User manual

Maciej Kozarzewski

Contents

1 Updates

For updates visit https://github.com/MaciejKozarzewski/AlphaGomoku/releases

2 What's inside the package?

AlphaGomoku is usually distributed with following contents:

2.1 Folder 'logs'

Detailed logs are stored in this folder.

2.2 Folder 'networks'

All files with neurall networks are stored here.

Note: The networks that are distributed have been optimized for inference and thus are not trainable.

2.3 File 'config.json'

This is the configuration file that is loaded by default. You can create more configs and load one of them when launching AG.

2.4 Executable 'pbrain-AlphaGomoku.exe'

This is the CPU-only version of the program (the same that participates in Gomocup). The only difference between this and the other versions is that this one has GPU support disabled so it does not require any CUDA- or OpenCL-related libraries to be present on your machine. Also it does not require installation of CUDA or OpenCL so you can run AG without even having a GPU.

2.5 Executable 'pbrain-AlphaGomoku_cuda.exe'

This is a version that supports both CPU and CUDA-enabled graphic cards (so only NVIDIA GPUs). In order to run this version it is required that CUDA is properly installed on the machine.

Note: For guides on how to install CUDA look here:
https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html
https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

Note: NVIDIA graphic cards usually support OpenCL, so you can use opencl version as well. However cuda version is slightly faster on CUDA-enabled GPUs.

2.6 Executable 'pbrain-AlphaGomoku_opencl.exe'

This is a version that supports both CPU and OpenCL-enabled graphic cards. However, in order to run this version it is required that CpenCL is properly installed on the machine.

Note: Although the OpenCL.dll is included in the package for Windows version, but still you may need to ensure you have proper driver installed, and if you don't then install it yourself. Check how to install OpenCL on your machine as the details vary between different hardware and operating systems.

Note: For Linux version you also need to install CLBlast library.

2.7 File 'user_manual.pdf'

This is the text that you are reading right now.

2.8 File 'protocols.pdf'

This is the documentation of all commands in all protocols supported by AG.

2.9 File 'swap2_openings.json'

This file contains pre-defined openings to be used for swap-like opening commands. The existance of this file is not required to run AG, but obviously it is necessary for swap/swap2 openings.

Note: Despite the name, those openings can also be used for swap opening rule, not only swap2.

2.10 A couple of libraries

Depending on the platform (Windows or Linux) there may be some extra libraries (.dll or .so) in the folder. AG won't run without them so don't delete them.

3 Manual configurating

The configuration file has several options that can be set by the user. Here are they:

3.1 version

String that encodes version of the configuration file (obviously). Must match the version of the AG. Format is "major.minor.revision" (for example "5.8.1").

3.2 protocol

String that describes which communication protocol will be used by AG. Allowed values are:

"gomocup"	base protocol used in Gomocup, as described in [?].
"extended_gomocup"	extended protocol based on the regular one. Supports various opening rules, analysis mode and other stuff. Documented in 'protocols.pdf'.
"yixinboard"	protocol used by YixinBoard GUI [?].

3.3 use_logging

Boolean flag that toggles logging to a file. Allowed values are:

"true"	logging is enabled.
"false"	logging is disabled.

3.4 always_ponder

Boolean flag that toggles automatic pondering. You can override this setting at runtime using appropriate INFO command. Allowed values are:

"true"	auto-pondering is enabled.
"false"	auto-pondering is disabled.

3.5 swap2_openings_file

String that specifies file where swap or swap2 openings are stored.

Note: AG will work if this file does not exists until you try to use swap or swap2 opening commands.

3.6 conv_networks

This table specifies paths to the convolutional neural networks that are used in the monte-carlo search. AG is distributed with a single network per rule, but if you somehow managed to get other networks, here is where you can set which network you want to use for each rule.

Note: Files specified here are assumed to be located inside "networks" folder.

3.7 nnue_networks

This table specifies paths to the NNUE-like neural networks that are used in the alpha-beta search. It is empty for now as they provide no benefit and thus are unused.

3.8 use_symmetries

Boolean flag that toggles applying random symmetries (flips, rotations) to the board before passing it to the neural network. Allowed values are:

"true"	symmetries are applied (stronger play).
"false"	board is passed as it is (weaker play but deterministic).

3.9 search_threads

Here you can set the number of threads that AG can use. This setting can be later changed by using appropriate INFO command. Obviously, must be larger or equal to 1.

Note: In order to properly enable multi-threading - read the next section.

3.10 devices

AG can utilize various computing devices to run its neural network - either CPU or GPU. But it has no way of knowing which device you want to use. This section is where you specify that. If you just want to get the best setup, skip to the chapter **??** and read about automatic configuration. But if you want to create some specific setup, continue reading.

