ca + 13cb + 23cc,$$
$$ab + 5ac + 7ba + 11bb + 2bc + 31ca + 19cb + 13cc,$$
$$ac + 5ba + 7bb + 11bc + 2ca + 31cb + 19cc,$$
$$ba + 5bb + 7bc + 11ca + 2cb + 31cc,$$
$$bb + 5bc + 7ca + 11cb + 2cc$$

A6

$$aa + 5ab + 7ac + 11ba + 2bb + 31bc + 19ca + 13cb + 23cc,$$
$$ab + 5ac + 7ba + 11bb + 2bc + 31ca + 19cb + 13cc,$$
$$ac + 5ba + 7bb + 11bc + 2ca + 31cb + 19cc,$$
$$ba + 5bb + 7bc + 11ca + 2cb + 31cc,$$
$$bb + 5bc + 7ca + 11cb + 2cc,$$
$$bc + 5ca + 7cb + 11cc$$

A7

$$aa + 5ab + 7ac + 11ba + 2bb + 31bc + 19ca + 13cb + 23cc,$$
$$ab + 5ac + 7ba + 11bb + 2bc + 31ca + 19cb + 13cc,$$
$$ac + 5ba + 7bb + 11bc + 2ca + 31cb + 19cc,$$
$$ba + 5bb + 7bc + 11ca + 2cb + 31cc,$$
$$bb + 5bc + 7ca + 11cb + 2cc,$$
$$bc + 5ca + 7cb + 11cc,$$
$$ca + 5cb + 7cc$$

A8

$$aa + 5ab + 7ac + 11ba + 2bb + 31bc + 19ca + 13cb + 23cc,$$
$$ab + 5ac + 7ba + 11bb + 2bc + 31ca + 19cb + 13cc,$$
$$ac + 5ba + 7bb + 11bc + 2ca + 31cb + 19cc,$$
$$ba + 5bb + 7bc + 11ca + 2cb + 31cc,$$
$$bb + 5bc + 7ca + 11cb + 2cc,$$
$$bc + 5ca + 7cb + 11cc,$$
$$ca + 5cb + 7cc,$$
$$cb + 5cc$$

C.1.2 Control Theory Problems

The following instances are examples from the paper by Helton and Wavrik on applying Gröbner bases in control theory [31].

HWEB The alphabet for this instance is $\{x, x', y, y', (1-xy)^{-1}, (1-yx)^{-1}\}$.

$$\begin{aligned}
&xx'-1, \quad (1-xy)^{-1}xy-(1-xy)^{-1}+1,\\
&x'x-1, \quad xy(1-xy)^{-1}-(1-xy)^{-1}+1,\\
&yy'-1, \quad (1-yx)^{-1}yx-(1-yx)^{-1}+1,\\
&y'y-1, \quad yx(1-yx)^{-1}-(1-yx)^{-1}+1
\end{aligned}$$

HWRES The alphabet for the HWRES instance is $\{x, x', (1-x)^{-1}\}$.

$$\begin{aligned}
&xx'-1, \quad (1-x)^{-1}x-(1-x)^{-1}+1,\\
&x'x-1, \quad x(1-x)^{-1}-(1-x)^{-1}+1
\end{aligned}$$

C.2 Other Free Instances

P4 The P4 instance is from the free algebra over the two letter alphabet $\{a, b\}$.

$$baa-aaa, \quad aba-aa$$

P6 The P6 instance is from the free algebra over the three letter alphabet $\{a, b, c\}$.

$$ccc+2ccb+3cca+5bcc+7aca, \quad bcc+11bab+13aaa$$

ELP The ELP instance is from the free algebra over the three letter alphabet $\{x, y, z\}$.

$$\begin{aligned}
&xxyy-xxzz, \quad xyyz,\\
&yyzz-yyxx, \quad yzzx,\\
&zzxx-zzyy, \quad zxxy
\end{aligned}$$

C.3 Path Algebras

This section includes the path algebra instances (other than the mesh algebra instances) used in the experimentation. Some of the path algebra problems were randomly generated and the parameters used to generate them are given in the next section.

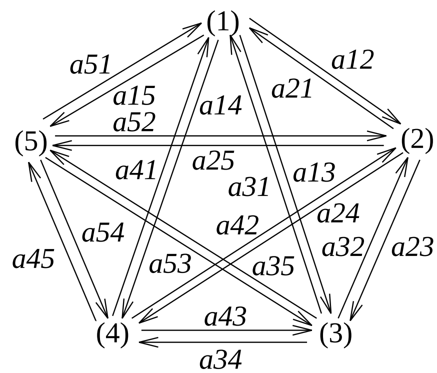

Figure C.1: Quiver for CGL, CGL1, and CG5.

C.3.1 CGL and Derivatives

The following three problems are based on the complete graph K_5 in Figure C.1.

CGL

$$a13a32 - a12, \quad a14a42 - a12, \quad a15a52 - a12, \quad a12a23 - a13,$$
$$a14a43 - a13, \quad a15a53 - a13, \quad a12a24 - a14, \quad a13a34 - a14,$$
$$a15a54 - a14, \quad a12a25 - a15, \quad a13a35 - a15, \quad a14a45 - a15,$$
$$a23a31 - a21, \quad a24a41 - a21, \quad a25a51 - a21, \quad a21a13 - a23,$$
$$a24a43 - a23, \quad a25a53 - a23, \quad a21a14 - a24, \quad a23a34 - a24,$$
$$a25a54 - a24, \quad a21a15 - a25, \quad a23a35 - a25, \quad a24a45 - a25,$$
$$a32a21 - a31, \quad a34a41 - a31, \quad a35a51 - a31, \quad a31a12 - a32,$$
$$a34a42 - a32, \quad a35a52 - a32, \quad a31a14 - a34, \quad a32a24 - a34,$$
$$a35a54 - a34, \quad a31a15 - a35, \quad a32a25 - a35, \quad a34a45 - a35,$$
$$a42a21 - a41, \quad a43a31 - a41, \quad a45a51 - a41, \quad a41a12 - a42,$$
$$a43a32 - a42, \quad a45a52 - a42, \quad a41a13 - a43, \quad a42a23 - a43,$$
$$a45a53 - a43, \quad a41a15 - a45, \quad a42a25 - a45, \quad a43a35 - a45,$$
$$a52a21 - a51, \quad a53a31 - a51, \quad a54a41 - a51, \quad a51a12 - a52,$$
$$a53a32 - a52, \quad a54a42 - a52, \quad a51a13 - a53, \quad a52a23 - a53,$$
$$a54a43 - a53, \quad a51a14 - a54, \quad a52a24 - a54, \quad a53a34 - a54$$

CGL1

$$
\begin{array}{cccc}
a13a32 - a12, & a14a42 - a12, & a15a52 - a12, & a13a32 - a14a42, \\
a13a32 - a15a52, & a14a42 - a15a52, & a12a23 - a13, & a14a43 - a13, \\
a15a53 - a13, & a12a23 - a14a43, & a12a23 - a15a53, & a14a43 - a15a53, \\
a12a24 - a14, & a13a34 - a14, & a15a54 - a14, & a12a24 - a13a34, \\
a12a24 - a15a54, & a13a34 - a15a54, & a12a25 - a15, & a13a35 - a15, \\
a14a45 - a15, & a12a25 - a13a35, & a12a25 - a14a45, & a13a35 - a14a45, \\
a23a31 - a21, & a24a41 - a21, & a25a51 - a21, & a23a31 - a24a41, \\
a23a31 - a25a51, & a24a41 - a25a51, & a21a13 - a23, & a24a43 - a23, \\
a25a53 - a23, & a21a13 - a24a43, & a21a13 - a25a53, & a24a43 - a25a53, \\
a21a14 - a24, & a23a34 - a24, & a25a54 - a24, & a21a14 - a23a34, \\
a21a14 - a25a54, & a23a34 - a25a54, & a21a15 - a25, & a23a35 - a25, \\
a24a45 - a25, & a21a15 - a23a35, & a21a15 - a24a45, & a23a35 - a24a45, \\
a32a21 - a31, & a34a41 - a31, & a35a51 - a31, & a32a21 - a34a41, \\
a32a21 - a35a51, & a34a41 - a35a51, & a31a12 - a32, & a34a42 - a32, \\
a35a52 - a32, & a31a12 - a34a42, & a31a12 - a35a52, & a34a42 - a35a52, \\
a31a14 - a34, & a32a24 - a34, & a35a54 - a34, & a31a14 - a32a24, \\
a31a14 - a35a54, & a32a24 - a35a54, & a31a15 - a35, & a32a25 - a35, \\
a34a45 - a35, & a31a15 - a32a25, & a31a15 - a34a45, & a32a25 - a34a45, \\
a42a21 - a41, & a43a31 - a41, & a45a51 - a41, & a42a21 - a43a31, \\
a42a21 - a45a51, & a43a31 - a45a51, & a41a12 - a42, & a43a32 - a42, \\
a45a52 - a42, & a41a12 - a43a32, & a41a12 - a45a52, & a43a32 - a45a52, \\
a41a13 - a43, & a42a23 - a43, & a45a53 - a43, & a41a13 - a42a23, \\
a41a13 - a45a53, & a42a23 - a45a53, & a41a15 - a45, & a42a25 - a45, \\
a43a35 - a45, & a41a15 - a42a25, & a41a15 - a43a35, & a42a25 - a43a35, \\
a52a21 - a51, & a53a31 - a51, & a54a41 - a51, & a52a21 - a53a31, \\
a52a21 - a54a41, & a53a31 - a54a41, & a51a12 - a52, & a53a32 - a52, \\
a54a42 - a52, & a51a12 - a53a32, & a51a12 - a54a42, & a53a32 - a54a42, \\
a51a13 - a53, & a52a23 - a53, & a54a43 - a53, & a51a13 - a52a23, \\
a51a13 - a54a43, & a52a23 - a54a43, & a51a14 - a54, & a52a24 - a54, \\
a53a34 - a54, & a51a14 - a52a24, & a51a14 - a53a34, & a52a24 - a53a34
\end{array}
$$

CG5

$$\begin{aligned}
&a13a32 - a14a42, \quad a13a32 - a15a52, \quad a14a42 - a15a52,\\
&a12a23 - a14a43, \quad a12a23 - a15a53, \quad a14a43 - a15a53,\\
&a12a24 - a13a34, \quad a12a24 - a15a54, \quad a13a34 - a15a54,\\
&a12a25 - a13a35, \quad a12a25 - a14a45, \quad a13a35 - a14a45,\\
&a23a31 - a24a41, \quad a23a31 - a25a51, \quad a24a41 - a25a51,\\
&a21a13 - a24a43, \quad a21a13 - a25a53, \quad a24a43 - a25a53,\\
&a21a14 - a23a34, \quad a21a14 - a25a54, \quad a23a34 - a25a54,\\
&a21a15 - a23a35, \quad a21a15 - a24a45, \quad a23a35 - a24a45,\\
&a32a21 - a34a41, \quad a32a21 - a35a51, \quad a34a41 - a35a51,\\
&a31a12 - a34a42, \quad a31a12 - a35a52, \quad a34a42 - a35a52,\\
&a31a14 - a32a24, \quad a31a14 - a35a54, \quad a32a24 - a35a54,\\
&a31a15 - a32a25, \quad a31a15 - a34a45, \quad a32a25 - a34a45,\\
&a42a21 - a43a31, \quad a42a21 - a45a51, \quad a43a31 - a45a51,\\
&a41a12 - a43a32, \quad a41a12 - a45a52, \quad a43a32 - a45a52,\\
&a41a13 - a42a23, \quad a41a13 - a45a53, \quad a42a23 - a45a53,\\
&a41a15 - a42a25, \quad a41a15 - a43a35, \quad a42a25 - a43a35,\\
&a52a21 - a53a31, \quad a52a21 - a54a41, \quad a53a31 - a54a41,\\
&a51a12 - a53a32, \quad a51a12 - a54a42, \quad a53a32 - a54a42,\\
&a51a13 - a52a23, \quad a51a13 - a54a43, \quad a52a23 - a54a43,\\
&a51a14 - a52a24, \quad a51a14 - a53a34, \quad a52a24 - a53a34
\end{aligned}$$

C.3.2 DCYC and ICYC

DCYC The quiver for the DCYC instance is shown in Figure C.2 and the relations are the following.

$$\begin{aligned}
&abc + di, \quad id + iaf, \quad fi + fgc + bhi,\\
&cab + hg, \quad ef + ebh, \quad dg + deb + afg,\\
&bca + fe, \quad gh + gcd, \quad he + hia + cde
\end{aligned}$$

ICYC The quiver for the ICYC instance is shown in Figure C.3 and the relations are the following.

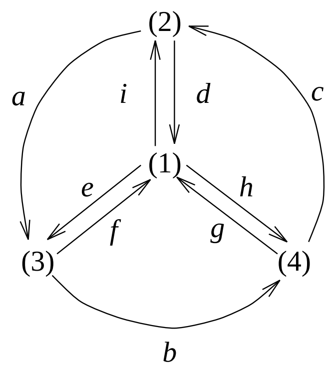

Figure C.2: Quiver for the DCYC Problem Instance.

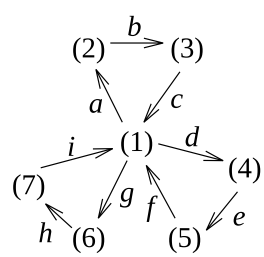

Figure C.3: Quiver for the ICYC Problem Instance.

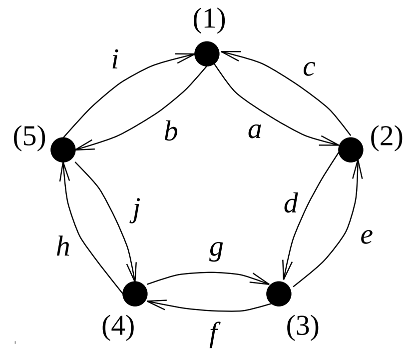

Figure C.4: Quiver for P5 Instance.

$$abcghi + def, \quad hidefg + hig, \quad bcghid + bcd,$$
$$hiabcg + hig, \quad cghiab + cab$$

C.3.3 P5

The instance P5 is a Froebenius algebra (see Rotman [50] for a definition). The quiver is shown in Figure C.4 and the relations are given below.

$$\begin{array}{ccccc} ac - bi, & de - ca, & fg - ed, & hj - gf, & ib - jh, \\ adfhia, & dfhiad, & fhiadf, & hiadfh, & iadfhi, \\ bjgecb, & jgecbj, & gecbjg, & ecbjge, & cbjgec \end{array}$$

C.3.4 Binary Tree Quivers

The three problems BT7, BT31, and M39 are mesh algebra instances whose quivers are binary trees (the first two), slightly modified trees (the latter).

BT7 This problem is a mesh algebra for the graph shown in Figure C.5, with the corresponding relations shown below.

$$\begin{array}{rll} a12b21 + a13b31, & b42a24, & b73a37 \\ a24b42 + a25b52 + b21a12, & b52a25, & \\ a36b63 + a37b73 + b31a13, & b63a36, & \end{array}$$

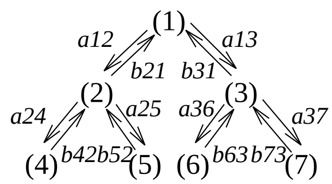

Figure C.5: Quiver for BT7 instance.

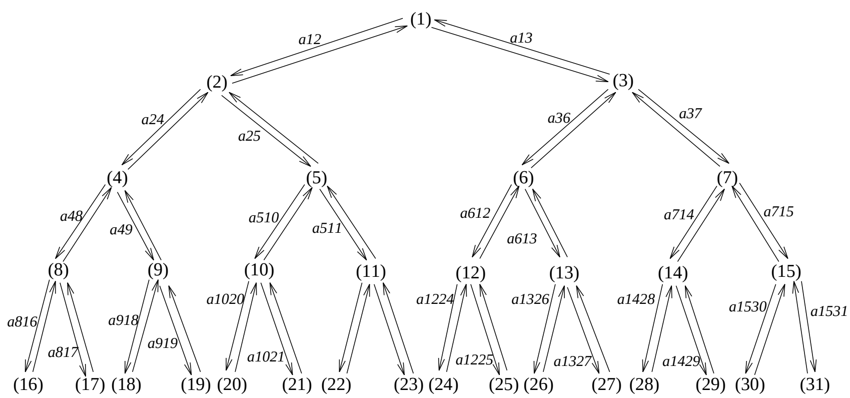

Figure C.6: Quiver for BT31 instance.

BT31 This problem is a mesh algebra for the graph shown in Figure C.6, with the corresponding relations shown below.

$$
\begin{array}{cc}
a12b12 + a13b13, & b816a816, \\
a24b24 + a25b25 + b12a12, & b817a817, \\
a48b48 + a49b49 + b24a24, & b918a918, \\
a510b510 + a511b511 + b25a25, & b919a919, \\
a36b36 + a37b37 + b13a13, & b1020a1020, \\
a612b612 + a613b613 + b36a36, & b1021a1021, \\
a714b714 + a715b715 + b37a37, & b1122a1122, \\
a816b816 + a817b817 + b48a48, & b1123a1123, \\
a918b918 + a919b919 + b49a49, & b1224a1224, \\
a1020b1020 + a1021b1021 + b510a510, & b1225a1225, \\
a1122b1122 + a1123b1123 + b511a511, & b1326a1326, \\
a1224b1224 + a1225b1225 + b612a612, & b1327a1327, \\
a1326b1326 + a1327b1327 + b613a613, & b1428a1428, \\
a1428b1428 + a1429b1429 + b714a714, & b1429a1429, \\
a1530b1530 + a1531b1531 + b715a715, & b1530a1530, \\
b1531a1531 &
\end{array}
$$

M39 This problem is a mesh algebra for the graph shown in Figure C.7, with the corresponding relations shown below.

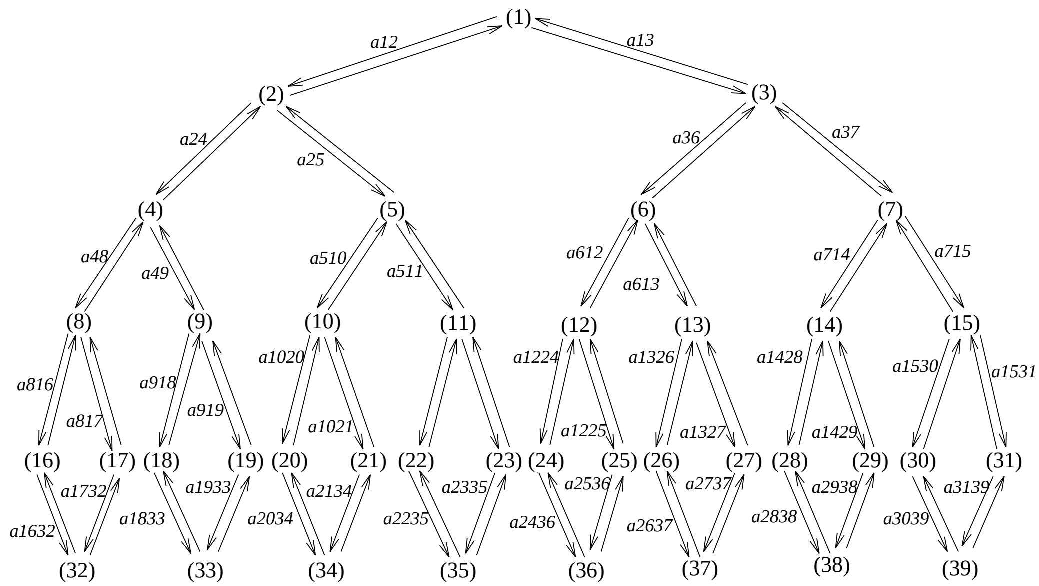

Figure C.7: Quiver for M39 instance.

$$
\begin{array}{cc}
a12b12 + a13b13, & a1632b1632 + b816a816, \\
a24b24 + a25b25 + b12a12, & a1732b1732 + b817a817, \\
a48b48 + a49b49 + b24a24, & a1833b1833 + b918a918, \\
a510b510 + a511b511 + b25a25, & a1933b1933 + b919a919, \\
a36b36 + a37b37 + b13a13, & a2034b2034 + b1020a1020, \\
a612b612 + a613b613 + b36a36, & a2134b2134 + b1021a1021, \\
a714b714 + a715b715 + b37a37, & a2235b2235 + b1122a1122, \\
a816b816 + a817b817 + b48a48, & a2335b2335 + b1123a1123, \\
a918b918 + a919b919 + b49a49, & a2436b2436 + b1224a1224, \\
a1020b1020 + a1021b1021 + b510a510, & a2536b2536 + b1225a1225, \\
a1122b1122 + a1123b1123 + b511a511, & a2637b2637 + b1326a1326, \\
a1224b1224 + a1225b1225 + b612a612, & a2737b2737 + b1327a1327, \\
a1326b1326 + a1327b1327 + b613a613, & a2838b2838 + b1428a1428, \\
a1428b1428 + a1429b1429 + b714a714, & a2938b2938 + b1429a1429, \\
a1530b1530 + a1531b1531 + b715a715, & a3039b3039 + b1530a1530, \\
a3139b3139 + b1531a1531 &
\end{array}
$$

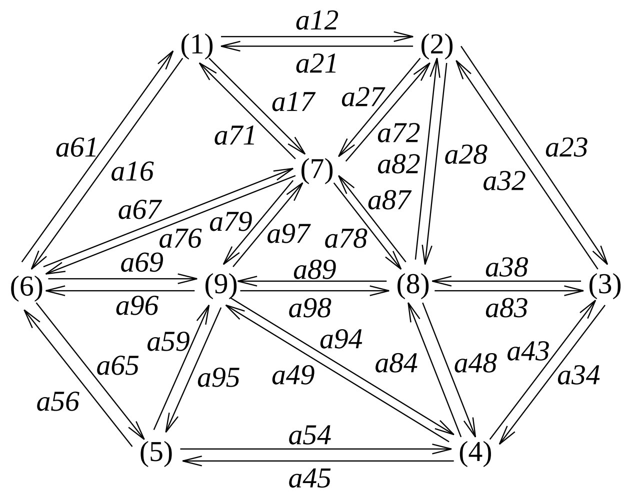

Figure C.8: M1 Quiver.

C.3.5 M1 and Derivatives

All of the following are mesh algebra problem instances, and all use the same quiver, which is shown in Figure C.8. The difference between them is how the paths in the relations cover the graph. All instances are mesh algebras.

M1 The M1 instance includes all mesh relations for every node of the graph.

$$a12a21 + a17a71 + a16a61,$$
$$a32a23 + a34a43 + a38a83,$$
$$a54a45 + a56a65 + a59a95,$$
$$a21a12 + a23a32 + a27a72 + a28a82,$$
$$a43a34 + a45a54 + a48a84 + a49a94,$$
$$a61a16 + a65a56 + a67a76 + a69a96,$$
$$a71a17 + a72a27 + a76a67 + a78a87 + a79a97,$$
$$a82a28 + a83a38 + a84a48 + a87a78 + a89a98,$$
$$a94a49 + a95a59 + a96a69 + a97a79 + a98a89.$$

MBFS

$$\begin{array}{ccc} a12a21 + a17a71 + a16a61, & a43a34, & a56a65, \\ a21a12 + a23a32 + a28a82, & a71a17, & a82a28, \\ a61a16 + a65a56 + a69a96, & a32a23 + a34a43, & a96a69. \end{array}$$

MDFS

$$\begin{array}{ccc} a21a12 + a23a32, & a54a45 + a56a65, & a12a21, \\ a32a23 + a34a43, & a65a56 + a69a96, & a79a97, \\ a43a34 + a45a54, & a96a69 + a97a79 + a98a89, & a89a98, \end{array}$$

MT1

$$\begin{array}{ccc} a17a71, & a49a94, & a71a17 + a76a67 + a78a87 + a79a97, \\ a28a82, & a59a95, & a82a28 + a83a38 + a87a78, \\ a38a83, & a67a76, & a94a49 + a95a59 + a97a79. \end{array}$$

MT2

$$\begin{array}{ccc} a17a71, & a49a94, & a71a17 + a76a67 + a78a87 + a79a97, \\ a28a82, & a59a95, & a82a28 + a83a38 + a87a78 + a89a98, \\ a38a83, & a67a76, & a94a49 + a95a59 + a97a79 + a98a89. \end{array}$$

MT3

$$\begin{array}{ccc} a17a71 + a16a61, & a49a94 + a45a54, & a71a17 + a76a67 + a78a87 + a79a97, \\ a28a82 + a23a32, & a59a95 + a54a45, & a82a28 + a83a38 + a87a78, \\ a38a83 + a32a23, & a67a76 + a61a16, & a94a49 + a95a59 + a97a79. \end{array}$$

MT4

$$\begin{array}{ccc} a12a21 + a16a61, & a61a16 + a67a76 + a69a96, & a21a12, \\ a43a34 + a45a54, & a76a67 + a78a87, & a34a43, \\ a54a45 + a59a95, & a95a59 + a96a69, & a87a78, \end{array}$$

MTRI

$$\begin{array}{ccc} a12a21 + a17a71, & a43a34 + a48a84, & a71a17 + a72a27 + a78a87 + a79a97, \\ a21a12 + a27a72, & a56a65 + a59a95, & a83a38 + a84a48 + a87a78 + a89a98, \\ a34a43 + a38a83, & a65a56 + a69a96, & a95a59 + a96a69 + a97a79 + a98a89. \end{array}$$

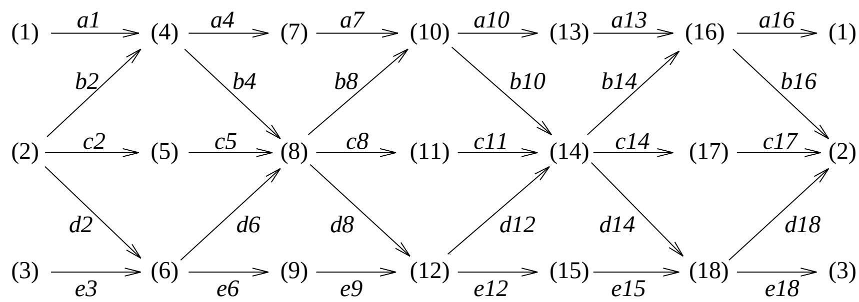

Figure C.9: Quiver of the MS Instance.

C.3.6 MS, MTB, MM

These three instances are mesh algebras defined in terms of graphs that have alternate paths of length two from (almost) every node. The MS instance is the simplest, with the other two making slight modifications of the graph. The relations for all three problem instances are sums of the uniform paths of length two between distinct nodes.

MS The quiver for the MS instance is shown in Figure C.9 and the relations are given below. Note that the nodes on the left and right sides of the diagram in Figure C.9 are the same (so the graph forms a tube).

$$\begin{array}{ccc}
a1a4, & a4a7 + b4b8, & b2b4 + c2c5 + d2d6, \\
a7a10, & a10a13 + b10b14, & b8b10 + c8c11 + d8d12, \\
a13a16, & a16a1 + b16b2, & b14b16 + c14c17 + d14d18 \\
e3e6, & d6d8 + e6e9, & \\
e9e12, & d12d14 + e12e15, & \\
e15e18, & d18d2 + e18e3, &
\end{array}$$

MTB The MTB instance is similar to the MS instance, except the quiver (Figure C.10) forms a torus.

