Learning Selection Strategies in Buchberger's Algorithm

Dylan Peifer [1] **Michael Stillman** [1] **Daniel Halpern-Leistner** [1]

Abstract

Studying the set of exact solutions of a system of polynomial equations largely depends on a single iterative algorithm, known as Buchberger's algorithm. Optimized versions of this algorithm are crucial for many computer algebra systems (e.g., Mathematica, Maple, Sage). We introduce a new approach to Buchberger's algorithm that uses reinforcement learning agents to perform S-pair selection, a key step in the algorithm. We then study how the difficulty of the problem depends on the choices of domain and distribution of polynomials, about which little is known. Finally, we train a policy model using proximal policy optimization (PPO) to learn S-pair selection strategies for random systems of binomial equations. In certain domains, the trained model outperforms state-of-the-art selection heuristics in total number of polynomial additions performed, which provides a proof-of-concept that recent developments in machine learning have the potential to improve performance of algorithms in symbolic computation.

1. Introduction

Systems of multivariate polynomial equations, such as

$$\begin{cases} 0 = f_1(x,y) = x^3 + y^2 \\ 0 = f_2(x,y) = x^2y - 1 \end{cases} \tag{1}$$

appear in many scientific and engineering fields, as well as many subjects in mathematics. The most fundamental question about such a system of equations is whether there exists an exact solution. If one can express the constant polynomial $h(x,y) = 1$ as a combination

$$h(x,y) = a(x,y)f_1(x,y) + b(x,y)f_2(x,y) \tag{2}$$

for some polynomials a and b, then there can be no solution, because the right hand side vanishes at any solution of the system, but the left hand side is always 1.

The converse also holds: the set of solutions with x and y in $\mathbb{C}$ is empty if and only if there exists a linear combination (2) for $h = 1$ (Hilbert, 1893). Thus the existence of solutions to (1) can be reduced to the larger problem of determining if a polynomial h lies in the *ideal* generated by these polynomials, which is defined to be the set $I = \langle f_1, f_2 \rangle$ of all polynomials of the form (2).

The key to solving this problem is to find a *Gröbner basis* for the system. This is another set of polynomials $\{g_1, \ldots, g_k\}$, potentially much larger than the original set, which generate the same ideal $I = \langle f_1, f_2 \rangle = \langle g_1, \ldots, g_k \rangle$, but for which one can employ a version of the Euclidean algorithm (discussed below) to determine if $h \in I$.

In fact, computing a Gröbner basis is the necessary first step in algorithms that answer a huge number of questions about the original system: eliminating variables, parametrizing solutions, studying geometric features of the solution set, etc. This has led to a wide array of scientific applications of Gröbner bases, wherever polynomial systems appear, including: computer vision (Duff et al., 2019), cryptography (Faugère et al., 2010), biological networks and chemical reaction networks (Arkun, 2019), robotics (Abłamowicz, 2010), statistics (Diaconis & Sturmfels, 1998; Sullivant, 2018), string theory (Gray, 2011), signal and image processing (Lin et al., 2004), integer programming (Conti & Traverso, 1991), coding theory (Sala et al., 2009), and splines (Cox et al., 2005).

Buchberger's algorithm (Buchberger, 1965; 2006) is the basic iterative algorithm used to find a Gröbner basis. As it can be costly in both time and space, this algorithm is the computational bottleneck in many applications of Gröbner bases. All direct algorithms for finding Gröbner bases (e.g., (Faugère, 1999; Faugère, 2002; Roune & Stillman, 2012; Eder & Faugère, 2017)) are variations of Buchberger's algorithm, and highly optimized versions of the algorithm are a key piece of computer algebra systems such as (CoCoA; Macaulay2; Magma; Maple; Mathematica; SageMath; Singular).

There are several points in Buchberger's algorithm which

[1]Department of Mathematics, Cornell University, Ithaca, NY, USA. Correspondence to: Daniel Halpern-Leistner <daniel.hl@cornell.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

depend on choices that do not affect the correctness of the algorithm, but can have a significant impact on performance. In this paper we focus on one such choice, called *pair selection*. We show that the problem of pair selection fits naturally into the framework of reinforcement learning, and claim that the rapid advancement in applications of deep reinforcement learning over the past decade has the potential to significantly improve the performance of the algorithm.

Our main contributions are the following:

1. Initiating the empirical study of Buchberger's algorithm from the perspective of machine learning.
2. Identifying a precise sub-domain of the problem, consisting of systems of binomials, that is directly relevant to applications, captures many of the challenging features of the problem, and can serve as a useful benchmark for future research.
3. Training a simple neural network model for pair selection which outperforms state-of-the art selection strategies by 20% to 40% in this domain, thereby demonstrating significant potential for future work.

1.1. Related Work

Several authors have applied machine learning to perform algorithm selection (Huang et al., 2019) or parameter selection (Xu et al., 2019) in problems related to Gröbner bases. While we are not aware of any existing work applying machine learning to improve the performance of Buchberger's algorithm, many authors have used machine learning to improve algorithm performance in other domains (Alvarez et al., 2017; Khalil et al., 2016). Recently, there has been progress using reinforcement learning to learn entirely new heuristics and strategies inside algorithms (Bengio et al., 2018), which is closest to our approach.

2. Gröbner Bases

In this section we give a focused introduction to Gröbner basis concepts that will be needed for Section 3. For a more general introduction to Gröbner bases and their uses, see (Cox et al., 2015; Mora, 2005).

Let $R = K[x_1, \dots, x_n]$ be the set of polynomials in variables $x_1, \dots, x_n$ with coefficients in some field K. Let $F = \{f_1, \dots, f_s\}$ be a set of polynomials in R, and consider $I = \langle f_1, \dots, f_s \rangle$ the ideal generated by F in R.

