Selection Strategies in Buchberger's Algorithm

Dylan Peifer

April 10, 2018

1 Gröbner Bases

Let $R = k[x_1, \ldots, x_n]$ be a polynomial ring over some field k. A *Gröbner basis* of an ideal $I \subseteq R$ is a special set of generators of the ideal I. Instead of a precise definition, let's start with a question.

Question 1. *Consider the ideal $I = \langle x^2 - y^3, xy^2 + x \rangle$ in the ring $\mathbb{Q}[x, y]$. Is $x^5 + x$ an element of I?*

This seems like a very simple question that should have a very simple answer. Elements of I look like $a_1(x^2 - y^3) + a_2(xy^2 + x)$ for $a_1, a_2 \in \mathbb{Q}[x, y]$. But we have no limits in size or degree of a_1 and a_2, and it is possible that some complex cancellation takes place across the terms of $a_1(x^2 - y^3)$ and $a_2(xy^2 + x)$ to get $x^5 + x$. How can we handle this?

Maybe a simpler question would help.

Question 2. *Consider the ideal $I = \langle x^2 + x - 2 \rangle$ in the ring $\mathbb{Q}[x]$. Is $h(x) = x^3 + 3x^2 + 5x + 4$ an element of I?*

Now elements of I look like $a(x^2 + x - 2)$ for $a \in \mathbb{Q}[x]$. In other words, every element of I is divisible by $x^2 + x - 2$, and we can test divisibility with the division algorithm. In particular, we can compute via polynomial long division

$$
\begin{array}{rcl|rcrcrcr}
 & & & & & x & + & 2 & & & \\
\cline{4-10}
x^2 & + & x - 2 & & & x^3 & + & 3x^2 & + & 5x & + & 4 \\
 & & & - & & (x^3 & + & x^2 & - & 2x) & & \\
\cline{4-10}
 & & & & & & & 2x^2 & + & 7x & + & 4 \\
 & & & & - & & & (2x^2 & + & 2x & - & 4) \\
\cline{4-10}
 & & & & & & & & & 5x & + & 8
\end{array}
$$

that $h(x) = (x + 2)(x^2 + x - 2) + (5x + 8)$ and thus $h(x)$ is not divisible by $x^2 + x - 2$. Therefore $h(x) \notin I$.

The division algorithm gave us a simple way to answer Question 2. With this as inspiration, we'd like to apply the division algorithm to Question 1. There are, however, two problems with directly applying the division algorithm to Question 1.

First, in Question 1 we have two generators of the ideal I, but the division algorithm requires one divisor. This is not actually a problem. With several divisors we simply choose

one appropriate divisor in each step of the division algorithm and keep track of a quotient for each divisor. If there are multiple choices for which divisor to use in a step we choose the first one in some predetermined ordering of the divisors. The end goal of the original division algorithm is an expression of the form $f = aq + r$ with quotient q and remainder r. Using multiple divisors will lead to an expression of the form $f = a_1q_1 + a_2q_2 + \cdots + a_sq_s + r$.

The second problem is more subtle. In each step of the division algorithm we divide the leading term of our divisor into the leading term of the dividend. Then we subtract away this multiple of the divisor from the dividend to remove the dividend's leading term. All of this depends upon the concept of a leading term, which in the single variable case is simply the term with highest exponent. But in the general multivariable case how do we decide if $x > y$ or if $xy^2 > x^2y$? In the end we just need to make some choice and be consistent. The choice we make is called a monomial order.

Definition 3. *Let x^α denote an arbitrary term where α is the vector of exponents. A* monomial order *on $R = k[x_1, \ldots, x_n]$ is a relation $>$ on the monomials of R such that $>$ is a total ordering, $>$ is a well-ordering, and if $x^\alpha > x^\beta$ then $x^\gamma x^\alpha > x^\gamma x^\beta$ for any x^γ (i.e., $>$ respects multiplication).*

These conditions are chosen precisely to guarantee that the steps of the division algorithm will still work. In particular, we need a total order so we can find the lead term of any polynomial, we need $>$ to respect multiplication so that when multiplying a divisor polynomial by a monomial the lead term remains the lead term, and we need a well-ordering so that the division algorithm terminates. There are several standard monomial orders.