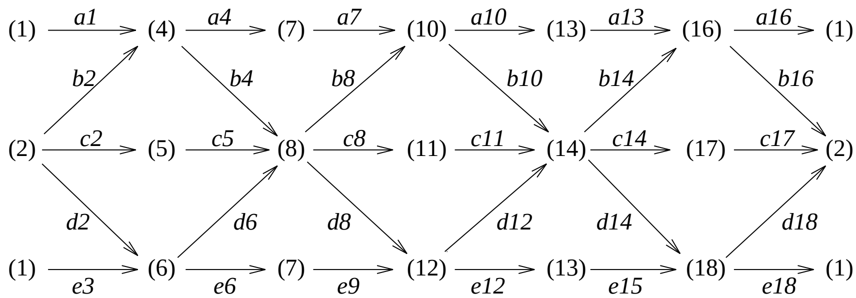

Figure C.10: Quiver of the MTB Instance.

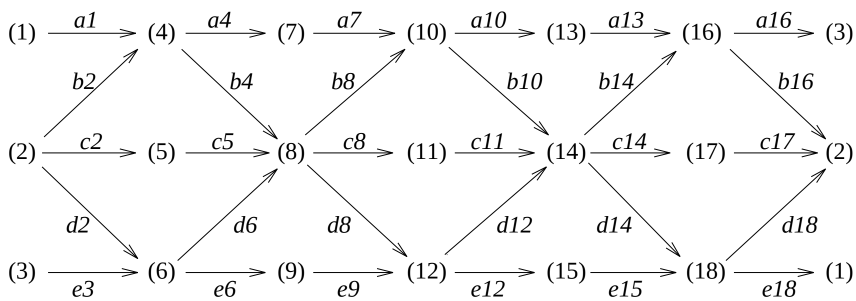

Figure C.11: Quiver of the MM Instance.

$$
\begin{array}{ccc}
a1a4, & a4a7 + b4b8, & b2b4 + c2c5 + d2d6, \\
a7a10, & a10a13 + b10b14, & b8b10 + c8c11 + d8d12, \\
a13a16, & a16a1 + b16b2, & b14b16 + c14c17 + d14d18. \\
e3e6, & d6d8 + e6e9, & \\
e9e12, & d12d14 + e12e15, & \\
e15e18, & d18d2 + e18e3, &
\end{array}
$$

MM The MM instance is also based on the MS instance. The difference is that the quiver (Figure C.11) is a moebius strip (the sequence of left and right nodes in the Figure is reversed on one-side is reversed from the other side).

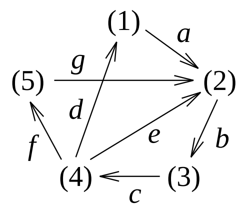

Figure C.12: Quiver for A51E Problem Instance.

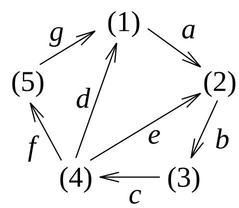

Figure C.13: Quiver for A51H Problem Instance.

$$\begin{array}{ccc}
a1a4, & a4a7 + b4b8, & b2b4 + c2c5 + d2d6, \\
a7a10, & a10a13 + b10b14, & b8b10 + c8c11 + d8d12, \\
a13a16, & a16e3 + b16d2, & b14b16 + c14c17 + d14d18. \\
e3e6, & d6d8 + e6e9, & \\
e9e12, & d12d14 + e12e15, & \\
e15e18, & d18b2 + e18a1, &
\end{array}$$

C.4 Random Instances

All of the random instances were generated using the algorithms given in Appendix D. The parameters for generating these problems are given here.

C.4.1 A51E and A51H

The A51E and A51H instances were generated for the graphs shown in Figure C.12 and Figure C.13. These graphs are modifications of a four node graph (from the A51 instance discussed in Chapter 5) by adding a fifth node in different positions.

Both instances were generated with the same parameters (other than the graph). The parameters, other than the graphs, are a seed of 97532791, an expected number of 2 polynomials in each uniform equivalence class, expected number of 3 terms in a polynomial, and the expected number of 3 loops in a path. In both instances, all monomial relations were removed.

C.4.2 AGS

The AGS instance has both a randomly generated graph and relations. The graph is acyclic and was generated using the seed 1719133751, size 15, and an edge probability of 0.5. The resulting graph has 58 arcs. The relations where generated using this graph with a seed of 97311519, an expected number of 2 polynomials for each uniform equivalence class, expected polynomial length of 10, and an expected number of 1 loop in a path. All monomial relations were then removed.

C.4.3 GL

The GL instance has both a randomly generated graph and relations. The graph is acyclic and was generated using the seed 1719133751, size 30, and an edge probability of 0.5. The relations where generated using this graph with a seed of 975332717, an expected number of 4 polynomials for each uniform equivalence class, expected polynomial length of 4, and an expected number of 1 loop in a path.

Appendix D

Problem Instance Generation

Some of the problem instances used in this research were randomly generated. A program that generated random graphs and a program that generated random generating sets were used. Both programs are written in Standard ML and are available (by ftp). The generation functions from these programs are given below.

D.1 Graph generation

Three functions are used for generating graphs. Each generates a different (but not disjoint) class of graphs. The function `gen_graph` generates directed graphs possibly with loops but no multiple edges; `gen_multigraph` generates directed multigraphs (possibly with loops and multiple edges); and `gen_acyclic` generates directed graphs with no cycles (and no multiple edges or loops).

All three functions take three arguments. The first argument is the number of vertices, the second argument is the probability, and the third function is the random number generator function. The program used to generate graphs using these functions is distributed with the Opal system.

D.2 Generating Set Generation

```
fun gen_graph (n, prob, rand) =
    let
        val i = ref 0 (* vertex counters *)
        and j = ref 0
        and p = ref 0.0 (* edge probability *)
        and g = new_graph n
    in
        i := 1;
        while (!i <= n) do
            (
            j := 1;
            while (!j <= n) do
                (
                if (!j <> !i) then
                    (
                    p := Random.norm ((rand ()));
                    if (!p <= prob) then
                        add_arc(!i, !j, g)
                    else ()
                    )
                else ();
                j := !j + 1
                );
            i := !i + 1
            );
        g
    end (* gen_graph *)
```

Figure D.1: Graph Generation Function.

```
fun gen_multigraph (n, prob, rand) =
    let
        val i = ref 0 (* vertex counters *)
        and j = ref 0
        and p = ref 0.0 (* edge probability *)
        and g = new_graph n
    in
        i := 1;
        while (!i <= n) do
            (
            j := 1;
            while (!j <= n) do
                (
                p := Random.norm ((rand ()));
                while (!p <= prob) do
                    (
                    add_arc(!i, !j, g);
                    p := Random.norm ((rand ()))
                    );
                j := !j + 1
                );
            i := !i + 1
            );
        g
    end (* gen_multigraph *)
```

Figure D.2: Multigraph Generation Function

```
fun gen_acyclic (n, prob, rand) =
    let
        val i = ref 0 (* vertex counters *)
        and j = ref 0
        and p = ref 0.0 (* edge probability *)
        and g = new_graph n
    in
        i := 1;
        while (!i <= n) do
            (
            j := !i + 1;
            while (!j <= n) do
                (
                p := Random.norm ((rand ()));
                if (!p <= prob) then
                    add_arc(!i, !j, g)
                else ();
                j := !j + 1
                );
            i := !i + 1
            );
        g
    end (* gen_acyclic *)
```

Figure D.3: Acyclic Graph Generation

```
fun gen_coefficient (rand) =
    let
        val num_primes = real(Primes.num())
        val r = Random.norm (rand ())
        val k = floor ((num_primes - 1.0) * r)
    in
        Poly.Field.mkelem(Integer.makestring(Primes.nth k))
    end
```

Figure D.4: Coefficient Generation Function.

```
fun successor_list ([],g) = []
    | successor_list (sg,g) =
    let
        val l1 = ref sg
        and l2 = ref sg
        and w_succ_l = ref []
        and succ_list = ref []
    in
        while (!l1 <> []) do
            (
            w_succ_l := [];
            l2 := sg;
            while (!l2 < > []) do
                (
                if Graph.arcExists(hd(!l1),hd(!l2),g) then
                    w_succ_l := (hd(!l2) :: !w_succ_l)
                else ();
                l2 := tl(!l2)
                );
            if (!w_succ_l <> []) then
                succ_list := (hd(!l1),!w_succ_l) :: (!succ_list)
            else ();
            l1 := tl(!l1)
            );
        !succ_list
    end (* successor_list *)
```

Figure D.5: Function to Compute Set of Successors of Node.

```
fun successors (v,[]) = []
    | successors (v,(h_v,l)::t) =
        if (v = h_v) then l
        else successors (v,t)
```

Figure D.6: Function to Compute Successors of Node in Graph.

```
fun gen_path(s, t, sg, g, path_pr, rand) =
    let
        fun select_arc (s,t,rand) =
            let
                val l = Graph.arcList(s,t,g)
                val r = Random.norm (rand ())
                val num_arcs = real (length l)
                val k = floor (r * num_arcs)
                val kth_label = List.nth (l,k)
                val arc = Graph.Arc(kth_label)
            in
                Poly.Path.mkpath(arc,g)
            end
        fun get_neighbor (c,rand) =
            let
                val N = successors(c,sg)
                val n = real (length N)
                val r = floor (n * Random.norm (rand ()))
            in
                List.nth(N,r)
                (* will raise Nth if no successors *)
            end
        val c = ref s
        and pth = ref (Poly.Path.mkpath(s,g))
        and p = ref 1.0
    in
```

Figure D.7: Function to Generate Paths (Part One).

```
    while (!p >= path_pr) do
        (
        while (!c <> t) do
            let
                val next = get_neighbor(!c,rand)
                val arc = select_arc (!c,next,rand)
            in
                pth := Poly.Path.compose(!pth,arc);
                c := next
            end;
        p := Random.norm ((rand ()));
        if (!p >= path_pr) then
            let
                val next = get_neighbor(!c,rand)
                val arc = select_arc (!c,next,rand)
            in
                pth := Poly.Path.compose(!pth,arc);
                c := next
            end
        else ()
        );
    !pth
  end (* gen_path *)
handle List.Nth => Poly.Path.zero
```

Figure D.8: Function to Generate Paths (Part Two).

```
fun gen_polynomial(s, t, sg, g, term_pr, path_pr, rand) =
    let
        val p = ref 0.0 (* term probability *)
        and ply = ref (Poly.zero)
    in
        p := Random.norm ((rand ()));
        while (!p >= term_pr) do
            let
                val pth = gen_path(s,t,sg,g,path_pr,rand)
                and k = gen_coefficient(rand)
                val m = Poly.mkmonomial(k,pth)
            in
                ply := Poly.addmon(m,!ply);
                p := Random.norm (rand ())
            end;
        !ply
    end
```

Figure D.9: Function to Generate Polynomials.

```
fun gen_generating_set (g, poly_pr, term_pr, path_pr, rand,out_str) =
    let
        val l = (Graph.listNodes g)
        val i = ref l (* vertex counters *)
        and j = ref l
        and p = ref 0.0 (* poly probability *)
        and Pset = ref (PS.emptyset())
        and ply = ref Poly.zero
        and ply_p = ref Poly.zero
        and sg' = ref []
        and sl = ref []
        fun valid_poly (p) =
                not (Poly.eqzero(p))
                andalso
                not (Poly.length(p) = 1 andalso
                Poly.Path.vertex(Poly.term(Poly.leadmon(p)))
                )
        fun print_poly (p,g,os,c) =
        if valid_poly (p) then
                Poly.pr(p,c,g,os)
        else
                ()
    in
```

Figure D.10: Function to Generate Polynomial Sets (Part One).

```
    while (!i <> []) do
        (
        j := l;
        while (!j <> []) do
            (
            p := Random.norm ((rand ()));
            if (!p >= poly_pr) then
                (
                sg' := Graph.effective_subgraph(hd(!i),hd(!j),g);
                sl := successor_list(!sg',g);
                if (!sl <> []) then
                    while (!p >= poly_pr) do
                        (
                        ply := gen_polynomial(hd(!i),hd(!j),
                            !sl, g,
                            term_pr,
                            path_pr,
                            rand);
                        print_poly(!ply_p,g,out_str,",");
                        ply_p := !ply;
                        p := Random.norm ( rand ())
                        )
            else ()
            )
            else ();
            j := tl(!j)
            );
    i := tl(!i)
    );
    print_poly(!ply_p,g,out_str,"")
end (* gen_generating_set *)
```

Figure D.11: Function to Generate Polynomial Sets (Part Two).

Appendix E

Experimental Results

E.1 Algorithm Experiments

E.1.1 Counts

Table E.1: Counts for Problem A4 Instance (Part One).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Eager, Deletion, Normal	l	48	38	38	14
Eager, Deletion, Normal	lr	48	38	38	14
Eager, Deletion, Shortest	l	48	38	38	14
Eager, Deletion, Shortest	lr	48	38	38	14
Eager, Reduction, Normal	l	48	38	38	14
Eager, Reduction, Normal	lr	48	38	38	14
Eager, Reduction, Shortest	l	48	38	38	14
Eager, Reduction, Shortest	lr	48	38	38	14
Hybrid, Reduction, Shortest	l	48	38	45	14
Hybrid, Reduction, Shortest	lr	48	38	45	14
Hybrid, Reduction, Normal	l	48	38	45	14
Hybrid, Reduction, Normal	lr	48	38	45	14
Lazy, Deletion, Normal	l	48	38	80	14
Lazy, Deletion, Normal	lr	48	38	80	14
Lazy, Deletion, Shortest	l	48	38	80	14
Lazy, Deletion, Shortest	lr	48	38	80	14
Lazy, Reduction, Shortest	l	48	38	80	14
Lazy, Reduction, Shortest	lr	48	38	80	14
Lazy, Reduction, Normal	l	48	38	80	14
Lazy, Reduction, Normal	lr	48	38	80	14
Eager, Deletion, Normal	li	51	41	41	14
Eager, Deletion, Normal	lv	51	41	41	14
Eager, Deletion, Shortest	i	51	41	41	14
Eager, Deletion, Shortest	li	51	41	41	14
Eager, Deletion, Shortest	lv	51	41	41	14
Eager, Deletion, Shortest	v	51	41	41	14
Eager, Reduction, Normal	li	51	41	41	14
Eager, Reduction, Normal	lv	51	41	41	14
Eager, Reduction, Shortest	i	51	41	41	14
Eager, Reduction, Shortest	li	51	41	41	14
Eager, Reduction, Shortest	lv	51	41	41	14
Eager, Reduction, Shortest	v	51	41	41	14

Table E.1: Counts for Problem A4 Instance (Part Two).

Configuration	Order	Reductions Total	Reductions Zero	Cardinality Triples	Cardinality Basis
Hybrid, Reduction, Shortest	i	51	41	47	14
Hybrid, Reduction, Shortest	li	51	41	47	14
Hybrid, Reduction, Shortest	lv	51	41	47	14
Hybrid, Reduction, Shortest	v	51	41	47	14
Hybrid, Reduction, Normal	li	51	41	47	14
Hybrid, Reduction, Normal	lv	51	41	47	14
Lazy, Deletion, Normal	li	51	41	80	14
Lazy, Deletion, Normal	lv	51	41	80	14
Lazy, Deletion, Shortest	i	51	41	80	14
Lazy, Deletion, Shortest	li	51	41	80	14
Lazy, Deletion, Shortest	lv	51	41	80	14
Lazy, Deletion, Shortest	v	51	41	80	14
Lazy, Reduction, Shortest	i	51	41	80	14
Lazy, Reduction, Shortest	li	51	41	80	14
Lazy, Reduction, Shortest	lv	51	41	80	14
Lazy, Reduction, Shortest	v	51	41	80	14
Lazy, Reduction, Normal	li	51	41	80	14
Lazy, Reduction, Normal	lv	51	41	80	14
Eager, Deletion, Normal	lri	60	49	49	15
Eager, Deletion, Normal	lrv	60	49	49	15
Eager, Deletion, Shortest	lri	60	49	49	15
Eager, Deletion, Shortest	lrv	60	49	49	15
Eager, Reduction, Normal	lri	60	49	49	15
Eager, Reduction, Normal	lrv	60	49	49	15
Eager, Reduction, Shortest	lri	60	49	49	15
Eager, Reduction, Shortest	lrv	60	49	49	15
Hybrid, Reduction, Shortest	lri	60	49	60	15
Hybrid, Reduction, Shortest	lrv	60	49	60	15
Hybrid, Reduction, Normal	lri	60	49	60	15
Hybrid, Reduction, Normal	lrv	60	49	60	15
Lazy, Deletion, Normal	lri	60	49	88	15
Lazy, Deletion, Normal	lrv	60	49	88	15

Table E.1: Counts for Problem A4 Instance (Part Three).

Configuration	Order	Reductions Total	Reductions Zero	Cardinality Triples	Cardinality Basis
Lazy, Deletion, Shortest	lri	60	49	88	15
Lazy, Deletion, Shortest	lrv	60	49	88	15
Lazy, Reduction, Shortest	lri	60	49	88	15
Lazy, Reduction, Shortest	lrv	60	49	88	15
Lazy, Reduction, Normal	lri	60	49	88	15
Lazy, Reduction, Normal	lrv	60	49	88	15
None, Deletion, Normal	l	90	80	80	14
None, Deletion, Normal	li	90	80	80	14
None, Deletion, Normal	lr	90	80	80	14
None, Deletion, Normal	lv	90	80	80	14
None, Reduction, Normal	l	90	80	80	14
None, Reduction, Normal	li	90	80	80	14
None, Reduction, Normal	lr	90	80	80	14
None, Reduction, Normal	lv	90	80	80	14
None, Deletion, Normal	lri	99	88	88	15
None, Deletion, Normal	lrv	99	88	88	15
None, Reduction, Normal	lri	99	88	88	15
None, Reduction, Normal	lrv	99	88	88	15
Eager, Deletion, Normal	v	440	140	112	29
Eager, Deletion, Normal	i	446	156	100	31
Eager, Reduction, Normal	v	500	220	114	71
Hybrid, Reduction, Normal	v	503	220	119	72
Eager, Reduction, Normal	i	503	237	115	76
Lazy, Reduction, Normal	v	504	220	332	71
Hybrid, Reduction, Normal	i	507	236	129	76
Lazy, Reduction, Normal	i	507	237	400	76
Lazy, Deletion, Normal	v	556	215	358	29
Lazy, Deletion, Normal	i	562	252	387	31
None, Deletion, Normal	i	694	15	75	21
None, Reduction, Normal	v	784	494	219	45
None, Reduction, Normal	i	824	451	203	45
None, Deletion, Normal	v	874	371	85	22

Table E.2: Counts for Problem A5 (Part One).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Eager, Deletion, Shortest	lri	35	30	30	11
Eager, Reduction, Shortest	lri	35	30	30	11
Hybrid, Reduction, Shortest	lri	35	30	34	11
Lazy, Deletion, Shortest	lri	35	30	44	11
Lazy, Reduction, Shortest	lri	35	30	44	11
Eager, Deletion, Shortest	l	40	33	33	12
Eager, Deletion, Shortest	lr	40	33	33	12
Eager, Deletion, Shortest	lrv	40	33	33	12
Eager, Deletion, Normal	l	40	33	33	12
Eager, Deletion, Normal	lr	40	33	33	12
Eager, Deletion, Normal	lri	40	33	33	12
Eager, Deletion, Normal	lrv	40	33	33	12
Eager, Reduction, Shortest	l	40	33	33	12
Eager, Reduction, Shortest	lr	40	33	33	12
Eager, Reduction, Shortest	lrv	40	33	33	12
Eager, Reduction, Normal	l	40	33	33	12
Eager, Reduction, Normal	lr	40	33	33	12
Eager, Reduction, Normal	lri	40	33	33	12
Eager, Reduction, Normal	lrv	40	33	33	12
Hybrid, Reduction, Shortest	l	40	33	39	12
Hybrid, Reduction, Shortest	lr	40	33	39	12
Hybrid, Reduction, Shortest	lrv	40	33	39	12
Hybrid, Reduction, Normal	l	40	33	39	12
Hybrid, Reduction, Normal	lr	40	33	39	12
Hybrid, Reduction, Normal	lri	40	33	39	12
Hybrid, Reduction, Normal	lrv	40	33	39	12
Lazy, Deletion, Normal	l	40	33	52	12
Lazy, Deletion, Normal	lr	40	33	52	12
Lazy, Deletion, Normal	lri	40	33	52	12
Lazy, Deletion, Normal	lrv	40	33	52	12
Lazy, Deletion, Shortest	l	40	33	52	12
Lazy, Deletion, Shortest	lr	40	33	52	12

Table E.2: Counts for Problem A5 (Part Two).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Lazy, Deletion, Shortest	lrv	40	33	52	12
Lazy, Reduction, Shortest	l	40	33	52	12
Lazy, Reduction, Shortest	lr	40	33	52	12
Lazy, Reduction, Shortest	lrv	40	33	52	12
Lazy, Reduction, Normal	l	40	33	52	12
Lazy, Reduction, Normal	lr	40	33	52	12
Lazy, Reduction, Normal	lri	40	33	52	12
Lazy, Reduction, Normal	lrv	40	33	52	12
Eager, Deletion, Shortest	i	42	34	34	13
Eager, Deletion, Shortest	li	42	34	34	13
Eager, Deletion, Shortest	lv	42	34	34	13
Eager, Deletion, Shortest	v	42	34	34	13
Eager, Deletion, Normal	li	42	34	34	13
Eager, Deletion, Normal	lv	42	34	34	13
Eager, Reduction, Shortest	i	42	34	34	13
Eager, Reduction, Shortest	li	42	34	34	13
Eager, Reduction, Shortest	lv	42	34	34	13
Eager, Reduction, Shortest	v	42	34	34	13
Eager, Reduction, Normal	li	42	34	34	13
Eager, Reduction, Normal	lv	42	34	34	13
Hybrid, Reduction, Shortest	i	42	34	41	13
Hybrid, Reduction, Shortest	li	42	34	41	13
Hybrid, Reduction, Shortest	lv	42	34	41	13
Hybrid, Reduction, Shortest	v	42	34	41	13
Hybrid, Reduction, Normal	li	42	34	41	13
Hybrid, Reduction, Normal	lv	42	34	41	13
Lazy, Deletion, Normal	li	42	34	67	13
Lazy, Deletion, Normal	lv	42	34	67	13
Lazy, Deletion, Shortest	i	42	34	67	13
Lazy, Deletion, Shortest	li	42	34	67	13
Lazy, Deletion, Shortest	lv	42	34	67	13
Lazy, Deletion, Shortest	v	42	34	67	13

Table E.2: Counts for Problem A5 (Part Three).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Lazy, Reduction, Shortest	i	42	34	67	13
Lazy, Reduction, Shortest	li	42	34	67	13
Lazy, Reduction, Shortest	lv	42	34	67	13
Lazy, Reduction, Shortest	v	42	34	67	13
Lazy, Reduction, Normal	li	42	34	67	13
Lazy, Reduction, Normal	lv	42	34	67	13
None, Deletion, Normal	l	59	52	52	12
None, Deletion, Normal	lr	59	52	52	12
None, Deletion, Normal	lri	59	52	52	12
None, Deletion, Normal	lrv	59	52	52	12
None, Reduction, Normal	l	59	52	52	12
None, Reduction, Normal	lr	59	52	52	12
None, Reduction, Normal	lri	59	52	52	12
None, Reduction, Normal	lrv	59	52	52	12
None, Deletion, Normal	li	75	67	67	13
None, Deletion, Normal	lv	75	67	67	13
None, Reduction, Normal	li	75	67	67	13
None, Reduction, Normal	lv	75	67	67	13
Eager, Deletion, Normal	v	201	89	64	21
Lazy, Reduction, Normal	v	201	114	190	46
Eager, Reduction, Normal	v	204	116	66	47
Eager, Reduction, Normal	i	204	116	67	47
Hybrid, Reduction, Normal	v	204	116	84	47
Hybrid, Reduction, Normal	i	204	116	85	47
Eager, Deletion, Normal	i	212	102	64	21
Lazy, Reduction, Normal	i	214	118	213	47
Lazy, Deletion, Normal	v	228	112	233	21
Lazy, Deletion, Normal	i	272	135	232	19
None, Deletion, Normal	v	336	137	99	13
None, Reduction, Normal	i	351	252	110	34
None, Reduction, Normal	v	356	238	109	33
None, Deletion, Normal	i	465	238	95	15

Table E.3: Counts for Problem A6 (Part One).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Eager, Deletion, Shortest	l	26	23	23	9
Eager, Deletion, Shortest	lr	26	23	23	9
Eager, Deletion, Normal	l	26	23	23	9
Eager, Deletion, Normal	lr	26	23	23	9
Eager, Reduction, Shortest	l	26	23	23	9
Eager, Reduction, Shortest	lr	26	23	23	9
Eager, Reduction, Normal	l	26	23	23	9
Eager, Reduction, Normal	lr	26	23	23	9
Hybrid, Reduction, Shortest	l	26	23	23	9
Hybrid, Reduction, Shortest	lr	26	23	23	9
Hybrid, Reduction, Normal	l	26	23	23	9
Hybrid, Reduction, Normal	lr	26	23	23	9
Lazy, Deletion, Normal	l	26	23	27	9
Lazy, Deletion, Normal	lr	26	23	27	9
Lazy, Deletion, Shortest	l	26	23	27	9
Lazy, Deletion, Shortest	lr	26	23	27	9
Lazy, Reduction, Shortest	l	26	23	27	9
Lazy, Reduction, Shortest	lr	26	23	27	9
Lazy, Reduction, Normal	l	26	23	27	9
Lazy, Reduction, Normal	lr	26	23	27	9
None, Deletion, Normal	l	30	27	27	9
None, Deletion, Normal	lr	30	27	27	9
None, Reduction, Normal	l	30	27	27	9
None, Reduction, Normal	lr	30	27	27	9
Eager, Deletion, Shortest	i	35	30	30	11
Eager, Deletion, Shortest	li	35	30	30	11
Eager, Deletion, Shortest	lri	35	30	30	11
Eager, Deletion, Shortest	lrv	35	30	30	11
Eager, Deletion, Shortest	lv	35	30	30	11
Eager, Deletion, Shortest	v	35	30	30	11
Eager, Deletion, Normal	li	35	30	30	11
Eager, Deletion, Normal	lri	35	30	30	11

