Alternatives in Implementing Noncommutative Gröbner Basis Systems

(Submitted to Proceedings of Workshop on Symbolic Rewriting Systems)

Benjamin J. Keller
Department of Computer Science
VPI&SU, Blacksburg, VA 24061, USA
keller@cs.vt.edu

30 January 1996

Abstract

Alternatives in implementing systems for computing Gröbner bases in path algebras (noncommutative polynomial rings) are considered and compared. Adapted forms of the standard variations to the Buchberger's algorithm (for commutative polynomial rings) are discussed, as is a pattern matching approach that finds the divisors and common multiples among the leading terms of a set of polynomials.

Results from preliminary experimentation with a prototype system are used to compare the different configurations of two variations (triple elimination and basis reduction). Eight problem instances split between two classes of problems (one over free algebras, the other over mesh algebras) are used to compare the configurations. An informal analysis suggests that order plays a larger role in determining the execution time for a problem instance than the algorithm. However, by comparing configurations for each of the admissible orders, some observations can be made about the algorithms.

1 Introduction

Since Mora introduced noncommutative Gröbner bases in 1986 [M86], few systems for computing them have been developed. Only two such systems are widely distributed: Bergman by Backelin (see description by Roos [R94]), and GRB by Feustel and Green. While recently more systems have been written

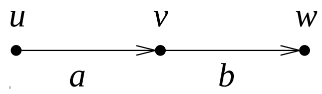

Figure 1: Example Graph

and even some commercial computer algebra systems will soon support noncommutative Gröbner bases, little is (generally) known about how to efficiently compute finite noncommutative Gröbner bases or finite subsets of infinite Gröbner bases. Fortunately, most of the improvements to Buchberger's algorithm for commutative Gröbner bases transfer well to the noncommutative case. Here we look at a number of these alternative improvements in an effort to begin to understand how to efficiently compute noncommutative Gröbner bases.

The paper begins by defining noncommutative polynomial rings as path algebras in the second section, and then reviews Buchberger's algorithm for the noncommutative case in the third section. Section 4 discusses alternative algorithms and data structures for implementing Buchberger's algorithm, and section 5 outlines a family of systems built to compare some of these alternatives experimentally. Finally, the results of preliminary experiments are presented in section 6.

2 Noncommutative Polynomial Rings

The noncommutative polynomial rings we consider are path algebras, which are quotients of free algebras that can be conveniently described in terms of a graph. Let K be a field. A path algebra $K\Gamma$ consists of K-linear combinations of finite paths in a directed multigraph Γ (called the *quiver*).

Let $\Gamma = (\Gamma_0, \Gamma_1)$ be a finite directed (multi)graph where $\Gamma_0 = \{v_1, \ldots, v_n\}$ is the set of *vertices*, and $\Gamma_1 = \{a_1, \ldots, a_m\}$ is the set of *arcs*. Let B be the set of finite directed paths in Γ, including the vertices as 0-length paths. The set $B \cup \{0\}$ is a monoid with zero under the product $p \cdot q$ of paths $p, q \in B$. The product $p \cdot q$ is the concatenation pq if the source vertex of q is the target vertex of p, and zero otherwise. The vertices are idempotents with respect to the product operation: $v \cdot v = v$ for $v \in \Gamma_0$. Also, vertices are absorbed when multiplied with a path for which the vertex is a source or target of the path ($v \cdot p = p$ or $p \cdot v = p$). Otherwise, a vertex acts as a zero.

The set of finite paths for the graph Γ in Figure 1 is $B = \{u, v, w, a, b, ab\}$. The nonzero products in B are $a \cdot b = ab$, $u \cdot a = a \cdot v = a$, $v \cdot b = b \cdot w = b$, $u \cdot u = u$, $v \cdot v = v$, and $w \cdot w = w$. All remaining products are zero, notably $b \cdot a = 0$.

For a field K and graph Γ, the path algebra $K\Gamma$ is the collection of finite K-linear combinations of

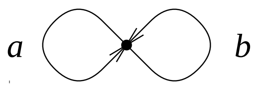

Figure 2: The Quiver for a Free Algebra

directed paths

$$f = \sum_{\beta \in B} k_\beta \beta$$

where $k_\beta \in K$ for $\beta \in B$. ($K\Gamma$ is the K-vector space with basis B.) $K\Gamma$ is a polynomial ring with unit the sum of vertices $\sum_{v \in \Gamma_0} v$. The elements of $K\Gamma$ are called *polynomials*. The set of terms (paths) for all nonzero monomials occurring in a polynomial f is called the *support* of f (denoted $\mathrm{supp}(f)$).

Path algebras are quotients of free associative algebras, and so free algebras are the special case where the quotient is the algebra itself. A free algebra $K\langle a, b\rangle$ corresponds to a path algebra $K\Gamma$ where Γ is the graph shown in Figure 2. For this graph the set of finite directed paths is (isomorphic to) the Kleene closure of the alphabet $\{a, b\}$ (the vertex plays the role of the empty string).

Generally a path algebra is used in place of certain quotients of a free algebra. There is a significant advantage to using path algebras in this situation, because the quotient relations are represented by the graph and so the quotient relations do not have to be considered in a Gröbner basis computation.

