ROPER:
A Genetic ROP-Chain Compiler Targetting Embedded Devices

Olivia Lucca Fraser

NIMS Lab, Dalhousie University

April 22, 2017

Progress Report # 7

3. Overview of Recent Results

In our last meeting, we saw how ROPER can evolve ROP-chains that perform a variety of tasks. Most interestingly, it can breed chains that exhibit learned or adaptive behaviour, and can solve a traditional benchmark classification problem: the classification of Iris specimens into species by petal and sepal measurements.

It's important to remember that this is just an example of ROPER's capabilities. What it shows is that it is possible, in principle, to automatically generate ROP-chain payloads that behave in an adaptive and 'intelligent' fashion.

Here, we'll be looking more closely at a particular method for encouraging population diversity and accelerating our populations' search for optimal solutions. Unless otherwise noted, we'll be focussing on the Iris classification task reviewed in our last meeting, but the principles are general, should be applicable to virtually any problem space on which ROPER is deployed.

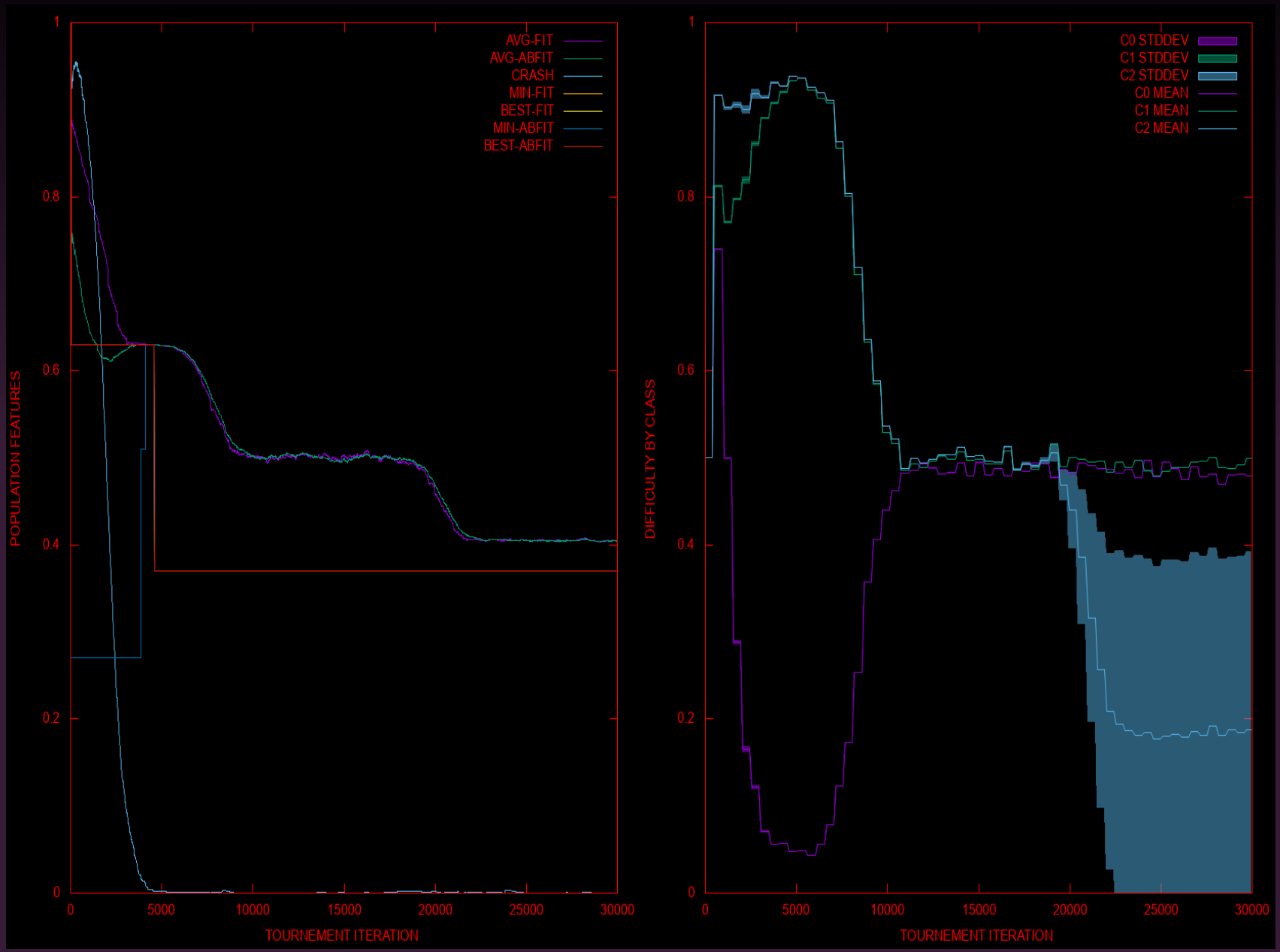
4. Low Hanging Fruit & its Consequences for Diversity

- A challenge facing any machine learning technique is to avoid getting trapped in merely *local* optima.
- This can happen, for example, if it hyperspecializes on a particularly simple portion – the "low hanging fruit" – of the problem set, while failing to adapt to more difficult problems.
- The phenomenon is analogous to a natural population over-adapting to a particularly hospitable niche.
- But in the wild, this is offset by an increase in competition and crowding, which increase the selective pressure acting on formerly hospitable niches. Low-hanging fruit doesn't last very long.