3.10.1 Decide which AG variant to use?

AG comes in 3 variants as described in section **??**. The first supports just CPU, second supports CPU and CUDA-enabled GPUs (only NVIDIA ones), while the third supports CPU and OpenCL-enabled GPUs (either NVIDIA, AMD or other).

3.10.2 Find available devices

After you picked a variant it is time to find out what kind of computing devices you have that AG can use. You can do it by running appropriate command line option (section **??**). Example output can be:

CPU : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz : 8 x AVX2 with 32033MB of memory

CUDA:0 : NVIDIA GeForce GTX 1080 Ti : 28 x SM 6.1 with 11169MB of memory

CUDA:1 : NVIDIA GeForce GTX 1080 Ti : 28 x SM 6.1 with 11172MB of memory

The names on the left, "CPU", "CUDA:0", "CUDA:1" are the identifiers of the device. Those are the names that have to be set in the configuration file. In the example case there is a CPU (there is always only one CPU so it doesn't have any number) and two CUDA GPUs, numbered 0 and 1.

Note: If you don't see some device here it means it cannot be used by AG. If you are sure that you have a GPU, check if you have installed everything properly (CUDA or OpenCL, depending on which variant you chose).

3.10.3 How many threads you want to use?

Each search thread in AG creates its own copy of the neural network on the device specified in the configuration file. This means that you have to add as many entries in the "devices" section as the number of thread you want to use. Here are some examples:

Example 1 - 4 thread, dual CUDA GPUs (literally the config used in unlimited tournament during Gomocup 2024). It's required to provide the name of the device and the batch_size (the number of states evaluated in parallel). The third parameter - omp_threads could have been ommited, it's unused anyway. Such config works with up to 4 threads.

```
"devices": [
    {
      "device": "CUDA:0",
      "batch_size": 128,
      "omp_threads": 1
    },
    {
      "device": "CUDA:0",
      "batch_size": 128,
      "omp_threads": 1
    },
    {
      "device": "CUDA:1",
      "batch_size": 128,
      "omp_threads": 1
    },
    {
      "device": "CUDA:1",
      "batch_size": 128,
      "omp_threads": 1
    }
  ],
```

Example 2 - 2 threads, single OpenCL GPU. Such config works with up to 2 threads.

```
"devices": [
    {
      "device": "OPENCL:0",
      "batch_size": 64
    },
    {
      "device": "OPENCL:0",
      "batch_size": 64
    }
  ],
```

Example 3 - 4 threads, all running on a CPU. Such config works with up to 4 threads.

```
"devices": [
    {
      "device": "CPU",
      "batch_size": 12
    },
    {
      "device": "CPU",
      "batch_size": 12
    },
    {
      "device": "CPU",
      "batch_size": 12
    },
    {
      "device": "CPU",
      "batch_size": 12
    }
  ],
```

How do you know what batch_size to use? Good question, this is why there is an automatic benchmarking and configuration in AG.

3.11 search_config

The only parameter you should change here is the "max_batch_size". It should be equal to the largest batch size out of all devices you set in the "devices" section.

3.12 mcts_config

There are two parameters that you can change here. To save memory and play better, AG utilizes some pruning of unpromising moves. "max_children" controls the maximum number of moves that will be considered in any position. "policy_expansion_threshold" controls the minimum policy probability (as calculated by neural network) for the move to be considered.

Note: If you want all moves to be included, set "max_children" to a value larger that board size, while "policy_expansion_threshold" should be set to 0.0

Note: Moves that are provably losing will always be pruned.

4 Reading logfiles

If you enable logging, each **START** command will create new logfile inside logs folder. In this file you can find all communication with the manager in lines starting with "Received : " and "Answered : ". Also, after each move you can find detailed info about search results, which can look like this:

4.1 First section - best moves

```
depth=40 : S= W21 : Q=0.974640 : 0.000015 : 0.025345 : Visits=13602 : Edges=32
BEST
CROSS  (17,16) : Xq17 : S= W21 : Q=0.983135 : 0.000009 : 0.016855 : Visits=13324 : P=0.013671
CROSS  ( 9,13) : Xn9  : S= -92 : Q=0.635456 : 0.000324 : 0.364220 : Visits=189 : P=0.598268
CROSS  (16,13) : Xn16 : S=-101 : Q=0.499617 : 0.000225 : 0.500158 : Visits=19 : P=0.082760
CROSS  (17,15) : Xp17 : S= +21 : Q=0.518970 : 0.000281 : 0.480749 : Visits=12 : P=0.050572
CROSS  (17,12) : Xm17 : S= -23 : Q=0.404419 : 0.000321 : 0.595261 : Visits=11 : P=0.061011
CROSS  (16,15) : Xp16 : S=-117 : Q=0.517730 : 0.000450 : 0.481820 : Visits=8 : P=0.037454
CROSS  (17,10) : Xk17 : S= -80 : Q=0.568752 : 0.000115 : 0.431134 : Visits=5 : P=0.017153
CROSS  (10,10) : Xk10 : S=-292 : Q=0.266853 : 0.000313 : 0.732835 : Visits=5 : P=0.032967
CROSS  ( 8,11) : Xl8  : S=-141 : Q=0.437899 : 0.000329 : 0.561772 : Visits=4 : P=0.021890
CROSS  (12,16) : Xq12 : S=+117 : Q=0.595978 : 0.000207 : 0.403816 : Visits=2 : P=0.007926
```

