

Introduction

ShashChess is a free UCI chess engine derived from Stockfish family chess engines.

The goal is to apply Alexander Shashin theory exposed on the following book :

<https://www.amazon.com/Best-Play-Method-Discovering-Strongest/dp/1936277468>

to improve

- base engine strength
- engine's behaviour on the different positions types (requiring the corresponding algorithm) :
 - Tal
 - Capablanca
 - Petrosian
 - the mixed ones
 - * Tal -Capablanca
 - * Capablanca-Petrosian
 - * Tal -Capablanca-Petrosian

Terms of use

Shashchess is free, and distributed under the ****GNU General Public License**** (GPL). Essentially, this

means that you are free to do almost exactly what you want with the program, including distributing

it among your friends, making it available for download from your web site, selling it (either by

itself or as part of some bigger software package), or using it as the starting point for a software

project of your own.

The only real limitation is that whenever you distribute ShashChess in some way, you must always

include the full source code, or a pointer to where the source code can be found. If you make any

changes to the source code, these changes must also be made available under the GPL.

For full details, read the copy of the GPL found in the file named `_Copying.txt_`.

Files

This distribution of ShashChessPro consists of the following files:

- Readme.md, the file you are currently reading.
- Copying.txt, a text file containing the GNU General Public License.
- src, a subdirectory containing the full source code, including a Makefile and the compilation scripts `makeAll.bat` (Windows) and `makeAll.sh` (Linux).

Uci options

Hash Memory

Hash

`_Integer, Default: 16, Min: 1, Max: 131072 MB (64-bit) : 2048 MB (32-bit)_`

The amount of memory to use for the hash during search, specified in MB (megabytes). This number should be smaller than the amount of physical memory for your system.

A modern formula to determine it is the following:

`_(T x S / 100) MB_`

where

`_T` = the average move time (in seconds)

S = the average node speed of your hardware_

A traditional formula is the following:

$$\frac{(N \times F \times T)}{512}$$

where

_N = logical threads number

F = clock single processor frequency (MB)

T = the average move time (in seconds)_

Clear Hash

Button to clear the Hash Memory.

If the Never Clear Hash option is enabled, this button doesn't do anything.

Threads

Integer, Default: 1, Min: 1, Max: 512

The number of threads to use during the search. This number should be set to the number of cores (physical+logical) in your CPU.

Ponder (checkbox)

Boolean, Default: True

Also called "Permanent Brain" : whether or not the engine should analyze when it is the opponent's turn.

Usually not on the configuration window.

MultiPV

Integer, Default: 1, Min: 1, Max: 500

The number of alternate lines of analysis to display.

Specify 1 to just get the best line. Asking for more lines slows down the search.

Usually not on the configuration window.

UCI_Chess960 (checkbox)

Whether or not ShashChess should play using Chess 960 mode. Usually not on the configuration window.

Move overhead

Default 30, min 0, max 5000

In ms, the default value seems to be the best on Linux systems, but must be increased for slow GUI like Fritz. In general, on Windows system it seems a good value to be 100.

Slow mover

Default 84, min 10, max 1000

"Time usage percent": how much the engine thinks on a move. Many engines seem to move faster and the engine is behind in time clock. With lower values it plays faster, with higher values slower - of course always within the time control.

Handicap mode

UCI_LimitStrength

Activate the strength limit by a weaker play in a random fashion to simulate human blunders.

UCI_Elo

Default 2850, min 1350, max 2850

UCI-protocol compliant version of Strength parameter. A very refined handicap mode based on the four famous sovietic chess school levels:

Internally the UCI_Elo value will be converted to a Strength value according to the following table:

- _beginner: elo < 2000_
- _intermediate: 2000 <= elo < 2200_

- `_advanced: 2200 <= elo < 2400_`
- `_expert: elo > 2400_`

Every school corresponds to a different evaluation function, more and more refined.

The UCI_Elo feature is controlled by the chess GUI, and usually doesn't appear in the configuration window.

Syzygy End Game table bases

Download at

[<http://olympuschess.com/egtb/sbases>] (<http://olympuschess.com/egtb/sbases>) (by Ronald De Man)

SyzygyPath

The path to the Syzygy endgame tablebases. this defines an absolute path on your computer to the tablebase files, also on multiple paths separated with a semicolon (;) character (Windows), the colon (:) character (OS X and Windows) character. The folder(s) containing the Syzygy EGTB files. If multiple folders are used, separate them by the ; (semicolon) character.