The definition of a Gröbner basis depends on a choice of *monomial order*, a well-order relation $>$ on the set of monomials $\{x^a = x_1^{a_1} \cdots x_n^{a_n} | a \in \mathbb{Z}_{\geq 0}^n\}$ such that $x^a > x^b$ implies $x^{a+c} > x^{b+c}$ for any exponent vectors a, b, c. Given a polynomial $f = \sum_a \lambda_a x^a$, we define the *leading term* $\mathrm{LT}(f) = \lambda_a x^a$, where the *leading monomial* $\mathrm{LM}(f) = x^a$ is the largest monomial with respect to the ordering $>$ that has $\lambda_a \neq 0$. An important example is the *grevlex* order, where $x^a > x^b$ if the total degree of x^a is greater than that of x^b, or they have the same degree, but the *last* non-zero entry of $a - b$ is *negative*. For example, in the grevlex order, we have $x_1 > x_2 > x_3$, $x_2^3 > x_1 x_2$, and $x_2^2 > x_1 x_3$.

Algorithm 1 Multivariate Division Algorithm

1: **Input:** a polynomial h and a set of polynomials $F = \{f_1, \dots, f_s\}$
2: **Output:** a remainder polynomial $r = \mathrm{reduce}(h, F)$
3: $r \leftarrow h$
4: **while** $\mathrm{LT}(f_i) | \mathrm{LT}(r)$ for some i **do**
5: choose i such that $\mathrm{LT}(f_i) | \mathrm{LT}(r)$
6: $r \leftarrow r - \frac{\mathrm{LT}(r)}{\mathrm{LT}(f_i)} f_i$
7: **end while**

Given a choice of monomial order $>$ and a set of polynomials $F = \{f_1, \dots, f_s\}$, the *multivariate division algorithm* takes any polynomial h and produces a remainder polynomial r, written $r = \mathrm{reduce}(h, F)$, such that $h - r \in \langle f_1, \dots, f_s \rangle$ and $\mathrm{LT}(f_i)$ does not divide $\mathrm{LT}(r)$ for any i. In this case we say that h *reduces to* r. The division algorithm is guaranteed to terminate, but the remainder can depend on the choice in line 5 of Algorithm 1.

Definition 1. *Given a monomial order, a* Gröbner basis G *of a nonzero ideal I is a set of generators $\{g_1, g_2, \dots, g_k\}$ of I such that any of the following equivalent conditions hold:*

$$\begin{aligned} &(i) && \mathrm{reduce}(h, G) = 0 \iff h \in I \\ &(ii) && \mathrm{reduce}(h, G) \text{ is unique for all } h \in R \\ &(iii) && \langle \mathrm{LT}(g_1), \mathrm{LT}(g_2), \dots, \mathrm{LT}(g_k) \rangle = \langle \mathrm{LT}(I) \rangle \end{aligned}$$

where $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(f) \mid f \in I \rangle$ is the ideal generated by the leading terms of all polynomials in I.

As mentioned in Section 1, a consequence of (i) is that given a Gröbner basis G for $\langle f_1, \dots, f_s \rangle$, the system of equations $f_1 = 0, \dots, f_s = 0$ has no solution over $\mathbb{C}$ if and only if $\mathrm{reduce}(1, G) = 0$, that is, if G contains a non-zero constant polynomial.

2.1. Buchberger's Algorithm

Buchberger's algorithm produces a Gröbner basis for the ideal $I = \langle f_1, \dots, f_s \rangle$ from the initial set $\{f_1, \dots, f_s\}$ by repeatedly producing and reducing combinations of the basis elements.

Definition 2. *Let $S(f, g) = \frac{x^\gamma}{\mathrm{LT}(f)} f - \frac{x^\gamma}{\mathrm{LT}(g)} g$, where $x^\gamma = \mathrm{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))$ 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.*

Theorem 1 (Buchberger's Criterion)**.** *Suppose the set of polynomials $G = \{g_1, g_2, \ldots, g_k\}$ generates the ideal I. If* $\text{reduce}(S(g_i, g_j), G) = 0$ *for all pairs g_i, g_j then G is a Gröbner basis of I.*

Example 1. *Fix $>$ to be grevlex. For the generating set $F = \{f_1, f_2\}$ in Equation (1),* $r = \text{reduce}(S(f_1, f_2), F) = y^3 + x$*. By construction, the set $G = \{f_1, f_2, r\}$ generates the same ideal as F and* $\text{reduce}(S(f_1, f_2), G) = 0$*, so we have eliminated this pair for the purposes of verifying the criterion at the expense of two new pairs. Luckily, in this example* $\text{reduce}(S(f_1, r), G) = 0$ *and* $\text{reduce}(S(f_2, r), G) = 0$*, so G is a Gröbner basis for $\langle f_1, f_2 \rangle$ with respect to the grevlex order.*

Generalizing this example, Theorem 1 naturally leads to Algorithm 2, which depends on several implementation choices: `select` in line 6, `reduce` in line 8, and `update` in line 10. Algorithm 2 is guaranteed to terminate regardless of these choices, but all three impact computational performance. Most improvements to Buchberger's algorithm have come from improved heuristics in these steps.

The simplest implementation of `update` is

$$\text{update}(P, G, r) = P \cup \{(f, r) : f \in G\},$$

but most implementations use special rules to eliminate some pairs a priori, so as to minimize the number of S-polynomial reductions performed. In fact, much recent research on improving the performance of Buchberger's algorithm (Faugère, 2002; Eder & Faugère, 2017) has focused on mathematical methods to eliminate as many pairs as possible. We use the standard pair elimination rules of (Gebauer & Möller, 1988) in all results in this paper.

The main choice in `reduce` occurs in line 5 of Algorithm 1. For our experiments, we always choose the smallest $\text{LT}(f_i)$ which divides r. We also modify Algorithm 1 to fully *tail reduce*, which leaves no term of r divisible by any $\text{LT}(f_i)$.

Algorithm 2 Buchberger's Algorithm

1: **Input:** a set of polynomials $\{f_1, \ldots, f_s\}$
2: **Output:** a Gröbner basis G of $I = \langle f_1, \ldots, f_s \rangle$
3: $G \leftarrow \{f_1, \ldots, f_s\}$
4: $P \leftarrow \{(f_i, f_j) : 1 \leq i < j \leq s\}$
5: **while** $|P| > 0$ **do**
6: $\quad (f_i, f_j) \leftarrow \text{select}(P)$
7: $\quad P \leftarrow P \setminus \{(f_i, f_j)\}$
8: $\quad r \leftarrow \text{reduce}(S(f_i, f_j), G)$
9: $\quad$ **if** $r \neq 0$ **then**
10: $\quad\quad P \leftarrow \text{update}(P, G, r)$
11: $\quad\quad G \leftarrow G \cup \{r\}$
12: $\quad$ **end if**
13: **end while**

Our focus is the implementation of `select`.

