CGP++ User Guide

February 9, 2024

1 Introduction

The user guide serves to give some specific details that can support the usage of `CGP++`. It therefore represent as supplementary and supporting element to the corresponding paper that introduces `GGP++` which is currently under peer-review.

Our repository on GitHub serves as the main resource for `CGP++`:
https://github.com/RomanKalkreuth/cgp-plusplus

2 Requirements

Since `CGP++` uses modern techniques of `C++`, it has to be compiled with a version of `GCC` that supports the `C++17` standard.

We recommend compiling `CGP++` by using the `G++` compiler with the following command: `g++ -std=c++17 -O3`

3 Build

For Linux, a classical Unix `makefile` is provided in the build folder. The command `make all` can be used to create the executable.

For building `CGP++` on Windows, we recommend, considering popular choices such as `Mingw-w64`[1], `MSVC`[2] or `VSCode`[3].

On MacOS, the use of texttttclang[4] in combination with `VSCode`[5] can be considered as an alternative to using `G++`.

[1] https://www.mingw-w64.org/
[2] https://visualstudio.microsoft.com/vs/features/cplusplus/
[3] https://code.visualstudio.com/docs/languages/cpp
[4] https://clang.llvm.org/
[5] https://code.visualstudio.com/docs/setup/mac

4 Configuration

The compiled executable can be used with the parameter file in the `data/parfile` folder and the following command line options. The files are located in the `data/` folder. The data and parameter file are passed to CGP via command line. Optionally, a checkpoint file can be passed to CGP this way. The CGP and run-specific options can be the passed to CGP via the commandl ine and parameter file configuration. Listing 1 shows the command line configuration and Listing 7 in the appendix gives an overview over the parameterization via parameter file.

```
usage: DATAFILE PARFILE CHECKPOINTFILE <options>

 -a  - search algorithm: 0 = one-plus-lambda; 1 = mu-plus-lambda
 -n  - number of function nodes
 -v  - number of variables
 -z  - number of constants
 -i  - number of inputs
 -o  - number of outputs
 -f  - number of functions
 -r  - maximum arity
 -p  - mutation rate
 -c  - crossover rate
 -m  - number of parents (mu)
 -l  - number of offspring (lambda)
 -e  - maximal number of fitness evaluations
 -g  - goal (ideal) fitness
 -j  - number of jobs
 -s  - global seed
 -1  - duplication rate
 -2  - max duplication depth
 -3  - inversion rate
 -4  - max inversion depth
```

Listing 1: Parameters that can be configured via the command line

5 Datafiles

Datafiles that describe the input-output matching are used to represent training and test datasets for experiments with black-box problems. Currently `CGP++` supports `.plu` and `.dat` files. For problems in the logic synthesis domain, truth tables are compressed format and stored in `.plu` files that have been commonly used in the past for this purpose. For symbolic regression bencmarks, we provide `.dat` files that describe the input-output matching of the training or test data. An example of a `.dat` file that represent the objective function of the `Koza1` benchmark is shown in 2.

```
.i 1
.o 1
.p 20
0.571965    1.19325
0.198192    0.2468
0.332413    0.491852
-0.630832   -0.325559
-0.89878    -0.164464
0.970743    3.71587
0.894716    3.05229
0.147635    0.173123
-0.445633   -0.296104
-0.664529   -0.321376
-0.425338   -0.288645
0.522597    1.01302
0.442076    0.762097
-0.659558   -0.32222
0.0462336   0.0484745
-0.0465327  -0.0444635
-0.172244   -0.146806
```

```
−0.781772    −0.274873
0.151308     0.178191
0.393863     0.634155
.e
```

Listing 2: Example of a checkpoint file.

6 Constants

`CGP++` supports the generation and usage of ephemeral random constants (ERC). The number of constants can be specified via the `num_constants` parameter and the type of the ERC can be configured via the `constant_type` setting.

7 Concurrency

`CGP++` supports evaluation concurrency that can be enabled by defining the `num_eval_threads` setting in the parameter file. Breeding concurrency is a planned feature.

8 Checkpointing

The generation of checkpoints can be enabled/disabled via the `checkpointing` setting. The checkpoint interval is defined with the `checkpoint_modulo` parameter. The checkpoints are stored in the `data_checkpoints` folder. A new folder with a unique (timestamp) name will be created for the experiment. An example of an checkpoint is provided in Listing 3.

```
generation_number 50
global_seed 1706766371186480657
genome 1,0,1,2,0,2,2,3,1,0,4,3,2,1,2,0,5,2,1,0,0,0,8,4,0,7,1,0,7,3,3
genome 1,0,1,2,0,2,2,3,1,2,4,3,2,1,2,0,5,2,1,0,5,0,8,4,0,7,1,0,7,3,3
genome 1,0,1,2,0,2,2,3,1,0,4,3,2,2,1,0,1,2,1,0,0,0,8,4,0,7,1,0,7,3,3
genome 1,0,1,2,0,2,2,0,1,0,4,3,2,2,1,0,1,2,1,0,0,0,8,4,0,7,1,0,7,1,3
genome 1,0,1,2,0,2,2,3,1,0,4,3,2,2,3,0,1,2,1,0,0,0,8,4,0,7,1,0,7,3,3
constant −0.74118
```

Listing 3: Example of a checkpoint file.