SyzygyProbeDepth

`_Integer, Default: 1, Min: 1, Max: 100_`

The probing tablebases depth (always the root position). If you don't have a SSD HD, you have to set it to maximize the depth and kn/s in infinite analysis and during a time equals to the double of that corresponding to half RAM size. Choice a test position with a few pieces on the board (from 7 to 12). For example:

- Fen: `_8/5r2/R7/8/1p5k/p3P3/4K3/8 w -- 0 1_` Solution : Ra4 (=)
- Fen: `_1R6/7k/1P5p/5p2/3K2p1/1r3P1P/8 b - - 1 1_`

Solution: 1...h5 !! (=)

SygyzyProbeLimit

Integer, Default: 6, Min: 0, Max: 6

How many pieces need to be on the board before ShashChess begins probing (even at the root).

Current default, obviously, is for 6-man.

Advanced Chess Analyzer

Advanced analysis options, highly recommended for CC play

Full depth threads

Integer, Default: 0, Min: 0, Max: 512

The number of settled threads to use for a full depth brute force search.

If the number is greater than threads number, all threads are for full depth brute force search.

Live Book section (thanks to Eman's author Khalid Omar for windows builds)

Live Book (checkbox)

Boolean, Default: False If activated, the engine uses the livebook as primary choice.

Live Book URL

The default is the online chessdb

[https://www.chessdb.cn/queryc_en/] (https://www.chessdb.cn/queryc_en/), a wonderful project by noobpwnftw (thanks to him!)

[<https://github.com/noobpwnftw/chessdb>] (<https://github.com/noobpwnftw/chessdb>)

[<http://talkchess.com/forum3/viewtopic.php?f=2&t=71764&highlight=chessdb>] (<http://talkchess.com/forum3/viewtopic.php?f=2&t=71764&highlight=chessdb>)

[t=71764&hlit=chessdb\)](#)

The private application can also learn from this live db.

Live Book Timeout

Default 5000, min 0, max 10000 Only for bullet games, use a lower value, for example, 1500.

Live Book Retry

Default 3, min 1, max 100 Max times the engine tries to contribute (if the corresponding option is activated: see below) to the live book. If 0, the engine doesn't use the livebook.

Live Book Diversity

Boolean, Default: False If activated, the engine varies its play, reducing conversely its strength because already the live chessdb is very large.

Live Book Contribute

Boolean, Default: False If activated, the engine sends a move, not in live chessdb, in its queue to be analysed. In this manner, we have a kind of learning cloud.

Live Book Depth

Default 100, min 1, max 100 Depth of live book moves.

Full depth threads

Default 0, min 0, max 512 The number of threads doing a full depth analysis (brute force). Useful in analysis of particular hard positions to limit the strong pruning's drawbacks.

Opening variety

Integer, Default: 0, Min: 0, Max: 40

To play different opening lines from default (0), if not from book (see below).

Higher variety -> more probable loss of ELO

* #### Use NNUE

Toggle between the NNUE and classical evaluation functions. If set to "true", the network parameters must be available to load from file (see also EvalFile), if they are not embedded in the binary.

Persisted Learning

Default is Off: no learning algorithm. The other values are "Standard" and "Self", this last to activate the [Q-learning] (https://youtu.be/ghRNvCVVJaA?list=PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv), optimized for self play. Some GUIs don't write the experience file in some game's modes because the uci protocol is differently implemented

The persisted learning is based on a collection of one or more positions stored with the following format (similar to in memory Stockfish Transposition Table):

- _best move_
- _board signature (hash key)_
- _best move depth_
- _best move score_
- _best move performance_ , a new parameter you can calculate with any learning application supporting this specification. An example is the private one, kernel of SaaS part of [ChessProbe] (<http://www.chessprobe.com>) AI portal. The idea is to calculate it based on pattern recognition concept. In the portal, you can also exploit the reports of another NLG (virtual trainer) application and buy the products in the digishop based on all this. This open-source part has the performance default. So, it doesn't use it. Clearly, even if already strong, this

private learning algorithm is a lot stronger as demonstrate here: [Graphical result](<https://github.com/amchess/BrainLearn/tree/master/tests/6-5.jpg>)

This file is loaded in an hashtable at the engine load and updated each time the engine receive quit or stop uci command.

When BrainLearn starts a new game or when we have max 8 pieces on the chessboard, the learning is activated and the hash table updated each time the engine has a best score

at a depth ≥ 4 PLIES, according to Stockfish aspiration window.

At the engine loading, there is an automatic merge to experience.bin files, if we put the other ones, based on the following convention:

&l t; fi l eType> &l t; qual i tyI ndex> . bi n

where

- _fi l eType=" experience" /" bi n"; _
- _qual i tyI ndex_ , an integer, incrementally from 0 on based on the file's quality assigned by the user (0 best quality and so on)

N. B.

Because of disk access, to be effective, the learning must be made at no bullet time controls (less than 5 minutes/game).