2.2. Selection Strategies

The selection strategy, which chooses the pair (f_i, f_j) to process next, is critically important for efficiency, as poor pair selection can add many unnecessary elements to the generating set before finding a Gröbner basis. While there is some research on selection (Faugère, 2002; Roune & Stillman, 2012), most is in the context of signature Gröbner bases and Faugere's F_5 algorithm. Other than these, most strategies to date depend on relatively simple human-designed heuristics. We use several well-known examples as benchmarks:

First: Among pairs with minimal j, select the one with minimal i. In other words, treat the pair set P as a queue.

Degree: Select the pair with minimal total degree of $\text{lcm}(\text{LM}(f_i), \text{LM}(f_j))$. If needed, break ties with First.

Normal: Select the pair with $\text{lcm}(\text{LM}(f_i), \text{LM}(f_j))$ minimal in the monomial order. If needed, break ties with First. In a degree order ($x^a > x^b$ if the total degree of x^a is greater than that of x^b), this is a refinement of Degree selection.

Sugar: Select the pair with minimal sugar degree, which is the degree $\text{lcm}(\text{LM}(f_i), \text{LM}(f_j))$ would have had if all input polynomials were homogenized. If needed, break ties with Normal. Presented in (Giovini et al., 1991).

Random: Select an element of the pair set uniformly at random.

Most implementations use Normal or Sugar selection.

2.3. Complexity

We will characterize the input to Buchberger's algorithm in terms of the number of variables (n), the maximal degree of a generator (d), and the number of generators (s). One measure of complexity is the maximum degree $\deg_{\max}(GB(I))$ of an element in the *unique* reduced minimal Gröbner basis for I.

When the coefficient field has characteristic 0, there is an upper bound $\deg_{\max}(GB(I)) \leq (2d)^{2^{n+1}}$ which is double exponential in the number of variables (Bayer & Mumford, 1993). There do exist ideals which exhibit double exponential behavior (Mayr & Meyer, 1982; Bayer & Stillman, 1988; Koh, 1998): there is a sequence of ideals $\{J_n\}$ where J_n is generated by quadratic homogeneous binomials in $22n - 1$ variables such that for any monomial order

$$2^{2^{n-1}-1} \leq \deg_{\max}(GB(J_n))$$

In the grevlex monomial order, the theoretical upper bounds on the complexity of Buchberger's algorithm are much better if the choice of generators is sufficiently generic. To

make this precise, for fixed n, d, s, the space of possible inputs, i.e., the space V of coefficients for each of the s generators, is finite dimensional. There is a subset $X \subset V$ of measure zero[1] such that for any point outside X,

$$\deg_{\max}(GB(I)) \leq (n+1)(d-1)+1$$

This implies that the size of $GB(I)$ is less than or equal to the number of monomials of degree less than or equal to $(n+1)(d-1)+1$, which grows like $\mathcal{O}(((n+1)d-1)^n)$.

It is expected, but not known, that it is rare for the maximum degree of a Gröbner basis element in the grevlex monomial order to be double exponential in the number of variables. Also, as early as the 1980's, it was realized that for many examples, the grevlex Gröbner basis was often much easier to compute than Gröbner bases for other monomial orders. For these reasons, the grevlex monomial order is a standard choice in Gröbner basis computations. We use grevlex throughout this paper for all of our experiments.

3. The Reinforcement Learning Problem

We model Buchberger's algorithm as a Markov Decision Process (MDP) in which an agent interacts with an environment to perform pair selection in line 6 of Algorithm 2.

Each pass through the `while` loop in line 5 of Algorithm 2 is a time step, in which the agent takes an action and receives a reward. At time step t, the agent's state $s_t = (G_t, P_t)$ consists of the current generating set G_t and the current pair set P_t. The agent must select a pair from the current set, so the set of allowable actions is $\mathcal{A}_t = P_t$. Once the agent selects an action $a_t \in \mathcal{A}_t$, the environment updates by removing the pair from the pair set, reducing the corresponding S-polynomial, and updating the generator and pair set if necessary.

After the environment updates, the agent receives a reward r_t which is -1 times the number of polynomial additions performed in the reduction of pair a_t, including the subtraction that produced the S-polynomial. This is a proxy for computational cost that is implementation independent, and thus useful for benchmarking against other selection heuristics. For simplicity, this proxy does not penalize monomial division tests or computing pair eliminations.

Each trajectory $\tau = (s_0, a_0, r_1, s_1, \ldots, r_T)$ is a sequence of steps in Buchberger's algorithm, and ends when the pair set is empty and the algorithm has terminated with a Gröbner basis. The agent's objective is to maximize the expected return $\mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} r_t]$, where $0 \leq \gamma \leq 1$ is a discount factor. With $\gamma = 1$, this is equivalent to minimizing the expected number of polynomial additions taken to produce a Gröbner basis.

[1]Technically, X is a closed algebraic subset. With coefficients in $\mathbb{R}$ or $\mathbb{C}$, this is measure zero in the usual sense.

This problem poses several interesting challenges from a machine learning perspective:

1. The size of the action set changes with each time step and can be very large.
2. There is a high variance in difficulty of problems of the same size.
3. The state changes shape with each time step, and the state space is unbounded in several dimensions: number of variables, degree and size of generators, number of generators, and size of coefficients.

3.1. The Domain: Random Binomial Ideals

Formulating Buchberger's algorithm as a reinforcement learning problem forces one to consider the question of what is a random polynomial. This is a significant departure from the typical framing of the Gröbner basis problem.

We have seen that Buchberger's algorithm performs much better than its worst case on generic choices of input. On the other hand, many of the ideals that arise in practice are far from generic in this sense. As n, d, and s grow, Gröbner basis computations tend to blow up in several ways simultaneously: (i) the number of polynomials in the generating set grows, (ii) the number of terms in each polynomial grows, and (iii) the size of the coefficients grows (e.g., rational numbers with very large denominators).