For two non-zero elements p and q of B, p is *uniform-equivalent* to q if for all $a, b \in B$

$$apb = 0 \iff aqb = 0.$$

So two paths are uniform-equivalent if the source vertex of p equals the source vertex of q and the target vertex of p equals the target vertex of q. All paths are uniform-equivalent in a free algebra where the graph only has one vertex. A polynomial in $K\Gamma$ is *uniform* if all the paths in its support are uniform-equivalent.

An arbitrary polynomial f in a path algebra can be decomposed into a sum of uniform polynomials $\{f_{u,v} : u, v \in \Gamma_0\}$ called the *uniform projections* of f. The decomposition is

$$f = \sum_{u,v \in \Gamma_0} ufv$$

where the composition using the pairs of vertices u and v identifies the separate projections $f_{u,v}$. A set of polynomials is *uniform-homogeneous* if whenever a nonzero element is in the set so are its uniform projections. Since each ideal is closed under multiplication by elements of the path algebra, all of the uniform projections of nonuniform elements of the ideal are elements of the ideal. Therefore, the ideals of a path algebra are uniform-homogeneous, and it is sufficient to consider only uniform generators.

3 Noncommutative Gröbner Bases

An admissible order $<$ on the set of finite paths is assumed. The *tip* of p is the maximal element of $\mathrm{supp}(p)$ with respect to the order $<$.

The algorithm for noncommutative Gröbner bases calculates *overlap-relations* for *triples* consisting of a pair of polynomials and an overlap of their tips. For a triple $t = \langle p, q, v \rangle$ the overlap relation is

$$o(p, q, v) = \frac{1}{\alpha} \cdot pb - \frac{1}{\beta} \cdot aq$$

where $tip_<(p) = av$, $tip_<(q) = vb$, and α and β are the leading coefficients of p and q respectively. The overlap determines a common multiple $\mathrm{cm}(t)$ of the tips avb that is canceled in the overlap relation. The algorithm must consider triples for all pairs of polynomials including a polynomial with itself.

In order for the computation to work properly, the initial generators must be uniform polynomials. For an arbitrary set of generators there is an equivalent set of uniform polynomials that generates the same ideal [FFG93]. Therefore, the set can be saturated using uniform projection before the Gröbner basis is computed.

The basic algorithm for noncommutative Gröbner bases as shown in Figure 3 is essentially the same as the Buchberger algorithm for the commutative case.

The input to the algorithm is the generating set (and an order, which is not shown). The INITIALIZE function both tip-reduces the elements of G, and builds the set T of triples for the tip-reduced generators. (Initialize iterates through the original set and calls the UPDATE function for each of the initial generators.) Within the loop the algorithm selects a triple and computes its reduced overlap relation. The reduced overlap relation is inserted into the working basis if it is nonzero. The loop continues until the triple set is empty, at which point the basis is totally reduced (in REDUCE_BASIS).

The algorithm shown in Figure 3 terminates only when the input instances has a finite Gröbner basis. For problem instances with infinite Gröbner bases the algorithm must use a modified termination test. When the generators are homogeneous, the termination test is the same, but a maximum degree bound is added as a parameter. In this case, a polynomial is only added to the working basis (and new triples formed) if its tip is shorter than the bound. If the generators are not homogeneous, a bound on the number of nonzero polynomial reductions is used, and the algorithm terminates when the bound is

GRÖBNER(G). Buchberger algorithm for instances with finite Gröbner bases.

INPUT: A set of generators G.
OUTPUT: A finite, totally reduced Gröbner basis for G.

```
▷ tip-reduce G and build triple set T
INITIALIZE(G, T);
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);
```

Figure 3: Buchberger's Algorithm

reached (or the triple set is empty). For both of these approaches the algorithm will eventually terminate with either a finite Gröbner basis or a finite subset of the basis.

4 Implementation Alternatives

The algorithm described above (Figure 3) leaves many implementation details unspecified. Many of these details are algorithmic, such as how the functions SELECT, UPDATE, and REDUCE_BASIS are performed; but other details such as data structures are also left open. Some of the alternatives for the algorithmic details and data structures are considered below.

4.1 Algorithms

As has been discovered in the setting of commutative polynomial rings, there are several ways to vary the basic Buchberger algorithm. These are the selection strategy (the SELECT function in Figure 3), 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.

4.1.1 Selection Strategy

A *selection strategy* is a method for selecting a triple for reduction. The role of a selection strategy is to avoid selection of pairs that reduce to zero and to support useless pair elimination. Most selection strategies use an order on the terms, but the order does not need to be the admissible order used for the polynomials (in reduction). If the polynomial and selection orders are the same, the strategy is the *normal* selection strategy.

4.1.2 Polynomial Reduction

Polynomial reduction is one of the major operations of the Gröbner basis computation. There are essentially two forms. The first is *total* reduction which finds the remainder of dividing a polynomial by all elements of a set of polynomials. Total division finds a normal form of the polynomial p where no tip of any of the elements of the set can divide any term of p. The second form of reduction is *tip* reduction, which finds a normal form where the tip of p is not divisible by the tip of a set element. A hybrid of these two approaches is to totally reduce the polynomial if it is tip reducible.