Here you can see up to 10 best moves in the current position. Let's investigate closer the first line.

CROSS (17,16)	is sign, row and column of a move (point 0,0 is in upper left corner).
Xq17	is a move encoded as text, in this case this is Cross (Black) at 17-th column and 18-th row (point a0 is in upper left corner).
S= W21	is alpha-beta score. In this case it is a sure win in 21 moves. Other values are "L" for loss, or "D" for draw or a number if the score is not proven.
Q=0.983135 : 0.000009 : 0.016855	are three numbers, probability of win, draw and loss respectively.
Visits=13324	is the number of MCTS simulations that traversed this move.
P=0.013671	is prior policy assigned by the network when this position was added to the tree. It represents the probability that this move is the best one in this position (at least that is what neural network thinks about this move).

4.2 Second section - distribution of playouts

Here you can find distribution of MCTS simulations over the board. For example a value of '13' means 1.3% of playouts went through this move (so you have to divide by 10 to get percents). Sometimes, some positions will be marked with "W" or "L" or "D" which means that this move was proven to be win or loss or draw, respectively (with the number of moves to this outcome).

```
    a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t
 0  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   0
 1  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   1
 2  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   2
 3  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   3
 4  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   4
 5  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   5
 6  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   6
 7  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   7
 8  _   _   _   _   _   _   _   _   _   _   _   _   O   O   O   X   _   _   _   _   8
 9  _   _   _   _   _   _   _   _   _   _   _   _   _   13  X   _   O   _   _   _   9
10  _   _   _   _   _   _   _   _   _   _   _   X   _   O   _   _   X   _   _   _   10
11  _   _   _   _   _   _   _   _   _   _   _   _   O   X   X   O   X   _   _   _   11
12  _   _   _   _   _   _   _   _   _   _   _   _   X   O   O   X   _   O   _   _   12
13  _   _   _   _   _   _   _   _   _   X   _   O   X   _   O   _   X   _   _   _   13
14  _   _   _   _   _   _   _   _   O   _   X   X   X   O   O   X   _   O   _   _   14
15  _   _   _   _   _   _   _   _   X   O   _   O   O   X   _   O   _   _   _   _   15
16  _   _   _   _   _   _   _   _   _   _   _   _   _    1  X   _   X   _   _   _   16
17  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _  W21  _   _   _   17
18  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   18
19  _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   _   19
    a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t
```

4.3 Third section - principal variation

Here you can see pricipal variation. Format is the same as in the first section.

```
CROSS  (17,16) : Xq17 : S= W21 : Q=0.983135 : 0.000009 : 0.016855 : Visits=13324 : P=0.013671
CIRCLE (15,16) : Oq15 : S= L18 : Q=0.012740 : 0.000005 : 0.987255 : Visits=8301 : P=0.878230
CROSS  (17,15) : Xp17 : S= W17 : Q=0.994129 : 0.000003 : 0.005869 : Visits=8094 : P=0.519155
CIRCLE (18,16) : Oq18 : S= L16 : Q=0.005850 : 0.000003 : 0.994147 : Visits=8093 : P=1.000000
CROSS  (17,14) : Xo17 : S= W15 : Q=0.999934 : 0.000000 : 0.000066 : Visits=7782 : P=0.466217
CIRCLE (17,13) : On17 : S= L14 : Q=0.000000 : 0.000000 : 1.000000 : Visits=1695 : P=0.218040
CROSS  (16,13) : Xn16 : S= W13 : Q=1.000000 : 0.000000 : 0.000000 : Visits=1694 : P=1.000000
```