The standard way to handle (iii) in evaluating Gröbner basis algorithms is to work over the finite field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ for a large prime number p. The choice $\mathbb{Z}/32003\mathbb{Z}$ is common, if seemingly arbitrary, and all of our experiments use this coefficient field. Finite field coefficients are already of use in many applications (Bettale et al., 2013). They also figure prominently in many state of the art Buchberger implementations with rational coefficients: the idea is to start with a generating set with integer coefficients, reduce mod p for several large primes, compute the Gröbner bases for each of the resulting systems over finite fields, then "lift" these Gröbner bases back to rational polynomials (Arnold, 2003).

In order to address (ii), we restrict our training to systems of polynomials with at most two terms. These are known as *binomials*. We will also assume neither term is a constant. If the input to Buchberger's algorithm is a set of binomials of this form, then all of the new generators added to the set will also have this form. This side-steps the thorny issue of how to represent a polynomial of arbitrary size to a neural network.

Restricting our focus to binomial ideals has several other benefits: We will show that using binomials typically avoids the known "easy" case when the dimension of the ideal, which is defined to be the dimension of the set of solutions of the corresponding system of equations, is zero. We

have also seen that some of the worst known examples with double exponential behavior are binomial systems. Finally, binomials capture the qualitative fact that many of the polynomials appearing in applications are sparse. In fact, several applications of Buchberger's algorithm, such as integer programming, specifically call for binomial ideals (Cox et al., 2005; Conti & Traverso, 1991).

We also remark that a model trained on binomials might be useful in other domains as well. Just as most standard selection strategies only consider the leading monomials of each pair, one could use a model trained on binomials to select pairs based on their leading *binomials*.

We performed experiments with two probability distributions on the set of binomials of degree $\leq d$ in s generators. The first, `weighted`, selects the degree of each monomial uniformly at random, then selects each uniformly at random among monomials of the chosen degree. The second, `uniform`, selects both monomials uniformly at random from the set of monomials of degree $\leq d$. The main difference between these two distributions is that `weighted` tends to produce more binomials of low total degree. Both distributions assign non-zero coefficients uniformly at random.

For the remainder of the paper, we will use the format "n-d-s (uniform/weighted)" to specify our distribution on s-tuples of binomials of degree $\leq d$ in n variables.

3.2. Statistics

We will briefly discuss the statistical properties of the problem in the domain of binomial ideals to highlight its features and challenges.

Difficulty increases with n: (Table 1) This is consistent with the double exponential behavior in the worst-case analysis.

Degree and Normal outperform First and Sugar: (Table 1) This pattern is consistent across all distributions in the range tested ($n = 3$, $d \leq 30$, $s \leq 20$). The fact that Sugar under-performs in an average-case analysis might reflect the fact that it was chosen because it improves performance on known sequences of challenging benchmark ideals in (Giovini et al., 1991).

Very high variance in difficulty: This is also illustrated in Table 1, especially as the number of variables increases. Figure 1 provides a more detailed view of a single distribution, demonstrating the large variance and long right tail that is typical of Gröbner basis calculations. This poses a particular challenge for the training of reinforcement learning models.

Dependence on s is subtle: For $n = 3$, there is is a spike in difficulty at four generators, followed by a drop/leveling off, and a slow increase after that (Figures 2 and 3). The spike is even more pronounced in $n > 3$ variables, where it occurs instead at $n + 1$ generators. The leveling off is consistent with the hypothesis that a low-degree generator, which is more likely for larger s, makes the problem easier, but this is eventually counteracted by the fact that increasing s always increases the minimum number of polynomial additions required. The fact that `weighted` is easier than `uniform` across values of d and s also supports this hypothesis.

Difficulty increases relatively slowly with d: The growth appears to be either linear or slightly sub-linear in d in the range tested (Figures 2 and 3).

Zero dimensional ideals are rare: (Table 2) For $n = 3$, $d = 20$, the hardest distribution is $s = 4$, in which case $.05\%$ of the ideals were zero dimensional. This increased to 21.2% using the `weighted` distribution and increasing to $s = 10$, which is still relatively rare. This also supports the hypothesis that the appearance of a generator of low degree makes the problem easier.

Table 1. Number of polynomial additions for different selection strategies on the same samples of 10000 ideals. Distributions are n-5-10 weighted. Table entries show mean [stddev].

n	FIRST	DEGREE	NORMAL	SUGAR
2	36.4 [7.24]	32.3 [5.71]	32.0 [5.49]	32.4 [6.15]
3	52.8 [17.9]	42.2 [13.2]	42.4 [13.1]	44.2 [15.1]
4	86.3 [40.9]	63.8 [28.5]	66.5 [29.8]	70.0 [32.9]
5	151. [85.7]	109. [58.8]	117. [64.4]	120. [68.7]
6	280. [174.]	198. [118.]	221. [132.]	223. [143.]
7	527. [359.]	379. [240.]	435. [277.]	430. [296.]
8	1030 [759.]	760. [510.]	887. [588.]	863. [639.]

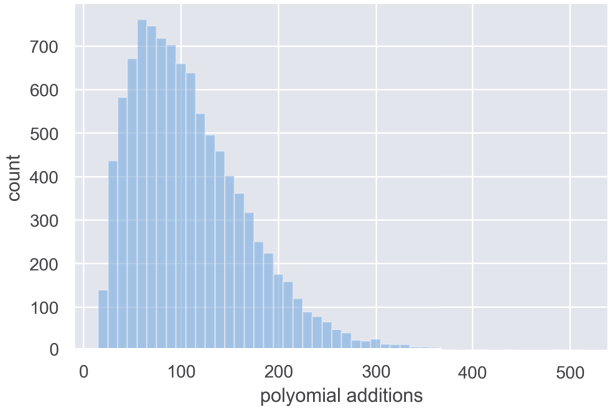


Figure 1. Histogram of polynomial additions in 5-5-10 weighted following Degree selection over 10000 samples.