However, it seems that reduction beyond the tip does not contribute to the basis computation until the reduced basis is found at the end of the computation. And, in fact, total reduction could hurt the computation since in the worst case it makes the polynomials longer. The implementation described below uses a hybrid between tip reduction and total reduction (as suggested by some comments of Giovini et al [GMNRT91]).

4.1.3 Basis Reduction

Sometimes during a Gröbner basis computation a new polynomial is computed that has a tip that divides the tip of some element of the working basis. Since the goal is a reduced basis where no tip divides another, it is better to remove the reducible element as soon as it is detected. Also, removing a reducible element helps ensure that it does not contribute to any new overlap computations.

The approach described by Gebauer and Möller [GM89] is called *redundant element deletion*. In this approach, a reducible element is removed from the working basis but any overlaps containing this element are kept. In this way, no new overlaps of the tip of the reducible element are found, but old overlaps which have not been considered are not lost. Another approach to basis reduction (which we call *element-reduction*) is to (tip-) reduce any reducible elements when a new polynomial is added. In this case, the set of triples must be updated to match the reduced form of the basis elements (the overlaps for the old tips are deleted and the new ones added).

It is not obvious which approach is best. The cost of redundant element deletion is that the reducible element must still be kept until all its overlaps are tested. It also complicates the implementation of the Gebauer-Möller elimination criteria, since for at least the elimination of new overlaps the basis is not reduced (and so equality tests must be performed). For element-reduction, the cost seems to be higher since the redundant elements are reduced rather than just deleted, and the old overlaps must be deleted and new ones found. However, if the reduced elements contribute to fewer (e.g., zero) overlap relations, the element-reduction strategy may be better than redundant element deletion. Which of these strategies is better is one of the major questions for experimentation.

4.1.4 Triple Elimination

The elimination of *useless pairs* (pairs for which the s-polynomial reduces to zero) is first considered by Buchberger [B79]. Buchberger defines several criteria that can be used to determine when a pair of polynomials can be eliminated. Gebauer and Möller [GM89] later derived similar criteria using syzygies, and Mora [M94] shows (using syzygies) that these criteria can be applied in computing noncommutative

Gröbner bases. (This result can also be proved very easily with an approach like that in the discussion by Becker and Weispfenning [BW93]. This proof approach gives more intuition for implementation.)

There are basically two strategies for using these criteria to eliminate useless pairs: that described by Buchberger and that described by Gebauer and Möller. The Buchberger strategy is to delay testing a pair for elimination until the pair is selected for reduction. The Gebauer-Möller strategy is to perform eliminations as soon as possible, which is when a new element is introduced to the Gröbner basis during a computation. (We refer to these as *lazy* and *eager* elimination strategies respectively.)

Elimination in the noncommutative case only uses criteria like the following: a triple t_1 can be eliminated if there is another triple t_2 whose common multiple $\mathrm{cm}(t_2)$ divides $\mathrm{cm}(t_1)$. Keeping the basis elements reduced helps simplify the tests for the criteria since none of the equality tests (primarily for the Gebauer-Möller criteria) need to be performed.

4.2 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).

4.2.1 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. The new overlaps are of the tip of h with the tips of the basis elements. 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 pattern, the tip set of the current generating set (basis).

Pattern matching using a set of patterns is called dictionary matching [AC75, AF91]. The original algorithm by Aho and Corasick [AC75] is for a static set (the dictionary) of patterns and uses an automata constructed from the dictionary set. The data structures used in this algorithm do not allow easy modifications to the automata, and so the addition or deletion of a pattern entails rebuilding the automata. Therefore, the time to make any modifications to the set is linear in the size of the dictionary.

Dynamic dictionary matching allows the dictionary to be changed. Amir and Farach [AF91] develop an algorithm for dynamic dictionary matching using suffix trees as the automata. Suffix trees are compacted tries (an edge may be labeled by arbitrary length strings) for the suffixes of a string. In dictionary matching the suffix tree holds the suffixes for the individual dictionary patterns (the tree is built as if for the string formed by concatenating the strings of the dictionary with each string terminated by a unique symbol). If one pattern may be a subword of another, then an additional data structure is required to keep track of these subword relationships. In both cases, the time required for modifications to the dictionary is independent of the dictionary size (basically linear in the length of the inserted or deleted pattern if subwords are allowed [AFGGP93]).

The dynamic dictionary matching scheme can be modified to handle the pattern matching problems of the noncommutative Gröbner basis computation. The basis elements are kept tip-reduced so that only the suffix tree is needed to implement the pattern matcher (and operations take only linear time). The suffix tree is built so that leaf nodes of the suffix tree corresponding to full patterns (tips) are distinguished from leaves that correspond to nontrivial suffixes of a pattern. These leaves are called *pattern* leaves and *suffix* leaves respectively.

The difference between the types of leaves comes out for the different searches. Consider, for example, the division search and the left-overlap search, which are much alike. In the left-overlap search, the input path t is scanned using the tree and if any suffix leaf is found then the corresponding suffix forms a left-overlap. For the division search *all* suffixes of t are scanned and any pattern leaf corresponds to a divisor path. The suffix and pattern leaves are used in a similar fashion for the other two searches.

