

Parameters in StelvioUI.ini

All user parameters with their default values.

General parameters:

- histogramMode = false. Instead of playing strategies, only count them and display them in a histogram. Default is false. Starts counting at the strategy defined by startAtStrategyNr.
- includeTestWithHalfMoveLess = false. In case true, then any input SPG is solved a second time but with the available half-moves reduced by one (say an SPG is 20.0 in re-solved with 19.5). With this option, Stelvio is guaranteed to find any shorter solution to an SPG (next to solutions with the given length).

Input / Output parameters:

- inputFileName = problems.txt
- outputFileName = problems_out.txt
- pgnOutput = false. Write solutions/cooks in pgn format. Actually, the output moves are always fully qualified (like in the non-pgn case), which is sufficient for use in tools like lichess.org.

Strategy read/write parameters:

- saveEveryXStrategiesToFile = -1. Save all found strategies to disk (in the subfolder /strategies/fen/...). The value -1 means nothing is saved. A value >1 means that this many strategies are written into a single file. A reasonable value is around 1000, so 1000 strategies per file (btw: a strategy requires about 2 kb of disk space).
- readStrategiesFromFile = false. In case this is true, instead of searching for strategies, Stelvio reads strategies previously written to disk using the saveEveryXStrategiesToFile option.

Strategy playing parameters:

- retractionMode = kingInCheck. Possible values = {none, kingInCheck}. In case this is set to kingInCheck, then: If the king is in check in the diagram position, then solving behaves differently: All possible last moves are initially searched for. All these last moves are then retracted one by one and the resulting shorter SPGs solved. The partial solving results are then combined into one overall result.

Parallelizing parameters:

In general, you should adjust the parameters according to your hardware, but also according to the SPG that you want to solve. To figure out what is best, I recommend playing a bit with different values for different problems and check how the solving times react. Btw: Histogram mode also comes with parallel calculation option.

- numStrategySeekers = 2. This can be set to a higher value in case your computer has multiple processors. In many cases, strategy seeking is fast anyway and this is not so useful. But in case the SPG has a lot of captures or promotions, then setting this to a higher value can be beneficial.

- `numStrategyPlayers = 1`. Setting this value is a bit tricky. Since strategy players sometimes require a lot of memory, it can be detrimental to performance to set this value >1 , in case you do not have enough memory. Because: If while playing a strategy, the position cache is saturated (which is displayed in the UI on the right), then playing becomes a lot slower. And since each strategy player needs its own position cache, this saturation will be reached quicker in case strategies are played in parallel. But in case the memory is sufficient for parallel playing, this option can speed up the solving process considerably.
- `strategyQueueSize = 100000`. The maximum number of strategies that the strategy seekers can put into the queue. When the queue is full, they have to wait until the strategy players have taken strategies off the queue.

Strategy seeking parameters:

- `printStrategies = false`. Print all strategies into the output file. This can easily fill your disk in some cases, as strategies can come in the tens of millions. Nice.
- `collectSeekPartMetricsAfterXCycles = 2`. The interval length for which strategy seeking information is gathered for displaying in the UI. Setting this to -1 gathers no information and is the fastest option. Setting this to 1 maximizes the UI updates at the expense of a slight performance penalty. The default 2 is a compromise between performance and update frequency.

Strategy condition parameters:

- `startAtStrategyNr = 1`. This can be set to a high value in case the first x strategies should be skipped. Also applies to histogram mode. Beware that strategy numbers are only well-defined in case of there is only 1 strategy seeker.

Strategy analysis parameters:

- `expensiveCollisionDetectionMode = default`. Possible values {default, on, off}. There are two ways for Stelvio to do collision detection:
 - A more involved analysis that might find more collisions at the expense of time consumed.
 - A less involved analysis that consumes less time but might find less collisions.
 - With `expensiveCollisionDetectionMode = on`, the first option is chosen.
 - With `expensiveCollisionDetectionMode = off`, the second option is chosen.
 - With `expensiveCollisionDetectionMode = default`, the first option is chosen in case the SPG has at least 28 pieces, otherwise the second option is chosen.
- `advancedCycleDetectionMode = default`. Possible values {default, on, off}. There are two ways for Stelvio to do cycle detection:
 - A more involved analysis that might find more cycles at the expense of time consumed.
 - A less involved analysis that consumes less time but might find less cycles.
 - With `advancedCycleDetectionMode = on`, the first option is chosen.
 - With `advancedCycleDetectionMode = off`, the second option is chosen.

- With `advancedCycleDetectionMode = default`, the first option is chosen in case the SPG has at least 28 pieces, otherwise the second option is chosen.

Cook parameters:

- `maxSolutionsPerCook = 2`. Number of cook-solutions to add to the output file per cook-strategy.
- `stopAfterXCooks = 1`. Abort the solving process after finding this number of cook strategies.
- `printCookStrategy = false`. Additionally add the strategy to the output file for found cooks.

Cache parameters:

- `positionCacheMaxOverallExponent = -1`. Technical value to limit the main cache size which is usually guessed by the given RAM available. It can be useful, albeit rarely, to downsize the cache in the special case when the rest of the needed artifacts require a lot of memory. One example would be a problem with many promoted rooks, as there are then a huge number of permutations in memory, which rook is what piece initially. Values below 25 are ignored. $2^{\text{positionCacheMaxOverallExponent}}$ is used.
- `positionCacheSplitExponent = 2`. How much should we split up the huge cache array into parts. Values 2-10. This has a performance impact, but has to be analyzed still to know what is best.
- `positionCacheMaxChainLength = 500`. How long should we look for an empty slot in the cache / try to find our position in the cache. Nobody knows what is best here, but value should probably be > 100 . This also has a performance impact.
- `positionCacheFullEvictionThreshold = 0.85`. The threshold in cache size that triggers eviction. Between 0 and 1. A value > 1 means never. This also has a performance impact.
- `positionCacheSizeAdjustment = 0`. The secondary cache can be made slightly larger or smaller. In case other artifacts need a lot of memory, this can be set to -1, decreasing this cache's size. The value 1 on the other hand increases the secondary cache to the maximum, but if you are unlucky, Stelvio will blow up with an out of memory error. Possible values are -1, 0, 1.