COS710: Assignment 2

u22571532

April 2025

1 Dataset

There are two data sets comprised of 12 and 20 variables, respectively, also having a target value for each record. The datasets represents CPU specs with the target value representing a performance of the computer. The two datasets are 227_cpu_small.tsv and 197_cpu_act.tsv, with 197_cpu_act.tsv being the dateset with 20 specs and a target value. The data is read into three corresponding arrays using the CSVReader.java with an array for the headers and target values each and a 2D array for the spec values.

Table 1: Dataset 227_cpu_small.tsv

lread	lwrite	scall	sread	swrite	fork	exec
6.0	2.0	1036.0	103.0	114.0	1.0	1.0
1.0	0.0	2165.0	205.0	101.0	0.4	1.2

Table 2: Continued Dataset 227_cpu_small.tsv

rchar	wchar	runqsz	freemem	freeswap	target
172076.0	355965.0	2.0	6527.0	1851864.0	90.0
43107.0	44139.0	3.0	130.0	1131931.0	88.0

Table 3: Dataset 197_cpu_act.tsv

lread	lwrite	scall	sread	swrite	fork	exec	rchar	wchar	pgout	ppgout
6.0	2.0	1036.0	103.0	114.0	1.0	1.0	172076.0	355965.0	0.0	0.0
1.0	0.0	2165.0	205.0	101.0	0.4	1.2	43107.0	44139.0	4.8	42.2

Table 4: Continued Dataset 197_cpu_act.tsv

pgfree	pgscan	atch	pgin	ppgin	pflt	vflt	runqsz	freemem	freeswap	target
0.0	0.0	0.0	2.0	4.0	73.6	89.0	2.0	6527.0	1851864.0	90.0
75.8	181.4	0.2	85.4	88.2	19.4	161.8	3.0	130.0	1131931.0	88.0

2 GP(Genetic Program) Algorithm

The GP makes use of a tree structure to make an individual of the population, representing a mathematical equation. The tree consists of two types of nodes, terminal and functional where functional nodes cannot be leaf nodes and terminal cannot be internal nodes.

- Terminal: represents the spec values of the CPU's well as constants ranging from 0 to 10
- Functional: represents the +, -, *, /, % and pow mathematical operators

The GP creates a population using given parameters : seed, epochs, population size, max tree depth, genetic operation split and data test & train split. During the run of the GP the best tree is stored to make sure the best solution found is not lost. The Genetic Programming algorithm implemented follows the evolutionary processes:

- **Initialization**: A random population is generated to fill the specified size. The population uses the grow method and therefore contains an even spread of trees of each depth up to the maximum depth with randomized fullness. The nodes are inserted at random to ensure great variation in the population. The root node is always a functional node.
- **Selection**: Tournament selection is used. Three random trees are chosen from the population and the one with the best fitness is chosen. Tournament selection is used to help nudge the population to have better individuals with each generation , with only three candidates for selection to ensure premature convergence doesn't happen.
- **Crossover**: Using two tournament selection calls, two parent trees are obtained. A random node is selected in the first parent tree. A second random node of the same type (function or terminal) is then selected from the second parent tree. These are the nodes where the crossover takes place. Deep copies of the trees are made to use for the next generation. The crossover event generates two individuals for the new population. This operator helps exploit the population to gain better results based on tree with already good results. Due to crossover being able to create bigger and bigger trees a penalty was made to prevent trees from growing larger than to depths more than the specified max depth.
- **Mutation**: The mutation uses the selection function to obtain a singular parent. A random node of the parent tree is selected and then randomly changed into a different spec, for terminal nodes, or operator, for functional nodes. Nodes stay the same type to avoid trees converging to an short tree local optima. A deep copy of the parent is used. This produces a one individual for the new generation.
- **Evaluation**: Average fitness is used to evaluate the individuals. The fitness of a run is obtained by with fitness = $|\text{target} - \text{score}|$. This means a lower fitness is better. The average fitness is then used at the end of a training cycle in the selection method. This is mean absolute error. To prevent trees from growing to large due to crossover trees that exceed the maximum depth plus two are provided with a very made score which leads to a very bad fitness. These tree scores are also not calculated and are just given to spare computational time.
- **Termination**: The GP terminates after the specified epochs are reach. This is to prevent overfitting as well as to long run-times with minimal improvements. Further if the best tree solution does not change within 100 generations the GP terminates and that solution is provided.