Read only learning

Boolean, Default: False

If activated, the learning file is only read.

Shashin section

Default: no option settled

The engine will determine dynamically the position's type starting from a "Capablanca/default positions".

If one or more (mixed algorithms/positions types at the boundaries) of the three following options are settled, it will force the initial position/algorithm understanding

Tal

Attack position/algorithm

Capablanca

Strategical algorithm (for quiescent positions)

Petrosian

Defense position/algorithm (the "reversed colors" Tal)

Acknowledgments

- Sergey Aleksandroviitch Kozlov for his very interesting patch and code on Sugar engine
- Omar Khalid for his great experience in microsoft c/cpp programming environment
- Alexei Chernakoff for his pretious suggestions about the android version and its contribution to it
- Dariusz Domagala for the Mac version
- The BrainFish, McBrain, CorChess, CiChess and Crystal authors for their very interesting derivative
- Obviously, the chess theorician Alexander Shashin, without whom I wouldn't had the idea of this engine

Stockfish community

ShashChess team

- engine owner and main developer: ICCF IM Andrea Manzo

(<https://www.iccf.com/player?id=241224>)

- IM Yohan Benitah for his professional chess understanding and help in testing against neural networks

- official tester: ICCF CCE and CCM Maurizio Platino

(<https://www.iccf.com/player?id=241094>)

- official tester: Maurizio Colbacchini, FSI 1N

- official tester and concept analyst: ICCF GM Fabio Finocchiaro (<https://www.iccf.com/player?id=240090>), 2012 ICCF world champion

- official tester Dennis Marvin (NDL) (overall the online learning)

- tester and concept analyst: ICCF GM Matjas Pirs

(<https://www.iccf.com/player?id=480232>), for his great experience and tests on positions analysis in different game's phases

Sorry If I forgot someone.

<h1 align="center">Stockfish NNUE</h1>

Overview

[![Build Status](<https://travis-ci.org/official-stockfish/Stockfish.svg?branch=master>)](<https://travis-ci.org/official-stockfish/Stockfish>)

[![Build Status](<https://ci.appveyor.com/api/projects/status/github/official-stockfish/Stockfish?branch=master&svg=true>)](<https://ci.appveyor.com/project/mcostalba/stockfish/branch/master>)

[Stockfish](<https://stockfishchess.org>) is a free, powerful UCI chess engine derived from Glaurung 2.1. Stockfish is not a complete chess program and requires a UCI-compatible graphical user interface (GUI) (e.g. XBoard with PolyGlott, Scid, Cute Chess, eboard, Arena, Sigma Chess, Shredder, Chess

Partner or Fritz) in order to be used comfortably. Read the documentation for your GUI of choice for information about how to use Stockfish with it.

The Stockfish engine features two evaluation functions for chess, the classical evaluation based on handcrafted terms, and the NNUE evaluation based on efficiently updateable neural networks. The classical evaluation runs efficiently on almost all CPU architectures, while the NNUE evaluation benefits from the vector intrinsics available on most CPUs (sse2, avx2, neon, or similar).

Files

This distribution of Stockfish consists of the following files:

- * Readme.md, the file you are currently reading.
- * Copying.txt, a text file containing the GNU General Public License version 3.
- * src, a subdirectory containing the full source code, including a Makefile that can be used to compile Stockfish on Unix-like systems.
- * a file with the .nnue extension, storing the neural network for the NNUE evaluation. Binary distributions will have this file embedded.

Note: to use the NNUE evaluation, the additional data file with neural network parameters needs to be available. Normally, this file is already

embedded in the binary or it can be downloaded.
The filename for the default (recommended) net can be found as the default value of the `EvalFile` UCI option, with the format `nn-[SHA256 first 12 digits].nnue` (for instance, `nn-c157e0a5755b.nnue`). This file can be downloaded from
` ``

[https://tests.stockfishchess.org/api/nn/\[filename\]](https://tests.stockfishchess.org/api/nn/[filename])
` ``

replacing `[filename]` as needed.

UCI options

Currently, Stockfish has the following UCI options:

- * ##### Threads
The number of CPU threads used for searching a position. For best performance, set this equal to the number of CPU cores available.
- * ##### Hash
The size of the hash table in MB. It is recommended to set Hash after setting Threads.
- * ##### Ponder
Let Stockfish ponder its next move while the opponent is thinking.
- * ##### MultiPV
Output the N best lines (principal variations, PVs) when searching.
Leave at 1 for best performance.
- * ##### EvalFile
The name of the file of the NNUE evaluation parameters. Depending on the GUI the filename might have to include the full path to the folder/directory that contains the file.