Table E.3: Counts for Problem A6 (Part Two).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Eager, Deletion, Normal	lrv	35	30	30	11
Eager, Deletion, Normal	lv	35	30	30	11
Eager, Reduction, Shortest	i	35	30	30	11
Eager, Reduction, Shortest	li	35	30	30	11
Eager, Reduction, Shortest	lri	35	30	30	11
Eager, Reduction, Shortest	lrv	35	30	30	11
Eager, Reduction, Shortest	lv	35	30	30	11
Eager, Reduction, Shortest	v	35	30	30	11
Eager, Reduction, Normal	li	35	30	30	11
Eager, Reduction, Normal	lri	35	30	30	11
Eager, Reduction, Normal	lrv	35	30	30	11
Eager, Reduction, Normal	lv	35	30	30	11
Hybrid, Reduction, Shortest	i	35	30	34	11
Hybrid, Reduction, Shortest	li	35	30	34	11
Hybrid, Reduction, Shortest	lri	35	30	34	11
Hybrid, Reduction, Shortest	lrv	35	30	34	11
Hybrid, Reduction, Shortest	lv	35	30	34	11
Hybrid, Reduction, Shortest	v	35	30	34	11
Hybrid, Reduction, Normal	li	35	30	34	11
Hybrid, Reduction, Normal	lri	35	30	34	11
Hybrid, Reduction, Normal	lrv	35	30	34	11
Hybrid, Reduction, Normal	lv	35	30	34	11
Lazy, Deletion, Normal	li	35	30	44	11
Lazy, Deletion, Normal	lri	35	30	44	11
Lazy, Deletion, Normal	lrv	35	30	44	11
Lazy, Deletion, Normal	lv	35	30	44	11
Lazy, Deletion, Shortest	i	35	30	44	11
Lazy, Deletion, Shortest	li	35	30	44	11
Lazy, Deletion, Shortest	lri	35	30	44	11
Lazy, Deletion, Shortest	lrv	35	30	44	11
Lazy, Deletion, Shortest	lv	35	30	44	11
Lazy, Deletion, Shortest	v	35	30	44	11

Table E.3: Counts for Problem A6 (Part Three).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Lazy, Reduction, Shortest	i	35	30	44	11
Lazy, Reduction, Shortest	li	35	30	44	11
Lazy, Reduction, Shortest	lri	35	30	44	11
Lazy, Reduction, Shortest	lrv	35	30	44	11
Lazy, Reduction, Shortest	lv	35	30	44	11
Lazy, Reduction, Shortest	v	35	30	44	11
Lazy, Reduction, Normal	li	35	30	44	11
Lazy, Reduction, Normal	lri	35	30	44	11
Lazy, Reduction, Normal	lrv	35	30	44	11
Lazy, Reduction, Normal	lv	35	30	44	11
None, Deletion, Normal	li	49	44	44	11
None, Deletion, Normal	lri	49	44	44	11
None, Deletion, Normal	lrv	49	44	44	11
None, Deletion, Normal	lv	49	44	44	11
None, Reduction, Normal	li	49	44	44	11
None, Reduction, Normal	lri	49	44	44	11
None, Reduction, Normal	lrv	49	44	44	11
None, Reduction, Normal	lv	49	44	44	11
Eager, Reduction, Normal	i	79	54	39	20
Eager, Reduction, Normal	v	79	54	39	20
Hybrid, Reduction, Normal	i	79	54	39	20
Hybrid, Reduction, Normal	v	79	54	39	20
Lazy, Reduction, Normal	i	79	54	50	20
Lazy, Reduction, Normal	v	79	54	50	20
Eager, Deletion, Normal	v	80	51	39	11
Lazy, Deletion, Normal	v	81	52	77	11
Eager, Deletion, Normal	i	82	53	39	11
Lazy, Deletion, Normal	i	83	54	69	11
None, Reduction, Normal	i	109	90	48	17
None, Reduction, Normal	v	109	90	48	17
None, Deletion, Normal	i	145	121	67	11
None, Deletion, Normal	v	153	127	59	11

Table E.4: Counts for Problem A7 (Part One).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Eager, Deletion, Shortest	i	26	24	24	9
Eager, Deletion, Shortest	l	26	24	24	9
Eager, Deletion, Shortest	li	26	24	24	9
Eager, Deletion, Shortest	lr	26	24	24	9
Eager, Deletion, Shortest	lri	26	24	24	9
Eager, Deletion, Shortest	lrv	26	24	24	9
Eager, Deletion, Shortest	lv	26	24	24	9
Eager, Deletion, Shortest	v	26	24	24	9
Eager, Deletion, Normal	l	26	24	24	9
Eager, Deletion, Normal	li	26	24	24	9
Eager, Deletion, Normal	lr	26	24	24	9
Eager, Deletion, Normal	lri	26	24	24	9
Eager, Deletion, Normal	lrv	26	24	24	9
Eager, Deletion, Normal	lv	26	24	24	9
Eager, Reduction, Shortest	i	26	24	24	9
Eager, Reduction, Shortest	l	26	24	24	9
Eager, Reduction, Shortest	li	26	24	24	9
Eager, Reduction, Shortest	lr	26	24	24	9
Eager, Reduction, Shortest	lri	26	24	24	9
Eager, Reduction, Shortest	lrv	26	24	24	9
Eager, Reduction, Shortest	lv	26	24	24	9
Eager, Reduction, Shortest	v	26	24	24	9
Eager, Reduction, Normal	l	26	24	24	9
Eager, Reduction, Normal	li	26	24	24	9
Eager, Reduction, Normal	lr	26	24	24	9
Eager, Reduction, Normal	lri	26	24	24	9
Eager, Reduction, Normal	lrv	26	24	24	9
Eager, Reduction, Normal	lv	26	24	24	9
Hybrid, Reduction, Shortest	i	26	24	24	9
Hybrid, Reduction, Shortest	l	26	24	24	9
Hybrid, Reduction, Shortest	li	26	24	24	9
Hybrid, Reduction, Shortest	lr	26	24	24	9

Table E.4: Counts for Problem A7 (Part Two).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Hybrid, Reduction, Shortest	lri	26	24	24	9
Hybrid, Reduction, Shortest	lrv	26	24	24	9
Hybrid, Reduction, Shortest	lv	26	24	24	9
Hybrid, Reduction, Shortest	v	26	24	24	9
Hybrid, Reduction, Normal	l	26	24	24	9
Hybrid, Reduction, Normal	li	26	24	24	9
Hybrid, Reduction, Normal	lr	26	24	24	9
Hybrid, Reduction, Normal	lri	26	24	24	9
Hybrid, Reduction, Normal	lrv	26	24	24	9
Hybrid, Reduction, Normal	lv	26	24	24	9
Lazy, Deletion, Normal	l	26	24	27	9
Lazy, Deletion, Normal	li	26	24	27	9
Lazy, Deletion, Normal	lr	26	24	27	9
Lazy, Deletion, Normal	lri	26	24	27	9
Lazy, Deletion, Normal	lrv	26	24	27	9
Lazy, Deletion, Normal	lv	26	24	27	9
Lazy, Deletion, Shortest	i	26	24	27	9
Lazy, Deletion, Shortest	l	26	24	27	9
Lazy, Deletion, Shortest	li	26	24	27	9
Lazy, Deletion, Shortest	lr	26	24	27	9
Lazy, Deletion, Shortest	lri	26	24	27	9
Lazy, Deletion, Shortest	lrv	26	24	27	9
Lazy, Deletion, Shortest	lv	26	24	27	9
Lazy, Deletion, Shortest	v	26	24	27	9
Lazy, Reduction, Shortest	i	26	24	27	9
Lazy, Reduction, Shortest	l	26	24	27	9
Lazy, Reduction, Shortest	li	26	24	27	9
Lazy, Reduction, Shortest	lr	26	24	27	9
Lazy, Reduction, Shortest	lri	26	24	27	9
Lazy, Reduction, Shortest	lrv	26	24	27	9
Lazy, Reduction, Shortest	lv	26	24	27	9
Lazy, Reduction, Shortest	v	26	24	27	9

Table E.4: Counts for Problem A7 (Part Three).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Lazy, Reduction, Normal	l	26	24	27	9
Lazy, Reduction, Normal	li	26	24	27	9
Lazy, Reduction, Normal	lr	26	24	27	9
Lazy, Reduction, Normal	lri	26	24	27	9
Lazy, Reduction, Normal	lrv	26	24	27	9
Lazy, Reduction, Normal	lv	26	24	27	9
Eager, Deletion, Normal	i	28	25	24	9
Eager, Deletion, Normal	v	28	25	24	9
Eager, Reduction, Normal	i	28	25	24	9
Eager, Reduction, Normal	v	28	25	24	9
Hybrid, Reduction, Normal	i	28	25	24	9
Hybrid, Reduction, Normal	v	28	25	24	9
Lazy, Reduction, Normal	i	28	25	28	9
Lazy, Reduction, Normal	v	28	25	28	9
Lazy, Deletion, Normal	i	28	25	31	9
Lazy, Deletion, Normal	v	28	25	31	9
None, Deletion, Normal	l	29	27	27	9
None, Deletion, Normal	li	29	27	27	9
None, Deletion, Normal	lr	29	27	27	9
None, Deletion, Normal	lri	29	27	27	9
None, Deletion, Normal	lrv	29	27	27	9
None, Deletion, Normal	lv	29	27	27	9
None, Reduction, Normal	l	29	27	27	9
None, Reduction, Normal	li	29	27	27	9
None, Reduction, Normal	lr	29	27	27	9
None, Reduction, Normal	lri	29	27	27	9
None, Reduction, Normal	lrv	29	27	27	9
None, Reduction, Normal	lv	29	27	27	9
None, Reduction, Normal	i	35	32	28	9
None, Reduction, Normal	v	35	32	28	9
None, Deletion, Normal	i	42	39	31	9
None, Deletion, Normal	v	42	39	31	9

Table E.5: Counts for Problem A8 (Part One).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Eager, Deletion, Shortest	i	26	25	25	9
Eager, Deletion, Shortest	l	26	25	25	9
Eager, Deletion, Shortest	li	26	25	25	9
Eager, Deletion, Shortest	lr	26	25	25	9
Eager, Deletion, Shortest	lri	26	25	25	9
Eager, Deletion, Shortest	lrv	26	25	25	9
Eager, Deletion, Shortest	lv	26	25	25	9
Eager, Deletion, Shortest	v	26	25	25	9
Eager, Deletion, Normal	i	26	25	25	9
Eager, Deletion, Normal	l	26	25	25	9
Eager, Deletion, Normal	li	26	25	25	9
Eager, Deletion, Normal	lr	26	25	25	9
Eager, Deletion, Normal	lri	26	25	25	9
Eager, Deletion, Normal	lrv	26	25	25	9
Eager, Deletion, Normal	lv	26	25	25	9
Eager, Deletion, Normal	v	26	25	25	9
Eager, Reduction, Shortest	i	26	25	25	9
Eager, Reduction, Shortest	l	26	25	25	9
Eager, Reduction, Shortest	li	26	25	25	9
Eager, Reduction, Shortest	lr	26	25	25	9
Eager, Reduction, Shortest	lri	26	25	25	9
Eager, Reduction, Shortest	lrv	26	25	25	9
Eager, Reduction, Shortest	lv	26	25	25	9
Eager, Reduction, Shortest	v	26	25	25	9
Eager, Reduction, Normal	i	26	25	25	9
Eager, Reduction, Normal	l	26	25	25	9
Eager, Reduction, Normal	li	26	25	25	9
Eager, Reduction, Normal	lr	26	25	25	9
Eager, Reduction, Normal	lri	26	25	25	9
Eager, Reduction, Normal	lrv	26	25	25	9
Eager, Reduction, Normal	lv	26	25	25	9
Eager, Reduction, Normal	v	26	25	25	9

Table E.5: Counts for Problem A8 (Part Two).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Hybrid, Reduction, Shortest	i	26	25	25	9
Hybrid, Reduction, Shortest	l	26	25	25	9
Hybrid, Reduction, Shortest	li	26	25	25	9
Hybrid, Reduction, Shortest	lr	26	25	25	9
Hybrid, Reduction, Shortest	lri	26	25	25	9
Hybrid, Reduction, Shortest	lrv	26	25	25	9
Hybrid, Reduction, Shortest	lv	26	25	25	9
Hybrid, Reduction, Shortest	v	26	25	25	9
Hybrid, Reduction, Normal	i	26	25	25	9
Hybrid, Reduction, Normal	l	26	25	25	9
Hybrid, Reduction, Normal	li	26	25	25	9
Hybrid, Reduction, Normal	lr	26	25	25	9
Hybrid, Reduction, Normal	lri	26	25	25	9
Hybrid, Reduction, Normal	lrv	26	25	25	9
Hybrid, Reduction, Normal	lv	26	25	25	9
Hybrid, Reduction, Normal	v	26	25	25	9
Lazy, Deletion, Normal	i	26	25	27	9
Lazy, Deletion, Normal	l	26	25	27	9
Lazy, Deletion, Normal	li	26	25	27	9
Lazy, Deletion, Normal	lr	26	25	27	9
Lazy, Deletion, Normal	lri	26	25	27	9
Lazy, Deletion, Normal	lrv	26	25	27	9
Lazy, Deletion, Normal	lv	26	25	27	9
Lazy, Deletion, Normal	v	26	25	27	9
Lazy, Deletion, Shortest	i	26	25	27	9
Lazy, Deletion, Shortest	l	26	25	27	9
Lazy, Deletion, Shortest	li	26	25	27	9
Lazy, Deletion, Shortest	lr	26	25	27	9
Lazy, Deletion, Shortest	lri	26	25	27	9
Lazy, Deletion, Shortest	lrv	26	25	27	9
Lazy, Deletion, Shortest	lv	26	25	27	9
Lazy, Deletion, Shortest	v	26	25	27	9

Table E.5: Counts for Problem A8 (Part Three).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Lazy, Reduction, Shortest	i	26	25	27	9
Lazy, Reduction, Shortest	l	26	25	27	9
Lazy, Reduction, Shortest	li	26	25	27	9
Lazy, Reduction, Shortest	lr	26	25	27	9
Lazy, Reduction, Shortest	lri	26	25	27	9
Lazy, Reduction, Shortest	lrv	26	25	27	9
Lazy, Reduction, Shortest	lv	26	25	27	9
Lazy, Reduction, Shortest	v	26	25	27	9
Lazy, Reduction, Normal	i	26	25	27	9
Lazy, Reduction, Normal	l	26	25	27	9
Lazy, Reduction, Normal	li	26	25	27	9
Lazy, Reduction, Normal	lr	26	25	27	9
Lazy, Reduction, Normal	lri	26	25	27	9
Lazy, Reduction, Normal	lrv	26	25	27	9
Lazy, Reduction, Normal	lv	26	25	27	9
Lazy, Reduction, Normal	v	26	25	27	9
None, Deletion, Normal	i	28	27	27	9
None, Deletion, Normal	l	28	27	27	9
None, Deletion, Normal	li	28	27	27	9
None, Deletion, Normal	lr	28	27	27	9
None, Deletion, Normal	lri	28	27	27	9
None, Deletion, Normal	lrv	28	27	27	9
None, Deletion, Normal	lv	28	27	27	9
None, Deletion, Normal	v	28	27	27	9
None, Reduction, Normal	i	28	27	27	9
None, Reduction, Normal	l	28	27	27	9
None, Reduction, Normal	li	28	27	27	9
None, Reduction, Normal	lr	28	27	27	9
None, Reduction, Normal	lri	28	27	27	9
None, Reduction, Normal	lrv	28	27	27	9
None, Reduction, Normal	lv	28	27	27	9
None, Reduction, Normal	v	28	27	27	9

Table E.6: Counts for Problem BT7 (Part One).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Eager, Deletion, Shortest	l	10	5	5	12
Eager, Deletion, Normal	l	10	5	5	12
Eager, Reduction, Shortest	l	10	5	5	12
Eager, Reduction, Normal	l	10	5	5	12
Hybrid, Reduction, Shortest	l	10	5	5	12
Hybrid, Reduction, Normal	l	10	5	5	12
Lazy, Deletion, Normal	l	10	5	5	12
Lazy, Deletion, Shortest	l	10	5	5	12
Lazy, Reduction, Shortest	l	10	5	5	12
Lazy, Reduction, Normal	l	10	5	5	12
None, Deletion, Normal	l	10	5	5	12
None, Reduction, Normal	l	10	5	5	12
None, Reduction, Normal	l	10	5	5	12
Eager, Reduction, Normal	v	31	12	12	17
Hybrid, Reduction, Normal	v	31	12	16	17
Lazy, Reduction, Normal	v	31	12	26	17
Eager, Deletion, Shortest	lv	31	14	16	17
Eager, Deletion, Shortest	v	31	14	16	17
Eager, Reduction, Shortest	lv	31	14	16	17
Eager, Reduction, Shortest	v	31	14	16	17
Eager, Deletion, Normal	lv	31	14	17	17
Eager, Reduction, Normal	lv	31	14	17	17
Hybrid, Reduction, Shortest	lv	31	14	19	17
Hybrid, Reduction, Shortest	v	31	14	19	17
Hybrid, Reduction, Normal	lv	31	14	20	17
Lazy, Deletion, Shortest	lv	31	14	29	17
Lazy, Deletion, Shortest	v	31	14	29	17
Lazy, Reduction, Shortest	lv	31	14	29	17
Lazy, Reduction, Shortest	v	31	14	29	17
Lazy, Deletion, Normal	lv	31	14	30	17
Lazy, Reduction, Normal	lv	31	14	30	17
Eager, Deletion, Normal	v	32	13	13	17

Table E.6: Counts for Problem BT7 (Part Two).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Lazy, Deletion, Normal	v	32	13	35	17
Eager, Deletion, Normal	lri	37	17	16	19
Eager, Reduction, Normal	lri	37	17	16	19
Eager, Deletion, Shortest	lri	37	17	17	19
Eager, Reduction, Shortest	lri	37	17	17	19
Hybrid, Reduction, Shortest	lri	37	17	20	19
Hybrid, Reduction, Normal	lri	37	17	22	19
Lazy, Deletion, Shortest	lri	37	17	38	19
Lazy, Reduction, Shortest	lri	37	17	38	19
Lazy, Deletion, Normal	lri	37	17	40	19
Lazy, Reduction, Normal	lri	37	17	40	19
Eager, Deletion, Shortest	lr	52	25	18	24
Eager, Reduction, Shortest	lr	52	25	18	24
Hybrid, Reduction, Shortest	lr	52	25	18	24
Eager, Deletion, Normal	lr	52	25	19	24
Eager, Reduction, Normal	lr	52	25	19	24
Hybrid, Reduction, Normal	lr	52	25	19	24
Lazy, Deletion, Shortest	lr	52	25	28	24
Lazy, Reduction, Shortest	lr	52	25	28	24
Lazy, Deletion, Normal	lr	52	25	30	24
Lazy, Reduction, Normal	lr	52	25	30	24
Eager, Deletion, Normal	lrv	55	25	19	26
Eager, Reduction, Normal	lrv	55	25	19	26
Eager, Deletion, Shortest	lrv	55	25	20	26
Eager, Deletion, Normal	li	55	25	20	26
Eager, Reduction, Shortest	lrv	55	25	20	26
Eager, Reduction, Normal	li	55	25	20	26
Eager, Deletion, Shortest	i	55	25	22	26
Eager, Deletion, Shortest	li	55	25	22	26
Eager, Reduction, Shortest	i	55	25	22	26
Eager, Reduction, Shortest	li	55	25	22	26
Hybrid, Reduction, Normal	lrv	55	25	22	26

Table E.6: Counts for Problem BT7 (Part Three).

		Reductions		Cardinality	
Configuration	Order	Total	Zero	Triples	Basis
Hybrid, Reduction, Shortest	lrv	55	25	24	26
Hybrid, Reduction, Normal	li	55	25	25	26
Hybrid, Reduction, Shortest	i	55	25	26	26
Hybrid, Reduction, Shortest	li	55	25	26	26
Lazy, Deletion, Shortest	i	55	25	44	26
Lazy, Deletion, Shortest	li	55	25	44	26
Lazy, Deletion, Shortest	lrv	55	25	44	26
Lazy, Reduction, Shortest	i	55	25	44	26
Lazy, Reduction, Shortest	li	55	25	44	26
Lazy, Reduction, Shortest	lrv	55	25	44	26
Lazy, Deletion, Normal	lrv	55	25	45	26
Lazy, Reduction, Normal	lrv	55	25	45	26
Lazy, Deletion, Normal	li	55	25	47	26
Lazy, Reduction, Normal	li	55	25	47	26
Eager, Deletion, Normal	i	67	27	13	26
Eager, Reduction, Normal	i	67	27	13	28
Hybrid, Reduction, Normal	i	67	27	16	28
Lazy, Reduction, Normal	i	67	27	28	28
Lazy, Deletion, Normal	i	67	27	29	26
None, Deletion, Normal	lr	140	81	54	36
None, Reduction, Normal	lr	140	81	54	36
None, Reduction, Normal	v	154	91	61	27
None, Deletion, Normal	lv	154	97	84	27
None, Reduction, Normal	lv	154	97	84	27
None, Deletion, Normal	li	235	112	116	42
None, Reduction, Normal	li	235	112	116	42
None, Deletion, Normal	lrv	235	128	112	42
None, Reduction, Normal	lrv	235	128	112	42
None, Deletion, Normal	lri	260	146	171	35
None, Reduction, Normal	lri	260	146	171	35
None, Reduction, Normal	i	267	122	90	46
None, Deletion, Normal	i	279	127	100	42
None, Deletion, Normal	v	379	227	146	40

Table E.7: Counts for Problem BT31 (Part One).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
None, Deletion, Normal	lr	14	6	5	37
None, Reduction, Normal	lr	14	6	5	37
Eager, Deletion, Shortest	lr	17	8	4	39
Eager, Reduction, Shortest	lr	17	8	4	39
Hybrid, Reduction, Shortest	lr	17	8	4	39
Lazy, Deletion, Shortest	lr	17	8	4	39
Lazy, Reduction, Shortest	lr	17	8	4	39
Eager, Deletion, Normal	lr	17	8	5	39
Eager, Reduction, Normal	lr	17	8	5	39
Hybrid, Reduction, Normal	lr	17	8	5	39
Lazy, Deletion, Normal	lr	17	8	5	39
Lazy, Reduction, Normal	lr	17	8	5	39
Eager, Deletion, Normal	v	31	12	9	41
Eager, Reduction, Normal	v	31	12	9	41
Hybrid, Reduction, Normal	v	31	12	13	41
Lazy, Deletion, Normal	v	31	12	22	41
Lazy, Reduction, Normal	v	31	12	22	41
Eager, Deletion, Shortest	lv	31	14	15	41
Eager, Deletion, Shortest	v	31	14	15	41
Eager, Deletion, Normal	lv	31	14	15	41
Eager, Reduction, Shortest	lv	31	14	15	41
Eager, Reduction, Shortest	v	31	14	15	41
Eager, Reduction, Normal	lv	31	14	15	41
Hybrid, Reduction, Shortest	lv	31	14	18	41
Hybrid, Reduction, Shortest	v	31	14	18	41
Hybrid, Reduction, Normal	lv	31	14	18	41
Lazy, Deletion, Shortest	lv	31	14	28	41
Lazy, Deletion, Shortest	v	31	14	28	41
Lazy, Reduction, Shortest	lv	31	14	28	41
Lazy, Reduction, Shortest	v	31	14	28	41
Lazy, Deletion, Normal	lv	31	14	29	41
Lazy, Reduction, Normal	lv	31	14	29	41

Table E.7: Counts for Problem BT31 (Part Two).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Eager, Deletion, Normal	lri	37	17	14	43
Eager, Reduction, Normal	lri	37	17	14	43
Eager, Deletion, Shortest	lri	37	17	16	43
Eager, Reduction, Shortest	lri	37	17	16	43
Hybrid, Reduction, Normal	lri	37	17	18	43
Hybrid, Reduction, Shortest	lri	37	17	19	43
Lazy, Deletion, Shortest	lri	37	17	36	43
Lazy, Reduction, Shortest	lri	37	17	36	43
Lazy, Deletion, Normal	lri	37	17	38	43
Lazy, Reduction, Normal	lri	37	17	38	43
None, Deletion, Normal	lri	38	15	26	40
None, Reduction, Normal	lri	38	15	26	40
None, Deletion, Normal	v	41	15	19	40
None, Reduction, Normal	v	41	15	19	40
None, Deletion, Normal	lv	41	16	28	40
None, Reduction, Normal	lv	41	16	28	40
None, Deletion, Normal	l	196	88	102	83
None, Reduction, Normal	l	196	88	102	83
Eager, Deletion, Shortest	l	208	100	78	99
Eager, Deletion, Normal	l	208	100	78	99
Eager, Reduction, Shortest	l	208	100	78	99
Eager, Reduction, Normal	l	208	100	78	99
Hybrid, Reduction, Shortest	l	208	100	79	99
Hybrid, Reduction, Normal	l	208	100	79	99
Lazy, Deletion, Normal	l	208	100	119	99
Lazy, Deletion, Shortest	l	208	100	119	99
Lazy, Reduction, Shortest	l	208	100	119	99
Lazy, Reduction, Normal	l	208	100	119	99
None, Deletion, Normal	lrv	212	68	116	83
None, Reduction, Normal	lrv	212	68	116	83
None, Deletion, Normal	i	212	76	24	83
None, Reduction, Normal	i	212	76	24	83

Table E.7: Counts for Problem BT31 (Part Two).

