**Greater Easy Common Divisor and standard basis completion algorithms**

**ANDRÉ GALLIGO**, **LOÏC POTTIER**, **CARLO TRAVERSO**

\[\text{†Université de Nice et INRIA – ‡Università di Pisa}\]

**INTRODUCTION**

The computation of a standard basis (also called Gröbner basis) of a multivariate polynomial ideal over a field \( K \) is crucial in many applications. The problem is intrinsically time and space consuming, and many researches aim to improve the basic algorithm due to B. Buchberger \([Bu1]\). One has investigated the problem of the optimal choice of the term ordering depending from the use that has to be made of the results, \([BS]\), and the systematic elimination of unnecessary reductions \([Bu2]\, [GM]\, [Po]\). We can call all these problems "combinatorial complexity problems".

The present paper considers arithmetic complexity problems; our main problem is how to limit the growth of the coefficients in the algorithms and the complexity of the field operations involved. The problem is important with every ground field, with the obvious exception of finite fields.

The ground field is often represented as the field of fractions of some explicit domain, which is usually a quotient of a finite extension of \( \mathbb{Z} \), and the computations are hence reduced to these domains.

The problem of coefficient growth and complexity already appeared in the calculation of the GCD of two univariate polynomials, which is indeed a very special case of standard basis computation; the PRS algorithms of Brown and Collins operate the partial coefficient simplifications predicted by a theorem, hence succeeding in controlling this complexity.

Our approach looks for analogies with these algorithms, but a general structure theorem is missing, hence our approach relies on a limited search of coefficient simplifications. The basic idea is the following: since the GCD is usually costly, we can use in its place the "greatest between the common divisors that are easy to compute" (the GECD), this suggestion allowing different instances. A set of such instances is based on the remark that if you have elements in factorized form, then many common divisors are immediately evident. Since irreducible factorization, even assuming that it exists in our domain, is costly, we use a partial factorization basically obtained using a "lazy multiplication" technique, i.e. performing coefficient multiplications only if they are unavoidable. The resulting algorithms were tested with a "simulated" implementation on the integers, and the results suggest that a complete implementation should be very efficient, at least when the coefficient domain is a multivariate rational function field.

1. **STANDARD BASIS COMPLETION ALGORITHM REVISITED**

Let \( A \) be a domain, \( K \) its quotient field, and \( X = (x_1, \ldots, x_n) \) a set of indeterminates. A polynomial in \( K[X] \) is a sum of monomials, each one being composed of a non-zero coefficient and a multiplicative term (term for short), i.e. a product of indeterminates.

A **term-ordering** is a total ordering on the set of terms, making it an ordered monoid, and such that 1, identified with the empty term, is the minimum of the monoid. From now on, we assume that a term-ordering is given.

Research with the contribution of Ministero della Pubblica Istruzione (Italy) and C.N.R.S. (France).
If $f$ is a non-zero polynomial, define $Lm(f)$, $Lt(f)$, $Lc(f)$ (leading monomial, leading term, leading coefficient of $f$) being respectively the monomial of $f$ with the maximum term, its term and its coefficient. Similarly, define the sets $Lm(S)$, $Lt(S)$, $Lc(S)$ for any subset $S \subseteq K[X]$. 

A polynomial (a subset) is called monic if its leading coefficient(s) is (are) 1.

A standard basis is a finite set $G$ of generators of an ideal $I \subseteq K[X]$ such that $Lm(G)$ generates the ideal spanned by $Lm(I)$. By noetherianity of $K[X]$ any family of polynomials can be completed into a standard basis; the effective computation is not straightforward.

B. Buchberger gave in [Bu1] the first completion procedure.

In this paper we propose an improved approach whenever $K$ is not a finite field.

We describe here the Buchberger algorithm as an "algorithm scheme", i.e. as a series of named (but undefined) data structures and subalgorithms. Our algorithms will consist in alternative definitions for some of these structures and algorithms, keeping unchanged the overall structure of the algorithm, and undefined some of the subalgorithms (meaning that any correct algorithm available can be used).

**Data structures.**

The data structures for the algorithm are:

1. a polynomial basis $G$ of $I$
2. a set $S$ of elements of $I$ (the simplifiers); in the original Buchberger algorithm this coincides with $G$.
3. a set $B$ of pairs of elements of $G$ (the pairs to process)
4. additional auxiliary structures (non existing in the original algorithm)

**Subalgorithms.**

The subalgorithms are defined as functions, having arguments, values and side-effects on the data structures (no side effect in the description means that side-effects are not necessary to the correctness).


B (Selection-strategy) Arguments: none. Values: either an element $\sigma$ of $B$; or empty ($\emptyset$). Side-effects: delete $\sigma$ from $B$.

C (Compute-$Sp$) Arguments: a pair $\sigma$. Values: a polynomial.

D (Simplification-strategy). Arguments: a polynomial $f$. Value: either an element $g_i \in G$, and a term $T$ such that $T \cdot Lt(g_i)$ appears as a term of $f$; or empty.

E (Simplify). Argument: two polynomials and a term. Value: a polynomial


H (Final-post-processing) Argument: none; value: a polynomial basis (the final result). This procedure may use both Simplification-strategy and Simplify.

A simplification strategy is called a $Lt$-simplification if we always have that $T \cdot Lt(g_i) = Lt(f)$, and a full simplification if we have $\emptyset$ as value only if no $Lt(g_i)$ divides a term of $f$.

To simplify the descriptions, we will always assume that we have an $Lt$-simplification, but the generalization is straightforward.

The algorithm scheme is the following:

1. Initialize