- **Error Prevention**: The GP allows duplicates of the same tree to from and does not penalize this them. Leave nodes can not be function nodes, and their are checks in place to prevent and fix this. Non-leave nodes can not be terminal nodes, and their are checks in place to prevent and fix this. Mutations don't mutate nodes into different types to prevent the loss of entire trees. Crossovers only crossover the same type of node to prevent trees to shorten to the point of very non-optimal trees. Further null checks are present every where to prevent program crashes. Operators that have limitations like division and mod have checks to make sure they use a divisor of 0.

3 Parameters

The parameters used in the GP implementation look as follows:

Table 5: GP Default Parameters

Parameter	Value
Seed	Sys Time
Epochs	500
Population Size	100
Max Depth	5
Genetic Operator Split	0.5
Train/Test Split	0.8

Testing was done to determine the affect of each parameter (except the seed as it is always different for the tests). A variety of 5 different values were used for each parameter to see the affect. Each value ran 10 GP simulations. The average MAE(Mean Absolute Error) of each parameter value as well as the best run of that parameter value was saved. In the functions, the spec labels are place holders for their values in the dataset. A Genetic Operator Split of 0.3 indicates that 30% of the new population is made by mutation and 70% by crossover. Similarly, a Train/Test split of 0.8 shows that 80% of the dataset will be used for training and 20% for testing.

To find the most suitable parameters each parameter was changed five times while the other values parameters stayed the same as in Table 5. The results of those test were used to create a combination of parameters to test. Experimenting with these combinations lead to the parameters found in Table 6 that were used for the initial GP runs.

Table 6: GP Final Parameters for initial GP

Parameter	Value
Seed	Sys Time
Epochs	200
Population Size	200
Max Depth	6
Genetic Operator Split	0.7
Train/Test Split	0.7

As can be seen in Table 6 the max depth, 6, is quite big, this seems to be the case as their are quite a few variables/specs to be used in the function and this allows for greater utility. The mutation rate is also relatively high at 70% indicating that exploration is very important due to the amount of variables. The epochs are set as 200 as no increase in solution quality was constantly found with more generations. The population size of 200 was enough to gain great

variability in the population. The epochs and population size were aimed to be kept small to save computational power and time, so that the max depth can be slightly increase without it greatly affecting the time it takes to find a solution.
The parameters for the GP after transferring the solution looks as follows:

Table 7: GP Final Parameters for Transfer GP

Parameter	Value
Seed	Sys Time
Epochs	250
Population Size	250
Max Depth	7
Genetic Operator Split	0.7
Train/Test Split	0.7
New Population Duplicates	0.25
New Population Mutants	0.50
New Population New Individuals	0.25

As can be seen in 7 the epochs, population size and max depth are increased. The epochs and population size increase is to allow the GP time and space to incorporate and experiment with the new CPU specs in it's reservoir. Due to the increase in specs the maximum depth increase also allows the ability to incorporate more specs in the solution. Further more there are three new parameters namely New Population duplicates,mutants and new individuals. These represent the ratio of individuals initialized the new population by GP after a solution has been transferred. Duplicates referring to deep copies of the transferred tree, mutants referring to mutations of the transferred tree and new individuals referring to totally new trees being made, with the growth method within the space provided. The mutations and new trees make use of the new specs available in the larger 197_cpu_act.tsv dataset.