Configuration	Order	Reductions Total	Reductions Zero	Cardinality Triples	Cardinality Basis
None, Deletion, Normal	li	212	76	115	83
None, Reduction, Normal	li	212	76	115	83
Eager, Deletion, Normal	i	224	108	23	107
Eager, Reduction, Normal	i	224	108	23	107
Hybrid, Reduction, Normal	i	224	108	23	107
Lazy, Deletion, Normal	i	224	108	36	107
Lazy, Reduction, Normal	i	224	108	36	107
Eager, Deletion, Normal	lrv	224	108	76	107
Eager, Reduction, Normal	lrv	224	108	76	107
Eager, Deletion, Normal	li	224	108	79	107
Eager, Reduction, Normal	li	224	108	79	107
Eager, Deletion, Shortest	i	224	108	80	107
Eager, Deletion, Shortest	li	224	108	80	107
Eager, Reduction, Shortest	i	224	108	80	107
Eager, Reduction, Shortest	li	224	108	80	107
Eager, Deletion, Shortest	lrv	224	108	88	107
Eager, Reduction, Shortest	lrv	224	108	88	107
Hybrid, Reduction, Normal	lrv	224	108	89	107
Hybrid, Reduction, Normal	li	224	108	92	107
Hybrid, Reduction, Shortest	i	224	108	100	107
Hybrid, Reduction, Shortest	li	224	108	100	107
Hybrid, Reduction, Shortest	lrv	224	108	104	107
Lazy, Deletion, Shortest	i	224	108	180	107
Lazy, Deletion, Shortest	li	224	108	180	107
Lazy, Reduction, Shortest	i	224	108	180	107
Lazy, Reduction, Shortest	li	224	108	180	107
Lazy, Deletion, Normal	lrv	224	108	181	107
Lazy, Reduction, Normal	lrv	224	108	181	107
Lazy, Deletion, Shortest	lrv	224	108	182	107
Lazy, Reduction, Shortest	lrv	224	108	182	107
Lazy, Deletion, Normal	li	224	108	184	107
Lazy, Reduction, Normal	li	224	108	184	107

Table E.8: Counts for Problem M39 (Part One).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Eager, Deletion, Shortest	i	0	0	0	0
Eager, Deletion, Shortest	l	0	0	0	0
Eager, Deletion, Shortest	li	0	0	0	0
Eager, Deletion, Shortest	lrv	0	0	0	0
Eager, Deletion, Normal	i	0	0	0	0
Eager, Deletion, Normal	l	0	0	0	0
Eager, Deletion, Normal	li	0	0	0	0
Eager, Deletion, Normal	lrv	0	0	0	0
Eager, Reduction, Shortest	i	0	0	0	0
Eager, Reduction, Shortest	l	0	0	0	0
Eager, Reduction, Shortest	li	0	0	0	0
Eager, Reduction, Shortest	lrv	0	0	0	0
Eager, Reduction, Normal	i	0	0	0	0
Eager, Reduction, Normal	l	0	0	0	0
Eager, Reduction, Normal	li	0	0	0	0
Eager, Reduction, Normal	lrv	0	0	0	0
Hybrid, Reduction, Shortest	i	0	0	0	0
Hybrid, Reduction, Shortest	l	0	0	0	0
Hybrid, Reduction, Shortest	li	0	0	0	0
Hybrid, Reduction, Shortest	lrv	0	0	0	0
Hybrid, Reduction, Normal	i	0	0	0	0
Hybrid, Reduction, Normal	l	0	0	0	0
Hybrid, Reduction, Normal	li	0	0	0	0
Hybrid, Reduction, Normal	lrv	0	0	0	0
Lazy, Deletion, Normal	i	0	0	0	0
Lazy, Deletion, Normal	l	0	0	0	0
Lazy, Deletion, Normal	li	0	0	0	0
Lazy, Deletion, Normal	lrv	0	0	0	0
Lazy, Deletion, Shortest	i	0	0	0	0
Lazy, Deletion, Shortest	l	0	0	0	0
Lazy, Deletion, Shortest	li	0	0	0	0
Lazy, Deletion, Shortest	lrv	0	0	0	0

Table E.8: Counts for Problem M39 (Part Two).

Configuration	Order	Reductions Total	Reductions Zero	Cardinality Triples	Cardinality Basis
Lazy, Reduction, Shortest	i	0	0	0	0
Lazy, Reduction, Shortest	l	0	0	0	0
Lazy, Reduction, Shortest	li	0	0	0	0
Lazy, Reduction, Shortest	lrv	0	0	0	0
Lazy, Reduction, Normal	i	0	0	0	0
Lazy, Reduction, Normal	l	0	0	0	0
Lazy, Reduction, Normal	li	0	0	0	0
Lazy, Reduction, Normal	lrv	0	0	0	0
None, Deletion, Normal	i	0	0	0	0
None, Deletion, Normal	l	0	0	0	0
None, Deletion, Normal	li	0	0	0	0
None, Deletion, Normal	lrv	0	0	0	0
None, Reduction, Normal	i	0	0	0	0
None, Reduction, Normal	l	0	0	0	0
None, Reduction, Normal	li	0	0	0	0
None, Reduction, Normal	lrv	0	0	0	0
None, Deletion, Normal	lr	14	6	5	37
None, Reduction, Normal	lr	14	6	5	37
Eager, Deletion, Shortest	lr	17	8	4	39
Eager, Reduction, Shortest	lr	17	8	4	39
Hybrid, Reduction, Shortest	lr	17	8	4	39
Lazy, Deletion, Shortest	lr	17	8	4	39
Lazy, Reduction, Shortest	lr	17	8	4	39
Eager, Deletion, Normal	lr	17	8	5	39
Eager, Reduction, Normal	lr	17	8	5	39
Hybrid, Reduction, Normal	lr	17	8	5	39
Lazy, Deletion, Normal	lr	17	8	5	39
Lazy, Reduction, Normal	lr	17	8	5	39
Eager, Deletion, Normal	v	31	12	9	41
Eager, Reduction, Normal	v	31	12	9	41
Hybrid, Reduction, Normal	v	31	12	13	41
Lazy, Deletion, Normal	v	31	12	22	41

Table E.8: Counts for Problem M39 (Part Three).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Lazy, Reduction, Normal	v	31	12	22	41
Eager, Deletion, Shortest	lv	31	14	15	41
Eager, Deletion, Shortest	v	31	14	15	41
Eager, Deletion, Normal	lv	31	14	15	41
Eager, Reduction, Shortest	lv	31	14	15	41
Eager, Reduction, Shortest	v	31	14	15	41
Eager, Reduction, Normal	lv	31	14	15	41
Hybrid, Reduction, Shortest	lv	31	14	18	41
Hybrid, Reduction, Shortest	v	31	14	18	41
Hybrid, Reduction, Normal	lv	31	14	18	41
Lazy, Deletion, Shortest	lv	31	14	28	41
Lazy, Deletion, Shortest	v	31	14	28	41
Lazy, Reduction, Shortest	lv	31	14	28	41
Lazy, Reduction, Shortest	v	31	14	28	41
Lazy, Deletion, Normal	lv	31	14	29	41
Lazy, Reduction, Normal	lv	31	14	29	41
Eager, Deletion, Normal	lri	37	17	14	43
Eager, Reduction, Normal	lri	37	17	14	43
Eager, Deletion, Shortest	lri	37	17	16	43
Eager, Reduction, Shortest	lri	37	17	16	43
Hybrid, Reduction, Normal	lri	37	17	18	43
Hybrid, Reduction, Shortest	lri	37	17	19	43
Lazy, Deletion, Shortest	lri	37	17	36	43
Lazy, Reduction, Shortest	lri	37	17	36	43
Lazy, Deletion, Normal	lri	37	17	38	43
Lazy, Reduction, Normal	lri	37	17	38	43
None, Deletion, Normal	lri	38	15	26	40
None, Reduction, Normal	lri	38	15	26	40
None, Deletion, Normal	v	41	15	19	40
None, Reduction, Normal	v	41	15	19	40
None, Deletion, Normal	lv	41	16	28	40
None, Reduction, Normal	lv	41	16	28	40

Table E.9: Counts for Problem P5 (Part One).

Configuration	Order	Reductions		Cardinality	
		Total	Zero	Triples	Basis
Eager, Reduction, Normal	l	69	43	22	41
Eager, Reduction, Normal	lr	69	43	22	41
Eager, Reduction, Shortest	l	69	43	22	41
Eager, Deletion, Normal	l	69	43	22	41
Eager, Deletion, Normal	lr	69	43	22	41
Eager, Deletion, Shortest	l	69	43	22	41
Eager, Reduction, Shortest	lr	69	43	25	41
Eager, Deletion, Shortest	lr	69	43	25	41
Hybrid, Reduction, Shortest	lr	69	43	25	41
Hybrid, Reduction, Normal	l	69	43	27	41
Hybrid, Reduction, Normal	lr	69	43	27	41
Hybrid, Reduction, Shortest	l	69	43	27	41
Lazy, Reduction, Shortest	lr	69	43	274	41
Lazy, Deletion, Shortest	lr	69	43	274	41
Lazy, Reduction, Normal	l	69	43	301	41
Lazy, Reduction, Normal	lr	69	43	301	41
Lazy, Reduction, Shortest	l	69	43	301	41
Lazy, Deletion, Normal	l	69	43	301	41
Lazy, Deletion, Normal	lr	69	43	301	41
Lazy, Deletion, Shortest	l	69	43	301	41
Eager, Reduction, Shortest	lv	122	75	36	62
Eager, Reduction, Shortest	v	122	75	36	62
Eager, Deletion, Shortest	lv	122	75	36	62
Eager, Deletion, Shortest	v	122	75	36	62
Eager, Reduction, Normal	lv	122	75	39	62
Eager, Deletion, Normal	lv	122	75	39	62
Eager, Reduction, Normal	li	122	75	40	62
Eager, Deletion, Normal	li	122	75	40	62
Eager, Reduction, Shortest	i	122	75	43	62
Eager, Reduction, Shortest	li	122	75	43	62
Eager, Deletion, Shortest	i	122	75	43	62
Eager, Deletion, Shortest	li	122	75	43	62

Table E.9: Counts for Problem P5 (Part Two).

Configuration	Order	Reductions Total	Reductions Zero	Cardinality Triples	Cardinality Basis
Hybrid, Reduction, Shortest	lv	122	75	53	62
Hybrid, Reduction, Shortest	v	122	75	53	62
Hybrid, Reduction, Normal	li	122	75	55	62
Hybrid, Reduction, Normal	lv	122	75	56	62
Hybrid, Reduction, Shortest	i	122	75	64	62
Hybrid, Reduction, Shortest	li	122	75	64	62
Lazy, Reduction, Shortest	i	122	75	556	62
Lazy, Reduction, Shortest	li	122	75	556	62
Lazy, Deletion, Shortest	i	122	75	556	62
Lazy, Deletion, Shortest	li	122	75	556	62
Lazy, Reduction, Shortest	lv	122	75	629	62
Lazy, Reduction, Shortest	v	122	75	629	62
Lazy, Deletion, Shortest	lv	122	75	629	62
Lazy, Deletion, Shortest	v	122	75	629	62
Lazy, Reduction, Normal	li	122	75	648	62
Lazy, Reduction, Normal	lv	122	75	648	62
Lazy, Deletion, Normal	li	122	75	648	62
Lazy, Deletion, Normal	lv	122	75	648	62
Eager, Reduction, Shortest	lri	130	79	36	66
Eager, Deletion, Shortest	lri	130	79	36	66
Eager, Reduction, Normal	lri	130	79	39	66
Eager, Reduction, Normal	lrv	130	79	39	66
Eager, Deletion, Normal	lri	130	79	39	66
Eager, Deletion, Normal	lrv	130	79	39	66
Eager, Reduction, Shortest	lrv	130	79	43	66
Eager, Deletion, Shortest	lrv	130	79	43	66
Hybrid, Reduction, Normal	lrv	130	79	53	66
Hybrid, Reduction, Normal	lri	130	79	55	66
Hybrid, Reduction, Shortest	lri	130	79	55	66
Hybrid, Reduction, Shortest	lrv	130	79	63	66
Lazy, Reduction, Shortest	lrv	130	79	611	66
Lazy, Deletion, Shortest	lrv	130	79	611	66

Table E.9: Counts for Problem P5 (Part Three).

Configuration	Order	Reductions Total	Reductions Zero	Cardinality Triples	Cardinality Basis
Lazy, Reduction, Shortest	lri	130	79	697	66
Lazy, Deletion, Shortest	lri	130	79	697	66
Lazy, Reduction, Normal	lri	130	79	712	66
Lazy, Reduction, Normal	lrv	130	79	712	66
Lazy, Deletion, Normal	lri	130	79	712	66
Lazy, Deletion, Normal	lrv	130	79	712	66
Eager, Deletion, Normal	i	136	85	32	62
Eager, Reduction, Normal	i	139	88	33	66
Hybrid, Reduction, Normal	i	139	88	33	66
Lazy, Reduction, Normal	i	139	88	163	66
Lazy, Deletion, Normal	i	141	90	188	62
Eager, Deletion, Normal	v	142	90	32	62
Eager, Reduction, Normal	v	148	96	34	67
Hybrid, Reduction, Normal	v	148	96	34	67
Lazy, Reduction, Normal	v	148	96	168	67
Lazy, Deletion, Normal	v	153	101	204	62
None, Reduction, Normal	l	468	442	301	41
None, Reduction, Normal	lr	468	442	301	41
None, Deletion, Normal	l	468	442	301	41
None, Deletion, Normal	lr	468	442	301	41
None, Reduction, Normal	li	984	937	648	62
None, Reduction, Normal	lv	984	937	648	62
None, Deletion, Normal	li	984	937	648	62
None, Deletion, Normal	lv	984	937	648	62
None, Reduction, Normal	i	1069	1018	163	66
None, Reduction, Normal	v	1070	1018	168	67
None, Reduction, Normal	lri	1097	1046	712	66
None, Reduction, Normal	lrv	1097	1046	712	66
None, Deletion, Normal	lri	1097	1046	712	66
None, Deletion, Normal	lrv	1097	1046	712	66
None, Deletion, Normal	i	1541	1478	303	61
None, Deletion, Normal	v	1618	1553	313	62

E.1.2 Times

Table E.10: Times for Problem A4 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Hybrid, Reduction, Normal	l	0.0410	3.5322	0.0973	3.6704
Hybrid, Reduction, Shortest	l	0.0410	3.5351	0.0991	3.6752
Hybrid, Reduction, Shortest	lr	0.0435	3.5701	0.1006	3.7142
Hybrid, Reduction, Normal	lr	0.0485	3.5717	0.1011	3.7213
Lazy, Reduction, Normal	l	0.0304	3.7042	0.1019	3.8365
Lazy, Reduction, Shortest	lr	0.0338	3.7082	0.1016	3.8437
Lazy, Reduction, Shortest	l	0.0286	3.7143	0.1028	3.8457
Lazy, Deletion, Normal	l	0.0331	3.7286	0.1027	3.8645
Lazy, Deletion, Shortest	lr	0.0355	3.7253	0.1042	3.8650
Lazy, Deletion, Shortest	l	0.0293	3.7400	0.1025	3.8718
Lazy, Reduction, Normal	lr	0.0336	3.7597	0.0987	3.8920
Lazy, Deletion, Normal	lr	0.0355	3.7682	0.1036	3.9073
Hybrid, Reduction, Shortest	v	0.0465	4.1214	0.1178	4.2857
Lazy, Reduction, Shortest	v	0.0358	4.1770	0.1173	4.3301
Lazy, Reduction, Shortest	i	0.0420	4.2084	0.1185	4.3689
Eager, Deletion, Normal	l	0.0525	4.2263	0.1052	4.3840
Hybrid, Reduction, Normal	li	0.0570	4.2124	0.1210	4.3904
Eager, Deletion, Shortest	l	0.0491	4.2370	0.1045	4.3907
Eager, Deletion, Shortest	lr	0.0585	4.2369	0.1037	4.3991
Lazy, Reduction, Normal	li	0.0462	4.2365	0.1177	4.4004
Eager, Deletion, Normal	lr	0.0588	4.2645	0.1102	4.4335
Eager, Reduction, Normal	l	0.0496	4.2801	0.1038	4.4336
Lazy, Deletion, Shortest	v	0.0339	4.2815	0.1214	4.4368
Eager, Reduction, Shortest	l	0.0490	4.3015	0.1039	4.4544
Lazy, Reduction, Shortest	lv	0.0369	4.3127	0.1212	4.4708

Table E.10: Times for Problem A4 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Hybrid, Reduction, Shortest	i	0.0531	4.2969	0.1216	4.4716
Hybrid, Reduction, Shortest	lv	0.0475	4.3039	0.1205	4.4719
Eager, Reduction, Normal	lr	0.0540	4.3169	0.1065	4.4775
Hybrid, Reduction, Normal	lv	0.0506	4.3335	0.1237	4.5078
Lazy, Reduction, Normal	lv	0.0384	4.3549	0.1197	4.5130
Lazy, Deletion, Shortest	i	0.0437	4.3648	0.1241	4.5325
Lazy, Reduction, Shortest	li	0.0438	4.3861	0.1214	4.5513
Hybrid, Reduction, Shortest	li	0.0537	4.3754	0.1234	4.5524
Lazy, Deletion, Normal	li	0.0490	4.4086	0.1232	4.5808
Eager, Reduction, Shortest	lr	0.0565	4.4221	0.1067	4.5853
Lazy, Deletion, Shortest	lv	0.0364	4.4348	0.1252	4.5963
Lazy, Deletion, Normal	lv	0.0361	4.4994	0.1247	4.6603
Lazy, Deletion, Shortest	li	0.0471	4.5374	0.1287	4.7131
Eager, Deletion, Shortest	v	0.0586	4.6234	0.1199	4.8019
Eager, Deletion, Shortest	i	0.0646	4.6677	0.1210	4.8533
Eager, Deletion, Normal	li	0.0650	4.7672	0.1192	4.9514
Eager, Reduction, Shortest	v	0.0584	4.7731	0.1214	4.9529
Eager, Deletion, Shortest	lv	0.0599	4.7659	0.1417	4.9675
Eager, Reduction, Shortest	i	0.0649	4.8395	0.1231	5.0275
Eager, Deletion, Shortest	li	0.0656	4.8927	0.1244	5.0827
Eager, Deletion, Normal	lv	0.0608	4.9266	0.1226	5.1100
Eager, Reduction, Shortest	lv	0.0597	4.9259	0.1248	5.1105
Eager, Reduction, Normal	li	0.0655	4.9266	0.1224	5.1145
Eager, Reduction, Shortest	li	0.0672	4.9649	0.1267	5.1588
Eager, Reduction, Normal	lv	0.0608	5.0607	0.1258	5.2472
Lazy, Reduction, Normal	lrv	0.0457	5.5288	0.1476	5.7221

Table E.10: Times for Problem A4 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Hybrid, Reduction, Normal	lrv	0.0592	5.6169	0.1481	5.8242
Hybrid, Reduction, Shortest	lrv	0.0567	5.6828	0.1538	5.8933
Lazy, Reduction, Shortest	lrv	0.0515	5.7039	0.1485	5.9039
Lazy, Deletion, Normal	lrv	0.0476	5.7680	0.1518	5.9674
Lazy, Deletion, Shortest	lrv	0.0482	5.8333	0.1579	6.0393
Lazy, Reduction, Shortest	lri	0.0424	5.9514	0.1503	6.1441
Hybrid, Reduction, Shortest	lri	0.0547	6.0071	0.1513	6.2131
Hybrid, Reduction, Normal	lri	0.0563	6.0499	0.1486	6.2548
Lazy, Reduction, Normal	lri	0.0446	6.1114	0.1489	6.3049
Lazy, Deletion, Normal	lri	0.0434	6.2436	0.1556	6.4425
Lazy, Deletion, Shortest	lri	0.0413	6.3268	0.1540	6.5221
Eager, Deletion, Shortest	lrv	0.0687	6.3920	0.1521	6.6128
Eager, Deletion, Normal	lrv	0.0692	6.5925	0.1522	6.8140
Eager, Deletion, Shortest	lri	0.0656	6.6386	0.1472	6.8513
Eager, Reduction, Shortest	lrv	0.0699	6.6315	0.1575	6.8590
Eager, Reduction, Normal	lrv	0.0699	6.8467	0.1546	7.0711
Eager, Reduction, Shortest	lri	0.0657	6.8629	0.1508	7.0794
Eager, Deletion, Normal	lri	0.0661	6.9330	0.1478	7.1469
Eager, Reduction, Normal	lri	0.0621	7.2829	0.1532	7.4982
Eager, Deletion, Normal	i	0.0621	822.7675	0.1274	822.9570
Lazy, Deletion, Normal	i	0.0459	875.6408	0.1239	875.8106
Hybrid, Reduction, Normal	i	0.0549	914.6888	0.1164	914.8601
Lazy, Reduction, Normal	i	0.0448	962.6742	0.1133	962.8323
Eager, Reduction, Normal	i	0.0659	964.2230	0.1128	964.4017
Eager, Deletion, Normal	v	0.0595	965.1622	0.1232	965.3449
Eager, Reduction, Normal	v	0.0589	976.5486	0.1103	976.7178
Lazy, Deletion, Normal	v	0.0340	1013.7048	0.1166	1013.8554
Lazy, Reduction, Normal	v	0.0364	1056.0256	0.1085	1056.1705
Hybrid, Reduction, Normal	v	0.0489	1068.0977	0.1216	1068.2682

Table E.11: Times for Problem A5 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Reduction, Shortest	lri	0.0600	1.3778	0.0623	1.5002
Hybrid, Reduction, Shortest	lri	0.0795	1.4040	0.0648	1.5483
Lazy, Deletion, Shortest	lri	0.0617	1.5135	0.0660	1.6412
Hybrid, Reduction, Shortest	l	0.0431	1.5627	0.0657	1.6715
Lazy, Reduction, Shortest	l	0.0283	1.5821	0.0644	1.6748
Hybrid, Reduction, Normal	l	0.0431	1.5686	0.0645	1.6762
Lazy, Reduction, Normal	l	0.0289	1.5922	0.0660	1.6870
Lazy, Deletion, Shortest	l	0.0289	1.6000	0.0673	1.6963
Lazy, Deletion, Normal	l	0.0294	1.6144	0.0663	1.7102
Lazy, Reduction, Shortest	lr	0.0462	1.6050	0.0673	1.7185
Hybrid, Reduction, Normal	lr	0.0586	1.5955	0.0661	1.7202
Lazy, Reduction, Normal	lr	0.0446	1.6092	0.0666	1.7203
Hybrid, Reduction, Shortest	lr	0.0594	1.6193	0.0656	1.7443
Lazy, Deletion, Shortest	lr	0.0472	1.6294	0.0690	1.7456
Lazy, Deletion, Normal	lr	0.0490	1.6432	0.0684	1.7605
Eager, Deletion, Shortest	lri	0.1202	1.5867	0.0632	1.7701
Eager, Reduction, Shortest	lri	0.1205	1.6191	0.0647	1.8043
Lazy, Reduction, Normal	lrv	0.0613	1.9142	0.0766	2.0521
Lazy, Reduction, Shortest	lrv	0.0636	1.9467	0.0824	2.0927
Hybrid, Reduction, Normal	lrv	0.0770	1.9587	0.0789	2.1146
Eager, Deletion, Shortest	l	0.0629	2.0152	0.0679	2.1460
Eager, Deletion, Normal	l	0.0627	2.0144	0.0711	2.1482
Eager, Reduction, Normal	l	0.0617	2.0279	0.0687	2.1582
Eager, Reduction, Shortest	l	0.0616	2.0322	0.0685	2.1623
Lazy, Deletion, Shortest	lrv	0.0652	2.0238	0.0855	2.1745
Hybrid, Reduction, Shortest	lrv	0.0775	2.0345	0.0813	2.1932
Eager, Deletion, Shortest	lr	0.0875	2.0384	0.0701	2.1960
Lazy, Reduction, Normal	lri	0.0466	2.0884	0.0764	2.2114

Table E.11: Times for Problem A5 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Reduction, Shortest	lr	0.0868	2.0603	0.0717	2.2189
Eager, Reduction, Normal	lr	0.0822	2.0698	0.0694	2.2214
Hybrid, Reduction, Normal	lri	0.0613	2.1014	0.0777	2.2404
Lazy, Deletion, Normal	lrv	0.0644	2.1042	0.0846	2.2532
Lazy, Deletion, Normal	lri	0.0485	2.1569	0.0817	2.2872
Hybrid, Reduction, Shortest	v	0.0540	2.1654	0.0865	2.3059
Eager, Deletion, Normal	lr	0.0823	2.1653	0.0704	2.3180
Lazy, Reduction, Shortest	v	0.0370	2.2036	0.0857	2.3264
Lazy, Reduction, Shortest	i	0.0625	2.1842	0.0901	2.3368
Lazy, Deletion, Shortest	v	0.0386	2.2422	0.0876	2.3683
Hybrid, Reduction, Shortest	lv	0.0569	2.2434	0.0871	2.3873
Lazy, Reduction, Normal	li	0.0683	2.2324	0.0880	2.3888
Lazy, Reduction, Shortest	lv	0.0393	2.2630	0.0879	2.3901
Hybrid, Reduction, Normal	li	0.0835	2.2265	0.0871	2.3970
Lazy, Deletion, Normal	li	0.0701	2.2364	0.0930	2.3995
Hybrid, Reduction, Normal	lv	0.0586	2.2681	0.0877	2.4143
Lazy, Reduction, Shortest	li	0.0639	2.2598	0.0919	2.4156
Hybrid, Reduction, Shortest	i	0.0768	2.2565	0.0893	2.4226
Hybrid, Reduction, Shortest	li	0.0795	2.2562	0.0953	2.4309
Lazy, Reduction, Normal	lv	0.0410	2.3071	0.0867	2.4348
Lazy, Deletion, Shortest	lv	0.0412	2.3273	0.0908	2.4593
Lazy, Deletion, Shortest	i	0.0655	2.2662	0.1284	2.4602
Eager, Deletion, Normal	lrv	0.1008	2.3020	0.0792	2.4819
Lazy, Deletion, Normal	lv	0.0439	2.3664	0.0922	2.5025
Lazy, Deletion, Shortest	li	0.0679	2.3494	0.0996	2.5170
Eager, Deletion, Shortest	lrv	0.1024	2.3372	0.0828	2.5224

Table E.11: Times for Problem A5 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Reduction, Normal	lrv	0.1008	2.3762	0.0806	2.5576
Eager, Deletion, Normal	lri	0.0843	2.4393	0.0785	2.6021
Eager, Reduction, Shortest	lrv	0.1033	2.4212	0.0861	2.6106
Eager, Reduction, Normal	lri	0.0856	2.5037	0.0807	2.6700
Eager, Deletion, Shortest	v	0.0820	2.5704	0.0871	2.7395
Eager, Deletion, Shortest	i	0.1026	2.5719	0.0931	2.7677
Eager, Deletion, Normal	li	0.0999	2.5924	0.0905	2.7827
Eager, Deletion, Shortest	li	0.1057	2.6370	0.0935	2.8363
Eager, Reduction, Normal	li	0.1026	2.6546	0.0926	2.8498
Eager, Reduction, Shortest	lv	0.0820	2.6981	0.0912	2.8712
Eager, Deletion, Normal	lv	0.0844	2.7137	0.0895	2.8876
Eager, Deletion, Shortest	lv	0.0880	2.7205	0.0892	2.8977
Eager, Reduction, Shortest	li	0.1076	2.7053	0.0961	2.9090
Eager, Reduction, Shortest	v	0.0804	2.7663	0.0901	2.9368
Eager, Reduction, Normal	lv	0.0858	2.7849	0.0906	2.9614
Eager, Reduction, Shortest	i	0.1061	2.7633	0.0986	2.9680
Hybrid, Reduction, Normal	i	0.0811	42.8628	0.0898	43.0337
Lazy, Reduction, Normal	v	0.0388	44.0198	0.0838	44.1424
Eager, Deletion, Normal	i	0.0981	44.7913	0.0925	44.9819
Hybrid, Reduction, Normal	v	0.0566	46.0504	0.0835	46.1905
Eager, Reduction, Normal	i	0.0994	47.5199	0.0890	47.7083
Lazy, Reduction, Normal	i	0.0641	48.6772	0.0836	48.8249
Eager, Deletion, Normal	v	0.0820	49.6946	0.0925	49.8690
Lazy, Deletion, Normal	v	0.0420	50.6951	0.0925	50.8296
Eager, Reduction, Normal	v	0.0829	54.6354	0.0917	54.8100
Lazy, Deletion, Normal	i	0.0669	76.3032	0.0939	76.4641