Other locations, such as the directory that contains the binary and the working directory, are also searched.

* ##### UCI_AnalyseMode

An option handled by your GUI.

* ##### UCI_Chess960

An option handled by your GUI. If true, Stockfish will play Chess960.

* ##### UCI_ShowWDL

If enabled, show approximate WDL statistics as part of the engine output.

These WDL numbers model expected game outcomes for a given evaluation and game play for engine self-play at fishtest LTC conditions (60+0.6s per game).

* ##### UCI_LimitStrength

Enable weaker play aiming for an Elo rating as set by UCI_Elo. This option overrides Skill Level.

* ##### UCI_Elo

If enabled by UCI_LimitStrength, aim for an engine strength of the given Elo.

This Elo rating has been calibrated at a time control of 60s+0.6s and anchored to CCRL 40/4.

* ##### Skill Level

Lower the Skill Level in order to make Stockfish play weaker (see also UCI_LimitStrength).

Internally, MultiPV is enabled, and with a certain probability depending on the Skill Level a weaker move will be played.

* ##### SyzygyPath

Path to the folders/directories storing the Syzygy tablebase files. Multiple directories are to be separated by ";" on Windows and

by ":" on Unix-based operating systems. Do not use spaces around the ";" or ":".

Example:

```
`C:\tablebases\wdl 345; C:\tablebases\wdl 6; D:\tablebases\dtz 345; D:\tablebases\dtz6`
```

It is recommended to store .rtbw files on an SSD. There is no loss in storing the .rtbz files on a regular HD. It is recommended to verify all md5 checksums of the downloaded tablebase files (`md5sum -c checksum.md5``) as corruption will lead to engine crashes.

* ##### SyzygyProbeDepth

Minimum remaining search depth for which a position is probed. Set this option to a higher value to probe less aggressively if you experience too much slowdown (in terms of nps) due to TB probing.

* ##### Syzygy50MoveRule

Disable to let fifty-move rule draws detected by Syzygy tablebase probes count as wins or losses. This is useful for ICCF correspondence games.

* ##### SyzygyProbeLimit

Limit Syzygy tablebase probing to positions with at most this many pieces left (including kings and pawns).

* ##### Contempt

A positive value for contempt favors middle game positions and avoids draws, effective for the classical evaluation only.

* ##### Analysis Contempt

By default, contempt is set to prefer the side to move. Set this option to "White" or "Black" to analyse with contempt for that side, or "Off" to disable contempt.

* ##### Move Overhead

Assume a time delay of x ms due to network and GUI overheads. This is useful to avoid losses on time in those cases.

* ##### Slow Mover

Lower values will make Stockfish take less time in games, higher values will make it think longer.

* ##### nodetime

Tells the engine to use nodes searched instead of wall time to account for elapsed time. Useful for engine testing.

* ##### Clear Hash

Clear the hash table.

* ##### Debug Log File

Write all communication to and from the engine into a text file.

A note on classical and NNUE evaluation

Both approaches assign a value to a position that is used in alpha-beta (PVS) search to find the best move. The classical evaluation computes this value as a function of various chess concepts, handcrafted by experts, tested and tuned using fishtest.

The NNUE evaluation computes this value with a neural network based on basic inputs (e.g. piece positions only). The network is optimized and trained on the evaluations of millions of positions at moderate

search depth.

The NNUE evaluation was first introduced in shogi, and ported to Stockfish afterward.

It can be evaluated efficiently on CPUs, and exploits the fact that only parts of the neural network need to be updated after a typical chess move.

[The nodchip repository] (<https://github.com/nodchip/Stockfish>) provides additional tools to train and develop the NNUE networks.

On CPUs supporting modern vector instructions (avx2 and similar), the NNUE evaluation results in stronger playing strength, even if the nodes per second computed by the engine is somewhat lower (roughly 60% of nps is typical).

Note that the NNUE evaluation depends on the Stockfish binary and the network parameter file (see EvalFile). Not every parameter file is compatible with a given Stockfish binary. The default value of the EvalFile UCI option is the name of a network that is guaranteed to be compatible with that binary.

What to expect from Syzygbases?

If the engine is searching a position that is not in the tablebases (e.g. a position with 8 pieces), it will access the tablebases during the search.

If the engine reports a very large score (typically 153.xx), this means that it has found a winning line into a tablebase position.