Example 4. *Lexicographic order (lex) is defined by $\alpha > \beta$ if the leftmost nonzero component of $\alpha - \beta$ is positive. For example, $x > y > z$, $xy > y^4$, and $xz > y^2$.*

Example 5. *Graded reverse lexicographic order (grevlex) is defined by $\alpha > \beta$ if $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ and the rightmost nonzero entry of $\alpha - \beta$ is negative. For example, $x > y > z$, $y^3 > x$, and $y^2 > xz$.*

Now let's apply the multivariate division algorithm to Question 1. Using lex order we can compute

$$
\begin{array}{rcl|l}
 & & q_1: \; x^3 - xy & \\
 & & q_2: \; x^2y - y^2 + 1 & \\
\hline
x^2 - y^3 & & & \\
xy^2 + x & & x^5 + x & \\
 & - & (x^5 - x^3y^3) & \\
\hline
 & & x^3y^3 + x & \\
 & - & (x^3y^3 + x^3y) & \\
\hline
 & & -x^3y + x & \\
 & - & (-x^3y + xy^4) & \\
\hline
 & & -xy^4 + x & \\
 & - & (-xy^4 - xy^2) & \\
\hline
 & & xy^2 + x & \\
 & - & (xy^2 + x) & \\
\hline
 & & 0 &
\end{array}
$$

so that $x^5 + x = (x^3 - xy)(x^2 - y^3) + (x^2y - y^2 + 1)(xy^2 + x)$. This means $x^5 + x \in I$, and we've answered Question 1.

Definition 6. *When F is an ordered set of polynomials and dividing h by the $f_i \in F$ using the division algorithm leads to the remainder r we write $h^F \to r$ and say h* reduces *to r.*

It is certainly the case that if F generates an ideal and applying our new method yields $h^F \to 0$ then h is an element of the ideal, since the division algorithm has given us an expression for h in terms of the generators. But what if we produce a nonzero remainder? Does this mean that the element is not in the ideal? Unfortunately, this is not the case.

Example 7. *Using the ideal from Question 1, note that $y^2(x^2-y^3)-x(xy^2+x) = -x^2-y^5 \in I$. However, multivariate division in lex order produces the nonzero remainder $-y^5 - y^3$.*

To fix this, consider starting the multivariate division algorithm with a dividend that is an element of I. Then in each step of the division algorithm we subtract an element of I and the remaining dividend is then also an element of I. If it is the case that every leading term of an element of I is divisible by some leading term of our divisors, then we can take a step at any nonzero dividend, and the division algorithm must then lead to a zero remainder. Thus what we need is a generating set of I such that every element of I has leading term divisible by some leading term of the generating set. This is exactly one way to define a Gröbner basis.

Definition 8. *Given a monomial order, let $\mathrm{LT}(f)$ be the leading term of f. Similarly, let $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(f) \mid f \in I \rangle$ be the ideal generated by all leading terms of I.*

Definition 9. *Given a monomial order, a* Gröbner basis *G of a nonzero ideal I is a set of generators $\{g_1, g_2, \ldots, g_s\} \subseteq I$ such that any of the following equivalent conditions hold:*

$$\begin{aligned} &(i) \quad f^G \to 0 \iff f \in I \\ &(ii) \quad f^G \text{ is unique for all } f \in R \\ &(iii) \quad \langle \mathrm{LT}(g_1), \mathrm{LT}(g_2), \ldots, \mathrm{LT}(g_s) \rangle = \langle \mathrm{LT}(I) \rangle \end{aligned}$$

Property (i) gives a definitive answer to Question 1. In general, Gröbner bases are useful for computing a wide variety of information in commutative algebra and algebraic geometry.

2 Buchberger's Algorithm

We've established that Gröbner bases have useful properties for computation, but we haven't shown how to find them or even demonstrated that they always exist. Buchberger first presented an algorithm to compute Gröbner bases in [2]. The start of the algorithm is the following definition.