5. Implementing Niching through Fitness Sharing

- In order to address this issue, we first need to keep track of where, in the problem space, the overfitting occurs. Where is the low-hanging fruit?
- To do this, we tag each problem in our space with a ‘difficulty’ field, which keeps track of how our specimens perform on it, on average.
- Since the whole point of tracking difficulty is to have it transform dynamically over the course of the evolution, we’ll update these scores every so many iterations.
- On the next slide, we plot the progress of the population’s best and average fitness scores on the left, and the difficulty rations of our problems on the right – plotted by class mean and standard deviation.

6. Tracking Niches without Crowding

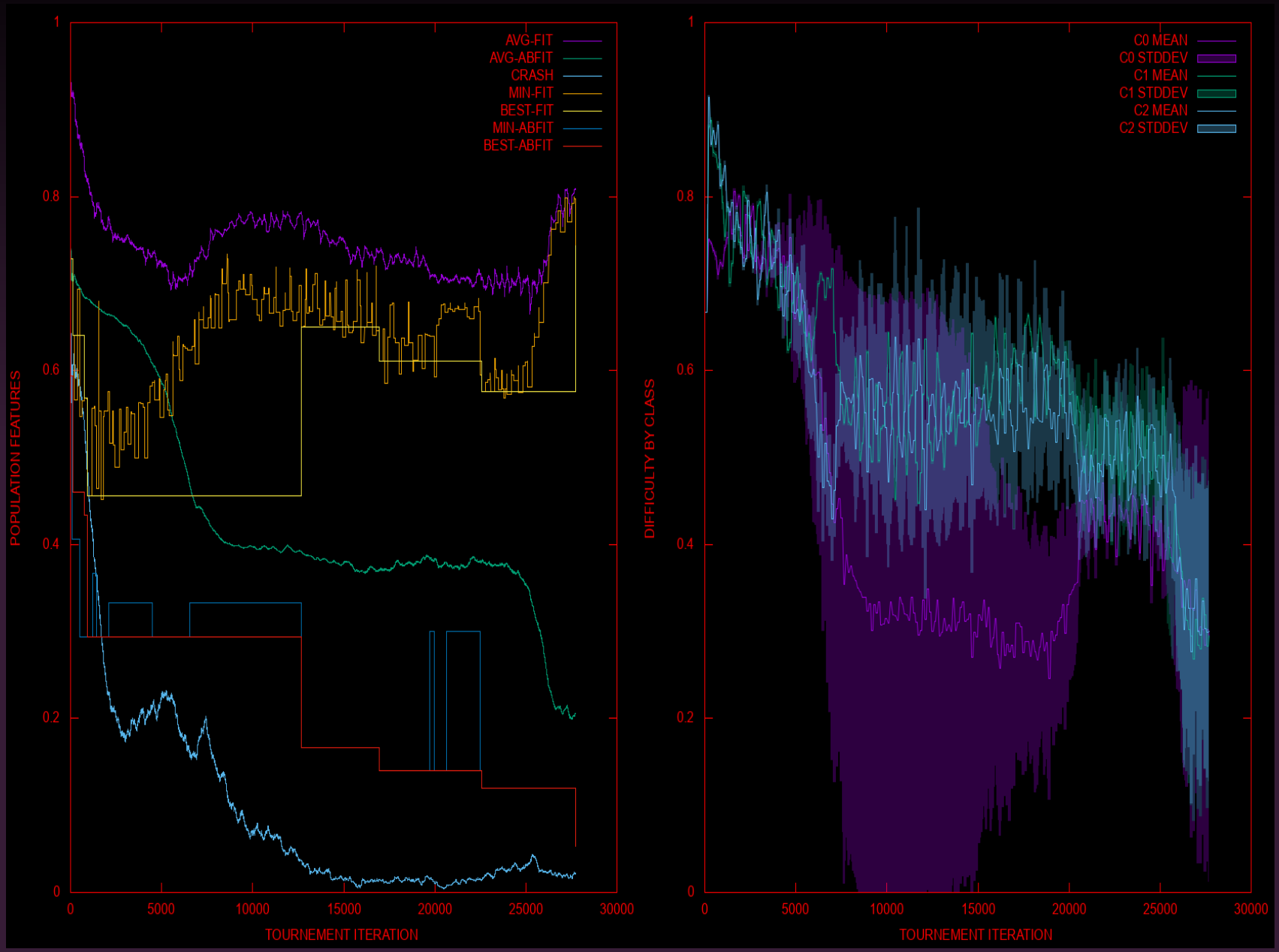


7. Crowding Implemented as Fitness Sharing

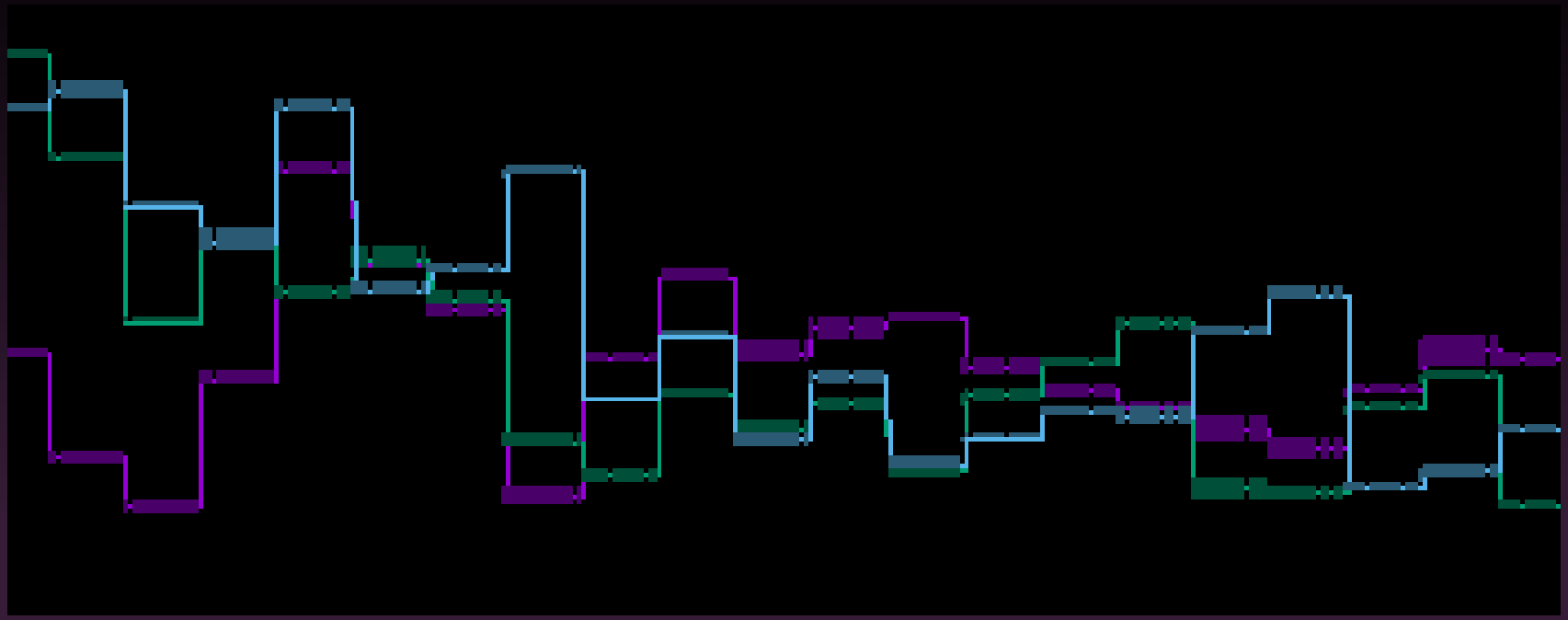
- We haven't yet changed anything in the way each specimen's fitness is evaluated. The graph only shows us how the population is performing, with respect to each class of problems.
- But we can use this information to tweak our fitness function in ways relevant to niching.

- All that we need to do is to scale the fitness points awareded for each problem with respect to that problem's difficulty. The rewards for solving 'difficult' problems (uncrowded niches) will be greater than those awarded for solving 'easy' problems (crowded niches).

8. Niching with Crowding



9. Dynamic Braiding of Difficulty by Niche



A detailed view of the intricate braiding of niche availability that takes place once we enable fitness sharing. The image is an enlargement of the right panel of the graph on the last slide, focussing on the region between iterations 3000 and 5000.

Because the environment perennially adjusts to the population's strengths and weaknesses, no specimen encounters the exact same fitness space as its distant ancestors, and cannot benefit from overfitting, or a diet of exclusively low-hanging fruit.

10. The Next Step

```
hdr: 33, typ: SNEK::OUTPUT, len: 12
WORDS: (0 0 0)
..............
..............
..............
..............
..............
......@.......
.......*......
.......*......
.......*......
.......*......
..............
..............
..............
..............
SCORE: (24)
PARAMS: (0 8 64 1 1)
```

I am currently working to train ROPER to handle dynamic problem spaces, beginning with simple games, like Snake and Pong. These are ideal candidates because the relevant features of the game state can be fully represented with just a small number of variables, and similarly low-dimensional controls can be sent back to the game.

The first step, already complete, is to set up an interface for ROPER and the game to communicate, over a TCP channel. For the duration of each game, the game sends packets of information to ROPER, representing its game state. The active chain processes this state information, and returns a direction for the game. The cycle repeats until the game is finished.