An alternative to the dynamic dictionary matching approach is to use a tabular data structure for the automata like that described by Sims [S94]. But this structure is basically that used in the (static) dictionary matching approach of Aho and Corasick [AC75]. Since most Gröbner basis computations add many new polynomials to the generating set the dynamic suffix tree structure is needed. (A tabular data structure might be better for a fixed set of polynomials as in polynomial reduction.)

4.3 Triple Sets

The key to efficient Gröbner basis computation seems to be 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 exponential in the number of generators).

The triple set data structure needs to support adding and deleting elements as well as selecting 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 a list sorted in the order of the strategy ensures that selection is constant time. The problem then is that the insertion and deletion operations must maintain the sorted order.

There is a time-space tradeoff with respect to keeping or computing the common multiples for triples. The common multiple is needed for sorting the triples and is also needed in triple elimination (described above).

5 An Implementation

A prototype system has been written in Standard ML to study the interactions between the algorithmic variations described above. The system is actually a family of different prototypes that implement six different configurations of the algorithmic variations (each configuration are built by compilation). The six configurations are combinations of the two approaches to basis reduction (eager, redundant element deletion) and three approaches to triple elimination (none, Buchberger strategy, Gebauer-Möller).

The system is built using the SML module facilities of signatures, structures, and functors. The relationships between the primary functors and structures of the system are shown in Figure 4. (The system consists of many other structures, some of which are implementations of data structures from the SML/NJ library.) In the figure, the square boxes represent the functors and the rounded boxes the structures created using the functors with arguments the structures connected by the incoming arrows.

The different configurations of the system are primarily determined by the "Buchberger" functor which has one form for each configuration (so there are six files containing the different versions of the functor). Most of the differences are localized to the `update` function which handles inserting a nonzero element into the generating set, along with keeping the set tip-reduced, and adding triples to the triple set (and for the Gebauer-Möller strategy, eliminating triples). The only other structure that varies for the different configurations is the polynomial set structure, which is different for the redundant element

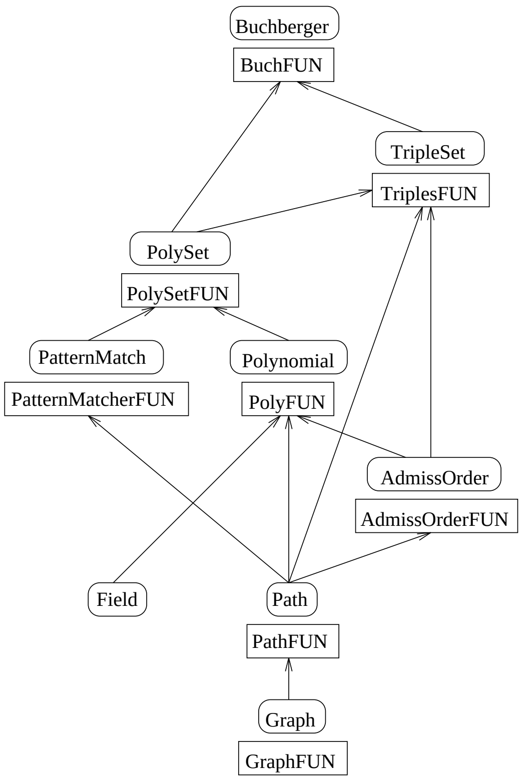

Figure 4: Structure of the NGRB Prototype.

deletion approach to basis reduction.

Triple sets are implemented as a doubly linked list (using SML references) sorted in the order used by the selection strategy. All the triples for a pair of polynomials are stored as a tuple consisting of references to the two polynomials and a list of overlap lengths (sorted longest to shortest). 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 in tests for triple elimination and for deleting triples when a polynomial is reduced during basis reduction (not done in redundant element deletion).

The 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 tips 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. The polynomials are kept sorted by the admissible order. Paths are also implemented by an SML datatype 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 (currently fixed to be 32117, although the structure allows the characteristic to be changed).

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). Actually, only the function to build lexicographic orders takes an alphabetic order as an argument, the others (length, weight, dualization, vector lexicographic, and inverse vector lexicographic) take orders on paths as arguments and produce orders on paths. So, a lexicographic order would be built first and then an admissible order could be built by using one of the other orders (of course, not all compositions of these functions produce admissible orders).

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 4, 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.

6 Experimentation

The six configurations of the prototype provide an opportunity to compare some of the algorithmic variations for noncommutative problems. Some preliminary experiments have been run and are discussed here.

Each of the prototype configurations were built using the SML of New Jersey compiler (version 0.93) on an IBM RS/6000 model 530H running AIX 3.2. The different configurations were run on several problem instances and the run times collected. The observations from eight of these problem instances are discussed here.

The eight problem instances are divided between five instances in a free algebra, and three instances from mesh algebras, both of which are expressed as path algebras. The free algebra instances are size k subsets (taken as prefixes of the list, for $k = 4, \ldots, 8$) of the following quadratic relations over three variables:

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

These free algebras instances are identified as IH3-k where k is the size of the generating set.

