Deletion Schemes for Classifier Systems

Tim Kovacs
School of Computer Science
The University of Birmingham
T.Kovacs@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~tyk
Introduction

- Talk is about low level mechanics of a Learning Classifier System (LCS).

- Goal is to optimise a particular mechanism in a particular system: how do we select rules to delete in Wilson's XCS?

- This is apparently the first reported study devoted to deletion in LCS.

- We’ll see that XCS has a special requirement for deletion.
  - Catering to it speeds genetic search and reduces population size.
Reproduction in GAs

• Generational
  – Replaces entire population during a reproductive event.

• Steady State GA
  – Replaces only a few individuals during an event.
  – To maintain a fixed population size we need to delete as many as we’ve generated.
  – …so in addition to selecting individuals to reproduce, we need to select individuals to delete.
• Despite this classification there are really two issues:
  – How much of one generation gets copied to the next.
  – Whether we apply selective pressure when deleting rules.
Wilson (1995)* used two schemes with XCS:

**t1** – each rule estimates the average number of rules which match the strings it matches. Its deletion probability is proportional to this estimate.

**t2** – as t1 except deletion is much more likely if the rule’s fitness is below a threshold.

1. **Strength** (prediction): 
   
   \[ s_j \leftarrow s_j + \beta (R - s_j) \]
   
   where \(0 < \beta \leq 1\) controls the learning rate and \(R\) is the reward from the environment.

2. **Prediction error:** 
   
   \[ \varepsilon_j \leftarrow \varepsilon_j + \beta \left( \frac{|R - s_j|}{R_{max}} - \varepsilon_j \right) \]
   
   where \(R_{max}\) is the highest reward possible in any state.

3. **Accuracy:** 
   
   \[ \kappa_j = \begin{cases} 
   1 & \text{if } \varepsilon_j \leq \varepsilon_o \\
   e^{(\ln \alpha)(\varepsilon_j - \varepsilon_o)/\varepsilon_o} \times 0.1 & \text{otherwise}
   \end{cases} \]
   
   where \(\varepsilon_o\) controls the tolerance for prediction error and \(\alpha < 1\) controls the rate of decline in accuracy when \(\varepsilon_o\) is exceeded.

4. **Relative accuracy:** 
   
   \[ \kappa'_j = \frac{\kappa_j}{\sum_{x \in [A]} \kappa_x} \]

5. **Fitness:** 
   
   \[ F_j \leftarrow F_j + \beta (\kappa'_j - F_j) \]
• We can think of accuracy as the consistency of a rule’s strength over time. It takes many samples to estimate accuracy.

• A rule’s fitness is initially 0. It must be used many times to gain fitness because first its accuracy must be evaluated.
• A new scheme called t3 takes XCS’s fitness calculation delay into account:
  – If a rule has been updated \(< x\) times use t1.
  – Otherwise use t2 (i.e. add low fitness penalty).

• \(x\) is called t3’s delay.

• t3 protects new classifiers from the deletion penalty so that accurate ones have time to gain fitness.