4 Transfer

The initial GP is run five times in sequence with different seeds. The best solution is then transferred. This is done to make sure a good solution is chosen as the randomness of GPs may sometimes lead non optimal solutions. It was observed that their is a extremely high chance for a good for a very good initial solution to be found within in five GP runs. The parameters in 6 are used for these five runs. The solution chosen for transfer is then given to a new GP instance that takes the parameters in Table 7. The transferred tree is then used as a template to create the specified ratio copies in the new initial population as well as the ratio of mutants. The tree's nodes are mutated, at random, a total of TreeNodeCount/2 times. The same node can be mutated more than once and the mutations make use of all the specs of the new larger dataset. The remaining part of the initial population is then filled in with newly constructed trees using the new dataset they are made using the growth method, meaning there is an equal amount of trees created of each depth up to the max depth. This format of of initializing the new population allows us to ensure the transferred solution is used allowing good solutions from the start, as well as good way starting point for exploration with the mutants. The batch of new trees further help promote the use of the new specs for a solution. The GP then proceeded to run the same as the initial GP optimizing the transferred solution.

5 Results

Table 8 contains the results of 10 consecutive GP runs using the parameters of Table 6 and 7.

Table 8: Results of 10 GP Runs

Run	Seed	MAE	Function
0	1458000029	3.1816	$(((((((pflt - pgout) + (fork + scall))/(fork * scall)) - (pgin/7.0)) + ((((10.0 * 7.0) + (9.0 \bmod freemem)) - (fork - 10.0)) \bmod (freeswap/5.0))) \bmod (exec - scall)) + ((((10.0 * 7.0) + ((pgin/atch) - (pgfree - 10.0))) - (fork/pgscan))/(swrite \bmod rchar)) - (0.0 + fork))$
1	1459728041	3.6818	$(((((10.0 - fork) + (((9.0 + 5.0) - (fork + fork)) \bmod (freemem/ppgin))) + (7.0 * 10.0)) \bmod ((((runqsz - 5.0) * (fork - freeswap))/(wchar * scall)) * (atch - freeswap))) \bmod (((1.0 - runqsz) + ((((vflt/fork)/(exec + runqsz)) * (pgin + ppgout)) - (swrite/pgfree))) * ((10.0 - lwrite) * ((10.0 - pgout) * ((10.0 \bmod fork) * ((pgscan - sread) - (pgin - freeswap))))))$
2	1461815656	5.0370	$(((((((fork * 6.0) * (5.0 * sread)) + (sread \bmod pgout)) + (((5.0 * 5.0) + (5.0 * 6.0)) * (sread * fork))) + (sread + pflt))/(sread * fork)) \bmod (((((((sread \bmod sread) - (scall/sread)) - (ppgout/6.0)) \bmod (freeswap + pgin)) \bmod (((pgfree + pgfree) \bmod (rchar - pflt)) * ((freeswap \bmod pgin) * ((5.0 * 5.0) + (5.0 * 6.0))))) + (freeswap - fork)))$
3	1463954660	3.5672	$(((((((10.0 - fork) + (10.0 * 3.0)) + (10.0 + 10.0)) + (10.0 + 7.0)) + (8.0 - fork)) + ((10.0 - fork) \bmod ((10.0 - fork) - (((sread - freemem) - (exec - ppgin)) \bmod (8.0 - fork))))) \bmod (freeswap/((((10.0 + atch)/(freeswap + lwrite)) * (runqsz * scall)) * (lwrite + ((swrite + sread) - (swrite * fork))))))$
4	1465693520	3.3929	$((((8.0 * 6.0) + (8.0 - fork)) + ((((5.0 * 6.0) - (fork \bmod ppgin)) + (8.0 - fork)) - (pgin/6.0))) \bmod ((atch - freeswap)/((((8.0 - pgout) - (ppgout \bmod sread)) * (runqsz * runqsz)) - (((atch - ppgin) \bmod (scall * 6.0)) + ((8.0 * pgout) - (ppgout \bmod sread))))))$
5	1467451032	4.0207	$(((((((2.0/freeswap) * (lread - swrite)) - (swrite/freemem)) + (((exec/freeswap)/(lread - swrite)) - (runqsz \bmod freemem))) + (6.0 * 8.0)) + (6.0 * 8.0)) - ((((pgout - 0.0) \bmod (fork + pgin)) + ((((scall - pgout)/(sread - freeswap)) \bmod (scall * ppgin)) + (((fork/pgout) \bmod (pgin \bmod pflt)) + (fork + fork)))) \bmod (ppgin * pflt)))$