Definition 10. *Let $S(f,g) = \frac{x^\gamma}{\mathrm{LT}(f)} f - \frac{x^\gamma}{\mathrm{LT}(g)} g$ where x^γ is the least common multiple of the leading monomials of f and g. This is the S-*polynomial *of f and g, where S stands for subtraction or syzygy.*

S-polynomials provide a natural way to cancel leading terms, which seems likely to reveal lead terms in the ideal that our current generating set can't divide (note that Example 7 was an S-polynomial). The surprising fact is that the S-polynomials reveal all the potential problems with a generating set.

Theorem 11 (Buchberger's Criterion)**.** *Let $G = \{g_1, g_2, \ldots, g_s\}$ generate some ideal I. If $S(g_i, g_j)^G \to 0$ for all pairs g_i, g_j then G is a Gröbner basis of I.*

Proof. We give a brief sketch of the detailed proof from [3]. Suppose $f \in I$. Then f has an expression

$$f = \sum_{i=1}^{s} a_i g_i$$

in terms of the g_i. We want to show that $\mathrm{LT}(f)$ is divisible by some $\mathrm{LT}(g_i)$. Since each term in the sum has a lead term which is a multiple of some $\mathrm{LT}(g_i)$, the only way that f has a lead term which is not a multiple of a $\mathrm{LT}(g_i)$ is if lead terms in the above sum cancel. We can rewrite any complex cancellation of lead terms in the sum in terms of pairwise cancellations from S-polynomials, and then since $S(g_i, g_j)^G \to 0$ the division algorithm gives us a way to rewrite these expressions without cancellation. With cancellations removed, the new expression for f must then have lead term a multiple of some $\mathrm{LT}(g_i)$, so G satisfies condition (iii) of Definition 9. □

This theorem leads naturally to a basic algorithm, known as Buchberger's algorithm, which computes a Gröbner basis of I from any starting generating set of I. The idea is to check all S-polynomials of our current generating set. If any S-polynomials do not reduce to 0, we force them to reduce to 0 by adding their reduction to our generating set. Of course adding new terms to our generating set means we have many more S-polynomials to consider, but eventually this process will terminate. Once it does, the current generating set is a Gröbner basis.

A simple form of Buchberger's algorithm is written here as Algorithm 1. A detailed proof of the correctness and termination of this form of Buchberger's algorithm can be found in [3]. In the algorithm we assume we have a function select that will select a pair from the set of pairs P. A simple implementation of select could treat P as a list and select the first pair in the list. More detailed strategies can improve performance and will be addressed in Section 3.2.

The existence of Buchberger's algorithm guarantees that a Gröbner basis exists for any ideal I, and the algorithm even shows us how to find it. This Gröbner basis can be much larger and more complicated than the original generating set.

Example 12. *Consider the ideal $I = \langle xy^2 + z, xz + 3y, x^2 + yz \rangle$. Using grevlex, Buchberger's algorithm produces the Gröbner basis*

$$G = \{xy^2 + z, \quad xz + 3y, \quad x^2 + yz, \quad -3y^3 + z^2, \quad -3y - \frac{1}{3}z^3, \quad \frac{1}{243}z^8 + z\}$$

where the pairs $(1,2), (1,3), (1,5)$ did not reduce to zero.

The Gröbner basis produced by Buchberger's algorithm may contain redundant elements. We can remove these elements to obtain minimal and reduced Gröbner bases.

Algorithm 1 Buchberger's Algorithm

input a set of polynomials $\{f_1, \ldots, f_s\}$
output a Gröbner basis of $I = \langle f_1, \ldots, f_s \rangle$
procedure BUCHBERGER($\{f_1, \ldots, f_s\}$)
    $G \leftarrow \{f_1, \ldots, f_s\}$ ▷ the current basis
    $m \leftarrow s$ ▷ the size of the current basis
    $P \leftarrow \{(i, j) \mid 1 \leq i < j \leq m\}$ ▷ the remaining pairs to process
    while $|P| > 0$ **do**
        $(i, j) \leftarrow \text{select}(P)$
        $P \leftarrow P \setminus \{(i, j)\}$
        $r \leftarrow S(f_i, f_j)^G$
        if $r \neq 0$ **then**
            $f_{m+1} \leftarrow r$
            $G \leftarrow G \cup \{f_{m+1}\}$
            $m \leftarrow m + 1$
            $P \leftarrow P \cup \{(i, m) \mid 1 \leq i < m\}$
        end if
    end while
    return G
end procedure

Definition 13. *A* minimal *Gröbner basis is a Gröbner basis* $G = \{g_1, \ldots, g_s\}$ *such that* $\mathrm{LT}(g_i)$ *does not divide* $\mathrm{LT}(g_j)$ *for any* $i \neq j$.