Table E.12: Times for Problem A6 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Reduction, Shortest	l	0.0327	0.9034	0.0419	0.9779
Lazy, Reduction, Normal	l	0.0322	0.9066	0.0416	0.9804
Lazy, Deletion, Normal	l	0.0335	0.9061	0.0435	0.9832
Lazy, Deletion, Shortest	l	0.0347	0.9100	0.0430	0.9876
Hybrid, Reduction, Normal	l	0.0497	0.8991	0.0399	0.9888
Hybrid, Reduction, Shortest	l	0.0497	0.9018	0.0406	0.9921
Lazy, Reduction, Shortest	lr	0.0614	0.9007	0.0422	1.0042
Lazy, Reduction, Normal	lr	0.0594	0.9128	0.0437	1.0159
Lazy, Deletion, Shortest	lr	0.0623	0.9111	0.0450	1.0184
Hybrid, Reduction, Shortest	lr	0.0759	0.9006	0.0438	1.0204
Lazy, Deletion, Normal	lr	0.0615	0.9232	0.0461	1.0307
Hybrid, Reduction, Normal	lr	0.0767	0.9123	0.0428	1.0318
Eager, Deletion, Shortest	l	0.0906	1.0170	0.0438	1.1513
Eager, Deletion, Normal	l	0.0914	1.0173	0.0443	1.1529
Eager, Reduction, Normal	l	0.0900	1.0249	0.0447	1.1596
Eager, Reduction, Shortest	l	0.0908	1.0262	0.0439	1.1609
Eager, Deletion, Normal	lr	0.1143	1.0354	0.0451	1.1949
Eager, Reduction, Normal	lr	0.1160	1.0430	0.0455	1.2045
Eager, Deletion, Shortest	lr	0.1257	1.0366	0.0449	1.2071
Eager, Reduction, Shortest	lr	0.1241	1.0457	0.0459	1.2157
Lazy, Reduction, Shortest	v	0.0462	1.3069	0.0617	1.4148
Lazy, Reduction, Shortest	i	0.0817	1.2898	0.0624	1.4339
Lazy, Deletion, Shortest	v	0.0486	1.3342	0.0616	1.4444
Hybrid, Reduction, Shortest	v	0.0730	1.3245	0.0612	1.4587
Lazy, Reduction, Normal	lv	0.0501	1.3526	0.0640	1.4668
Lazy, Reduction, Normal	li	0.0861	1.3184	0.0640	1.4685

Table E.12: Times for Problem A6 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Hybrid, Reduction, Shortest	i	0.0988	1.3148	0.0627	1.4763
Lazy, Deletion, Shortest	i	0.0835	1.3362	0.0650	1.4848
Lazy, Reduction, Shortest	li	0.0838	1.3421	0.0640	1.4898
Lazy, Deletion, Shortest	lv	0.0509	1.3764	0.0637	1.4909
Hybrid, Reduction, Shortest	lv	0.0721	1.3612	0.0616	1.4949
Lazy, Reduction, Shortest	lri	0.0595	1.3734	0.0630	1.4960
Hybrid, Reduction, Normal	li	0.1076	1.3324	0.0652	1.5052
Lazy, Reduction, Normal	lrv	0.0780	1.3680	0.0621	1.5082
Lazy, Reduction, Normal	lri	0.0608	1.3853	0.0634	1.5096
Lazy, Deletion, Normal	lv	0.0559	1.3898	0.0654	1.5111
Lazy, Deletion, Normal	li	0.0893	1.3721	0.0661	1.5275
Hybrid, Reduction, Normal	lv	0.0812	1.3835	0.0638	1.5285
Hybrid, Reduction, Shortest	li	0.1035	1.3639	0.0646	1.5321
Lazy, Deletion, Shortest	li	0.0873	1.3829	0.0679	1.5382
Hybrid, Reduction, Normal	lri	0.0806	1.3976	0.0643	1.5426
Hybrid, Reduction, Shortest	lri	0.0799	1.3983	0.0644	1.5426
Lazy, Reduction, Shortest	lrv	0.0792	1.4042	0.0661	1.5495
Lazy, Deletion, Shortest	lri	0.0633	1.4215	0.0663	1.5512
Lazy, Deletion, Normal	lri	0.0655	1.4314	0.0659	1.5628
Hybrid, Reduction, Normal	lrv	0.1011	1.4024	0.0629	1.5664
Lazy, Deletion, Normal	lrv	0.0816	1.4190	0.0659	1.5665
Lazy, Deletion, Shortest	lrv	0.0821	1.4611	0.0674	1.6107
Hybrid, Reduction, Shortest	lrv	0.0989	1.4487	0.0653	1.6129
Lazy, Reduction, Shortest	lv	0.0477	1.5595	0.0640	1.6712
Eager, Deletion, Shortest	v	0.1132	1.5448	0.0630	1.7210

Table E.12: Times for Problem A6 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Deletion, Shortest	i	0.1467	1.5339	0.0643	1.7448
Eager, Reduction, Shortest	v	0.1128	1.5872	0.0640	1.7640
Eager, Deletion, Shortest	lv	0.1154	1.5886	0.0623	1.7664
Eager, Deletion, Normal	li	0.1424	1.5739	0.0654	1.7817
Eager, Deletion, Shortest	li	0.1496	1.5762	0.0666	1.7924
Eager, Reduction, Shortest	lv	0.1155	1.6180	0.0642	1.7976
Eager, Reduction, Shortest	i	0.1493	1.5819	0.0672	1.7984
Eager, Deletion, Normal	lri	0.1166	1.6173	0.0651	1.7990
Eager, Deletion, Normal	lv	0.1168	1.6345	0.0658	1.8171
Eager, Deletion, Normal	lrv	0.1419	1.6259	0.0626	1.8304
Eager, Reduction, Normal	lri	0.1160	1.6557	0.0651	1.8368
Eager, Reduction, Normal	li	0.1433	1.6274	0.0679	1.8387
Eager, Reduction, Shortest	li	0.1518	1.6209	0.0675	1.8402
Eager, Reduction, Normal	lv	0.1164	1.6710	0.0655	1.8530
Eager, Deletion, Shortest	lrv	0.1516	1.6521	0.0665	1.8702
Eager, Reduction, Normal	lrv	0.1433	1.6712	0.0653	1.8798
Eager, Reduction, Shortest	lri	0.1206	1.7017	0.0661	1.8884
Eager, Deletion, Shortest	lri	0.1205	1.7182	0.0641	1.9028
Eager, Reduction, Shortest	lrv	0.1526	1.8261	0.0700	2.0487
Lazy, Reduction, Normal	i	0.0813	2.4901	0.0624	2.6338
Lazy, Deletion, Normal	i	0.0845	2.6476	0.0660	2.7981
Lazy, Reduction, Normal	v	0.0473	2.6879	0.0631	2.7983
Lazy, Deletion, Normal	v	0.0494	2.8089	0.0645	2.9228
Hybrid, Reduction, Normal	i	0.1006	2.7758	0.0635	2.9399
Hybrid, Reduction, Normal	v	0.0712	2.9647	0.0617	3.0976
Eager, Deletion, Normal	i	0.1377	3.2461	0.0657	3.4496
Eager, Deletion, Normal	v	0.1130	3.5952	0.0650	3.7732
Eager, Reduction, Normal	i	0.1390	3.6068	0.0660	3.8119
Eager, Reduction, Normal	v	0.1143	3.8084	0.0680	3.9906

Table E.13: Times for Problem A7 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Reduction, Shortest	l	0.0390	0.7123	0.0422	0.7936
Lazy, Reduction, Normal	l	0.0384	0.7158	0.0445	0.7987
Lazy, Deletion, Shortest	l	0.0456	0.7197	0.0415	0.8068
Lazy, Deletion, Normal	l	0.0398	0.7273	0.0417	0.8087
Hybrid, Reduction, Normal	l	0.0677	0.7084	0.0408	0.8169
Hybrid, Reduction, Shortest	l	0.0686	0.7090	0.0406	0.8183
Lazy, Reduction, Shortest	lr	0.0749	0.7277	0.0427	0.8453
Lazy, Reduction, Normal	lr	0.0734	0.7358	0.0444	0.8537
Lazy, Deletion, Shortest	lr	0.0770	0.7400	0.0434	0.8604
Hybrid, Reduction, Shortest	lr	0.0974	0.7218	0.0412	0.8604
Lazy, Deletion, Normal	lr	0.0764	0.7496	0.0440	0.8700
Hybrid, Reduction, Normal	lr	0.0980	0.7340	0.0420	0.8740
Lazy, Reduction, Shortest	v	0.0605	0.8601	0.0520	0.9727
Eager, Deletion, Normal	l	0.1324	0.8093	0.0418	0.9835
Eager, Deletion, Shortest	l	0.1329	0.8127	0.0430	0.9886
Eager, Reduction, Shortest	l	0.1312	0.8177	0.0422	0.9911
Eager, Reduction, Normal	l	0.1316	0.8169	0.0436	0.9921
Lazy, Deletion, Shortest	v	0.0611	0.8774	0.0549	0.9934
Lazy, Reduction, Normal	v	0.0606	0.8900	0.0508	1.0014
Lazy, Reduction, Shortest	lv	0.0613	0.8933	0.0515	1.0061
Hybrid, Reduction, Shortest	v	0.0847	0.8727	0.0501	1.0074
Lazy, Reduction, Normal	lv	0.0657	0.8967	0.0519	1.0144
Lazy, Deletion, Shortest	lv	0.0635	0.9129	0.0535	1.0300
Lazy, Reduction, Shortest	i	0.0988	0.8823	0.0511	1.0321
Lazy, Deletion, Normal	v	0.0634	0.9190	0.0523	1.0348
Hybrid, Reduction, Normal	v	0.0858	0.9085	0.0504	1.0447
Lazy, Reduction, Shortest	lrv	0.1018	0.8887	0.0545	1.0450
Hybrid, Reduction, Shortest	lv	0.0865	0.9087	0.0512	1.0464

Table E.13: Times for Problem A7 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Deletion, Normal	lv	0.0665	0.9259	0.0541	1.0465
Eager, Deletion, Normal	lr	0.1647	0.8379	0.0440	1.0466
Lazy, Reduction, Normal	i	0.0983	0.9005	0.0503	1.0491
Eager, Reduction, Normal	lr	0.1640	0.8469	0.0436	1.0544
Hybrid, Reduction, Shortest	i	0.1202	0.8839	0.0510	1.0550
Hybrid, Reduction, Normal	lv	0.0892	0.9145	0.0513	1.0550
Lazy, Reduction, Normal	lrv	0.0996	0.9023	0.0538	1.0557
Lazy, Deletion, Shortest	i	0.1010	0.9082	0.0537	1.0629
Lazy, Reduction, Normal	li	0.1023	0.9102	0.0522	1.0646
Eager, Deletion, Shortest	lr	0.1868	0.8385	0.0453	1.0706
Lazy, Reduction, Shortest	li	0.1002	0.9184	0.0525	1.0711
Eager, Reduction, Shortest	lr	0.1814	0.8466	0.0458	1.0738
Lazy, Reduction, Normal	lri	0.0709	0.9513	0.0529	1.0751
Lazy, Deletion, Shortest	lrv	0.1068	0.9158	0.0559	1.0785
Lazy, Deletion, Normal	lrv	0.1016	0.9282	0.0559	1.0856
Hybrid, Reduction, Shortest	lrv	0.1282	0.9096	0.0538	1.0915
Hybrid, Reduction, Shortest	li	0.1252	0.9175	0.0535	1.0962
Hybrid, Reduction, Normal	i	0.1248	0.9226	0.0515	1.0989
Lazy, Deletion, Shortest	li	0.1035	0.9411	0.0545	1.0991
Lazy, Deletion, Normal	li	0.1099	0.9381	0.0543	1.1023
Hybrid, Reduction, Normal	lrv	0.1275	0.9244	0.0531	1.1050
Hybrid, Reduction, Normal	li	0.1283	0.9261	0.0510	1.1054
Lazy, Deletion, Normal	i	0.1026	0.9477	0.0555	1.1058
Lazy, Deletion, Normal	lri	0.0740	0.9791	0.0551	1.1082
Lazy, Deletion, Shortest	lri	0.0712	0.9812	0.0566	1.1090
Hybrid, Reduction, Normal	lri	0.0945	0.9691	0.0541	1.1177

Table E.13: Times for Problem A7 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Hybrid, Reduction, Shortest	lri	0.0949	0.9696	0.0534	1.1179
Eager, Deletion, Shortest	v	0.1576	0.9371	0.0510	1.1457
Eager, Reduction, Shortest	v	0.1588	0.9640	0.0523	1.1750
Eager, Deletion, Shortest	lv	0.1601	0.9668	0.0522	1.1791
Eager, Deletion, Normal	v	0.1529	0.9862	0.0522	1.1913
Eager, Deletion, Normal	lv	0.1606	0.9919	0.0534	1.2059
Eager, Deletion, Shortest	i	0.2087	0.9472	0.0518	1.2077
Eager, Reduction, Shortest	lv	0.1605	0.9955	0.0523	1.2083
Eager, Reduction, Normal	v	0.1529	1.0024	0.0532	1.2084
Eager, Deletion, Normal	i	0.1859	0.9858	0.0519	1.2237
Eager, Reduction, Normal	lv	0.1569	1.0183	0.0540	1.2293
Eager, Deletion, Normal	lrv	0.1890	0.9875	0.0532	1.2296
Eager, Deletion, Shortest	lrv	0.2090	0.9688	0.0541	1.2319
Eager, Deletion, Shortest	lri	0.1636	1.0160	0.0544	1.2340
Eager, Deletion, Normal	li	0.1895	0.9989	0.0514	1.2398
Eager, Deletion, Normal	lri	0.1555	1.0324	0.0539	1.2419
Eager, Deletion, Shortest	li	0.2113	0.9792	0.0516	1.2420
Eager, Reduction, Shortest	i	0.2117	0.9814	0.0515	1.2446
Eager, Reduction, Shortest	lri	0.1621	1.0411	0.0553	1.2585
Eager, Reduction, Normal	lrv	0.1905	1.0142	0.0539	1.2585
Eager, Reduction, Shortest	lrv	0.2092	1.0028	0.0539	1.2659
Eager, Reduction, Normal	lri	0.1530	1.0642	0.0541	1.2712
Eager, Reduction, Shortest	li	0.2130	1.0114	0.0530	1.2774
Lazy, Reduction, Shortest	lri	0.0679	1.1589	0.0560	1.2828
Eager, Reduction, Normal	li	0.1966	1.0362	0.0553	1.2881
Eager, Reduction, Normal	i	0.1893	1.0535	0.0570	1.2997

Table E.14: Times for Problem A8 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Reduction, Normal	l	0.0509	0.5731	0.0416	0.6657
Lazy, Reduction, Shortest	l	0.0505	0.5761	0.0411	0.6678
Lazy, Deletion, Shortest	l	0.0512	0.5832	0.0421	0.6764
Lazy, Deletion, Normal	l	0.0512	0.5839	0.0422	0.6774
Hybrid, Reduction, Shortest	l	0.0836	0.5671	0.0427	0.6934
Lazy, Reduction, Normal	lr	0.0913	0.6041	0.0421	0.7375
Lazy, Deletion, Normal	lr	0.0912	0.6119	0.0434	0.7465
Lazy, Reduction, Shortest	lr	0.0967	0.6091	0.0435	0.7493
Lazy, Deletion, Shortest	lr	0.0954	0.6110	0.0451	0.7514
Hybrid, Reduction, Shortest	lr	0.1232	0.5850	0.0437	0.7519
Eager, Deletion, Shortest	l	0.1897	0.6331	0.0445	0.8673
Eager, Deletion, Normal	l	0.1904	0.6364	0.0440	0.8707
Eager, Reduction, Normal	l	0.1889	0.6388	0.0441	0.8717
Eager, Reduction, Shortest	l	0.1885	0.6430	0.0449	0.8764
Lazy, Reduction, Normal	v	0.0686	0.7739	0.0511	0.8935
Lazy, Reduction, Shortest	v	0.0681	0.7805	0.0526	0.9012
Lazy, Deletion, Shortest	v	0.0693	0.7899	0.0527	0.9119
Lazy, Deletion, Normal	v	0.0718	0.7908	0.0526	0.9152
Lazy, Reduction, Shortest	lv	0.0694	0.8000	0.0528	0.9221
Hybrid, Reduction, Shortest	v	0.0997	0.7776	0.0506	0.9279
Lazy, Reduction, Normal	lv	0.0718	0.8053	0.0521	0.9293
Eager, Deletion, Normal	lr	0.2307	0.6680	0.0457	0.9443
Lazy, Deletion, Shortest	lv	0.0711	0.8204	0.0541	0.9455
Eager, Reduction, Normal	lr	0.2299	0.6715	0.0466	0.9480
Lazy, Deletion, Normal	lv	0.0747	0.8251	0.0548	0.9546
Hybrid, Reduction, Shortest	lv	0.1010	0.8088	0.0519	0.9617
Eager, Deletion, Shortest	lr	0.2629	0.6584	0.0447	0.9661
Eager, Reduction, Shortest	lr	0.2615	0.6659	0.0455	0.9729

Table E.14: Times for Problem A8 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Reduction, Normal	i	0.1222	0.8018	0.0521	0.9762
Lazy, Reduction, Shortest	i	0.1254	0.8034	0.0534	0.9823
Lazy, Reduction, Normal	lrv	0.1209	0.8194	0.0545	0.9948
Lazy, Reduction, Shortest	lrv	0.1248	0.8179	0.0543	0.9970
Lazy, Reduction, Normal	lri	0.0824	0.8654	0.0552	1.0029
Hybrid, Reduction, Shortest	i	0.1543	0.7965	0.0525	1.0034
Lazy, Reduction, Shortest	lri	0.0818	0.8693	0.0543	1.0053
Lazy, Deletion, Shortest	i	0.1298	0.8234	0.0552	1.0084
Lazy, Deletion, Normal	i	0.1275	0.8291	0.0552	1.0118
Lazy, Reduction, Normal	li	0.1269	0.8369	0.0522	1.0160
Lazy, Reduction, Shortest	li	0.1295	0.8348	0.0519	1.0162
Lazy, Deletion, Normal	lrv	0.1261	0.8461	0.0559	1.0282
Lazy, Deletion, Shortest	lrv	0.1303	0.8446	0.0548	1.0298
Hybrid, Reduction, Shortest	lrv	0.1569	0.8246	0.0533	1.0348
Lazy, Deletion, Shortest	lri	0.0839	0.8952	0.0563	1.0354
Hybrid, Reduction, Shortest	lri	0.1133	0.8713	0.0550	1.0396
Hybrid, Reduction, Shortest	li	0.1580	0.8309	0.0521	1.0410
Lazy, Deletion, Normal	li	0.1346	0.8581	0.0541	1.0468
Eager, Deletion, Normal	v	0.2083	0.7869	0.0533	1.0485
Lazy, Deletion, Shortest	li	0.1322	0.8649	0.0541	1.0512
Eager, Reduction, Normal	v	0.2087	0.8088	0.0529	1.0704
Eager, Deletion, Shortest	v	0.2153	0.8151	0.0520	1.0824
Eager, Reduction, Shortest	v	0.2146	0.8377	0.0559	1.1082
Eager, Deletion, Shortest	lv	0.2168	0.8393	0.0539	1.1099
Eager, Deletion, Normal	i	0.2475	0.8136	0.0536	1.1148
Lazy, Deletion, Normal	lri	0.0862	0.9772	0.0569	1.1203

Table E.14: Times for Problem A8 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Deletion, Normal	lv	0.2153	0.8640	0.0541	1.1334
Eager, Reduction, Shortest	lv	0.2163	0.8648	0.0533	1.1344
Eager, Reduction, Normal	i	0.2543	0.8357	0.0537	1.1437
Eager, Reduction, Normal	lv	0.2157	0.8837	0.0551	1.1544
Eager, Deletion, Normal	lri	0.2067	0.9125	0.0539	1.1731
Eager, Deletion, Shortest	lri	0.2285	0.8963	0.0536	1.1784
Eager, Deletion, Shortest	i	0.2931	0.8340	0.0540	1.1812
Eager, Deletion, Normal	lrv	0.2549	0.8751	0.0558	1.1859
Eager, Deletion, Normal	li	0.2549	0.8837	0.0546	1.1931
Eager, Deletion, Shortest	lrv	0.2879	0.8529	0.0567	1.1975
Eager, Reduction, Shortest	lri	0.2281	0.9161	0.0548	1.1990
Eager, Reduction, Normal	lri	0.2083	0.9411	0.0540	1.2034
Eager, Deletion, Shortest	li	0.2964	0.8654	0.0545	1.2162
Eager, Reduction, Shortest	i	0.2973	0.8645	0.0550	1.2168
Eager, Reduction, Normal	lrv	0.2569	0.9052	0.0570	1.2191
Eager, Reduction, Shortest	lrv	0.2906	0.8844	0.0566	1.2316
Eager, Reduction, Shortest	li	0.3001	0.8883	0.0565	1.2449
Eager, Reduction, Normal	li	0.2577	0.9411	0.0583	1.2571

Table E.15: Times for Problem BT31 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Reduction, Shortest	lr	0.1418	0.3733	0.2621	0.7772
Lazy, Reduction, Normal	lr	0.1376	0.3838	0.2622	0.7836
Lazy, Deletion, Shortest	lr	0.1421	0.3903	0.265	0.7974
Lazy, Deletion, Normal	lr	0.142	0.3969	0.2652	0.8041
Hybrid, Reduction, Shortest	lr	0.1432	0.4384	0.2646	0.8462
Hybrid, Reduction, Normal	lr	0.1453	0.4399	0.265	0.8502
Eager, Deletion, Shortest	lr	0.2626	0.4731	0.3049	1.0405
Eager, Deletion, Normal	lr	0.2637	0.4881	0.3063	1.058
Eager, Reduction, Shortest	lr	0.2782	0.4862	0.3295	1.094
Eager, Reduction, Normal	lr	0.2765	0.5011	0.3261	1.1037
Eager, Deletion, Shortest	v	0.2722	10.5592	0.4205	11.2519
Eager, Deletion, Shortest	lv	0.2712	10.663	0.4242	11.3585
Eager, Deletion, Normal	lv	0.2831	10.7481	0.4216	11.4528
Eager, Deletion, Normal	v	0.2832	10.7953	0.4208	11.4992
Lazy, Reduction, Shortest	v	0.145	11.0627	0.3814	11.5891
Lazy, Reduction, Shortest	lv	0.1475	11.0601	0.3845	11.5921
Eager, Reduction, Shortest	v	0.2856	10.9174	0.4504	11.6533
Lazy, Reduction, Normal	lv	0.1468	11.2268	0.3853	11.7588
Eager, Reduction, Shortest	lv	0.2864	11.1093	0.4496	11.8453
Lazy, Reduction, Normal	v	0.147	11.422	0.379	11.948
Eager, Reduction, Normal	lv	0.2949	11.2005	0.4535	11.9489
Eager, Reduction, Normal	v	0.2906	11.3476	0.4486	12.0868
Hybrid, Reduction, Shortest	v	0.1533	12.5389	0.4029	13.0951
Lazy, Reduction, Normal	l	0.154	11.8096	1.1763	13.1399
Lazy, Reduction, Shortest	l	0.1566	11.9266	1.1838	13.2669
Hybrid, Reduction, Normal	lv	0.1539	12.8132	0.4067	13.3738
Hybrid, Reduction, Shortest	lv	0.154	12.839	0.4049	13.3978
Hybrid, Reduction, Normal	l	0.2035	12.133	1.1927	13.5292

Table E.15: Times for Problem BT31 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Hybrid, Reduction, Shortest	l	0.1986	12.1417	1.1982	13.5386
Lazy, Deletion, Shortest	v	0.1486	13.0957	0.4139	13.6582
Hybrid, Reduction, Normal	v	0.1565	13.1148	0.4012	13.6726
Lazy, Deletion, Shortest	lv	0.1495	13.3429	0.4154	13.9078
Lazy, Deletion, Normal	lv	0.1515	13.3689	0.4136	13.9341
Lazy, Deletion, Shortest	l	0.1563	12.8366	1.2024	14.1954
Lazy, Deletion, Normal	l	0.1623	12.8582	1.202	14.2226
Lazy, Deletion, Normal	v	0.15	13.6746	0.4127	14.2373
Lazy, Reduction, Normal	i	0.1544	21.4526	1.9464	23.5534
Hybrid, Reduction, Normal	i	0.2048	23.5898	2.019	25.8136
Lazy, Reduction, Shortest	i	0.1566	24.5842	1.9444	26.6851
Lazy, Deletion, Normal	i	0.1609	24.5954	2.0589	26.8152
Lazy, Reduction, Shortest	li	0.1582	24.7653	1.962	26.8855
Eager, Deletion, Normal	i	0.8284	23.9538	2.1489	26.9311
Lazy, Reduction, Shortest	lrv	0.1578	24.9391	1.9266	27.0235
Hybrid, Reduction, Shortest	i	0.2036	25.374	2.0166	27.5942
Hybrid, Reduction, Shortest	li	0.2044	25.4512	2.0317	27.6873
Hybrid, Reduction, Shortest	lrv	0.2044	25.7527	1.9975	27.9546
Eager, Reduction, Normal	i	0.8633	25.3023	2.2916	28.4573
Lazy, Deletion, Shortest	i	0.1644	28.1052	2.0511	30.3207
Lazy, Deletion, Shortest	li	0.1648	28.3279	2.0671	30.5599
Lazy, Deletion, Shortest	lrv	0.1645	28.4601	2.032	30.6566
Hybrid, Reduction, Normal	li	0.2024	28.763	2.0306	30.996
Hybrid, Reduction, Normal	lrv	0.2059	28.9052	1.9994	31.1105
Lazy, Reduction, Normal	lrv	0.1608	29.8247	1.9161	31.9016
Lazy, Reduction, Normal	li	0.1557	30.2217	1.9588	32.3362
Eager, Deletion, Shortest	lri	0.2738	33.9856	0.727	34.9864
Eager, Deletion, Normal	lri	0.2866	34.4148	0.8263	35.5278

Table E.15: Times for Problem BT31 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Reduction, Shortest	lri	0.2856	35.242	0.7225	36.2501
Eager, Reduction, Normal	lri	0.2989	35.6484	0.7426	36.6899
Lazy, Deletion, Normal	lrv	0.164	34.5595	2.0317	36.7552
Lazy, Deletion, Normal	li	0.1647	34.6847	2.0697	36.9191
Lazy, Reduction, Normal	lri	0.1472	36.4436	0.7332	37.324
Lazy, Reduction, Shortest	lri	0.1468	36.5219	0.7612	37.43
Hybrid, Reduction, Normal	lri	0.1552	41.688	0.7258	42.569
Hybrid, Reduction, Shortest	lri	0.1553	41.9505	0.7586	42.8645
Lazy, Deletion, Normal	lri	0.1513	43.1049	0.7165	43.9727
Lazy, Deletion, Shortest	lri	0.1508	43.7201	0.797	44.668
Eager, Deletion, Shortest	l	0.6833	52.8854	1.1773	54.746
Eager, Reduction, Shortest	l	0.7147	53.5832	1.161	55.4589
Eager, Reduction, Normal	l	0.7164	53.5831	1.1677	55.4672
Eager, Deletion, Normal	l	0.6827	53.9543	1.2678	55.9048
Eager, Deletion, Shortest	i	0.6211	67.9186	1.8804	70.4201
Eager, Deletion, Shortest	li	0.6274	68.603	1.9191	71.1495
Eager, Deletion, Shortest	lrv	0.6253	69.5065	1.8911	72.0229
Eager, Reduction, Shortest	lrv	0.6525	69.5562	1.8548	72.0636
Eager, Reduction, Shortest	i	0.6531	69.9353	2.001	72.5895
Eager, Reduction, Shortest	li	0.6523	71.4303	1.9703	74.0529
Eager, Deletion, Normal	li	0.836	95.6068	1.9381	98.3808
Eager, Deletion, Normal	lrv	0.853	95.7256	1.907	98.4855
Eager, Reduction, Normal	lrv	0.8876	95.9664	1.8983	98.7523
Eager, Reduction, Normal	li	0.8693	96.0827	1.9383	98.8904