Continued from previous page

Run	Seed	MAE	Function
6	1469915783	6.8482	$((scall/sread)+((((((8.0 \bmod rchar)-(exec \bmod scall))+(10.0*7.0)) \bmod (((atch-freeswap)-(pgfree/runqsz))*(freeswap/rchar))) \bmod (((freemem \bmod freeswap)*((sread/lread) \bmod (freeswap-lread)))*(((rchar-sread)+(sread-rchar)) \bmod (atch+pgfree)))) \bmod (((freemem \bmod freeswap)*((sread/pgscan)*(wchar-sread)))+(((rchar-sread)+(sread-rchar)) \bmod (atch+runqsz)))))$
7	1472328830	4.8061	$(((((((freeswap/vflt)/(fork/6.0))/(scall+runqsz))-(9.0*10.0))-(9.0/runqsz)) \bmod (((((ppgin-pgfree)-(rchar+sread))/(rchar-exec))/(atch+exec))-(exec+runqsz)))+(9.0*10.0))$
8	1473450547	3.9840	$(((swrite*8.0)/(((rchar*0.0)+((1.0/3.0)+((freemem/swrite)+(vflt/swrite))))+((((freemem/swrite) \bmod (scall-ppgout))-(6.0+freeswap))+(((vflt/swrite) \bmod (vflt*pgfree)) \bmod (6.0+swrite))))))+((8.0*10.0)+(((1.0/fork)-(pgin/7.0))+((1.0-fork)+((pflt/sread)+(9.0-fork))))))$
9	1475304999	4.3356	$(((((pflt+sread)/(sread*fork))+((10.0*8.0) \bmod (lread-scall))) \bmod ((((pgfree-sread)/(swrite*fork))/((runqsz-8.0)-(runqsz-sread)))*((freemem/sread)-(((pgscan/vflt)+(fork*freeswap))-((swrite \bmod fork)+(freemem/freeswap))))))+((8.0+sread)-(sread+exec)))$
	Average MAE		4.2855
	Standard Deviation		1.0229
	Best Seed		1458000029
	Best MAE		3.1816
	Best Solution Equation		$(((((((pflt-pgout)+(fork+scall))/(fork*scall))-(pgin/7.0))+((((10.0*7.0)+(9.0 \bmod freemem))-(fork-10.0)) \bmod (freeswap/5.0))) \bmod (exec-scall))+((((10.0*7.0)+((pgin/atch)-(pgfree-10.0)))-(fork/pgscan))/(swrite \bmod rchar))-(0.0+fork))$

After the final tests we saw that the MAE of the runs is 4.2855 . The standard deviation was found to be 1.0229, meaning the GP is very consistent with minimal outliers.

The best solution found as represented in Table 9 had an MAE of 3.1816 meaning on average the solution derives a value that is out with 3.1816 from the target value. This is good as the target values have great variation between them. This solution has a depth of 8 in it's tree form

meaning it is not to computationally complex and is within the non penalized depth range.

Table 9: The best result achieved in the 10 runs

Seed	MAE	Function
1458000029	3.1816	$(((((((pflt - pgout) + (fork + scall))/(fork * scall)) - (pgin/7.0)) + ((((10.0 * 7.0) + (9.0 \bmod freemem)) - (fork - 10.0)) \bmod (freeswap/5.0))) \bmod (exec - scall)) + ((((10.0 * 7.0) + ((pgin/atch) - (pgfree - 10.0))) - (fork/pgscan))/(swrite \bmod rchar)) - (0.0 + fork))$

6 Conclusion

In this study, we explored the effectiveness of Transfer learning in Genetic Programming (GP) for symbolic regression tasks in predicting performance of a computer based on CPU specs. Through extensive parameter testing, we identified configurations that optimize performance by balancing exploration and exploitation within the search space. It was found that the transferring of a solution obtained from a smaller dataset can increase the performance of finding a good solution within a larger and more complex dataset set of the same type.