Example 14. *The basis in Example 12 is not minimal, but we can remove redundant polynomials to produce the minimal basis*

$$xz + 3y, \quad x^2 + yz, \quad -3y - \frac{1}{3}z^3, \quad \frac{1}{243}z^8 + z.$$

Definition 15. *A* reduced *Gröbner basis is a minimal Gröbner basis* $G = \{g_1, \ldots, g_s\}$ *such that each* g_i *is monic and* $\mathrm{LT}(g_i)$ *does not divide any term in* g_j *for any* $i \neq j$.

Example 16. *The basis in Example 14 is not reduced, but we can divide each polynomial by its lead coefficient and reduce with respect to the others to produce the reduced basis*

$$xz - \frac{1}{3}z^3, \quad x^2 - \frac{1}{9}z^4, \quad y + \frac{1}{9}z^3, \quad z^8 + 243z.$$

Like reduced echelon form in linear algebra, the reduced Gröbner basis of an ideal is actually unique, which is another useful property of Gröbner bases. Most implementations of Buchberger's algorithm will automatically return this reduced Gröbner basis by minimalizing and interreducing as we did in Example 14 and Example 16.

3 Improving Buchberger's Algorithm

There are many ways to improve the basic outline of Buchberger's algorithm presented in Section 2. Two major directions for improvement are pair elimination and pair selection.

3.1 Pair Elimination

In Algorithm 1 we reduce all S-polynomials $S(g_i, g_j)$ for the eventual Gröbner basis $G = \{g_1, \ldots, g_s\}$. These reductions are the most expensive part of the algorithm, and careful examination of a full proof of Theorem 11 shows that not all reductions are actually necessary to guarantee a Gröbner basis. Pair elimination is the process of removing some S-polynomials from consideration without actually reducing them. There are two main criteria for eliminating pairs.

Lemma 17 (LCM Criterion)**.** *If the lead monomials of g_i, g_j are relatively prime (i.e., their product is their least common multiple) then $S(g_i, g_j)$ has standard representation and thus can be ignored in Buchberger's algorithm.*

Lemma 18 (Chain Criterion)**.** *Let* $\mathrm{LM}(g)$ *denote the lead monomial of g. If* $\mathrm{LM}(g_k)$ *divides* $\mathrm{lcm}(\mathrm{LM}(g_i), \mathrm{LM}(g_j))$ *then*

$$S(g_i, g_j) = \frac{\mathrm{lcm}(\mathrm{LM}(g_i), \mathrm{LM}(g_j))}{\mathrm{lcm}(\mathrm{LM}(g_i, g_k))} S(g_i, g_k) - \frac{\mathrm{lcm}(\mathrm{LM}(g_i), \mathrm{LM}(g_j))}{\mathrm{lcm}(\mathrm{LM}(g_j, g_k))} S(g_j, g_k)$$

and thus $S(g_i, g_j)$ can be ignored as long as $S(g_i, g_k)$ and $S(g_j, g_k)$ are considered (or otherwise eliminated) in Buchberger's algorithm.

These are listed as criteria 2 and criteria 1 in [1]. As usual, detailed proofs can be found in [3]. The Gebauer-Möller rules [6] give a way to efficiently use these two criteria during an actual run of Buchberger's algorithm, and they are a key part of all major implementations.

A theoretical way to understand the two criteria is that we only need to check the S-polynomials corresponding to a minimal set of generators for the syzygy module on lead terms of $g_1, \ldots, g_s$. Following the Gebauer-Möller rules does not guarantee that only a minimal generating set is checked, but in practice the difference is small, and trying to eliminate all unneeded pairs is typically more computationally expensive than the eliminated reductions. Storing even more information about generators during the algorithm in order to eliminate even more pairs is one way to understand recent developments in signature based algorithms like F_5 [5, 4], so research still continues in pair elimination.