Mesh algebras are path algebras that are determined by the graph. The graph must have pairs of inverse arrows between adjacent vertices (if an arc exists, so does 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 5 are

$$\{aa' + bb' + cc', a'a, b'b, c'c\}\,.$$

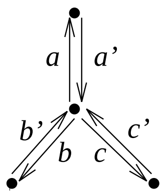

Figure 5: Example Graph for Mesh Algebra.

The mesh algebra instances used in the experiment are for binary trees with seven nodes (instance BTM7) and thirty-one nodes (instance BTM31), and for a thirty-nine node graph which is the 31 vertex tree with an additional eight vertices (each of which is linked to one of the eight pairs of leaves).

All of these problem instances are homogeneous and so were run with a degree bound. The smaller mesh algebra instance (BTM7) was run with a degree bound of 10, and all the others were run with a bound of 5.

Eight admissible orders were also used. These are length lex, length rev lex, length inverse vector lex, length vector lex, length reverse inverse vector lex (the analogue of the "optimal" order for the commutative case), length reverse vector lex, vector lex, and inverse vector lex.

The experiment was run by using each configuration to find the partial Gröbner basis for each problem instance for the eight orders. During each run the prototype was the only user process running. No replication was used (other experiments indicate low variation of the execution time for replications under these conditions).

Informal analysis is used. In general, the observations indicate that order is responsible for more variation than the algorithms. This is easy to see in the results for the free algebra instance with 5 relations (Table 2). Table 2 is sorted by execution time. The first thirty-six entries correspond to the length orders used; with a 7 second difference between the execution time for slowest length order run and the fastest non-length order run. This pattern also appears in the results for the instances with four and six relations but is less apparent in the other free instances.

While the trend is not so strong for the mesh algebra instances, order also appears to be a dominant factor. However, for the mesh algebra instances the length orders do not dominate the others. Table 1 shows the results for the BTM31 instance. A possible explanation for the scrambling of the orders might be that the relationship between the underlying graph and the orders plays a role in determining which order is better. This is suggested by the fact that the length reverse lexicographic and inverse vector lexicographic orders dominate the others. Although, it is important to note that for the smaller

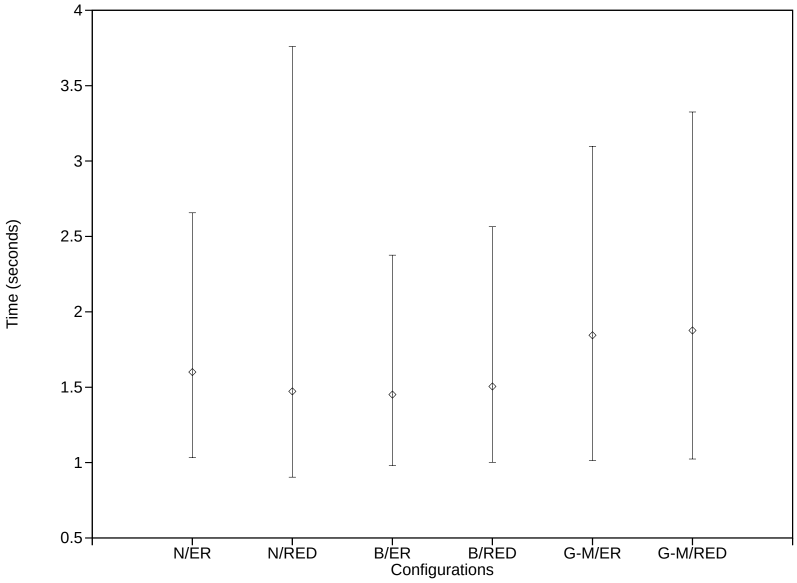

Figure 6: Median and Range of observations for IH3-6

BTM7 instance the opposite pattern occurs (the length lexicographic order dominates the length reverse lexicographic order).

To compare the algorithms, the results are presented by problem instance. Table 3 and Table 4 shows the median and range (as high and low values) for each problem instance and configuration. Graphs of these values for IH3-6 (the free instance with 6 relations) and BTM31 are shown in Figure 6 and Figure 7. (In Figure 7 and Figure 6 the configurations are indicated by a symbol of the form *elimination technique/basis reduction technique*, where the elimination techniques are "N" for none, "B" for Buchberger, and "G-M" for Gebauer-Möller; and the reduction techniques are "RED" for redundant element deletion, and "ER" for element reduction.) Both graphs suggest that lazy elimination with the Buchberger criteria bounds the execution time better than the other forms, but this is about all that can be said (see the conclusions for a discussion of the implementation of the Gebauer-Möller elimination strategy). Another observation that might be made (see Figure 6) is that element-reduction helps reduce maximum run times, but this is not generally true (see Figure 7).