Table E.16: Times for Problem BT7 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Deletion, Normal	l	0.0498	0.12	0.0467	0.2165
Eager, Deletion, Normal	lr	0.0468	1.2747	0.1046	1.426
Eager, Deletion, Normal	lv	0.0519	1.2435	0.1013	1.3967
Eager, Deletion, Normal	lrv	0.0486	1.7766	0.1296	1.9548
Eager, Deletion, Normal	li	0.0489	1.7753	0.1278	1.952
Eager, Deletion, Normal	lri	0.0506	1.5809	0.1161	1.7476
Eager, Deletion, Normal	v	0.0512	1.2828	0.1003	1.4344
Eager, Deletion, Normal	i	0.0486	1.1999	0.1236	1.3721
Eager, Deletion, Shortest	l	0.0488	0.1197	0.0475	0.216
Eager, Deletion, Shortest	lr	0.0486	1.3431	0.1044	1.4961
Eager, Deletion, Shortest	lv	0.0486	1.2073	0.1003	1.3562
Eager, Deletion, Shortest	lrv	0.0455	1.5908	0.1283	1.7646
Eager, Deletion, Shortest	li	0.0452	1.5501	0.122	1.7173
Eager, Deletion, Shortest	lri	0.0483	1.4991	0.1143	1.6616
Eager, Deletion, Shortest	v	0.0486	1.206	0.1006	1.3552
Eager, Deletion, Shortest	i	0.046	1.5498	0.1214	1.7172
Eager, Reduction, Shortest	l	0.0476	0.1185	0.0483	0.2144
Eager, Reduction, Shortest	lr	0.045	1.2374	0.1059	1.3883
Eager, Reduction, Shortest	lv	0.0479	1.2074	0.1034	1.3587
Eager, Reduction, Shortest	lrv	0.0456	1.4961	0.1286	1.6703
Eager, Reduction, Shortest	li	0.0448	1.541	0.1299	1.7157
Eager, Reduction, Shortest	lri	0.0474	1.4959	0.1172	1.6605
Eager, Reduction, Shortest	v	0.0499	1.201	0.1016	1.3525
Eager, Reduction, Shortest	i	0.0464	1.5349	0.1293	1.7105
Eager, Reduction, Normal	l	0.0485	0.1204	0.0481	0.217
Eager, Reduction, Normal	lr	0.0472	1.2619	0.105	1.4141
Eager, Reduction, Normal	lv	0.0503	1.2437	0.109	1.403
Eager, Reduction, Normal	lrv	0.0484	1.7611	0.1303	1.9398

Table E.16: Times for Problem BT7 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Reduction, Normal	li	0.0479	1.7782	0.1297	1.9558
Eager, Reduction, Normal	lri	0.0498	1.5892	0.1203	1.7594
Eager, Reduction, Normal	v	0.052	1.2719	0.101	1.4248
Eager, Reduction, Normal	i	0.048	1.1937	0.1255	1.3672
Hybrid, Reduction, Normal	l	0.0257	0.1133	0.038	0.177
Hybrid, Reduction, Normal	lr	0.0378	0.6481	0.0957	0.7816
Hybrid, Reduction, Normal	lv	0.0275	1.1126	0.0974	1.2375
Hybrid, Reduction, Normal	lrv	0.0398	0.9111	0.1111	1.062
Hybrid, Reduction, Normal	li	0.0409	0.9396	0.1158	1.0964
Hybrid, Reduction, Normal	lri	0.0275	1.4073	0.1206	1.5553
Hybrid, Reduction, Normal	v	0.0273	1.1987	0.0986	1.3245
Hybrid, Reduction, Normal	i	0.0403	1.0949	0.1143	1.2496
Hybrid, Reduction, Shortest	l	0.0266	0.1166	0.0391	0.1823
Hybrid, Reduction, Shortest	lr	0.0373	0.6498	0.0939	0.7809
Hybrid, Reduction, Shortest	lv	0.0275	1.0575	0.0988	1.1839
Hybrid, Reduction, Shortest	lrv	0.0369	0.8215	0.113	0.9713
Hybrid, Reduction, Shortest	li	0.0371	0.8372	0.115	0.9893
Hybrid, Reduction, Shortest	lri	0.0306	1.3284	0.1121	1.4712
Hybrid, Reduction, Shortest	v	0.0277	1.0464	0.0987	1.1728
Hybrid, Reduction, Shortest	i	0.0377	0.8368	0.1148	0.9893
Lazy, Deletion, Normal	l	0.0216	0.1013	0.0384	0.1613
Lazy, Deletion, Normal	lr	0.0268	0.6083	0.0937	0.7288
Lazy, Deletion, Normal	lv	0.0262	1.1335	0.0972	1.2569
Lazy, Deletion, Normal	lrv	0.031	0.9645	0.1171	1.1126
Lazy, Deletion, Normal	li	0.0305	1.0107	0.1157	1.1569
Lazy, Deletion, Normal	lri	0.0256	1.4159	0.1136	1.5551
Lazy, Deletion, Normal	v	0.0251	1.3566	0.0984	1.4801
Lazy, Deletion, Normal	i	0.0298	1.0627	0.1167	1.2093

Table E.16: Times for Problem BT7 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Deletion, Shortest	l	0.0208	0.0987	0.0384	0.1579
Lazy, Deletion, Shortest	lr	0.0262	0.6042	0.094	0.7244
Lazy, Deletion, Shortest	lv	0.0247	1.0615	0.101	1.1871
Lazy, Deletion, Shortest	lrv	0.0287	0.8215	0.1157	0.9658
Lazy, Deletion, Shortest	li	0.0285	0.8197	0.1165	0.9647
Lazy, Deletion, Shortest	lri	0.0252	1.3839	0.1141	1.5232
Lazy, Deletion, Shortest	v	0.0255	1.054	0.1002	1.1797
Lazy, Deletion, Shortest	i	0.0275	0.8174	0.1165	0.9614
Lazy, Reduction, Normal	l	0.0196	0.0994	0.0371	0.1561
Lazy, Reduction, Normal	lr	0.023	0.5783	0.0946	0.6958
Lazy, Reduction, Normal	lv	0.0217	1.0215	0.0903	1.1335
Lazy, Reduction, Normal	lrv	0.0265	0.8786	0.1091	1.0142
Lazy, Reduction, Normal	li	0.0261	0.9104	0.1138	1.0503
Lazy, Reduction, Normal	lri	0.0209	1.279	0.1088	1.4086
Lazy, Reduction, Normal	v	0.0214	1.1502	0.0904	1.2621
Lazy, Reduction, Normal	i	0.0251	0.9421	0.1143	1.0815
Lazy, Reduction, Shortest	l	0.0196	0.0976	0.0372	0.1544
Lazy, Reduction, Shortest	lr	0.0221	0.5731	0.0946	0.6898
Lazy, Reduction, Shortest	lv	0.0206	0.9733	0.0897	1.0836
Lazy, Reduction, Shortest	lrv	0.0245	0.7694	0.115	0.9089
Lazy, Reduction, Shortest	li	0.0232	0.7706	0.1136	0.9074
Lazy, Reduction, Shortest	lri	0.0217	1.2532	0.1075	1.3825
Lazy, Reduction, Shortest	v	0.0205	0.9573	0.089	1.0668
Lazy, Reduction, Shortest	i	0.0231	0.7658	0.1135	0.9024

Table E.17: Times for Problem M39 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Deletion, Normal	l	0.2069	0.0002	0.3075	0.5146
Eager, Deletion, Normal	lr	0.2961	0.5182	0.3696	1.184
Eager, Deletion, Normal	lv	0.3173	13.106	0.4854	13.9086
Eager, Deletion, Normal	lrv	0.217	0.0002	0.3186	0.5358
Eager, Deletion, Normal	li	0.2179	0.0002	0.3181	0.5362
Eager, Deletion, Normal	lri	0.3216	44.2162	0.9418	45.4796
Eager, Deletion, Normal	v	0.3148	12.9398	0.4928	13.7474
Eager, Deletion, Normal	i	0.2162	0.0002	0.3187	0.5352
Eager, Deletion, Shortest	l	0.2091	0.0002	0.3053	0.5146
Eager, Deletion, Shortest	lr	0.294	0.5057	0.3686	1.1684
Eager, Deletion, Shortest	lv	0.3066	12.8815	0.4844	13.6726
Eager, Deletion, Shortest	lrv	0.2164	0.0002	0.3189	0.5355
Eager, Deletion, Shortest	li	0.2177	0.0002	0.3204	0.5383
Eager, Deletion, Shortest	lri	0.307	40.2696	0.8213	41.398
Eager, Deletion, Shortest	v	0.3055	12.5886	0.4805	13.3746
Eager, Deletion, Shortest	i	0.2162	0.0002	0.3193	0.5357
Eager, Reduction, Shortest	l	0.2264	0.0002	0.3391	0.5657
Eager, Reduction, Shortest	lr	0.312	0.5202	0.3992	1.2314
Eager, Reduction, Shortest	lv	0.3241	13.2994	0.5173	14.1409
Eager, Reduction, Shortest	lrv	0.2355	0.0009	0.3536	0.5901
Eager, Reduction, Shortest	li	0.2375	0.0002	0.3517	0.5894
Eager, Reduction, Shortest	lri	0.3256	41.9637	0.8671	43.1565
Eager, Reduction, Shortest	v	0.3235	13.2809	0.5154	14.1199
Eager, Reduction, Shortest	i	0.2352	0.0002	0.35	0.5854
Eager, Reduction, Normal	l	0.2262	0.0002	0.3423	0.5688
Eager, Reduction, Normal	lr	0.3114	0.5335	0.3957	1.2406
Eager, Reduction, Normal	lv	0.3401	13.4152	0.5168	14.2721
Eager, Reduction, Normal	lrv	0.2428	0.0002	0.3486	0.5916

Table E.17: Times for Problem M39 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Reduction, Normal	li	0.2362	0.0004	0.3504	0.587
Eager, Reduction, Normal	lri	0.3431	48.3829	1.051	49.7771
Eager, Reduction, Normal	v	0.3402	13.5339	0.5137	14.3879
Eager, Reduction, Normal	i	0.2352	0.0002	0.351	0.5865
Hybrid, Reduction, Normal	l	0.1748	0.0002	0.2535	0.4285
Hybrid, Reduction, Normal	lr	0.1758	0.4718	0.3198	0.9674
Hybrid, Reduction, Normal	lv	0.1901	15.6689	0.4643	16.3232
Hybrid, Reduction, Normal	lrv	0.1869	0.0002	0.2673	0.4544
Hybrid, Reduction, Normal	li	0.1888	0.0002	0.2688	0.4578
Hybrid, Reduction, Normal	lri	0.1901	48.8861	0.876	49.9522
Hybrid, Reduction, Normal	v	0.1868	16.0138	0.4594	16.6601
Hybrid, Reduction, Normal	i	0.1866	0.0002	0.2734	0.4603
Hybrid, Reduction, Shortest	l	0.1735	0.0002	0.2552	0.429
Hybrid, Reduction, Shortest	lr	0.1763	0.4693	0.3182	0.9637
Hybrid, Reduction, Shortest	lv	0.188	15.6898	0.4655	16.3434
Hybrid, Reduction, Shortest	lrv	0.1874	0.0002	0.2678	0.4554
Hybrid, Reduction, Shortest	li	0.1879	0.0002	0.2702	0.4583
Hybrid, Reduction, Shortest	lri	0.1903	47.0328	0.8163	48.0395
Hybrid, Reduction, Shortest	v	0.1875	15.366	0.4587	16.0123
Hybrid, Reduction, Shortest	i	0.1864	0.0002	0.2686	0.4553
Lazy, Deletion, Normal	l	0.1727	0.0002	0.2568	0.4297
Lazy, Deletion, Normal	lr	0.1725	0.4325	0.3208	0.9258
Lazy, Deletion, Normal	lv	0.1828	16.2722	0.4756	16.9306
Lazy, Deletion, Normal	lrv	0.1839	0.0002	0.2774	0.4615
Lazy, Deletion, Normal	li	0.1848	0.0002	0.2766	0.4616
Lazy, Deletion, Normal	lri	0.1836	49.9161	0.886	50.9858
Lazy, Deletion, Normal	v	0.1831	16.5993	0.4656	17.2479
Lazy, Deletion, Normal	i	0.1839	0.0002	0.2746	0.4587

Table E.17: Times for Problem M39 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Deletion, Shortest	l	0.1728	0.0002	0.2562	0.4293
Lazy, Deletion, Shortest	lr	0.1725	0.428	0.3194	0.9199
Lazy, Deletion, Shortest	lv	0.1832	16.1608	0.4777	16.8216
Lazy, Deletion, Shortest	lrv	0.1831	0.0002	0.2754	0.4588
Lazy, Deletion, Shortest	li	0.1844	0.0002	0.2769	0.4615
Lazy, Deletion, Shortest	lri	0.1849	49.881	0.8921	50.9581
Lazy, Deletion, Shortest	v	0.1829	15.8043	0.4667	16.4539
Lazy, Deletion, Shortest	i	0.1851	0.0002	0.2764	0.4617
Lazy, Reduction, Normal	l	0.1688	0.0004	0.2522	0.4214
Lazy, Reduction, Normal	lr	0.1701	0.412	0.3134	0.8955
Lazy, Reduction, Normal	lv	0.1812	13.843	0.4394	14.4637
Lazy, Reduction, Normal	lrv	0.1801	0.0004	0.2658	0.4463
Lazy, Reduction, Normal	li	0.1811	0.0004	0.2655	0.447
Lazy, Reduction, Normal	lri	0.1806	42.2499	0.8165	43.2471
Lazy, Reduction, Normal	v	0.1794	13.9506	0.4369	14.5669
Lazy, Reduction, Normal	i	0.1795	0.0004	0.267	0.4469
Lazy, Reduction, Shortest	l	0.168	0.0004	0.2561	0.4246
Lazy, Reduction, Shortest	lr	0.1691	0.4087	0.3138	0.8915
Lazy, Reduction, Shortest	lv	0.1795	13.6142	0.4404	14.2341
Lazy, Reduction, Shortest	lrv	0.1781	0.0004	0.2652	0.4438
Lazy, Reduction, Shortest	li	0.1806	0.0004	0.2667	0.4478
Lazy, Reduction, Shortest	lri	0.1783	47.5508	1.0062	48.7352
Lazy, Reduction, Shortest	v	0.1776	13.3498	0.4359	13.9633
Lazy, Reduction, Shortest	i	0.1796	0.0004	0.2647	0.4447

Table E.18: Times for Problem P5 (Part One).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Deletion, Normal	l	0.2898	3.147	0.2455	3.6823
Eager, Deletion, Normal	lr	0.2854	3.1865	0.2489	3.7209
Eager, Deletion, Normal	lv	0.3011	12.0897	0.4108	12.8016
Eager, Deletion, Normal	lrv	0.3467	13.1878	0.439	13.9735
Eager, Deletion, Normal	li	0.3235	11.8829	0.4113	12.6177
Eager, Deletion, Normal	lri	0.2957	13.0373	0.4416	13.7746
Eager, Deletion, Normal	v	0.3437	10.6436	0.4084	11.3957
Eager, Deletion, Normal	i	0.369	10.5025	0.4164	11.2879
Eager, Deletion, Shortest	l	0.2907	3.1527	0.2438	3.6871
Eager, Deletion, Shortest	lr	0.294	3.0764	0.248	3.6184
Eager, Deletion, Shortest	lv	0.2903	8.8417	0.4099	9.5419
Eager, Deletion, Shortest	lrv	0.2942	11.09	0.4463	11.8305
Eager, Deletion, Shortest	li	0.2956	10.042	0.4144	10.7519
Eager, Deletion, Shortest	lri	0.2872	10.3493	0.4392	11.0758
Eager, Deletion, Shortest	v	0.2884	8.8435	0.4071	9.539
Eager, Deletion, Shortest	i	0.2947	10.0542	0.4079	10.7569
Eager, Reduction, Normal	l	0.2874	3.2103	0.2513	3.749
Eager, Reduction, Normal	lr	0.2868	3.2456	0.2494	3.7818
Eager, Reduction, Normal	lv	0.3034	12.2261	0.4156	12.9451
Eager, Reduction, Normal	lrv	0.3467	13.3324	0.4497	14.1289
Eager, Reduction, Normal	li	0.3287	12.2201	0.417	12.9658
Eager, Reduction, Normal	lri	0.2946	13.2255	0.4431	13.9632
Eager, Reduction, Normal	v	0.3478	10.9766	0.4103	11.7347
Eager, Reduction, Normal	i	0.3744	10.7836	0.4109	11.5689
Eager, Reduction, Shortest	l	0.2876	3.136	0.2496	3.6732
Eager, Reduction, Shortest	lr	0.2966	3.0632	0.2499	3.6096
Eager, Reduction, Shortest	lv	0.2861	8.7993	0.4149	9.5004
Eager, Reduction, Shortest	lrv	0.2963	11.0981	0.4813	11.8757

Table E.18: Times for Problem P5 (Part Two).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Eager, Reduction, Shortest	li	0.2978	10.0922	0.4165	10.8065
Eager, Reduction, Shortest	lri	0.2866	10.182	0.4408	10.9095
Eager, Reduction, Shortest	v	0.2865	8.8622	0.4153	9.5641
Eager, Reduction, Shortest	i	0.2964	9.9902	0.4167	10.7032
Hybrid, Reduction, Normal	l	0.1234	1.6647	0.2224	2.0105
Hybrid, Reduction, Normal	lr	0.1265	1.6636	0.2222	2.0123
Hybrid, Reduction, Normal	lv	0.1389	4.7442	0.375	5.258
Hybrid, Reduction, Normal	lrv	0.137	5.2272	0.4027	5.7669
Hybrid, Reduction, Normal	li	0.1392	4.5265	0.3764	5.0421
Hybrid, Reduction, Normal	lri	0.1366	5.1348	0.4004	5.6718
Hybrid, Reduction, Normal	v	0.1672	7.1297	0.3717	7.6686
Hybrid, Reduction, Normal	i	0.1856	6.7866	0.3745	7.3467
Hybrid, Reduction, Shortest	l	0.1215	1.6688	0.2242	2.0145
Hybrid, Reduction, Shortest	lr	0.1231	1.5804	0.2251	1.9287
Hybrid, Reduction, Shortest	lv	0.1223	4.0859	0.3736	4.5819
Hybrid, Reduction, Shortest	lrv	0.1228	4.811	0.4007	5.3345
Hybrid, Reduction, Shortest	li	0.1237	4.2614	0.38	4.7652
Hybrid, Reduction, Shortest	lri	0.1231	4.6428	0.3976	5.1635
Hybrid, Reduction, Shortest	v	0.1212	4.062	0.3767	4.5599
Hybrid, Reduction, Shortest	i	0.1239	4.2627	0.3757	4.7624
Lazy, Deletion, Normal	l	0.3087	7.6637	0.2261	8.1985
Lazy, Deletion, Normal	lr	0.2999	7.7544	0.2264	8.2807
Lazy, Deletion, Normal	lv	0.3554	48.4368	0.3792	49.1715
Lazy, Deletion, Normal	lrv	0.4445	59.4846	0.4106	60.3397
Lazy, Deletion, Normal	li	0.4407	47.1942	0.381	48.0159
Lazy, Deletion, Normal	lri	0.3474	59.3448	0.411	60.1032
Lazy, Deletion, Normal	v	0.5056	21.9209	0.3794	22.8059
Lazy, Deletion, Normal	i	1.0706	20.3236	0.3817	21.7759

Table E.18: Times for Problem P5 (Part Three).

Configuration	Order	Times (seconds)			Total
		Init.	Comp.	Set Reduction	Time
Lazy, Deletion, Shortest	l	0.3085	7.658	0.2285	8.195
Lazy, Deletion, Shortest	lr	0.3142	7.1285	0.2289	7.6716
Lazy, Deletion, Shortest	lv	0.31	38.4376	0.3788	39.1264
Lazy, Deletion, Shortest	lrv	0.3131	48.5854	0.4124	49.3109
Lazy, Deletion, Shortest	li	0.3119	37.6739	0.385	38.3708
Lazy, Deletion, Shortest	lri	0.411	48.8554	0.4144	49.6808
Lazy, Deletion, Shortest	v	0.3101	38.4175	0.3793	39.1069
Lazy, Deletion, Shortest	i	0.3157	37.7159	0.3788	38.4104
Lazy, Reduction, Normal	l	0.3078	7.6662	0.2193	8.1933
Lazy, Reduction, Normal	lr	0.3019	7.7381	0.2196	8.2596
Lazy, Reduction, Normal	lv	0.3469	47.2212	0.3709	47.939
Lazy, Reduction, Normal	lrv	0.4271	58.1344	0.396	58.9575
Lazy, Reduction, Normal	li	0.4248	46.019	0.3699	46.8136
Lazy, Reduction, Normal	lri	0.3358	58.0609	0.3986	58.7953
Lazy, Reduction, Normal	v	0.4735	19.0497	0.3675	19.8907
Lazy, Reduction, Normal	i	0.9966	18.5482	0.3694	19.9142
Lazy, Reduction, Shortest	l	0.3065	7.661	0.2189	8.1863
Lazy, Reduction, Shortest	lr	0.3102	7.0389	0.2238	7.5729
Lazy, Reduction, Shortest	lv	0.3094	38.2502	0.3694	38.929
Lazy, Reduction, Shortest	lrv	0.3125	48.5249	0.4043	49.2417
Lazy, Reduction, Shortest	li	0.3113	37.6128	0.3719	38.296
Lazy, Reduction, Shortest	lri	0.3097	49.0567	0.3978	49.7642
Lazy, Reduction, Shortest	v	0.3075	38.4302	0.3686	39.1064
Lazy, Reduction, Shortest	i	0.3119	37.7616	0.3707	38.4442

E.2 Order Experiments

Table E.19: Counts for Admissible Order Comparison with Problem A4.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	12.74	0.39	47	37	934	TRUE
1	l	18.89	0.53	44	34	1321	TRUE
1	li	12.79	0.39	47	37	934	TRUE
1	lri	19.13	0.59	56	45	1334	TRUE
1	lrv	22.24	0.66	56	45	1397	TRUE
1	lv	11.69	0.36	47	37	922	TRUE
1	v	11.65	0.36	47	37	922	TRUE
2	i	12.38	0.39	47	37	934	TRUE
2	l	19.31	0.53	44	34	1321	TRUE
2	li	12.42	0.39	47	37	934	TRUE
2	lri	19.46	0.59	56	45	1334	TRUE
2	lrv	21.14	0.64	56	45	1397	TRUE
2	lv	12.11	0.37	47	37	922	TRUE
2	v	12.14	0.36	47	37	922	TRUE
3	i	12.75	0.4	47	37	934	TRUE
3	l	19.16	0.53	44	34	1321	TRUE
3	li	12.49	0.39	47	37	934	TRUE
3	lri	19.5	0.59	56	45	1334	TRUE
3	lrv	21.29	0.64	56	45	1397	TRUE
3	lv	12.11	0.37	47	37	922	TRUE
3	v	12.01	0.37	47	37	922	TRUE
4	i	12.07	0.37	47	37	934	TRUE
4	l	19.82	0.55	44	34	1321	TRUE
4	li	12.12	0.38	47	37	934	TRUE
4	lri	20.84	0.62	56	45	1334	TRUE
4	lrv	21.09	0.64	56	45	1397	TRUE
4	lv	12.44	0.37	47	37	922	TRUE
4	v	12.4	0.38	47	37	922	TRUE
5	i	12.72	0.4	47	37	934	TRUE
5	l	19.02	0.53	44	34	1320	TRUE
5	li	12.8	0.4	47	37	934	TRUE
5	lri	19.25	0.6	56	45	1334	TRUE
5	lrv	22.28	0.67	56	45	1397	TRUE
5	lv	12.04	0.37	47	37	922	TRUE
5	v	11.73	0.36	47	37	922	TRUE

Table E.20: Counts for Admissible Order Comparison with Problem A5.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	4.38	0.28	38	30	480	TRUE
1	l	7.21	0.33	36	29	747	TRUE
1	li	4.36	0.28	38	30	480	TRUE
1	lri	5.11	0.28	36	29	540	TRUE
1	lrv	6.44	0.31	36	29	607	TRUE
1	lv	4.06	0.25	38	30	475	TRUE
1	v	4.06	0.24	38	30	475	TRUE
2	i	4.33	0.26	38	30	480	TRUE
2	l	7.42	0.34	36	29	753	TRUE
2	li	4.35	0.27	38	30	480	TRUE
2	lri	5.06	0.27	36	29	540	TRUE
2	lrv	6.22	0.31	36	29	607	TRUE
2	lv	4.19	0.25	38	30	475	TRUE
2	v	4.18	0.25	38	30	475	TRUE
3	i	4.31	0.28	38	30	480	TRUE
3	l	7.36	0.33	36	29	753	TRUE
3	li	4.33	0.27	38	30	480	TRUE
3	lri	5.04	0.28	36	29	540	TRUE
3	lrv	6.22	0.31	36	29	607	TRUE
3	lv	4.18	0.25	38	30	475	TRUE
3	v	4.19	0.24	38	30	475	TRUE
4	i	4.19	0.26	38	30	480	TRUE
4	l	7.59	0.35	36	29	753	TRUE
4	li	4.21	0.27	38	30	480	TRUE
4	lri	5.21	0.29	36	29	540	TRUE
4	lrv	6.19	0.31	36	29	607	TRUE
4	lv	4.23	0.25	38	30	475	TRUE
4	v	4.21	0.26	38	30	475	TRUE
5	i	4.35	0.28	38	30	479	TRUE
5	l	7.51	0.35	36	29	753	TRUE
5	li	4.34	0.28	38	30	479	TRUE
5	lri	5.89	0.31	36	29	540	TRUE
5	lrv	6.42	0.32	36	29	607	TRUE
5	lv	4.09	0.24	38	30	475	TRUE
5	v	4.07	0.25	38	30	475	TRUE