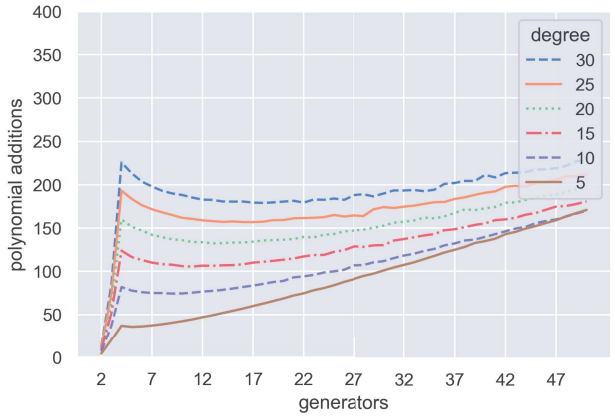


Figure 2. Average number of polynomial additions following Degree selection in $n = 3$ weighted. Each degree and generator point is the mean over 10000 samples for $s \leq 20$ and 1000 samples for $s > 20$.

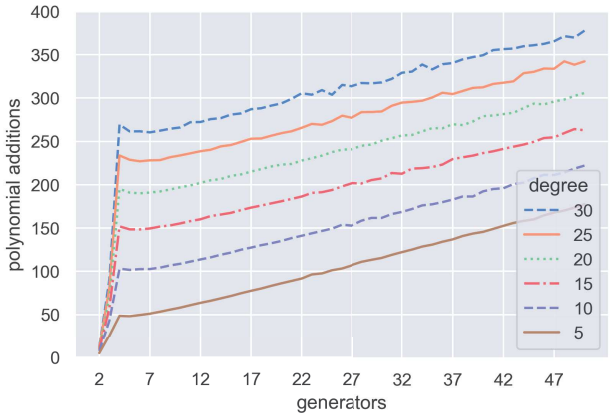


Figure 3. Average number of polynomial additions following Degree selection in $n = 3$ uniform. Each degree and generator point is the mean over 10000 samples for $s \leq 20$ and 1000 samples for $s > 20$.

Table 2. Dimension of the binomial ideals (i.e., the dimension of the solution set of the corresponding system of equations), in a sample of 10000 ($n = 3$, $d = 20$).

	WEIGHTED		UNIFORM	
dim	$s = 10$	$s = 4$	$s = 10$	$s = 4$
0	2121	178	58	5
1	7657	6231	8146	2932
2	223	3592	1797	7064

4. Experimental Setup

We train a neural network model to perform pair selection in Buchberger's algorithm.

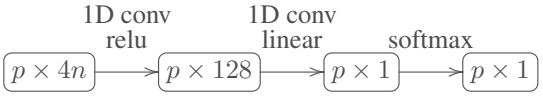
4.1. Network Structure

We represent a state $S_t = (G_t, P_t)$ as a matrix whose rows are obtained by concatenating the exponent vector of each pair. For n variables and p pairs, this results in a matrix of size $p \times 4n$. The environment is now partially observed, as the observation does not include the coefficients.

Example 2. *Let $n = 3$, and consider the state given by $G = \{xy^6 + 9y^2z^4, z^4 + 13z, xy^3 + 91xy^2\}$, where the terms of each binomial are shown in grevlex order, and $P = \{(1, 2), (1, 3), (2, 3)\}$. Mapping each pair to a row yields*

$$\left[\begin{array}{cccccc|cccccc} 1 & 6 & 0 & 0 & 2 & 4 & 0 & 0 & 4 & 0 & 0 & 1 \\ 1 & 6 & 0 & 0 & 2 & 4 & 1 & 3 & 0 & 1 & 2 & 0 \\ 0 & 0 & 4 & 0 & 0 & 1 & 1 & 3 & 0 & 1 & 2 & 0 \end{array}\right]$$

Our agent uses a policy network that maps each row to a single preference score using a series of dense layers. We implement these layers as 1D convolutions with 1×1 kernel in order to compute the preference score for all pairs simultaneously. The agent's policy, which is a probability distribution on the current pair set, is the softmax of these preference scores. In preliminary experiments, network depth did not appear to significantly affect performance, so we settled on the following architecture:



Due to its simplicity, it would in principle be easy to deploy this model in a production implementation of Buchberger's algorithm. The preference scores produced by the network could be used as sort keys for the pair set. Each pair would only need to be processed once, and we expect the relatively small matrix multiplies in this model to add minimal overhead in a careful implementation. In fact, most of the improvement was already achieved by a model with only four hidden units (see supplement).

However, given that real time performance of Buchberger's algorithm is highly dependent on sophisticated implementation details, we exclusively focus on implementation independent metrics, and defer the testing of real time performance improvements to future work.

4.2. Value Functions

A general challenge for policy gradient algorithms is the large variance in the estimate of expected rewards. This is exacerbated in our context by the large variance in difficulty of computing a Gröbner basis of different ideals from the

Table 3. Agent performance versus benchmark strategies in 3 variables and degree 20. Each line is a unique agent trained on the given distribution. Performance is mean[stddev] on 10000 new randomly sampled ideals from that distribution. Training times were 16 to 48 hours each on a c5n.xlarge instance through Amazon Web Services. Smaller numbers are better.

s	DISTRIBUTION	FIRST	DEGREE	NORMAL	SUGAR	RANDOM	AGENT	IMPROVEMENT
10	WEIGHTED	187.[73.1]	136.[50.9]	136.[51.2]	161.[66.9]	178.[68.3]	85.6[27.3]	37% [46%]
4	WEIGHTED	210.[101.]	160.[64.5]	160.[66.6]	185.[87.2]	203.[97.8]	101.[44.9]	37% [30%]
10	UNIFORM	352.[117.]	197.[55.7]	198.[57.1]	264.[88.5]	318.[103.]	141.[42.8]	28% [23%]
4	UNIFORM	317.[130.]	195.[70.0]	194.[70.0]	265.[107.]	303.[122.]	151.[56.4]	22% [19%]

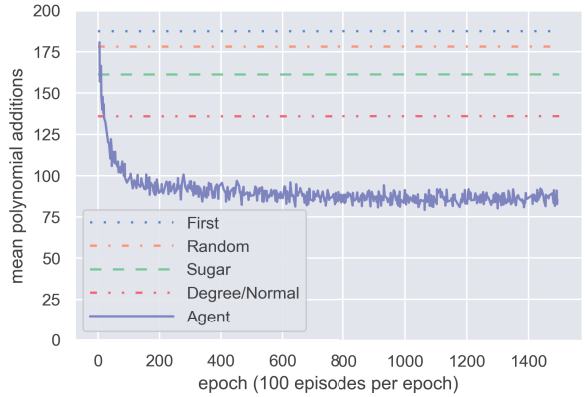


Figure 4. Mean performance during each epoch of training on the 3-20-10 weighted distribution. Dashed lines indicate mean performance of benchmark strategies on 10000 random ideals. Total training time was 16 hours on a c5n.xlarge instance through Amazon Web Services. Smaller numbers are better.