Table 1: Observations for BTM31

Elimination	Reduction	Order	Total Time
None	Red. Elem. Del.	length rev lex	0.611176
Buchberger	Red. Elem. Del.	length rev lex	0.616691
Buchberger	Elem. Red.	length rev lex	0.641583
None	Elem. Red.	length rev lex	0.703145
Gebauer-Möller	Elem. Red.	length rev lex	0.785195
Gebauer-Möller	Red. Elem. Del.	length rev lex	0.789362
Buchberger	Elem. Red.	invvec lex	5.823765
Buchberger	Red. Elem. Del.	invvec lex	5.958291
Buchberger	Elem. Red.	length rev invvec lex	6.786294
Buchberger	Elem. Red.	length lex	7.164073
Buchberger	Red. Elem. Del.	length lex	7.167553
None	Red. Elem. Del.	length lex	7.282425
Buchberger	Red. Elem. Del.	length rev invvec lex	7.685723
None	Elem. Red.	length lex	8.091098
Buchberger	Elem. Red.	length rev vec lex	8.715699
Buchberger	Elem. Red.	length invvec lex	8.977622
Buchberger	Red. Elem. Del.	length rev vec lex	9.196928
Buchberger	Red. Elem. Del.	length invvec lex	9.487765
Gebauer-Möller	Elem. Red.	inv vec lex	11.076892
Gebauer-Möller	Red. Elem. Del.	inv vec lex	11.492880
Gebauer-Möller	Elem. Red.	length inv vec rev lex	12.102485
Gebauer-Möller	Red. Elem. Del.	length inv vec rev lex	12.181052
Buchberger	Elem. Red.	vec lex	13.478362
Buchberger	Elem. Red.	length vec lex	13.619357
None	Red. Elem. Del.	length rev invvec lex	14.893701
None	Red. Elem. Del.	invvec lex	14.984867
Buchberger	Red. Elem. Del.	vec lex	15.605251
None	Elem. Red.	length rev invvec lex	15.730259
Buchberger	Red. Elem. Del.	length vec lex	15.784493
None	Elem. Red.	invvec lex	16.426852
None	Red. Elem. Del.	length rev vec lex	17.942698
None	Red. Elem. Del.	length invvec lex	18.287530
None	Elem. Red.	length rev vec lex	19.408382
None	Elem. Red.	length invvec lex	19.799269
Gebauer-Möller	Elem. Red.	length lex	32.736332
Gebauer-Möller	Red. Elem. Del.	length lex	34.700461
Gebauer-Möller	Elem. Red.	vec lex	38.299262
Gebauer-Möller	Elem. Red.	length vec lex	38.751934
Gebauer-Möller	Red. Elem. Del.	vec lex	38.955875
Gebauer-Möller	Red. Elem. Del.	length vec lex	39.251509
None	Red. Elem. Del.	vec lex	50.281610
Gebauer-Möller	Elem. Red.	length inv vec lex	50.296555
Gebauer-Möller	Elem. Red.	length rev vec lex	51.233215
None	Red. Elem. Del.	length vec lex	51.806459
Gebauer-Möller	Red. Elem. Del.	length rev vec lex	53.630812
Gebauer-Möller	Red. Elem. Del.	length inv vec lex	53.908103
None	Elem. Red.	vec lex	55.487283
None	Elem. Red.	length vec lex	57.728926

Table 2: Observations for IH3-5

Elimination	Reduction	Order	Total Time
Buchberger	Elem. Red.	length lex	1.725265
Buchberger	Red. Elem. Del.	length lex	1.765959
Buchberger	Elem. Red.	length rev lex	1.778634
Buchberger	Red. Elem. Del.	length rev lex	1.812240
Buchberger	Elem. Red.	length rev vec lex	2.012582
None	Red. Elem. Del	length lex	2.084776
Buchberger	Elem. Red.	length rev invvec lex	2.111142
Buchberger	Red. Elem. Del.	length rev vec lex	2.126380
None	Red. Elem. Del.	length rev lex	2.151160
None	Red. Elem. Del.	length rev vec lex	2.180021
Buchberger	Red. Elem. Del.	length rev invvec lex	2.185371
None	Red. Elem. Del.	length rev invvec lex	2.267101
Gebauer-Möller	Red. Elem. Del.	length lex	2.267254
Gebauer-Möller	Elem. Red.	length lex	2.279790
Gebauer-Möller	Elem. Red.	length rev lex	2.305130
Gebauer-Möller	Red. Elem. Del.	length rev lex	2.337828
None	Elem. Red.	length rev vec lex	2.370467
None	Elem. Red.	length lex	2.376715
Buchberger	Elem. Red.	length vec lex	2.385972
Buchberger	Elem. Red.	length invvec lex	2.388121
None	Elem. Red.	length rev lex	2.420498
None	Elem. Red.	length rev invvec lex	2.451320
Buchberger	Red. Elem. Del.	length invvec lex	2.452360
Buchberger	Red. Elem. Del.	length vec lex	2.457401
Gebauer-Möller	Elem. Red.	length rev vec lex	2.686846
Gebauer-Möller	Red. Elem. Del.	length rev vec lex	2.696631
Gebauer-Möller	Elem. Red.	length rev invvec lex	2.781055
Gebauer-Möller	Red. Elem. Del.	length rev invvec lex	2.816384
None	Red. Elem. Del.	length vec lex	3.141879
None	Red. Elem. Del.	length invvec lex	3.215582
Gebauer-Möller	Elem. Red.	length vec lex	3.319990
Gebauer-Möller	Elem. Red.	length invvec lex	3.335636
Gebauer-Möller	Red. Elem. Del.	length vec lex	3.335998
Gebauer-Möller	Red. Elem. Del.	length invvec lex	3.404937
None	Elem. Red.	length vec lex	3.448979
None	Elem. Red.	length invvec lex	3.460320
Buchberger	Elem. Red.	invvec lex	10.747109
Buchberger	Elem. Red.	vec lex	15.121369
Buchberger	Red. Elem. Del.	invvec lex	19.669387
Buchberger	Red. Elem. Del.	vec lex	21.406232
Gebauer-Möller	Elem. Red.	vec lex	31.051596
Gebauer-Möller	Elem. Red.	invvec lex	32.114485
Gebauer-Möller	Red. Elem. Del.	vec lex	39.209583
None	Elem. Red.	vec lex	44.186015
None	Elem. Red.	invvec lex	48.718180
Gebauer-Möller	Red. Elem. Del.	invvec lex	50.630340
None	Red. Elem. Del.	invvec lex	84.679806
None	Red. Elem. Del.	vec lex	471.390735

