Reinforcement Learning in Buchberger’s Algorithm

Dylan Peifer
Cornell University

**Summary**

1. Buchberger’s algorithm is the standard method for computing a Gröbner basis, and highly-tuned and optimized versions are a critical part of many computer algebra systems.
2. The efficiency of Buchberger’s algorithm strongly depends on a choice of selection strategy that determines the order in which S-polynomials are processed.
3. By phrasing Buchberger’s algorithm as a reinforcement learning problem and applying standard reinforcement learning techniques we can learn new selection strategies that can match or beat the existing state-of-the-art.

**Gröbner Bases**

Let $R = K[x_1, \ldots, x_n]$ be a polynomial ring over some field $K$ and $I = \langle f_1, \ldots, f_m \rangle \subseteq R$ be a nonzero ideal generated by polynomials $f_1, \ldots, f_m$.

Given a monomial order, a Gröbner basis $G$ of $I$ is a subset $\langle g_1, g_2, \ldots, g_n \rangle \subseteq I$ such that any of the following equivalent conditions hold:

(i) $g_i^2 = 0 \iff i \in I$
(ii) $g_i^2$ is unique for all $i \in R$
(iii) $\langle LT(g_1), LT(g_2), \ldots, LT(g_n) \rangle = \langle LT(f) \rangle$

where $LT(g)$ is the lead term of $g$ with respect to the monomial order and $i^2 \rightarrow r$ is the remainder under polynomial long division of $i$ by the polynomials in $G$.

**Buchberger’s Algorithm**

*Theorem (Buchberger’s Criterion):* Let $G = \lbrace g_1, g_2, \ldots, g_n \rbrace$ generate some ideal $I$. If $S(g_i, g_j) = \langle \text{lcm}(LT(g_i), LT(g_j)) \rangle$ for all pairs $g_i, g_j$, then $G$ is a Gröbner basis of $I$.

**Algorithm 1** Buchberger’s Algorithm

```plaintext
input: a set of polynomials $\lbrace f_1, \ldots, f_m \rbrace$
output: a Gröbner basis $G$ of $I = \langle f_1, \ldots, f_m \rangle$
procedure Buchberger(f_1, \ldots, f_m)
    $G = \lbrace f_1, \ldots, f_m \rbrace$  // the current basis
    while $P \neq 0$
do
        $t \leftarrow \text{select}(P)$
        $P \leftarrow P \cup \lbrace \langle f, g \rangle \rbrace$
        $r \leftarrow S(t, G)$
        if $r \neq 0$
            $G \leftarrow G \cup \lbrace r \rbrace$
        $P \leftarrow P \cup \lbrace \langle f, r \rangle \rbrace$
    end while
return $G$
end procedure
```

**Selection Strategies in Buchberger’s Algorithm**

The implementation of select does not affect correctness of Buchberger’s algorithm, but it is critical for efficiency. In general, good selection strategies pick “small” pairs first.

- **First**: among the pairs with minimal $j$, pick the pair with smallest $i$
- **Degree**: pick the pair with smallest degree of lcm(LT($i$), LT($j$))
- **Normal**: pick the pair with smallest lcm(LT($i$), LT($j$)) in the monomial order
- **Sugar**: pick the pair with smallest sugar degree of lcm(LT($i$), LT($j$))

**Example 1: 5 Homogeneous Binomial Quadratics**

Let $R = K[x, y, z]$ with grevlex ordering. Consider ideals $I$ generated by 5 random binomials of degree 2. Performed Buchberger with no pair elimination.

$G = \lbrace xy - 2, x^2 + y^2, xy + 8x - z^2 \rbrace$

In each epoch we perform 10 rollouts, compute future rewards for each state on each trajectory, baseline by the size of the current pair set in each state, and normalize these scores before performing the policy gradient step. Total training time was 45 minutes.

By examining the agent’s preferences for picking different pairs we see that it has learned an approximation to degree selection, the best strategy in this case.

**Example 2: 10 Nonhomogeneous Binomials of Degree ≤ 20**

Let $R = K[x, y, z]$ with grevlex ordering. Consider ideals $I$ generated by 10 random binomials of degree ≤ 20. Performed Buchberger with Gebauer-Möller pair elimination.

$G = \lbrace xy - 2, x^2 + y^2, xy + 8x - z^2 \rbrace$

Convert the state $S = \langle G, P \rangle$ to a matrix with rows the exponent vectors of the lead terms of each pair. Each step input this matrix to a neural network that learns the policy function.

After 12 hours of training the agent has learned a policy that averages 20% fewer pair reductions than the best known selection strategies.

---

**Reinforcement Learning**

Reinforcement learning problems can be phrased as the interaction of an agent and an environment.

- **Environment**: state $S_t$, reward $R_t$, transition $P_{t+1}$. Agent observes $S_t$ and receives $R_t$.
- **Agent**: action $A_t$, policy $\pi$, state-action pair $(S_t, A_t)$.

The agent chooses actions and the environment processes actions and gives back the updated state and a reward. The agent wants to maximize its return, which is the amount of reward it gets in the long run.

<table>
<thead>
<tr>
<th>CartPole</th>
<th>Chess</th>
<th>Buchberger</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 2 0 2 0</td>
<td>1 1 0 2 0 2 0</td>
<td>2 0 0 0 2 0 2</td>
</tr>
<tr>
<td>1 1 0 2 0 2 0</td>
<td>1 1 0 2 0 2 0</td>
<td>2 0 0 0 2 0 2</td>
</tr>
<tr>
<td>1 1 0 2 0 2 0</td>
<td>2 0 0 0 2 0 2</td>
<td>1 1 0 2 0 2 0</td>
</tr>
<tr>
<td>1 1 0 2 0 2 0</td>
<td>1 1 0 2 0 2 0</td>
<td>2 0 0 0 2 0 2</td>
</tr>
</tbody>
</table>

**CartPole Policy Gradient**

A policy $\pi$ is a function $\pi: A \times S \rightarrow K$ given by $\pi(a|s) = P(A_t = a | S_t = s)$.

Given an environment, the goal of reinforcement learning is to find a policy $\pi$ that maximizes $J_\pi(s) = \mathbb{E}_{\pi} \left[ \sum_{t=1}^{\infty} \gamma^t R_t \right]$. This is known as the expected return along trajectories obtained by following the policy $\pi$ one time through the environment, and the return of a trajectory is the sum of rewards along the trajectory.

<table>
<thead>
<tr>
<th>Policy Gradient</th>
<th>Example</th>
<th>First</th>
<th>Normal</th>
<th>Sugar</th>
<th>Random</th>
<th>Last</th>
<th>Strange</th>
</tr>
</thead>
<tbody>
<tr>
<td>wheelie</td>
<td>1 18</td>
<td>14</td>
<td>9</td>
<td>41</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reimer</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>katsura</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>eyes</td>
<td>67</td>
<td>64</td>
<td>94</td>
<td>149</td>
<td>295</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nosie</td>
<td>71</td>
<td>71</td>
<td>103</td>
<td>375</td>
<td>375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cyclet</td>
<td>666</td>
<td>620</td>
<td>343</td>
<td>793</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>katsura7</td>
<td>164</td>
<td>164</td>
<td>284</td>
<td>284</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>katsura4-lex</td>
<td>25</td>
<td>46</td>
<td>19</td>
<td>29</td>
<td>40</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>cyclet6-lex</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For some learning rate $\alpha$, we can incrementally improve the policy. Intuitively, we should increase the probability of taking the action we chose proportional to the future reward we received and the derivative of the log probability of choosing that action again.