Table E.21: Counts for Admissible Order Comparison with Problem A51E.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	25.07	0.02	3	0	385	TRUE
1	l	90.05	0.01	15	3	1797	TRUE
1	li	55.57	0.01	17	3	727	TRUE
1	lri	126.02	0.01	19	4	1937	TRUE
1	lrv	59.19	0.02	17	1	930	TRUE
1	lv	133.32	0.01	15	3	1780	TRUE
1	v	141.39	0.01	11	1	1116	TRUE
2	i	23.65	0.01	10	2	382	TRUE
2	l	20.12	0.01	10	2	395	TRUE
2	li	26	0.01	10	2	395	TRUE
2	lri	68.7	0.01	12	3	1369	TRUE
2	lrv	70.18	0.01	12	3	1376	TRUE
2	lv	21.91	0.01	10	2	390	TRUE
2	v	27.26	0.02	11	2	458	TRUE
3	l	18.96	0.01	9	1	425	TRUE
3	li	118.55	0.01	19	4	1879	TRUE
3	lri	20.08	0.01	10	2	444	TRUE
3	lrv	84	0.01	17	4	1634	TRUE
3	lv	20.29	0.01	9	1	411	TRUE
3	v	172.42	0.02	2	0	2501	TRUE
4	i	1026.69	0.01	19	1	3468	TRUE
4	l	40.38	0.01	11	1	702	TRUE
4	li	145.22	0.01	24	4	2082	TRUE
4	lri	48.48	0.01	15	3	596	TRUE
4	lrv	44.94	0.01	15	3	597	TRUE
4	lv	143.51	0.01	24	4	2077	TRUE
5	l	54.56	0.01	16	1	958	TRUE
5	li	103.31	9.14	24	4	1713	FALSE
5	lri	46.13	0.01	16	3	612	TRUE
5	lv	59.07	0.01	16	1	935	TRUE
5	lrv	137.48	0.01	23	4	2130	TRUE

Table E.22: Counts for Admissible Order Comparison with Problem A51H.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	l	36.15	34.99	26	6	1553	FALSE
1	li	37.18	36.73	23	3	1314	FALSE
1	lri	39.2	42.09	26	6	1464	FALSE
1	lrv	41.32	37.77	23	3	1312	FALSE
1	lv	43.22	46.49	26	6	1524	FALSE
2	i	59.14	33.75	23	3	1150	FALSE
2	l	43.79	23.46	23	3	1228	FALSE
2	li	51.81	27.89	23	3	1213	FALSE
2	lri	58.14	61.88	25	5	1964	FALSE
2	lrv	65.81	51.68	26	6	1918	FALSE
2	lv	62.05	34.99	23	3	1187	FALSE
3	i	61.52	1240.03	20	0	1775	FALSE
3	l	39.59	24.48	27	7	1394	FALSE
3	li	47.65	53.82	29	9	1834	FALSE
3	lri	40.3	26.59	27	7	1343	FALSE
3	lrv	47.63	48.55	29	9	1877	FALSE
3	lv	43.5	31.26	27	7	1365	FALSE
4	l	45.08	24.41	23	3	1228	FALSE
4	li	63.69	60.07	24	4	1981	FALSE
4	lri	48.22	28.85	23	3	1189	FALSE
4	lrv	47.86	28.57	23	3	1191	FALSE
4	lv	66.73	62.62	26	6	2009	FALSE
5	i	215.82	764.51	22	2	1692	FALSE
5	l	32.41	24.05	23	3	1084	FALSE
5	li	61.42	64.98	24	4	2012	FALSE
5	lri	33.21	31.51	23	3	1068	FALSE
5	lrv	57.78	55.13	24	4	1797	FALSE
5	lv	42.09	34.41	25	5	1239	FALSE

Table E.23: Counts for Admissible Order Comparison with Problem A6.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	3.07	0.2	31	26	379	TRUE
1	l	2.94	0.13	22	19	357	TRUE
1	li	3.05	0.19	31	26	379	TRUE
1	lri	3.79	0.2	31	26	405	TRUE
1	lrv	3.57	0.21	31	26	410	TRUE
1	lv	2.91	0.18	31	26	372	TRUE
1	v	2.9	0.17	31	26	372	TRUE
2	i	3.01	0.18	31	26	379	TRUE
2	l	2.99	0.14	22	19	357	TRUE
2	li	3	0.19	31	26	379	TRUE
2	lri	3.32	0.19	31	26	405	TRUE
2	lrv	3.44	0.21	31	26	410	TRUE
2	lv	2.98	0.18	31	26	372	TRUE
2	v	3	0.18	31	26	372	TRUE
3	i	3	0.19	31	26	379	TRUE
3	l	2.87	0.13	22	19	357	TRUE
3	li	3.01	0.19	31	26	379	TRUE
3	lri	3.31	0.19	31	26	405	TRUE
3	lrv	3.46	0.2	31	26	410	TRUE
3	lv	3.05	0.18	31	26	372	TRUE
3	v	2.99	0.18	31	26	372	TRUE
4	i	2.94	0.18	31	26	379	TRUE
4	l	2.95	0.13	22	19	357	TRUE
4	li	2.92	0.19	31	26	379	TRUE
4	lri	4.03	0.21	31	26	405	TRUE
4	lrv	3.41	0.21	31	26	410	TRUE
4	lv	3.04	0.18	31	26	372	TRUE
4	v	3.04	0.18	31	26	372	TRUE
5	i	3.04	0.17	31	26	379	TRUE
5	l	2.95	0.14	22	19	357	TRUE
5	li	3.05	0.19	31	26	379	TRUE
5	lri	3.27	0.18	31	26	405	TRUE
5	lrv	3.57	0.21	31	26	410	TRUE
5	lv	2.94	0.18	31	26	372	TRUE
5	v	2.9	0.18	31	26	372	TRUE

Table E.24: Counts for Admissible Order Comparison with Problem AGS.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	3.31	4.09	27	3	315	TRUE
1	l	3.06	2.73	31	2	250	TRUE
1	li	4.17	3.48	31	3	309	TRUE
1	lri	3.45	2.83	32	3	261	TRUE
1	lrv	4.06	3.38	32	3	307	TRUE
1	lv	4.12	3.28	31	2	250	TRUE
1	v	2.81	2.85	27	1	204	TRUE
2	i	1.91	3.04	21	1	195	TRUE
2	l	2.89	2.64	30	2	242	TRUE
2	li	3.86	3.1	32	3	270	TRUE
2	lri	4.21	2.93	33	4	297	TRUE
2	lrv	4.68	3.69	32	3	324	TRUE
2	lv	3.95	2.94	31	4	281	TRUE
2	v	2.19	2.3	17	3	198	TRUE
3	i	2.13	2.59	19	3	158	TRUE
3	l	3.79	3.01	32	3	281	TRUE
3	li	4.37	3.32	32	4	296	TRUE
3	lri	4.1	3.32	33	3	273	TRUE
3	lrv	4.38	3.77	36	5	295	TRUE
3	lv	3.89	3.09	30	2	258	TRUE
3	v	3.45	3.71	26	1	268	TRUE
4	i	1.08	2.68	13	1	133	TRUE
4	l	4.18	3.64	35	5	306	TRUE
4	li	3.54	2.84	29	2	228	TRUE
4	lri	4.59	4.02	36	5	329	TRUE
4	lrv	4.3	3.38	33	3	284	TRUE
4	lv	3.78	3.34	29	2	266	TRUE
4	v	1.96	2.08	12	1	151	TRUE
5	i	1.96	3.39	17	1	216	TRUE
5	l	3.6	2.82	33	4	294	TRUE
5	li	3.33	2.95	31	2	248	TRUE
5	lri	5.16	3.32	35	5	349	TRUE
5	lrv	2.84	2.93	31	3	225	TRUE
5	lv	4.5	3.28	35	5	324	TRUE
5	v	3.47	3.07	29	3	243	TRUE

Table E.25: Counts for Admissible Order Comparison with Problem BT7.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	1.44	0.28	55	25	61	FALSE
1	l	0.31	0.1	10	5	27	TRUE
1	li	1.44	0.28	55	25	61	FALSE
1	lri	2.1	0.25	37	17	128	FALSE
1	lrv	1.43	0.29	55	25	59	FALSE
1	lv	1.84	0.25	31	14	121	FALSE
1	v	1.95	0.25	31	14	121	FALSE
2	i	2.42	0.22	32	16	172	FALSE
2	l	1.15	0.26	38	20	87	FALSE
2	li	2.45	0.21	32	16	172	FALSE
2	lri	1.56	0.28	56	27	71	FALSE
2	lrv	1.32	0.23	42	21	75	FALSE
2	lv	1.4	0.27	48	24	61	FALSE
2	v	1.41	0.27	48	24	61	FALSE
3	i	3.17	0.33	49	26	184	FALSE
3	l	1.27	0.25	52	28	79	FALSE
3	li	3.18	0.33	49	26	184	FALSE
3	lri	1.47	0.3	47	24	67	FALSE
3	lrv	1.63	0.28	56	27	80	FALSE
3	lv	2.28	0.28	51	27	141	FALSE
3	v	2.25	0.26	51	27	141	FALSE
4	i	2.29	0.27	51	27	141	FALSE
4	l	0.31	0.1	10	5	27	TRUE
4	li	2.29	0.27	51	27	141	FALSE
4	lri	1.66	0.29	56	27	81	FALSE
4	lrv	2.46	0.38	43	19	172	FALSE
4	lv	1.14	0.23	42	21	58	FALSE
4	v	1.14	0.23	42	21	58	FALSE
5	i	2.47	0.31	41	21	158	FALSE
5	l	1.27	0.25	52	28	79	FALSE
5	li	2.46	0.31	41	21	158	FALSE
5	lri	1.44	0.28	55	25	61	FALSE
5	lrv	1.35	0.23	42	21	75	FALSE
5	lv	1.54	0.28	55	25	74	FALSE
5	v	1.58	0.31	55	25	74	FALSE

Table E.26: Counts for Admissible Order Comparison with Problem CG5.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	24.25	1.52	553	444	1530	FALSE
1	l	25.19	1.4	400	344	2443	FALSE
1	li	23.42	1.44	553	444	1530	FALSE
1	lri	40.25	2.17	754	556	2786	FALSE
1	lrv	34.63	1.84	746	545	2180	FALSE
1	lv	20.94	1.42	464	403	1726	FALSE
1	v	20.92	1.41	464	403	1726	FALSE
2	i	23.07	1.36	501	427	1919	FALSE
2	l	17.84	1.16	350	305	1932	TRUE
2	li	23.04	1.36	501	427	1919	FALSE
2	lri	30.92	1.76	633	522	2406	FALSE
2	lrv	29.97	1.72	607	495	2290	FALSE
2	lv	21.79	1.37	497	428	1710	FALSE
2	v	21.68	1.37	497	428	1710	FALSE
3	i	25.38	1.57	500	425	1961	FALSE
3	l	16.39	1.12	326	284	1704	TRUE
3	li	23.44	1.45	500	425	1961	FALSE
3	lri	32.72	1.7	657	568	2541	FALSE
3	lrv	37.19	1.93	673	578	2461	FALSE
3	lv	20.67	1.32	460	391	1487	FALSE
3	v	20.32	1.3	460	391	1487	FALSE
4	i	25.83	1.42	572	500	2168	FALSE
4	l	19.35	1.17	353	302	1912	FALSE
4	li	26.11	1.47	572	500	2168	FALSE
4	lri	30.93	1.72	639	528	2272	FALSE
4	lrv	28.55	1.54	634	543	2055	FALSE
4	lv	21.54	1.29	494	417	1654	FALSE
4	v	21.51	1.28	494	417	1654	FALSE
5	i	24.37	1.39	522	446	2000	FALSE
5	l	14.08	0.96	268	234	1385	FALSE
5	li	24.21	1.39	522	446	2000	FALSE
5	lri	39.7	1.84	742	558	3035	FALSE
5	lrv	34.34	1.74	672	564	2563	FALSE
5	lv	28.38	1.5	569	484	2419	FALSE
5	v	28.19	1.5	569	484	2419	FALSE

Table E.27: Counts for Admissible Order Comparison with Problem CGL.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	1.04	0.13	68	65	59	TRUE
1	l	5.31	0.61	296	279	652	TRUE
1	li	5.78	0.64	296	274	687	TRUE
1	lri	5.45	0.61	296	279	652	TRUE
1	lrv	5.75	0.64	296	274	687	TRUE
1	lv	5.41	0.61	296	279	652	TRUE
1	v	0.65	1.59	26	23	168	TRUE
2	i	0.56	0.27	27	23	76	TRUE
2	l	5.12	0.57	280	264	630	TRUE
2	li	5.77	0.62	296	276	688	TRUE
2	lri	5.62	0.61	296	279	671	TRUE
2	lrv	5.8	0.62	296	275	674	TRUE
2	lv	5.76	0.62	296	277	677	TRUE
2	v	0.27	0.19	19	18	32	TRUE
3	i	0.14	0.37	11	11	69	TRUE
3	l	4.88	0.53	280	264	632	TRUE
3	li	5.35	0.58	296	276	685	TRUE
3	lri	5.15	0.56	296	279	668	TRUE
3	lrv	5.52	0.58	296	276	673	TRUE
3	lv	5.1	0.57	296	279	663	TRUE
3	v	0.19	0.16	17	16	31	TRUE
4	i	0.34	0.16	19	17	33	TRUE
4	l	5.58	0.6	296	279	666	TRUE
4	li	6.25	0.66	312	291	723	TRUE
4	lri	5.84	0.62	296	277	665	TRUE
4	lrv	5.88	0.62	296	276	673	TRUE
4	lv	5.84	0.61	296	278	662	TRUE
4	v	0.39	0.2	19	17	41	TRUE
5	i	0.29	0.19	20	18	43	TRUE
5	l	5.58	0.6	312	292	695	TRUE
5	li	5.39	0.6	296	276	686	TRUE
5	lri	5.67	0.6	312	293	710	TRUE
5	lrv	6.1	0.64	312	290	728	TRUE
5	lv	5.65	0.59	312	293	711	TRUE
5	v	0.49	0.16	32	28	48	TRUE

Table E.28: Counts for Admissible Order Comparison with Problem CGL1.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	1.04	0.13	68	65	59	TRUE
1	l	5.36	0.61	296	279	652	TRUE
1	li	5.8	0.65	296	274	687	TRUE
1	lri	5.48	0.62	296	279	652	TRUE
1	lrv	5.79	0.65	296	274	687	TRUE
1	lv	5.45	0.61	296	279	652	TRUE
1	v	0.63	1.57	26	23	168	TRUE
2	i	0.58	0.26	27	23	77	TRUE
2	l	5.15	0.58	280	264	630	TRUE
2	li	5.73	0.62	296	276	688	TRUE
2	lri	5.62	0.62	296	279	671	TRUE
2	lrv	5.8	0.61	296	275	674	TRUE
2	lv	5.79	0.62	296	277	677	TRUE
2	v	0.27	0.19	19	18	32	TRUE
3	i	0.15	0.37	11	11	69	TRUE
3	l	4.92	0.53	280	264	632	TRUE
3	li	5.33	0.58	296	276	685	TRUE
3	lri	5.15	0.55	296	279	668	TRUE
3	lrv	5.42	0.57	296	276	673	TRUE
3	lv	5.12	0.57	296	279	663	TRUE
3	v	0.2	0.15	17	16	31	TRUE
4	i	0.34	0.16	19	17	33	TRUE
4	l	5.57	0.61	296	279	666	TRUE
4	li	6.27	0.65	312	291	723	TRUE
4	lri	5.91	0.61	296	277	665	TRUE
4	lrv	5.87	0.62	296	276	673	TRUE
4	lv	5.87	0.6	296	278	662	TRUE
4	v	0.39	0.2	19	17	41	TRUE
5	i	0.29	0.19	20	18	43	TRUE
5	l	5.9	0.62	312	292	695	TRUE
5	li	5.38	0.57	296	276	686	TRUE
5	lri	5.72	0.6	312	293	710	TRUE
5	lrv	5.87	0.6	312	290	728	TRUE
5	lv	5.76	0.57	312	293	711	TRUE
5	v	0.55	0.18	32	28	48	TRUE

Table E.29: Counts for Admissible Order Comparison with Problem DCYC.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	6.91	0.59	108	71	520	TRUE
1	l	7.57	0.7	135	69	599	TRUE
1	li	12.78	1.03	170	93	879	TRUE
1	lri	11.13	0.96	162	88	719	TRUE
1	lrv	13.57	1.02	192	109	849	TRUE
1	lv	9.28	0.76	136	71	669	TRUE
1	v	9.12	0.83	135	74	698	TRUE
2	i	6.7	0.79	118	72	504	TRUE
2	l	13.07	1.03	173	104	867	TRUE
2	li	9.36	0.81	150	89	667	TRUE
2	lri	11.15	0.98	167	95	838	TRUE
2	lrv	10.64	0.98	155	83	737	TRUE
2	lv	10.87	1	159	92	849	TRUE
2	v	6.45	0.53	101	61	488	TRUE
3	i	9.81	0.91	154	95	741	TRUE
3	l	8.12	0.74	144	90	746	TRUE
3	li	8.84	0.87	148	80	673	TRUE
3	lri	12.22	1.02	177	107	848	TRUE
3	lrv	9.43	0.9	162	97	685	TRUE
3	lv	12.59	0.93	184	118	1031	TRUE
3	v	6.81	0.59	119	80	481	TRUE
4	i	6.82	0.59	95	55	563	TRUE
4	l	8.3	0.77	146	88	697	TRUE
4	li	10.9	0.88	164	87	714	TRUE
4	lri	8.04	0.81	136	77	602	TRUE
4	lrv	9.67	0.85	153	83	689	TRUE
4	lv	11.26	0.89	178	107	781	TRUE
4	v	8.93	0.78	130	81	641	TRUE
5	i	9.36	0.73	116	75	745	TRUE
5	l	8.99	0.71	146	79	700	TRUE
5	li	12.03	0.91	168	87	752	TRUE
5	lri	8.84	0.83	150	85	640	TRUE
5	lrv	14.27	1.08	188	116	1004	TRUE
5	lv	9.47	0.86	147	77	678	TRUE
5	v	13.06	0.97	169	97	954	TRUE

Table E.30: Counts for Admissible Order Comparison with Problem ELP.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	3.52	0.24	99	85	142	TRUE
1	l	1.48	0.11	59	52	84	TRUE
1	li	3.28	0.24	99	85	142	TRUE
1	lri	1.47	0.11	59	52	87	TRUE
1	lrv	2.79	0.22	93	79	124	TRUE
1	lv	1.48	0.1	59	52	84	TRUE
1	v	1.47	0.12	59	52	84	TRUE
2	i	3.1	0.23	99	85	138	TRUE
2	l	1.49	0.11	59	52	84	TRUE
2	li	3.17	0.23	99	85	138	TRUE
2	lri	1.47	0.12	59	52	87	TRUE
2	lrv	2.8	0.21	93	79	121	TRUE
2	lv	1.48	0.11	59	52	84	TRUE
2	v	1.47	0.11	59	52	84	TRUE
3	i	1.43	0.12	59	52	84	TRUE
3	l	2.72	0.2	93	79	111	TRUE
3	li	1.44	0.11	59	52	84	TRUE
3	lri	3.68	0.22	116	102	166	TRUE
3	lrv	1.44	0.12	59	52	87	TRUE
3	lv	2.74	0.21	93	79	117	TRUE
3	v	2.76	0.2	93	79	117	TRUE
4	i	3.1	0.23	99	85	136	TRUE
4	l	1.49	0.11	59	52	84	TRUE
4	li	3.07	0.23	99	85	136	TRUE
4	lri	1.49	0.12	59	52	87	TRUE
4	lrv	2.79	0.22	93	79	119	TRUE
4	lv	1.48	0.11	59	52	84	TRUE
4	v	1.47	0.11	59	52	84	TRUE
5	i	1.45	0.11	59	52	84	TRUE
5	l	2.74	0.2	93	79	111	TRUE
5	li	1.55	0.12	59	52	84	TRUE
5	lri	3.72	0.22	116	102	167	TRUE
5	lrv	1.48	0.12	59	52	87	TRUE
5	lv	2.77	0.21	93	79	119	TRUE
5	v	2.76	0.21	93	79	119	TRUE

Table E.31: Counts for Admissible Order Comparison with Problem HWEB.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	2095.89	126.73	557	457	31839	FALSE
1	l	0.55	0.07	33	26	83	TRUE
1	li	2216.38	341.73	557	457	39582	FALSE
1	lri	1754.78	14.18	587	487	26742	FALSE
1	lrv	41132.8	354.12	562	462	545810	FALSE
1	lv	4996.49	14.32	587	487	57256	FALSE
1	v	0.27	0.05	17	13	38	TRUE
2	i	0.64	0.05	28	20	85	TRUE
2	l	1382.26	10.73	575	475	21351	FALSE
2	li	3618.24	9.99	567	467	44409	FALSE
2	lri	1451.14	10.69	575	475	21351	FALSE
2	lrv	19362.2	10.19	573	473	268246	FALSE
2	lv	1450.76	10.7	575	475	21351	FALSE
2	v	0.31	0.04	19	13	41	TRUE
3	i	0.55	0.05	29	22	78	TRUE
3	l	1299.1	10.44	574	474	21258	FALSE
3	lri	1299.08	9.88	574	474	21258	FALSE
3	lv	1399.07	10.43	574	474	21258	FALSE
3	v	0.32	0.04	19	13	41	TRUE
4	i	0.34	0.05	20	15	51	TRUE
4	lri	1309.95	9.96	570	470	20895	FALSE
4	lrv	1016.29	9.41	564	464	16440	
4	v	0.29	0.04	19	13	41	TRUE
5	i	0.34	0.05	20	15	51	TRUE
5	lri	1291.5	9.66	570	470	20895	FALSE
5	lrv	3311.5	9.58	564	464	40210	FALSE
5	v	0.29	0.04	19	13	41	TRUE

Table E.32: Counts for Admissible Order Comparison with Problem HWRES.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	0.1	0.02	12	10	16	TRUE
1	l	0.09	0.02	12	10	16	TRUE
1	li	0.09	0.03	12	10	16	TRUE
1	lri	0.1	0.02	12	10	16	TRUE
1	lrv	0.09	0.02	12	10	16	TRUE
1	lv	0.1	0.02	12	10	16	TRUE
1	v	0.1	0.02	12	10	16	TRUE
2	i	0.1	0.02	12	10	16	TRUE
2	l	0.1	0.02	12	10	16	TRUE
2	li	0.1	0.02	12	10	16	TRUE
2	lri	0.1	0.02	12	10	16	TRUE
2	lrv	0.1	0.02	12	10	16	TRUE
2	lv	0.1	0.02	12	10	16	TRUE
2	v	0.11	0.02	12	10	16	TRUE
3	i	0.1	0.02	12	10	16	TRUE
3	l	0.09	0.02	12	10	16	TRUE
3	li	0.1	0.02	12	10	16	TRUE
3	lri	0.1	0.02	12	10	16	TRUE
3	lrv	0.1	0.02	12	10	16	TRUE
3	lv	0.09	0.02	12	10	16	TRUE
3	v	0.1	0.03	12	10	16	TRUE
4	i	0.1	0.02	12	10	16	TRUE
4	l	0.1	0.02	12	10	16	TRUE
4	li	0.09	0.03	12	10	16	TRUE
4	lri	0.11	0.02	12	10	16	TRUE
4	lrv	0.09	0.02	12	10	16	TRUE
4	lv	0.1	0.02	12	10	16	TRUE
4	v	0.1	0.02	12	10	16	TRUE
5	i	0.11	0.02	12	10	16	TRUE
5	l	0.09	0.02	12	10	16	TRUE
5	li	0.1	0.02	12	10	16	TRUE
5	lri	0.09	0.02	12	10	16	TRUE
5	lrv	0.09	0.02	12	10	16	TRUE
5	lv	0.1	0.02	12	10	16	TRUE
5	v	0.1	0.02	12	10	16	TRUE

Table E.33: Counts for Admissible Order Comparison with Problem ICYC.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	2.07	0.26	40	23	62	TRUE
1	l	0.93	0.17	29	20	46	TRUE
1	li	2.23	0.28	48	33	124	TRUE
1	lri	1.02	0.17	29	20	53	TRUE
1	lrv	2.22	0.29	48	33	124	TRUE
1	lv	1.01	0.17	29	20	52	TRUE
1	v	0.8	0.17	25	17	38	TRUE
2	i	1.57	0.23	40	27	58	TRUE
2	l	2.18	0.3	48	33	125	TRUE
2	li	1.73	0.24	40	27	73	TRUE
2	lri	2.59	0.3	54	37	135	TRUE
2	lrv	1.86	0.21	40	25	80	TRUE
2	lv	1.7	0.24	40	27	83	TRUE
2	v	1.69	0.23	40	27	76	TRUE
3	i	1.63	0.22	40	27	58	TRUE
3	l	2.43	0.29	54	38	139	TRUE
3	li	1.68	0.23	40	27	71	TRUE
3	lri	2.48	0.28	54	38	139	TRUE
3	lrv	1.68	0.23	40	27	71	TRUE
3	lv	2.5	0.3	54	38	137	TRUE
4	l	2.39	0.29	54	38	139	TRUE
4	li	2.48	0.3	54	38	137	TRUE
4	lri	1.88	0.25	40	27	71	TRUE
4	lrv	1.67	0.24	40	27	71	TRUE
4	lv	2.46	0.3	54	38	137	TRUE
5	l	1.59	0.23	39	26	75	TRUE
5	li	2.3	0.27	47	30	106	TRUE
5	lri	1.59	0.23	39	26	68	TRUE
5	lrv	2.56	0.29	54	37	137	TRUE
5	lv	1.59	0.23	39	26	66	TRUE
5	v	1.54	0.23	39	26	55	TRUE

Table E.34: Counts for Admissible Order Comparison with Problem M1.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	1338.13	5.37	29	16	3373	FALSE
1	l	2.03	0.31	12	6	110	TRUE
1	li	1341.35	5.34	29	16	3373	FALSE
1	lri	8.4	2.06	40	16	349	FALSE
1	lrv	26.39	3.89	23	12	496	FALSE
1	lv	183.05	1.99	29	16	1491	FALSE
1	v	182.6	1.99	29	16	1491	FALSE
2	i	158.04	5.1	78	40	1289	FALSE
2	l	0.57	0.18	6	3	30	TRUE
2	li	158.51	5.07	78	40	1289	FALSE
2	lri	25.4	2.87	47	24	556	FALSE
2	lrv	25.42	4.23	57	27	588	FALSE
2	lv	113.74	2.83	62	34	1189	FALSE
2	v	112.51	2.83	62	34	1189	FALSE
3	i	236.33	4.74	71	39	1931	FALSE
3	l	0.44	0.18	6	3	28	TRUE
3	li	236.68	4.75	71	39	1931	FALSE
3	lri	49.88	3.57	68	37	969	FALSE
3	lrv	50.73	4.13	87	40	945	FALSE
3	lv	41.95	3.52	53	23	695	FALSE
3	v	41.78	3.51	53	23	695	FALSE
4	i	98.1	3.32	62	34	1052	FALSE
4	l	0	0.1	0	0	0	TRUE
4	li	98.35	3.33	62	34	1052	FALSE
4	lri	30.5	4.97	53	25	739	FALSE
4	lrv	28.34	5.76	49	24	623	FALSE
4	lv	72.45	2.8	70	37	1071	FALSE
4	v	73.3	2.91	70	37	1071	FALSE
5	i	81.34	3.46	87	47	1048	FALSE
5	l	0.56	0.17	6	3	32	TRUE
5	li	81.79	3.48	87	47	1048	FALSE
5	lri	40.49	6.83	56	26	866	FALSE
5	lrv	21.93	2.71	53	25	517	FALSE
5	lv	86.61	2.88	67	32	1095	FALSE
5	v	86.16	2.87	67	32	1095	FALSE