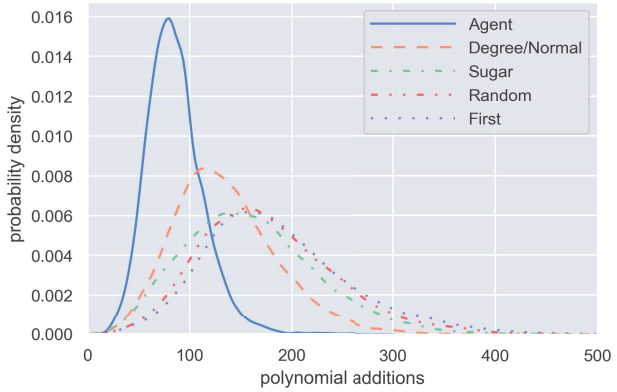


Figure 5. Estimated distribution of polynomial additions per ideal in the 3-20-10 weighted distribution for the fully trained agent from Figure 4, compared to benchmark strategies. Smaller numbers are better. (10000 samples, computed using kernel density estimation)

same distribution. We address this using Generalized Advantage Estimation (GAE) (Schulman et al., 2016), which uses a value function to produce a lower-variance estimator of expected returns while limiting the bias introduced. Our value function $V(G, P)$ is the number of polynomial additions required to complete a full run of Buchberger's algorithm starting with the state (G, P), using the Degree strategy. This is computationally expensive but significantly improves performance.

4.3. Training Algorithm

Our agents are trained with proximal policy optimization (PPO) (Schulman et al., 2017) using a custom implementation inspired by (Achiam, 2018). In each epoch we first sample 100 episodes following the current policy. Next, GAE with $\lambda = 0.97$ and $\gamma = 0.99$ is used to compute advantages, which are normalized over the epoch. Finally, we perform at most 80 gradient updates on the clipped surrogate PPO objective with $\epsilon = 0.2$ using Adam optimization with learning rate 0.0001. Early-stopping is performed when the sampled KL-divergence from the last policy exceeds 0.01.

4.4. Data Generation

There are no fixed train or test sets. Instead, the training and testing data are generated online by a function that builds an ideal at random from the distribution. The large size of the distributions prevents any over-fitting to a particular subset of ideals. For example, even ignoring coefficients, the total number of ideals in 3-20-10 weighted is roughly 10^{55}. The agent trained in Figure 4 saw 150000 ideals from this set generated at random during training. This agent was tested by running on a completely new generated set of 10000 ideals to produce the results in Table 3.

5. Experimental Results

Table 3 shows the final performance of agents which have been trained on several distributions with $n = 3$, $d = 20$. All agents use 22% to 37% fewer polynomial additions on average than the best benchmark strategies, and reduce the standard deviation in the number of polynomial additions by 19% to 46%. The improvement on `uniform` distributions, which tend to produce ideals of higher average difficulty, is not as large as the improvement on `weighted` distributions.

Figure 5 gives a more detailed view of the distribution of polynomial additions per ideal performed by the trained agent. Figure 4 shows the rapid convergence during training.

5.1. Interpretation

We have identified several components of the agents strategy: (a) the agent is mimicking Degree, (b) the agent prefers pairs whose S-polynomials are monomials, (c) the agent prefers pairs whose S-polynomials are low degree.

On 10000 sample runs of Buchberger's algorithm using a trained agent on 3-20-10 weighted, the average probability that the agent selected a pair which could be chosen by Degree was 43.5%. If there was a pair in the list whose S-polynomial was a monomial, the agent picked such a pair 31.7% of the time. The probability that the agent selected a pair whose S-polynomial had minimal degree (among S-polynomials), was 48.3%.

It is notable that (b) and (c) are not standard selection heuristics. When we hard-coded the strategy of selecting a pair with minimal degree S-polynomial, which we call TrueDegree, the average number of additions (3-20-10 weighted, 10000 samples) was 120.3, a 12% improvement over the Degree strategy. On the other hand, for the strategy which follows Degree but will first select any S-polynomial which is monomial, the average number of additions was 134.2, a 1.2% improvement over Degree. While neither hard-coded strategy achieves the 37% improvement of the agent over Degree, it is notable that these insights from the model led to understandable strategies that beat our benchmark strategies in this domain.

5.2. Variants of the Model

We found that the model performance decreased when we made any of the following modifications: only allowed the network to see the lead monomials of each pair; removed the value function; or substituted the value function with a naive "pairs left" value function which assigned $V(G, P) = |P|$. See Table 4. However, all of these trained models still outperform the best benchmark strategy, which is Degree.

Table 4. Performance of variants of the model. Entries show mean [stddev] of polynomial additions and performance drop relative to the original model on samples of 10000 ideals from 3-20-10 weighted distribution. Original model is 85.6 [27.3].

AGENT	ADDITIONS	DROP
PAIRSLEFT VALUE FUNCTION	95.2 [32.7]	11.2%
NO VALUE FUNCTION	103.2 [35.9]	20.6%
LEAD MONOMIAL ONLY	90.0[29.4]	5.4%

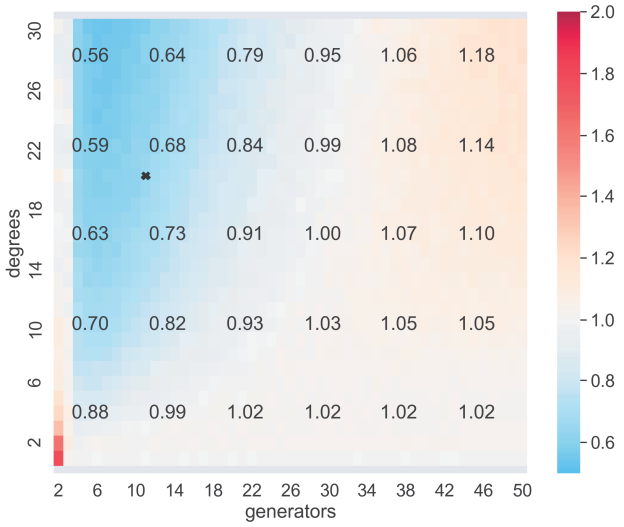


Figure 6. Testing a single agent on 3-d-s weighted distribution as d and s vary. Agent is trained on 3-20-10 weighted, indicated with an "X." Numbers are the ratio of mean polynomial additions by the agent to that of the best benchmark strategy, with numbers less than 1 indicating better performance by the agent. The agent was tested on 1000 random ideals in each distribution, and the strategies were tested on 10000 for $s \leq 20$ and 1000 for $s > 20$.