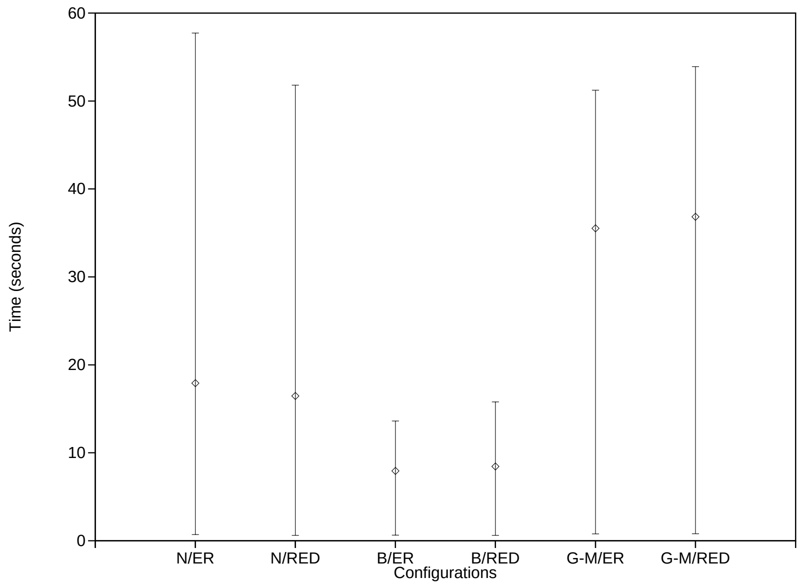

Figure 7: Median and Range of observations for BTM31

Table 3: Median and Range of Execution Times for Mesh Algebra Instances

Problem	Configuration	Minimum	Maximum	Median
btm7	Buch., Elem. Red.	0.155413	11.635136	5.1293135
btm7	Buch, Red. Elem. Del.	0.154301	22.247185	5.7059545
btm7	None, Red. Elem. Del.	0.141466	86.275077	6.26459
btm7	None, Elem. Red.	0.155473	90.045485	6.290878
btm7	G-M, Elem. Red.	0.203559	78.318407	15.1563695
btm7	G-M, Red. Elem. Del.	0.204941	112.749354	16.231269
btm31	Buch., Elem. Red.	0.641583	13.619357	7.939886
btm31	Buch, Red. Elem. Del.	0.616691	15.784493	8.4413255
btm31	None, Red. Elem. Del.	0.611176	51.806459	16.4637825
btm31	None, Elem. Red.	0.703145	57.728926	17.917617
btm31	G-M, Elem. Red.	0.785195	51.233215	35.517797
btm31	G-M, Red. Elem. Del.	0.789362	53.908103	36.828168
mm39	Buch, Red. Elem. Del.	0.423835	19.898315	0.577675
mm39	None, Red. Elem. Del.	0.454188	72.044216	0.5884625
mm39	Buch., Elem. Red.	0.453443	16.989247	0.601399
mm39	G-M, Elem. Red.	0.460756	50.201107	0.6778205
mm39	None, Elem. Red.	0.537412	80.198546	0.6800925
mm39	G-M, Red. Elem. Del.	0.467322	56.493257	0.680656