Table E.35: Counts for Admissible Order Comparison with Problem MBFS.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	2.14	0.39	72	34	64	FALSE
1	l	0.35	0.14	10	5	29	TRUE
1	li	2.13	0.39	72	34	64	FALSE
1	lri	2.55	0.5	40	16	133	FALSE
1	lrv	2.46	0.55	66	29	105	FALSE
1	lv	7.46	0.43	29	15	328	FALSE
1	v	6.87	0.41	29	15	328	FALSE
2	i	7.12	0.54	52	27	331	FALSE
2	l	1.3	0.34	38	20	68	FALSE
2	li	7.09	0.54	52	27	331	FALSE
2	lri	3.48	0.66	65	27	134	FALSE
2	lrv	2.91	0.49	45	22	152	FALSE
2	lv	2.63	0.46	58	26	95	FALSE
2	v	2.65	0.46	58	26	95	FALSE
3	i	1.96	0.3	44	22	113	FALSE
3	l	1.17	0.28	21	13	86	TRUE
3	li	1.97	0.29	44	22	113	FALSE
3	lri	3.13	0.47	34	15	177	FALSE
3	lrv	3.19	0.58	72	36	136	FALSE
3	lv	2.65	0.42	42	19	143	FALSE
3	v	2.64	0.41	42	19	143	FALSE
4	i	2.28	0.43	43	20	93	FALSE
4	l	2.47	0.42	52	28	156	FALSE
4	li	2.29	0.43	43	20	93	FALSE
4	lri	2.47	0.43	45	22	130	FALSE
4	lrv	2.45	0.53	57	26	119	FALSE
4	lv	4.94	0.61	58	31	251	FALSE
4	v	4.93	0.62	58	31	251	FALSE
5	i	2.74	0.44	42	19	140	FALSE
5	l	3.12	0.58	32	18	226	FALSE
5	li	2.74	0.44	42	19	140	FALSE
5	lri	3.23	0.58	72	36	145	FALSE
5	lrv	3.41	0.55	56	27	166	FALSE
5	lv	2.56	0.51	57	26	107	FALSE
5	v	2.57	0.5	57	26	107	FALSE

Table E.36: Counts for Admissible Order Comparison with Problem MDFS.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	1.24	0.22	33	14	52	FALSE
1	l	0.74	0.17	30	6	46	FALSE
1	li	1.24	0.22	33	14	52	FALSE
1	lri	0.74	0.17	30	6	46	FALSE
1	lrv	1.19	0.21	33	14	50	FALSE
1	lv	0.73	0.16	30	6	46	FALSE
1	v	0.73	0.16	30	6	46	FALSE
2	i	1.87	0.36	68	30	71	FALSE
2	l	0.55	0.19	24	10	31	FALSE
2	li	1.87	0.37	68	30	71	FALSE
2	lri	2.55	0.43	62	25	110	FALSE
2	lrv	1.64	0.38	55	26	72	FALSE
2	lv	2.17	0.38	59	28	112	FALSE
2	v	2.14	0.37	59	28	112	FALSE
3	i	2.24	0.42	73	34	99	FALSE
3	l	0.91	0.22	38	11	57	FALSE
3	li	2.26	0.42	73	34	99	FALSE
3	lri	1.83	0.44	51	24	83	FALSE
3	lrv	2.64	0.43	62	26	133	FALSE
3	lv	1.69	0.36	65	27	66	FALSE
3	v	1.68	0.36	65	27	66	FALSE
4	i	1.55	0.32	61	30	62	FALSE
4	l	1.5	0.36	37	18	88	FALSE
4	li	1.73	0.35	61	30	62	FALSE
4	lri	2.17	0.39	64	27	86	FALSE
4	lrv	1.82	0.44	51	24	79	FALSE
4	lv	2.34	0.4	73	38	109	FALSE
4	v	2.41	0.42	73	38	109	FALSE
5	i	2.49	0.4	66	30	123	FALSE
5	l	0.59	0.2	18	7	38	FALSE
5	li	2.47	0.4	66	30	123	FALSE
5	lri	1.75	0.32	46	24	76	FALSE
5	lrv	1.71	0.37	62	26	62	FALSE
5	lv	2.68	0.41	64	32	133	FALSE
5	v	2.95	0.43	64	32	133	FALSE

Table E.37: Counts for Admissible Order Comparison with Problem MM.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	1.72	0.67	44	22	63	TRUE
1	l	1.46	0.72	36	18	72	TRUE
1	li	1.73	0.68	44	22	63	TRUE
1	lri	1.26	0.58	28	14	65	TRUE
1	lrv	1.4	0.63	28	14	69	TRUE
1	lv	1.66	0.68	44	22	59	TRUE
1	v	1.66	0.69	44	22	59	TRUE
2	i	1.1	0.53	42	15	20	FALSE
2	l	0.42	0.39	18	5	7	TRUE
2	li	1.09	0.53	42	15	20	FALSE
2	lri	0.61	0.4	21	8	16	TRUE
2	lrv	1.16	0.58	44	22	31	TRUE
2	lv	1.61	0.57	49	23	50	FALSE
2	v	1.62	0.57	49	23	50	FALSE
3	i	2.78	0.84	82	32	61	FALSE
3	l	0.22	0.26	10	1	1	TRUE
3	li	2.76	0.82	82	32	61	FALSE
3	lri	0.17	0.21	9	2	4	TRUE
3	lrv	0.21	0.23	10	3	5	TRUE
3	lv	2.09	0.76	59	25	61	FALSE
3	v	2.07	0.76	59	25	61	FALSE
4	i	1.33	0.55	38	20	54	TRUE
4	l	0.2	0.25	9	1	1	TRUE
4	li	1.32	0.54	38	20	54	TRUE
4	lri	1.85	0.8	51	26	67	TRUE
4	lrv	1.79	0.69	58	28	52	FALSE
4	lv	0.83	0.4	30	15	31	TRUE
4	v	0.82	0.4	30	15	31	TRUE
5	i	0.13	0.19	8	1	1	TRUE
5	l	2.63	0.89	62	27	113	FALSE
5	li	0.13	0.18	8	1	1	TRUE
5	lri	4.27	1.16	92	40	143	FALSE
5	lrv	0.2	0.23	11	2	3	TRUE
5	lv	3.52	1.01	79	32	124	FALSE
5	v	3.6	1.05	79	32	124	FALSE

Table E.38: Counts for Admissible Order Comparison with Problem MS.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	3.99	0.91	60	36	165	TRUE
1	l	4.63	1.18	60	36	254	TRUE
1	li	4.01	0.9	60	36	165	TRUE
1	lri	5.06	1.23	60	36	254	TRUE
1	lrv	4.65	1.23	60	36	236	TRUE
1	lv	3.92	0.89	60	36	158	TRUE
1	v	3.9	0.91	60	36	158	TRUE
2	i	1.02	0.52	41	16	16	FALSE
2	l	0.44	0.4	18	5	7	TRUE
2	li	1.04	0.51	41	16	16	FALSE
2	lri	1.78	0.62	46	24	57	FALSE
2	lrv	1.89	0.67	46	22	58	TRUE
2	lv	0.91	0.43	26	12	29	TRUE
2	v	0.9	0.44	26	12	29	TRUE
3	i	2.5	0.77	78	31	55	FALSE
3	l	0.3	0.26	13	3	3	TRUE
3	li	2.37	0.73	78	31	55	FALSE
3	lri	0.22	0.22	11	3	5	TRUE
3	lrv	0.36	0.28	16	6	10	TRUE
3	lv	1.27	0.6	46	19	34	FALSE
3	v	1.27	0.6	46	19	34	FALSE
4	i	3.07	0.7	48	28	138	TRUE
4	l	0.19	0.23	9	1	1	TRUE
4	li	3.08	0.7	48	28	138	TRUE
4	lri	2.73	0.73	50	26	113	TRUE
4	lrv	2.33	0.71	67	37	82	FALSE
4	lv	0.61	0.4	24	10	22	TRUE
4	v	0.65	0.4	24	10	22	TRUE
5	i	0.13	0.2	8	1	1	TRUE
5	l	4.58	1.34	72	34	230	FALSE
5	li	0.14	0.18	8	1	1	TRUE
5	lri	4.03	1.09	90	38	117	FALSE
5	lrv	0.2	0.24	11	2	3	TRUE
5	lv	3.31	0.88	83	35	95	FALSE
5	v	3.33	0.88	83	35	95	FALSE

Table E.39: Counts for Admissible Order Comparison with Problem MT1.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	21.34	0.79	28	15	575	FALSE
1	l	2.22	0.5	69	36	107	FALSE
1	li	21.44	0.8	28	15	575	FALSE
1	lri	2.49	0.52	69	33	96	FALSE
1	lrv	8.39	1.13	33	16	348	FALSE
1	lv	2.58	0.46	69	33	86	FALSE
1	v	2.6	0.44	69	33	86	FALSE
2	i	3.77	0.49	47	26	122	FALSE
2	l	2.79	0.73	47	25	184	FALSE
2	li	3.81	0.48	47	26	122	FALSE
2	lri	4.06	0.88	44	20	188	FALSE
2	lrv	3.3	0.63	38	19	168	FALSE
2	lv	3.82	0.6	62	32	123	FALSE
2	v	3.8	0.59	62	32	123	FALSE
3	i	3.02	0.47	62	30	98	FALSE
3	l	4.8	0.51	36	17	249	FALSE
3	li	2.98	0.44	62	30	98	FALSE
3	lri	3.37	0.74	42	19	151	FALSE
3	lrv	2.72	0.52	62	30	119	FALSE
3	lv	15.84	0.75	46	24	436	FALSE
3	v	15.79	0.75	46	24	436	FALSE
4	i	12.17	0.83	45	23	415	FALSE
4	l	2.04	0.44	39	21	117	FALSE
4	li	12.22	0.82	45	23	415	FALSE
4	lri	3.7	0.74	62	32	175	FALSE
4	lrv	5.61	1.15	37	17	267	FALSE
4	lv	1.72	0.28	47	24	58	FALSE
4	v	1.7	0.28	47	24	58	FALSE
5	i	21.18	0.82	51	27	602	FALSE
5	l	2.11	0.44	47	26	114	FALSE
5	li	21.33	0.86	51	27	602	FALSE
5	lri	3.52	0.95	56	25	167	FALSE
5	lrv	4.15	0.76	42	21	191	FALSE
5	lv	3.07	0.44	62	30	104	FALSE
5	v	3.06	0.44	62	30	104	FALSE

Table E.40: Counts for Admissible Order Comparison with Problem MT2.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	100.98	1.29	29	16	1186	FALSE
1	l	3.46	0.78	69	36	183	FALSE
1	li	101.2	1.28	29	16	1186	FALSE
1	lri	4.14	0.84	69	33	180	FALSE
1	lrv	9.63	0.92	23	12	297	FALSE
1	lv	4.52	0.64	69	33	150	FALSE
1	v	4.53	0.64	69	33	150	FALSE
2	i	6.09	0.67	47	26	161	FALSE
2	l	4.06	0.95	31	18	233	FALSE
2	li	6.11	0.69	47	26	161	FALSE
2	lri	10.65	1.89	33	14	359	FALSE
2	lrv	9.36	2.05	37	17	396	FALSE
2	lv	7.57	0.84	62	32	210	FALSE
2	v	7.5	0.85	62	32	210	FALSE
3	i	4.54	0.65	62	30	140	FALSE
3	l	0	0.06	0	0	0	TRUE
3	li	4.38	0.59	62	30	140	FALSE
3	lri	7.77	1.81	33	14	320	FALSE
3	lrv	3.88	0.72	62	30	169	FALSE
3	lv	42.5	1.17	37	18	704	FALSE
3	v	43.01	1.18	37	18	704	FALSE
4	i	25.78	1.68	42	19	470	FALSE
4	l	2.51	0.52	39	21	131	FALSE
4	li	23.72	1.54	42	19	470	FALSE
4	lri	7.18	1.17	62	32	300	FALSE
4	lrv	9.42	2.15	33	14	344	FALSE
4	lv	3.62	0.44	47	24	112	FALSE
4	v	3.59	0.44	47	24	112	FALSE
5	i	49.89	1.14	52	27	879	FALSE
5	l	2.89	0.54	47	26	146	FALSE
5	li	49.95	1.13	52	27	879	FALSE
5	lri	4.83	1.57	33	14	208	FALSE
5	lrv	12.17	1.81	35	16	397	FALSE
5	lv	3.54	0.44	47	24	112	FALSE
5	v	3.53	0.43	47	24	112	FALSE

Table E.41: Counts for Admissible Order Comparison with Problem MT3.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	67.9	1.19	29	16	1055	FALSE
1	l	0.53	0.21	18	9	37	TRUE
1	li	68.15	1.2	29	16	1055	FALSE
1	lri	5.05	0.62	69	36	221	FALSE
1	lrv	11.4	1.6	33	16	387	FALSE
1	lv	5.53	0.68	69	36	232	FALSE
1	v	5.47	0.68	69	36	232	FALSE
2	i	4.1	0.7	70	36	184	FALSE
2	l	0.33	0.11	6	3	20	TRUE
2	li	4.09	0.69	70	36	184	FALSE
2	lri	6.68	1.28	44	20	268	FALSE
2	lrv	7.41	1.58	65	34	323	FALSE
2	lv	10.73	0.97	69	38	350	FALSE
2	v	10.81	0.98	69	38	350	FALSE
3	i	14.88	0.98	64	35	364	FALSE
3	l	0.19	0.09	6	3	12	TRUE
3	li	14.76	0.97	64	35	364	FALSE
3	lri	6.47	1.3	64	32	249	FALSE
3	lrv	3.86	0.65	55	29	179	FALSE
3	lv	12.63	0.84	61	34	336	FALSE
3	v	12.61	0.84	61	34	336	FALSE
4	i	8.5	0.93	61	32	292	FALSE
4	l	0.39	0.17	12	6	24	TRUE
4	li	8.6	1	61	32	292	FALSE
4	lri	6.42	1.11	65	31	276	FALSE
4	lrv	8.44	1.66	45	22	351	FALSE
4	lv	6.25	0.78	73	40	270	FALSE
4	v	6.19	0.77	73	40	270	FALSE
5	i	50.72	1.27	64	35	746	FALSE
5	l	0.27	0.1	6	3	19	TRUE
5	li	51.3	1.3	64	35	746	FALSE
5	lri	7.06	1.33	55	26	293	FALSE
5	lrv	8.36	1.42	64	34	331	FALSE
5	lv	15.13	1.01	66	35	369	FALSE
5	v	15.22	1.01	66	35	369	FALSE

Table E.42: Counts for Admissible Order Comparison with Problem MT4.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	2.05	0.32	38	15	91	FALSE
1	l	2.02	0.48	56	21	114	FALSE
1	li	1.93	0.3	38	15	91	FALSE
1	lri	2.32	0.44	62	23	107	FALSE
1	lrv	1.82	0.39	40	15	89	FALSE
1	lv	2.32	0.41	57	20	91	FALSE
1	v	2.33	0.41	57	20	91	FALSE
2	i	2.86	0.45	66	28	146	FALSE
2	l	1.69	0.45	44	17	100	FALSE
2	li	2.84	0.45	66	28	146	FALSE
2	lri	1.54	0.34	50	17	39	FALSE
2	lrv	1.98	0.4	48	20	91	FALSE
2	lv	2.53	0.4	46	18	128	FALSE
2	v	2.52	0.41	46	18	128	FALSE
3	i	2.46	0.36	48	23	114	FALSE
3	l	2.01	0.52	63	23	94	FALSE
3	li	2.17	0.33	48	23	114	FALSE
3	lri	1.87	0.53	50	18	82	FALSE
3	lrv	2.77	0.41	37	17	158	FALSE
3	lv	2.33	0.48	69	27	74	FALSE
3	v	2.34	0.48	69	27	74	FALSE
4	i	2.15	0.37	57	28	111	FALSE
4	l	0.63	0.17	16	7	42	FALSE
4	li	2.15	0.37	57	28	111	FALSE
4	lri	2.22	0.48	56	23	82	FALSE
4	lrv	1.25	0.3	48	15	35	FALSE
4	lv	3.46	0.46	54	24	159	FALSE
4	v	3.14	0.42	54	24	159	FALSE
5	i	1.84	0.34	49	22	75	FALSE
5	l	1.16	0.29	25	12	86	FALSE
5	li	1.84	0.34	49	22	75	FALSE
5	lri	2.97	0.48	49	21	148	FALSE
5	lrv	2.15	0.36	47	21	92	FALSE
5	lv	2.54	0.46	64	24	91	FALSE
5	v	2.56	0.45	64	24	91	FALSE

Table E.43: Counts for Admissible Order Comparison with Problem MTB.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	4.02	0.91	60	36	165	TRUE
1	l	4.63	1.18	60	36	254	TRUE
1	li	4	0.91	60	36	165	TRUE
1	lri	5.03	1.23	60	36	254	TRUE
1	lrv	4.68	1.26	60	36	236	TRUE
1	lv	3.91	0.91	60	36	158	TRUE
1	v	3.91	0.9	60	36	158	TRUE
2	i	1.04	0.52	41	16	16	FALSE
2	l	0.44	0.4	18	5	7	TRUE
2	li	1.12	0.54	41	16	16	FALSE
2	lri	1.79	0.64	46	24	57	FALSE
2	lrv	1.73	0.65	46	22	58	TRUE
2	lv	0.91	0.43	26	12	29	TRUE
2	v	0.9	0.43	26	12	29	TRUE
3	i	2.36	0.73	78	31	55	FALSE
3	l	0.29	0.27	13	3	3	TRUE
3	li	2.36	0.73	78	31	55	FALSE
3	lri	0.21	0.22	11	3	5	TRUE
3	lrv	0.37	0.27	16	6	10	TRUE
3	lv	1.27	0.61	46	19	34	FALSE
3	v	1.37	0.67	46	19	34	FALSE
4	i	3.28	0.73	48	28	138	TRUE
4	l	0.18	0.24	9	1	1	TRUE
4	li	3.12	0.7	48	28	138	TRUE
4	lri	2.99	0.75	50	26	113	TRUE
4	lrv	2.33	0.71	67	37	82	FALSE
4	lv	0.6	0.39	24	10	22	TRUE
4	v	0.61	0.39	24	10	22	TRUE
5	i	0.13	0.19	8	1	1	TRUE
5	l	4.57	1.33	72	34	230	FALSE
5	li	0.14	0.18	8	1	1	TRUE
5	lri	4.49	1.16	90	38	117	FALSE
5	lrv	0.2	0.23	11	2	3	TRUE
5	lv	3.32	0.89	83	35	95	FALSE
5	v	3.31	0.89	83	35	95	FALSE

Table E.44: Counts for Admissible Order Comparison with Problem MTRI.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	161.53	1.78	29	16	1454	FALSE
1	l	0.68	0.22	18	9	45	TRUE
1	li	161.55	1.77	29	16	1454	FALSE
1	lri	7.34	0.76	69	36	279	FALSE
1	lrv	10.26	1.23	23	12	313	FALSE
1	lv	6.28	0.81	69	36	258	FALSE
1	v	6.24	0.82	69	36	258	FALSE
2	i	43.76	1.46	55	27	626	FALSE
2	l	0.44	0.16	12	6	28	TRUE
2	li	43.78	1.47	55	27	626	FALSE
2	lri	13.42	1.92	69	33	380	FALSE
2	lrv	15.94	2.54	37	17	493	FALSE
2	lv	7.01	0.96	71	37	266	FALSE
2	v	6.96	0.95	71	37	266	FALSE
3	i	26.45	1.49	54	28	638	FALSE
3	l	0.26	0.15	10	5	17	TRUE
3	li	26.45	1.47	54	28	638	FALSE
3	lri	14.39	1.45	55	29	377	FALSE
3	lrv	9.2	1.54	64	31	332	FALSE
3	lv	9.22	1.26	40	16	285	FALSE
3	v	9.16	1.24	40	16	285	FALSE
4	i	35.57	1.21	60	33	585	FALSE
4	l	0.23	0.11	6	3	15	TRUE
4	li	35.69	1.24	60	33	585	FALSE
4	lri	8.79	2.03	67	32	349	FALSE
4	lrv	6.18	1.51	62	29	233	FALSE
4	lv	20.18	1.5	56	28	495	FALSE
4	v	20.16	1.51	56	28	495	FALSE
5	i	15.5	1.23	76	41	362	FALSE
5	l	0.33	0.16	10	5	21	TRUE
5	li	15.66	1.27	76	41	362	FALSE
5	lri	11.84	2	41	21	388	FALSE
5	lrv	13.17	1.13	39	21	345	FALSE
5	lv	6.78	1.1	59	31	245	FALSE
5	v	6.76	1.1	59	31	245	FALSE

Table E.45: Counts for Admissible Order Comparison with Problem P4.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	0.1	0.01	6	5	9	TRUE
1	l	0.1	0.01	6	5	14	TRUE
1	li	0.09	0.02	6	5	9	TRUE
1	lri	0.1	0.02	6	5	14	TRUE
1	lrv	0.1	0.01	6	5	9	TRUE
1	lv	0.09	0.02	6	5	14	TRUE
1	v	0.12	0.01	6	5	19	TRUE
2	i	0.12	0.01	6	5	19	TRUE
2	l	0.09	0.02	6	5	9	TRUE
2	li	0.1	0.01	6	5	14	TRUE
2	lri	0.09	0.02	6	5	9	TRUE
2	lrv	0.09	0.02	6	5	14	TRUE
2	lv	0.09	0.02	6	5	9	TRUE
2	v	0.09	0.02	6	5	9	TRUE

Table E.46: Counts for Admissible Order Comparison with Problem P5.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	7.13	0.89	122	75	284	TRUE
1	l	2.82	0.46	69	43	182	TRUE
1	li	6.71	0.85	122	75	284	TRUE
1	lri	7.02	0.81	130	79	216	TRUE
1	lrv	7.02	0.83	130	79	216	TRUE
1	lv	6.66	0.91	122	75	296	TRUE
1	v	6.69	0.91	122	75	296	TRUE
2	i	7.53	0.93	130	79	280	TRUE
2	l	2.81	0.44	65	41	192	TRUE
2	li	7.52	0.93	130	79	280	TRUE
2	lri	6.74	0.81	128	78	220	TRUE
2	lrv	5.87	0.76	130	82	199	TRUE
2	lv	5.89	0.83	135	87	270	TRUE
2	v	5.9	0.85	135	87	270	TRUE
3	i	6.15	0.76	130	83	260	TRUE
3	l	2.82	0.44	65	41	192	TRUE
3	li	6.09	0.76	130	83	260	TRUE
3	lri	6.92	0.87	136	83	167	TRUE
3	lrv	7.16	0.86	135	82	191	TRUE
3	lv	6.89	0.89	149	95	244	TRUE
3	v	6.91	0.89	149	95	244	TRUE
4	i	6.88	0.89	149	95	248	TRUE
4	l	3.05	0.5	69	43	182	TRUE
4	li	6.8	0.88	149	95	248	TRUE
4	lri	6.91	0.83	133	81	188	TRUE
4	lrv	7.01	0.83	130	79	218	TRUE
4	lv	6.9	0.91	122	75	320	TRUE
4	v	7.18	0.94	122	75	320	TRUE
5	i	6.96	0.89	149	95	248	TRUE
5	l	3.02	0.5	69	43	182	TRUE
5	li	6.91	0.89	149	95	248	TRUE
5	lri	7.05	0.86	136	83	171	TRUE
5	lrv	7.12	0.85	133	81	175	TRUE
5	lv	6.1	0.76	130	83	260	TRUE
5	v	6.34	0.78	130	83	260	TRUE

Table E.47: Counts for Admissible Order Comparison with Problem P6.

Alpha. Order	Admiss. Order	Comp. Time	Reduction Time	Total Reductions	Zero Reductions	Simple Reductions	Finite
1	i	223.76	2.81	42	6	1395	TRUE
1	l	2.43	0.19	24	10	108	TRUE
1	li	224.63	2.82	42	6	1395	TRUE
1	lri	1.89	0.14	27	6	64	TRUE
1	lrv	22.14	1.68	48	4	400	TRUE
1	lv	2.49	0.17	38	6	77	TRUE
1	v	2.52	0.16	38	6	77	TRUE
2	i	0.5	0.11	8	0	22	TRUE
2	l	254.77	1.65	30	5	814	TRUE
2	li	0.5	0.11	8	0	22	TRUE
2	lri	14.24	3.62	22	2	244	TRUE
2	lrv	0.36	0.07	6	2	15	TRUE
2	lv	78.31	4.58	30	2	704	TRUE
2	v	78.03	4.57	30	2	704	TRUE
3	i	46.15	3.74	26	2	596	TRUE
3	l	0.38	0.15	4	2	30	TRUE
3	li	46.3	3.76	26	2	596	TRUE
3	lri	0.36	0.08	6	2	15	TRUE
3	lrv	7.07	4.16	25	3	187	TRUE
3	lv	0.54	0.12	8	1	23	TRUE
3	v	0.54	0.11	8	1	23	TRUE
4	i	6.84	0.31	52	8	142	TRUE
4	l	0.72	0.14	8	0	19	TRUE
4	li	6.89	0.31	52	8	142	TRUE
4	lri	13.11	0.61	30	2	281	TRUE
4	lrv	4.24	0.41	49	5	105	TRUE
4	lv	1.96	0.34	17	1	63	TRUE
4	v	1.91	0.34	17	1	63	TRUE
5	i	3.53	0.21	41	7	112	TRUE
5	l	27.09	0.89	33	6	487	TRUE
5	li	3.54	0.21	41	7	112	TRUE
5	lri	15.52	1.58	45	4	284	TRUE
5	lrv	2.39	0.17	35	6	78	TRUE
5	lv	250.81	3.11	42	5	1237	TRUE
5	v	249.71	3.08	42	5	1237	TRUE

VITA

Benjamin J. Keller was born in Iowa City, Iowa on July 6, 1964. Mr. Keller received his Bachelor of Science degree in Computer Science in December of 1986 from Western Kentucky University. In September of 1990, Mr. Keller completed his Master of Science in Computer Science at Virginia Tech by defending his thesis *An Algebraic Model of Software Evolution*. After a semester at the University of Iowa, Mr. Keller returned in the Spring of 1991 to Virginia Tech to finish his Ph.D. studies. Mr. Keller has joined the faculty of Montana Tech of the University of Montana in Butte, Montana.