Table 5. Agent performance outside of training distribution. Performance is mean[stddev] on a sample of 10000 random ideals.

3-20-10 DISTRIBUTION

TRAIN \ TEST	WEIGHTED	UNIFORM
WEIGHTED	85.6[27.3]	140.[45.7]
UNIFORM	89.3[29.0]	141.[42.8]

3-20-4 DISTRIBUTION

TRAIN \ TEST	WEIGHTED	UNIFORM
WEIGHTED	101.[44.9]	158.[67.9]
UNIFORM	107.[42.6]	151.[56.4]

5.3. Generalization across Distributions

A major question in machine learning is the ability of a model to generalize outside of its training distribution. Table 5 shows reasonable generalization between `uniform` and `weighted` distributions.

Figure 6 shows that a model trained on 3-20-10 weighted has similar performance at nearby values of d and s, as compared to the performance of the best benchmark strategy. Agents can also be trained on a mix of distributions by randomly selecting a training distribution at each epoch. Choosing uniformly from $5 \leq d \leq 30$ and $4 \leq s \leq 20$ yields

agents with 1-5% worse performance at 3-20-10 weighted and 1-10% better performance away from it, though performance does eventually degrade as in Figure 6.

5.4. Future directions

It would be interesting to extend these results to more variables and non-binomial ideals. In the interest of establishing a simple proof-of-concept, we have left a thorough investigation of these questions for future research, but we have done some preliminary experiments.

In the direction of increasing n, we trained and tested our model (with the same hyperparameters) on binomial ideals in five variables. The agents use on average 48% fewer polynomial additions than the best benchmark strategy in the 5-10-10 weighted distribution, 28% fewer in 5-5-10 weighted, and 11% fewer in the 5-5-10 uniform distribution. We could not increase the degree further or perform a full hyperparameter search due to computational constraints.

In the non-binomial setting, we tested our agent on a toy model for sparse polynomials. We sampled generators for our random ideals by drawing a binomial from the `weighted` distribution, then adding k monomial terms drawn from the same distribution, where k is sampled from a Poisson distribution with parameter λ.

The fully trained agent from Figure 4 had mixed results when tested on this non-binomial distribution. The distribution for the agent's performance is bimodal, with it outperforming all benchmarks on many ideals but behaving essentially randomly on others, see Figure 7 and Figure 8. As a result, the agent significantly underperformed the best benchmark on average, see Table 6, but still had the best median performance for $\lambda = 0.1, 0.2$. TrueDegree, the strategy derived from the model in Section 5.1, outperforms the best benchmark in mean by 19% to 33% for all λ.

Table 6. Mean polynomial additions of several strategies tested on samples of 10000 non-binomial ideals. Agent was trained on the 3-20-10 weighted binomial distribution. Benchmark refers to the Normal strategy, the best performing benchmark in this case.

λ	AGENT	TRUEDEGREE	BENCHMARK
0.1	4.17E+3	625.	872.
0.3	2.16E+4	3138.	4693.
0.5	4.96E+4	8436.	1.14E+4

Finally, the value function used for training with GAE, one of the main contributors to our performance improvement, effectively squares the complexity by doing a full rollout at every step. Therefore, a more efficient modeled value function is crucial for scaling these results both to higher numbers of variables and to non-binomials.

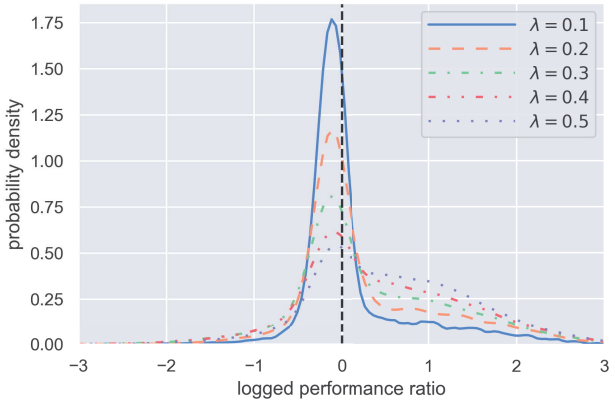


Figure 7. Agent performance on non-binomial ideals. The logged performance ratio is the base-10 log of agent polynomial additions to best benchmark strategy on each of a sample of 10000 ideals. Values less than 0 indicate better performance by the agent.

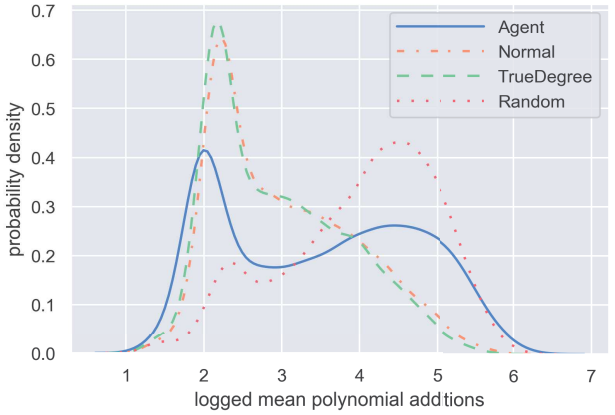


Figure 8. Estimated distribution of base-10 log of polynomial additions per ideal with $\lambda = 0.5$, compared to benchmark strategies. (10000 samples, computed using kernel density estimation)

6. Conclusion

We have introduced the Buchberger environment, a challenging reinforcement learning problem with important ramifications for the performance of computer algebra software. We have identified binomial ideals as an interesting domain for this problem that is tractable, maintains many of the problem's interesting features, and can serve as a benchmark for future research.