If the engine is given a position to search that is in the tablebases, it

will use the tablebases at the beginning of the search to preselect all good moves, i.e. all moves that preserve the win or preserve the draw while taking into account the 50-move rule. It will then perform a search only on those moves. **The engine will not move immediately**, unless there is only a single good move. **The engine likely will not report a mate score even if the position is known to be won.**

It is therefore clear that this behaviour is not identical to what one might be used to with Nalimov tablebases. There are technical reasons for this difference, the main technical reason being that Nalimov tablebases use the DTM metric (distance-to-mate), while Syzygybases use a variation of the DTZ metric (distance-to-zero, zero meaning any move that resets the 50-move counter). This special metric is one of the reasons that Syzygybases are more compact than Nalimov tablebases, while still storing all information needed for optimal play and in addition being able to take into account the 50-move rule.

Large Pages

Stockfish supports large pages on Linux and Windows. Large pages make the hash access more efficient, improving the engine speed, especially on large hash sizes. Typical increases are 5..10% in terms of nodes per second, but speed increases up to 30% have been measured. The support is

automatic. Stockfish attempts to use large pages when available and will fall back to regular memory allocation when this is not the case.

Support on Linux

Large page support on Linux is obtained by the Linux kernel transparent huge pages functionality. Typically, transparent huge pages are already enabled and no configuration is needed.

Support on Windows

The use of large pages requires "Lock Pages in Memory" privilege. See [Enable the Lock Pages in Memory Option (Windows)](<https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/enable-the-lock-pages-in-memory-option-windows>) on how to enable this privilege, then run [RAMMap](<https://docs.microsoft.com/en-us/sysinternals/downloads/rammap>) to double-check that large pages are used. We suggest that you reboot your computer after you have enabled large pages, because long Windows sessions suffer from memory fragmentation which may prevent Stockfish from getting large pages: a fresh session is better in this regard.

Compiling Stockfish yourself from the sources

Stockfish has support for 32 or 64-bit CPUs, certain hardware instructions, big-endian machines such as Power PC, and other platforms.

On Unix-like systems, it should be easy to compile Stockfish directly from the source code with the included Makefile in the folder ``src``. In general it is recommended to run ``make help`` to see a list of make targets with corresponding descriptions.

```
...  
    cd src  
    make help  
    make net  
    make build ARCH=x86-64-modern  
...
```

When not using the Makefile to compile (for instance with Microsoft MSVC) you need to manually set/unset some switches in the compiler command line; see file `*types.h*` for a quick reference.

When reporting an issue or a bug, please tell us which version and compiler you used to create your executable. These informations can be found by typing the following commands in a console:

```
...  
    ./stockfish compiler  
...
```

Understanding the code base and participating in the project

Stockfish's improvement over the last couple of years has been a great community effort. There are a few ways to help contribute to its growth.

Donating hardware

Improving Stockfish requires a massive amount of testing. You can donate your hardware resources by installing the [FishTest Worker] (<https://github.com/glinnscott/fishtest/wiki/Running-the-worker:-overview>) and view the current tests on [FishTest] (<https://tests.stockfishchess.org/tests>).

Improving the code

If you want to help improve the code, there are several valuable resources:

- * [In this wiki,] (<https://www.chessprogramming.org>) many techniques used in Stockfish are explained with a lot of background information.

- * [The section on Stockfish] (<https://www.chessprogramming.org/Stockfish>) describes many features and techniques used by Stockfish. However, it is generic rather than being focused on Stockfish's precise implementation. Nevertheless, a helpful resource.

- * The latest source can always be found on [GitHub] (<https://github.com/official-stockfish/Stockfish>). Discussions about Stockfish take place in the [FishCooking] (<https://groups.google.com/forum/#!forum/fishcooking>) group and engine testing is done on [FishTest] (<https://tests.stockfishchess.org/tests>). If you want to help improve Stockfish, please read this [guide] (<https://github.com/glinnscott/fishtest/wiki/Creating-my-first-test>) first, where the basics of Stockfish development are explained.

Terms of use

Stockfish is free, and distributed under the ****GNU General Public License version 3**** (GPL v3). Essentially, this means that you are free to do almost exactly what you want with the program, including distributing it among your friends, making it available for download from your web site, selling it (either by itself or as part of some bigger software package), or using it as the starting point for a software project of your own.

The only real limitation is that whenever you distribute Stockfish in some way, you must always include the full source code, or a pointer to where the source code can be found. If you make any changes to the source code, these changes must also be made available under the GPL.

For full details, read the copy of the GPL v3 found in the file named ***Copying.txt***.