Table 4: Median and Range of Execution Times for Free Algebra Instances

Problem	Configuration	Minimum	Maximum	Median
ih3-4	Buch., Elem. Red.	4.160653	196.82224	5.0623175
ih3-4	Buch, Red. Elem. Del.	4.296054	213.544353	5.255193
ih3-4	G-M, Elem. Red.	6.150199	359.464591	7.0841845
ih3-4	G-M, Red. Elem. Del.	6.285215	438.314409	7.3798725
ih3-4	None, Red. Elem. Del.	5.889392	832.314803	7.5759955
ih3-4	None, Elem. Red.	6.383612	519.902379	8.498162
ih3-5	Buch., Elem. Red.	1.725265	15.121369	2.248557
ih3-5	Buch, Red. Elem. Del.	1.765959	21.406232	2.3188655
ih3-5	None, Red. Elem. Del.	2.084776	471.390735	2.70449
ih3-5	None, Elem. Red.	2.370467	48.71818	2.9501495
ih3-5	G-M, Elem. Red.	2.27979	32.114485	3.0505225
ih3-5	G-M, Red. Elem. Del.	2.267254	50.63034	3.076191
ih3-6	Buch., Elem. Red.	0.980669	2.375679	1.4513495
ih3-6	None, Red. Elem. Del.	0.903515	3.759574	1.4720825
ih3-6	Buch, Red. Elem. Del.	1.001914	2.564641	1.504633
ih3-6	None, Elem. Red.	1.032732	2.657063	1.600416
ih3-6	G-M, Elem. Red.	1.013729	3.097546	1.8443465
ih3-6	G-M, Red. Elem. Del.	1.023987	3.325579	1.875984
ih3-7	None, Red. Elem. Del.	0.738389	0.971269	0.9508755
ih3-7	Buch., Elem. Red.	0.801289	1.033022	1.0084515
ih3-7	None, Elem. Red.	0.873977	1.05306	1.0302285
ih3-7	Buch, Red. Elem. Del.	0.815441	1.063571	1.042638
ih3-7	G-M, Elem. Red.	0.865834	1.216853	1.1627835
ih3-7	G-M, Red. Elem. Del.	0.876751	1.228598	1.174545
ih3-8	None, Red. Elem. Del.	0.619991	0.916206	0.861669
ih3-8	Buch., Elem. Red.	0.669124	0.973505	0.9278505
ih3-8	None, Elem. Red.	0.690881	0.99897	0.948173
ih3-8	Buch, Red. Elem. Del.	0.681967	1.007736	0.9535495
ih3-8	G-M, Elem. Red.	0.760649	1.157871	1.074102
ih3-8	G-M, Red. Elem. Del.	0.766744	1.178689	1.0806105

7 Conclusions

The efficient implementation of a system for computing noncommutative Gröbner bases is a difficult task. Which configuration of alternative algorithms is best is not obvious, and analytic techniques for comparing these configurations are effectively impossible. Clearly, an experimental approach is required to compare these algorithms.

The experimental results presented above are a beginning in this comparison of algorithms. While these results do not allow a good comparison of all of the configurations (since the implementation of the Gebauer-Möller elimination strategy performs much worse than it should), they do give some indication of which choices should be made in a system. In particular, the results do indicate that triple elimination techniques can be implemented in a way so that the variation of run-times, regardless of order, can be bounded. The results also suggest that the element-reduction approach to basis reduction is a better choice than redundant element deletion (further experimentation is being conducted to determine whether element-reduction may actually aid reaching a solution in fewer steps).

The fact that the implementation of the Gebauer-Möller elimination strategy performs so poorly is an indication that the relevant operations on the triples and triple sets are not fast enough. Gebauer-Möller elimination requires being able to quickly determine which triples can be eliminated when a new element is inserted into the basis. In the prototype implementation, this search requires that the common multiple be tested for each triple in the set (as a linear search of the list). Unless the number of triples eliminated is very high, this search technique is simply too expensive. All this really indicates is that more care is needed in implementing this form of elimination.

References

[AC75] A. V. Aho and M. J. Corasick. “Efficient string matching: an aid to bibliographic search,” *Communications of the ACM* **18**, 6 (1975) 333–340.

[AF91] A. Amir and M. Farach. “Adaptive Dictionary Matching,” in *32nd Foundations of Computer Science*, IEEE Press, 1991, 760–766.

[AFGGP93] A. Amir, M. Farach, Z. Galil, R. Giancarlo, and K. Park. “Dynamic dictionary matching,” *Journal of Computer and Systems Sciences*, **49**, 2 (1994) 208–222.

[BW93] T. Becker and V. Weispfenning. *Gröbner Basis: A Computational Approach to Commutative Algebra*, Springer-Verlag, 1993.

[B79] B. Buchberger. “A criterion for detecting unnecessary reductions in the construction of Gröbner-bases,” in *EUROSAM ‘79*, E. W. Ng (ed.), LNCS# 72, Springer-Verlag, 1979, 3–21.

[FFG93] 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 (1993) 727–739.

[GM89] R. Gebauer and H. M. Möller. "On an installation of Buchberger's Algorithm", in *Computational Aspects of Commutative Algebra*, L. Robbiano (ed.), Academic Press, 1989, 141–152.

[GMNRT91] A. Giovini, T. Mora, G. Niesi, L. Robbiano and C. Traverso. " 'One sugar cube, please' or Selection strategies in the Buchberger algorithm", in *ISSAC '91*, ACM, 1991, 49–54.

[M86] F. Mora. "Groebner bases for noncommutative polynomial rings," in *Algebraic Algorithms and Error-Correcting Codes*, J. Calmet (ed.), LNCS# 229, Springer-Verlag, Berlin, 1986, 353–362.

[M94] T. Mora. "An introduction to commutative and non-commutative Gröbner bases," *Theoretical Computer Science*, **134**, 1 (1994) 131–173.

[R94] J.-E. Roos. "A computer-aided study of the graded Lie algebra of a local commutative noetherian ring," *Journal of Pure and Applied Algebra*, **91** (1994) 255–315.

[S94] C. C. Sims. *Computation with Finitely Presented Groups*, Cambridge University Press, New York, 1994.