Standard reinforcement learning algorithms with simple models can develop strategies that improve over state-of-the-art in this domain. This illustrates a direction in which modern developments in machine learning can improve the performance of critical algorithms in symbolic computation.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback and corrections, and David Eisenbud for useful discussions. Dylan Peifer and Michael Stillman were partially supported by NSF Grant No. DMS-1502294, and Daniel Halpern-Leistner was partially supported by NSF Grant No. DMS-1762669.

References

Abłamowicz, R. Some applications of Gröbner bases in robotics and engineering. In Bayro-Corrochano, E. and Scheuermann, G. (eds.), *Geometric Algebra Computing: in Engineering and Computer Science*, pp. 495–517. Springer London, London, 2010.

Achiam, J. Spinning Up in Deep Reinforcement Learning. 2018. URL https://spinningup.openai.com.

Alvarez, A. M., Louveaux, Q., and Wehenkel, L. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

Arkun, Y. Detection of biological switches using the method of Gröbner bases. *BMC Bioinformatic*, 20, 2019.

Arnold, E. A. Modular algorithms for computing Gröbner bases. *J. Symbolic Comput.*, 35(4):403–419, 2003.

Bayer, D. and Mumford, D. What can be computed in algebraic geometry? In *Computational algebraic geometry and commutative algebra (Cortona, 1991)*, Sympos. Math., XXXIV, pp. 1–48. Cambridge Univ. Press, Cambridge, 1993.

Bayer, D. and Stillman, M. On the complexity of computing syzygies. *J. Symbolic Comput.*, 6(2-3):135–147, 1988.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *CoRR*, abs/1811.06128, 2018.

Bettale, L., Faugère, J.-C., and Perret, L. Cryptanalysis of HFE, Multi-HFE and Variants for Odd and Even Characteristic. *Designs, Codes and Cryptography*, 69(1):1 – 52, 2013.

Buchberger, B. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

Buchberger, B. 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.

CoCoA. A system for doing computations in commutative algebra, Abbott, J., Bigatti, A. M., and Robbiano, L., 2019. URL http://cocoa.dima.unige.it.

Conti, P. and Traverso, C. Buchberger algorithm and integer programming. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pp. 130–139. Springer, 1991.

Cox, D. A., Little, J., and O'Shea, D. *Using algebraic geometry*. Graduate Texts in Mathematics. Springer, New York, second edition, 2005.

Cox, D. A., Little, J., and O'Shea, D. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, Cham, fourth edition, 2015.

Diaconis, P. and Sturmfels, B. Algebraic algorithms for sampling from conditional distributions. *Ann. Statist.*, 26 (1):363–397, 1998.

Duff, T., Kohn, K., Leykin, A., and Pajdla, T. PLMP - point-line minimal problems in complete multi-view visibility. *CoRR*, abs/1903.10008, 2019.

Eder, C. and Faugère, J.-C. A survey on signature-based algorithms for computing Gröbner bases. *J. Symbolic Comput.*, 80(3):719–784, 2017.

Faugère, J.-C. A new efficient algorithm for computing Gröbner bases (F_4). *J. Pure Appl. Algebra*, 139(1-3): 61–88, 1999.

Faugère, J.-C. 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*, pp. 75–83. ACM, New York, 2002.

Faugère, J.-C., Safey El Din, M., and Spaenlehauer, P.-J. Computing loci of rank defects of linear matrices using Gröbner bases and applications to cryptology. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, pp. 257–264. ACM, New York, 2010.

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

Giovini, A., Mora, T., Niesi, G., Robbiano, L., and Traverso, C. "One sugar cube, please" or selection strategies in the Buchberger algorithm. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pp. 49–54. ACM, New York, 1991.

Gray, J. A simple introduction to Gröbner basis methods in string phenomenology. *Adv. High Energy Phys.*, 2011:12, 2011.

Hilbert, D. Über die vollen Invariantensysteme. *Math. Ann.*, 42(3):313–373, 1893.

Huang, Z., England, M., Wilson, D. J., Bridge, J., Davenport, J. H., and Paulson, L. C. Using machine learning to improve cylindrical algebraic decomposition. *Mathematics in Computer Science*, 13(4):461–488, 2019.

Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. Learning to branch in mixed integer programming. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 724–731. AAAI Press, 2016.

Koh, J. Ideals generated by quadrics exhibiting double exponential degrees. *J. Algebra*, 200(1):225–245, 1998.

Lin, Z., Xu, L., and Wu, Q. Applications of Gröbner bases to signal and image processing: a survey. *Linear Algebra Appl.*, 391:169–202, 2004.

Macaulay2. A software system for research in algebraic geometry, Grayson, D. and Stillman, M., 2019. URL http://www.math.uiuc.edu/Macaulay2/.

Magma. Algebra system, Bosma, W., Cannon, J. and Playoust, C., 2019. URL http://magma.maths.usyd.edu.au.

Maple. Maplesoft, 2019. URL https://maplesoft.com.

Mathematica. Wolfram, S., 2019. URL https://www.wolfram.com/mathematica.

Mayr, E. W. and Meyer, A. R. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.*, 46(3):305–329, 1982.

Mora, T. *Solving polynomial equation systems. II*, volume 99 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2005.

Roune, B. H. and Stillman, M. Practical Gröbner basis computation. In *Proceedings of the 2012 International Symposium on Symbolic and Algebraic Computation*, pp. 203–210. ACM, New York, 2012.

SageMath. The Sage Mathematics Software System, 2019. URL https://www.sagemath.org.

Sala, M., Mora, T., Perret, L., Sakata, S., and Traverso, C. (eds.). *Gröbner bases, coding, and cryptography*. Springer-Verlag, Berlin, 2009.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *Proceeding of the 4th International Conference on Learning Representations (ICLR 2016)*, 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Singular. A computer algebra system for polynomial computations, Decker, W., Greuel, G.M., Pfister, G., and Schönemann, H., 2019. URL http://www.singular.uni-kl.de.

Sullivant, S. *Algebraic statistics*. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2018.

Xu, W., Hu, L., Tsakiris, M. C., and Kneip, L. Online stability improvement of Gröbner basis solvers using deep learning. *2019 International Conference on 3D Vision (3DV)*, pp. 544–552, 2019.