4.4 Fourth section - statistics and timings

Here you can find the timing of various stages of the search. Typically they have following format: "name" : total time : number of events : average time of each event. On Windows the clock resolution is sometimes too low to capture some of the events, you will see 0.0s then.

```
----NNEvaluator----
total samples = 873
avg batch size = 2.025522 // this is the average number of positions evaluated by the neural network in each step.
pack    = 0.001443s : 431 : 3.347564 us // this is the time of packing the data to network input
compute = 4.680242s : 431 : 10.859031 ms // this is the time spent on computation
unpack  = 0.002133s : 431 : 4.948260 us // this is the time of unpacking the data from network output

----SearchStats----
nb_duplicate_nodes     = 0 // in parallel MCTS it can happen that the same node will be evaluated more than once
nb_information_leaks   = 218 // number of mismatch of evaluation between graph edge and node
nb_wasted_expansions   = 0 // number of unnecessarily expanded nodes
nb_proven_states       = 12573 // number of visited states that have already been proven
nb_network_evaluations = 873 // number of positions evaluated by the neural network
nb_node_count          = 965 // number of evaluated nodes
select   = 0.034291s : 13756 : 2.492781 us // time of selection phase in MCTS
solve    = 0.374076s : 965 : 387.643834 us // time used by alpha-beta search
schedule = 0.001101s : 965 : 1.141347 us // time used to schedule positions for neural network evaluation
generate = 0.005064s : 965 : 5.247358 us // time used to generate legal actions for each position
expand   = 0.002326s : 965 : 2.410674 us // time of tree expansion phase in MCTS
backup   = 0.001310s : 965 : 1.357617 us // time of backup phase in MCTS

----NodeCacheStats----
load factor = 0.032000
stored nodes    = 2103
allocated nodes = 40000
stored edges    = 30913
allocated edges = 700000
seek     = 0.084930s : 201805 : 0.420850 us // time used for seeking an element in the cache
insert   = 0.094467s : 88673 : 1.065341 us // time used for inserting elements to the cache
remove   = 0.000016s : 17 : 0.929412 us // time used for removing elements from the cache
resize   = 0.000000s : 0 : 0.000000 us // time used for cache resize
cleanup  = 0.048654s : 17 : 2.862006 ms // time used for cache cleanup (after each position change)

memory usage = 1MB // estimated memory usage
```

5 Automatic configuration

If you just want the best setup, this is your way to go:

1. Run automatic benchmarking of AG (unless you already did that) by running appropriate command as described in section **??**. This may take a few minutes and will create file "benchmark.json".

2. Run automatic configuration of AG (once again, see section **??**). This will create "config.json" with the most performant setup out of all available devices.

And that's it, enjoy.

6 Troubleshooting

When facing any technical problems, like for example:

- program crashed
- program doesn't start at all
- program complains about incorrect configuration options
- program cannot find some file
- and many other reasons

you are expected to run the self-check feature of AG. You can do it in a two ways:

- Launch AG from command line with --selfcheck option
- Launch AG and type selfcheck

Both variantes will run some test procedure that saves the results to a file named 'selfcheck.txt' (created in the working directory of the executable). Then create the issue on github https://github.com/MaciejKozarzewski/AlphaGomoku/issues where you can describe the problem (don't forget to attach selfcheck.txt).

7 Bug reporting

When you find some incorrect behaviour at runtime, like for example:

- some command does not behave like it is described in the protocol
- program makes illegal or forbidden move
- program runs abnormally slow or fast
- you found error in this document
- and many other reasons

and you want this issue to be fixed, do the following:

1. Prepare the list of steps required to reproduce the problem
2. Reproduce the problem yourself but with logging enabled
3. Create a bug report on github with all the above information

The better you describe the problem, the easier and faster it will be to fix it.

Note: It is **NOT** a bug if AG doesn't find the best move in some position. The only situation you should report is when AG found forced win and then didn't win the game.

8 Command line arguments

There are several command line options that can be used. All of them can also be typed after launching AG (with the same names but without '--' prefix) and before passing any Gomoku protocol-related commands.

There are following options:

--help	prints help message
--version	prints info about version
--list-devices	prints list of all detected devices that AG can use
--load-config	specifies path to the configuration file to be used
--configure	runs automatic configuration
--benchmark	runs automatic benchmark, required for auto-configuration
--selfcheck	runs a self-check that tests numerous potentially problematic features of AG, then saves the results to a file. For more details see section **??**

9 Frequently asked questions

9.1 I set max memory to X but AlphaGomoku is using much less

AG uses exactly as much memory as it needs at the moment. It is perfectly fine that memory consumption is lower than the specified maximum.

9.2 There is a win in X but AlphaGomoku played a longer one

AG is not designed to find the shortest win - it tries to find ANY win instead. Shorter wins are slightly more favoured over longer ones, but there are no guarantees here.

References

[1] Petr Laštovička, New Gomocup Protocol

[2] YixinBoard protocol