Report for CSCI6506 Sandbox 3

Jéssica Pauli de Castro Bonson (B00617515)

Implementation of the canonical GP for multi-class classification improved with sampling, class-wise detection rate (DR) and modular behavior used the following final configurations:

Parameters	**Thyroid dataset**	**Shuttle dataset**
Program Population Size	200	200
Team Population Size	100	100
Total Generations	40	40
Mutation Single Instruction Rate	0.9	0.9
Mutation Instruction Set Rate	0.9	0.9
Mutation Team Rate Set	0.9	0.9
Max Mutations per Program	8	10
Total Runs	50	20
Min. Program Size	16	20
Initial Program Size	32	40
Max. Program Size	64	80
Min. Team Size	2	2
Initial Team Size	3	7
Max. Team Size	6	14
Extra Calculation Registers	2	3
Fitness Function	Accuracy	Accuracy
Sampling Heuristic	$\tau/\|C\|$	$\tau/\|C\|$
Sample Size (τ)	420 (140 per class)	420 (60 per class)
Replacement Rate	0.1	0.1
Selector Operator	Selection for Modular Behavior with Proportional Selection of Teams	
Variation Op.	Mutation Single Instruction, Mutation Instruction Set, Mutation Team Set	
Stop Criterion	While current_generation < generation_total	
Best Solution		
- accuracy	0.93	0.847
- DR	0.888	0.267

– How does the classification performance compare to the GP solution from Sandbox 2?

Besides the Sandbox 3 implementation changes, the code had the following changes to be comparable with Sandbox 2:

- Added two new mutation operators, so the algorithm still can produce solutions with varying length of teams and instructions even without the crossover;
- To further compensate the lack of crossover, the 'Max Mutations per Program' parameter allows that more than one mutation happens to a program per generation, so the algorithm isn't limited to changing just one instruction per generation.
- Using twice the number of generations, to balance the fact that now the programs change much slower, since the crossover would change a random set of instructions between programs, while the current mutations can just add or remove one instruction up to the 'Max Mutations per Program' each time. And also because team mutations are able to change a smaller fraction of the solution.
- The replacement rate is equal to the previous crossover rate, so the % of the worst solutions being removed per generation is the same.

Observation:

- I am using the best solution for the Shuttle dataset based on accuracy, instead of the fitness function reported in Sandbox 2. Because I ended up removing the previous fitness function from the new version of the code, but the results are very similar, since the results from the fitness functions were statistically the same.

Classification performance comparison:

Dataset	**Thyroid**		**Shuttle**	
Sandbox	**2**	**3**	**2**	**3**
accuracy	0.915	0.93	0.825	0.847
DR	0.857	0.888	0.249	0.267
(ACC+DR)/2	0.886	0.909	0.537	0.557
Generation	13	32	12	37
Avg. Instructions	38	32.66	45	40.25

So the modular behavior seems to have increased the classification performance considerably. I think a even better quality could be achieved with a higher number of generations and, for the Shuttle dataset, by forcing the teams to have at least one action for each class. Another future improvements could be to reduce the number of instructions and registers, since the programs now are focused in specific classes.

– How many BidGP individuals are used to express a solution in your champion individual? Is there just one BidGP individual per action or do you find that for some classes more BidGP individuals are utilized?

	Thyroid Solution	**Shuttle Solution**
team size	3	4
team members*	'(1788:32), 0, 34, 1' '(180:1), 1, 32, 42' '(111:1), 2, 32, 22'	'(95:1), 0, 40, 20', '(593:17), 0, 42, 5', '(153:1), 3, 40, 20', '(126:1), 3, 40, 14'

*Format: [(program_id:generation_id), action, instructions_len, total_teams_program_participates)]

For the Thyroid dataset there was used just one action per class, since its data seems to be less complex. For the Shuttle dataset 2 actions per class were necessary, and the solution ignored 5 out of the 7 classes.

– How many input attributes does a BidGP individual employ? Is GP able to associate specific attributes with specific actions?

	Thyroid Solution
Class 0	1 program that used* 5 inputs (3, 12, 16, 18, 19) and used 1 register (0)
Class 1	1 program that used* 10 inputs (0, 1, 3, 5, 6, 8, 9, 11, 16, 18) and used all registers.
Class 2	1 program that used* 8 inputs (0, 2, 5, 6, 12, 15, 19, 20) and used all registers.
Ignored features: 4, 7, 10, 13, 14, 17 (**total:** 5)	
Ignored classes: None	

*non-intron instructions only

	Shuttle Solution
Class 0	2 programs, where the first used* 6 inputs (0, 1, 2, 4, 5, 7) and 3 registers (0, 1, 2), and the second used* 7 inputs (1, 2, 3, 5, 6, 7, 8) and all registers.
Class 3	2 programs, where the first used* 4 inputs (0, 3, 7, 8) and 3 registers (0, 1, 2), and the second used* 8 inputs (0, 2, 3, 4, 5, 6, 7, 8) and all registers.
Ignored features: None	
Ignored classes: 1, 2,4, 5, 6 (**total:** 5)	

*non-intron instructions only

Yes, I think GP is able to associate specific attributes with specific actions. For example, for the Shuttle dataset both programs for action 0 used the attribute 1, while neither program for class 3 used this attribute. And the action 0 for the Thyroid dataset was able to obtain a good recall (0.90) using just 5 inputs, so it seems that they are the relevant inputs for this class.

Note:
I didn't implemented any metric to access that, but I think in some cases one of the programs for the same actions may be the only one actually doing classification (e.g.: The first program for class 3 is almost entirely made of introns, while the second one is a way more complex program). It may be interesting to implement a metric to detect 'introns' programs.

– What can you say about the types of classes that teams learn to classify first during evolution?
For both datasets, the teams most of the time learned to classify the biggest class first. For the Thyroid dataset, most solutions would use at least 2 actions to be able to get a good accuracy for class 0 (73 samples), so it seems that these classes contain quite complex data to be learned. This also happened to a lesser extend to class 1 (177 samples). Class 2 (3178 samples) most of the time was assigned just 1 action and obtained a good accuracy anyway.
For the Shuttle dataset, most run's solutions would use 2 to 5 actions to be able to classify class 0 (11478 samples) with a good accuracy, so it seems to contains very complex data. Usually the solution would have actions for class 0 and three other classes, ignoring the rest. These three classes frequently were the next biggest classes (2, 3, and 4).

– Identify 2 other approaches for constructing teams of programs. What are their respective advantages / disadvantages compared to the framework adopted in this sandbox?
The other approaches could be to add another minimal criteria for a legal team such as:

1. Each team has a fixed size, with a program for each class.
2. Teams have variable sizes, but each team must have at least one action per class.

Both approaches have the advantage of ensuring that all classes will be learned by at least one program, while the current approach has a tendency of ignoring classes that are much smaller than the others. So they would be useful in the case of anomaly detection, and when ignoring the smaller classes isn't acceptable. But for cases where this is an acceptable outcome, the current approach is better since it will focus on the most significant classes.
An advantage from the 2 approach regarding the approach 1 is that it would have a better performance for classes that have samples with multiple patterns, since different programs for the same action would be able to learn the different patterns. While approach 1 would have a better runtime performance than both approach 2 and the current approach.