9 Example Runs

9.1 Logic Synthesis

The following call to CGP++ triggers a run with a simple 1-Bit adder benchmark by using a 1+1-ES as search algorithm and a simple CGP with 10 function nodes and point mutation:

```
./cgp data/plufiles/add1c.plu data/parfiles/cgp.params -a 0 -b 10 -n 10 -v 1 -z 0 -o
1 -f 4 -r 2 -m 1 -l 1 -p 0.1 -c 0.0 -e 1000000 -j 10 -g 0
```

9.2 Symbolic Regression

The following call triggers a run with the simple Koza1 benchmark (quartic polynomial) by using a $4 + 16$-ES as search algorithm and discrete recombination:

```
./cgp data/datfiles/koza1.dat data/parfiles/cgp.params -a 1 -b 10 -n 10 -v 1 -z 0 -o
1 -f 4 -r 2 -m 4 -l 16 -p 0.1 -c 0.5 -e 10000000 -j 10 -g 0.01
```

10 Program Output

The overview of the configuration of the parameters can be printed by enabling the `print_configuration` in the parameter file. Generational output inside of a run can be activated/deactivated by setting `report_during_job`. The report interval can be configured via the `report_interval` setting. Metrics of the respective instances such as number of fitness evaluations, best fitness and runtime can be printed by activating the `report_during_job` setting. Listing 4 shows the output of the `CGP++` configuration, Listing 5 the generational output inside of a job and Listing 6 the output of metrics for respective jobs.

```
CGP++ CONFIGURATION
Number of function nodes: 10
Levels back: 10

Number of functions: 4
Maximum arity: 2

Number of variables: 1
Number of constants: 1

Number of inputs: 2
Number of outputs: 1

Crossover rate: 0.5
Mutation rate: 0.1

Number of parents (mu): 4
Number of offspring (lambda): 8

Ideal fitness value: 0.01

Number of jobs: 100
Maximum number of fitness evaluations: 1000000000
Maximum number of generations: 125000000

Global seed: 1707214445176910306

Functions: [1] ADD [2] SUB [3] MUL [4] DIV
Constants: [1] -0.416173

Recombination: Discrete Crossover
Mutation: [1] Probabilistic Point [2] Inversion
[3] Duplication

Algorithm: mu-plus-lambda
```

Listing 4: Output of the CGP++ configuration

```
Generation # 10 :: Best Fitness: 11.2734
Generation # 20 :: Best Fitness: 11.2734
Generation # 30 :: Best Fitness: 11.2734
Generation # 40 :: Best Fitness: 4.51127
Generation # 50 :: Best Fitness: 4.51127
Generation # 60 :: Best Fitness: 4.51127
Generation # 70 :: Best Fitness: 4.34047
Generation # 80 :: Best Fitness: 3.42921
```

```
Generation # 90 :: Best Fitness: 2.56594
Ideal fitness has been reached in generation # 94
```

Listing 5: Generational output inside of a run.

```
Job # 1 :: Evaluations: 1980264 :: Best Fitness: 2.08206e-05 :: Runtime (s): 40.5062
Job # 2 :: Evaluations: 167528 :: Best Fitness: 2.11298e-05 :: Runtime (s): 2.47478
Job # 3 :: Evaluations: 113000 :: Best Fitness: 2.10702e-05 :: Runtime (s): 2.098
Job # 4 :: Evaluations: 49800 :: Best Fitness: 2.11298e-05 :: Runtime (s): 0.590241
Job # 5 :: Evaluations: 8832 :: Best Fitness: 2.08057e-05 :: Runtime (s): 0.128364
```

Listing 6: Output of metrics for respective jobs.

11 Python Binding

A this time, we do not provide bindings for python. However, a gentle tutorial to create own Python bindings can be found here: https://nwcpp.org/Aug-2021.html

A Appendix

```
algorithm                      - 0 = one-plus-lambda, 1 = mu-plus-lambda
levels_back                    - type: integer

num_jobs                       - type: integer
num_function_nodes             - type: integer
num_variables                  - type: integer
num_constants                  - type: integer
constant_type                  - 0 = Koza
num_outputs                    - type: integer
num_functions                  - type: integer
max_arity                      - type: integer

num_parents                    - type: integer
num_offspring                  - type: integer

max_fitness_evaluations        - type: integer
ideal_fitness                  - type: generic
minimizing_fitness             - 0 = false, 1 = true

crossover_type                 - 0 = block, 1 = discrete
crossover_rate                 - type: float,  range: [0.0, 1.0]

probabilistic_point_mutation   - 0 = deactivated, 1 = activated
single_active_gene_mutation    - 0 = deactivated, 1 = activated
inversion_mutation             - 0 = deactivated, 1 = activated
duplication_mutation           - 0 = deactivated, 1 = activated
point_mutation_rate            - type: float,  range: [0.0, 1.0]
duplication_rate               - type: float,  range: [0.0, 1.0]
inversion_rate                 - type: float,  range: [0.0, 1.0]
max_duplication_depth          - type: integer
max_inversion_depth            - type: integer

neutral_genetic_drift          - 0 = deactivated, 1 = activated
simple_report_type             - 0 = deactivated, 1 = activated
print_configuration            - 0 = deactivated, 1 = activated
evaluate_expression            - 0 = deactivated, 1 = activated

num_eval_threads               - type: integer
generate_random_seed           - 0 = deactivated, 1 = activated
global_seed                    - type: long long

report_during_job              - 0 = deactivated, 1 = activated
report_after_job               - 0 = deactivated, 1 = activated
report_simple                  - 0 = deactivated, 1 = activated
report_interval                - 0 = deactivated, 1 = activated

checkpointing                  - 0 = deactivated, 1 = activated
checkpoint_modulo              - type: integer

write_statfile                 - 0 = deactivated, 1 = activated
```

Listing 7: Parameters that can be configured in the paramter file