3.2 Pair Selection

In Algorithm 1 we assumed we had a function select that will select a pair (i, j) from the set of pairs P. For correctness, it does not matter how select is implemented, and a simple implementation such as treating the pair set as a list and selecting the first pair is sufficient. The efficiency of Buchberger's algorithm, however, strongly depends on the selection strategy, as poor selection can generate many additional redundant generators during the computation.

Our general intuition is that Buchberger's algorithm is a reduction algorithm like LLL or the Euclidean algorithm. With that in mind we would like to choose small things first, as this is likely to reveal small generators that are more likely to quickly reduce terms in subsequent steps. In selecting pairs, "small" typically means that the lead monomials of the pair have small least common multiple in the monomial order or in total degree. As

generators tend to get larger and more complicated in later steps of the algorithm, our naive method of selecting the first pair is also in some ways selecting on smallness. There are four major strategies along these lines.

- First: among pairs with minimal j, select the pair with minimal i (i.e., treat the pair set as a queue)
- Degree: select a pair with minimal degree of $\operatorname{lcm}(\operatorname{LM}(f_i), \operatorname{LM}(f_j))$
- Normal: select a pair with smallest $\operatorname{lcm}(\operatorname{LM}(f_i), \operatorname{LM}(f_j))$ in the monomial order
- Sugar: select a pair with smallest sugar degree, which is the total degree $S(f_i, f_j)$ would have had if we homogenized at the beginning

First is the only strategy above that uniquely determines a pair. Other strategies may need to break ties, and for this purpose we will have Sugar break ties with Normal, and Normal and Degree break ties with First.

Normal selection was proposed by Buchberger in his initial presentation of the algorithm [2] and is very common in implementations. Sugar is presented in [8], tends to have similar or better performance in worst case examples, and is likely being used in some form in most serious implementations. For comparison with these good strategies, we can also consider strategies that select pairs at random or do the opposite of the above strategies. This gives an additional five strategies.

- Random: select a pair uniformly at random
- Last: among pairs with maximal j, select the pair with maximal i (i.e., treat the pair set as a stack)
- Codegree: select a pair with maximal degree of $\operatorname{lcm}(\operatorname{LM}(f_i), \operatorname{LM}(f_j))$
- Strange: select a pair with largest $\operatorname{lcm}(\operatorname{LM}(f_i), \operatorname{LM}(f_j))$ in the monomial order
- Spice: select a pair with largest sugar degree, which is the total degree $S(f_i, f_j)$ would have had if we homogenized at the beginning

For breaking ties we will have Spice break ties with Strange, and Strange and Codegree break ties with Last. Note that Last, Codegree, Strange, and Spice are cute names I made up for the strategies that do the opposite of First, Degree, Normal, and Sugar. As we will see, they have very poor performance and are not even listed or considered elsewhere.

To evaluate these strategies we will use them in Buchberger's algorithm and count the total number of S-polynomial reductions performed (i.e., the number of times we go through the while loop) before the algorithm terminates on several example ideals. This is a rough measure of the computational cost, with smaller numbers better, that is independent of hardware or implementation details. The example ideals we will consider are listed in Table 1, and consist of the small example from Question 1 and several instances of five standard parametrized families of benchmarking ideals. Results are listed in Table 2. None of the

example	generators
ex1	$\langle x^2 - y^3, xy^2 + x \rangle$
cyclic3	$\langle x + y + z, xy + xz + yz, xyz - 1 \rangle$
eco3	$\langle xyz + xz - 1, yz - 2, x + y + 1 \rangle$
katsura3	$\langle x + 2y + 2z - 1, x^2 + 2y^2 + 2z^2 - x, 2xy + 2yz - y \rangle$
noon3	$\langle 10xy^2 + 10xz^2 - 11x + 10, 10x^2y + 10yz^2 - 11y + 10, 10x^2z + 10y^2z - 11z + 10 \rangle$
reimer3	$\langle 2x^2 - 2y^2 + 2z^2 - 1, 2x^3 - 2y^3 + 2z^3 - 1, 2x^4 - 2y^4 + 2z^4 - 1 \rangle$

Table 1: A collection of example ideals. A single example using parameter value 3 is listed for each of the five parametrized families. Increasing the parameter increases the number of variables, degrees, and generators. Larger examples are listed in [7].

	First	Degree	Normal	Sugar	Random	Last	Codegree	Strange	Spice
ex1	2	2	2	2	2.00[0.00]	2	2	2	2
cyclic3	2	2	2	2	2.47[0.50]	3	3	3	3
eco3	2	2	2	2	2.00[0.00]	2	2	2	2
katsura3	4	4	4	4	4.00[0.00]	4	4	4	4
noon3	17	17	17	17	18.67[1.86]	23	27	18	18
reimer3	22	21	23	24	24.34[2.17]	25	29	29	29
cyclic4	11	11	11	11	13.53[2.34]	19	19	19	19
eco4	10	10	10	10	11.05[1.23]	12	12	12	12
katsura4	10	10	10	10	12.39[1.79]	17	17	18	18
noon4	71	71	71	71	92.19[12.46]	138	242	546	546
reimer4	154	95	91	101	228.73[84.32]	-	-	-	-
cyclic5	121	110	107	114	179.01[56.14]	-	-	-	-
eco5	28	27	24	24	33.84[6.10]	51	48	53	53
katsura5	28	28	28	28	43.41[10.92]	70	73	86	86
noon5	262	262	262	262	353.88[42.24]	-	-	-	-
reimer5	757	212	211	411	-	-	-	-	-
cyclic6	439	660	620	412	-	-	-	-	-
eco6	69	72	61	64	94.76[15.23]	177	244	315	315
katsura6	66	66	66	66	122.35[32.80]	754	768	797	797
noon6	887	887	887	887	-	-	-	-	-
reimer6	-	687	589	2505	-	-	-	-	-
cyclic7	2552	5882	5781	2750	-	-	-	-	-
eco7	164	158	144	156	270.63[63.08]	-	-	-	-
katsura7	164	164	164	164	318.33[104.94]	-	-	-	-
noon7	2885	2885	2885	2885	-	-	-	-	-
reimer7	-	1726	1504	-	-	-	-	-	-

Table 2: Reductions per strategy on the example ideals. All computations were performed in grevlex with coefficient field $\mathbb{Z}/32003\mathbb{Z}$. For Random we report mean and standard deviation over 100 trials. Blank entries indicate that the computation did not finish after 1 hour.

ideals we use have Gröbner bases computations that are difficult on modern hardware, though with large enough parameters these ideal families do become infeasible.

Overall, we see the expected result that Normal and Sugar are consistently better strategies than the others. Last, Codegree, Stange, and Spice quickly make even these small examples infeasible to compute, and First, Degree, Normal, and Sugar are always better than Random. For noon and katsura, the four good strategies have the same performance, but they differ, sometimes dramatically, on many other examples. It is notable that no strategy is best on all examples, and each of First, Degree, Normal, and Sugar is best on at least one example. This suggests that trying different strategies when faced with a difficult ideal could be useful. Optimal performance, which is the minimal number of reductions to compute a Gröbner basis for these examples is, as far as I know, unknown, and there does not appear to be much modern research on selection strategies.

References

[1] Bruno Buchberger. Groebner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory II*, chapter 4, pages 89–127. D. Reidel Publishing Company, 1985.

[2] Bruno Buchberger. An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *J. Symbolic Comput.*, 41(3-4):475–511, 2006. Translated from the 1965 German original by Michael P. Abramson.

[3] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, Cham, fourth edition, 2015.

[4] Christian Eder and Jean-Charles Faugère. A survey on signature-based algorithms for computing Gröbner bases. *J. Symbolic Comput.*, 80(part 3):719–784, 2017.

[5] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83. ACM, New York, 2002.

[6] Rüdiger Gebauer and H. Michael Möller. On an installation of Buchberger's algorithm. *J. Symbolic Comput.*, 6(2-3):275–286, 1988.

[7] V. P. Gerdt, Yu. A. Blinkov, and D. A. Yanovich. Janet basis examples. `http://invo.jinr.ru/examples.phtml`.

[8] A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. 'One sugar cube, please' or Selection strategies in the Buchberger algorithm. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pages 49–54. ACM, New York, 1991.