# 6.106 Project 4 Final Write-Up

AmarBot: Ishank Agrawal, Richard Chen, Viktor Fukala, Andrew Liu

Dec 13, 2023

## Contents

# 0 Project Log

| Date | Start | Person | Duration | Description |
|---|---|---|---|---|
| Sat November 18 | 2:00 pm | A, I, V | 3 | reading up on the project |
| Sun Novemeber 19 | 10:00am | I | 2 | create correctness test suite, python interfacing with UCI |
| Mon November 20 | 4:30 pm | V | 1 | maintaining monarch positions |
| Mon November 29 | 8:00 pm | everyone | 3 | read literation on NNUE |
| Mon November 20 | 6:00 pm | everyone | 3 | project design document |
| Fri December 1 | 5:00 pm | V | 2 | testing scripts |
| Fri December 1 | 8:00 pm | V | 2 | unsuccessfully trying to remove extra sentinel squares |
| Fri December 1 | 8:00 pm | R | 6 | unsuccessfully tried to represent entire board with uint64 |
| Fri December 1 | 10:00 pm | V | 1 | `piece_t` using bitfields |
| Sat December 2 | 0:30 am | V | 1 | removing laser coverage and testing that |
| Sat December 2 | 4pm | I | 3 | create eval function testing suites, profile. Noticed that `abs_qi` bottleneck for eval computation |
| Sat December 2 | 4pm | R | 3 | small optimizations: inlining, precomputation of reciprocal floats |
| Sat December 2 | 4pm | R | 3 | changed to 10 x 10 board |
| Sun December 3 | 10:00 pm | V | 4 | optimizations (`is_draw`, sort, inlining, etc.) |
| Mon December 4 | 12:00 am | A | 12 | more robust testing scripts, set up collab to retrain weights and start training games, turn various misc. functions into lookup tables, other small optimizations |
| Mon December 4 | 7:00 pm | V | 2 | measuring playing power to find best bot |
| Mon December 4 | 4:00pm | I | 2 | maintaining `abs_qi` as a part of the board state |
| Mon December 4 | 5:00pm | everyone | 5 | merging branches, including lookup tables, final touch ups |
| Thu December 7 | 7:00 pm | everyone | 3 | beta write-up |
| Fri December 8 | 6:00 pm | A | 24 | implemented young siblings wait along with a number of other search heuristics |
| Sat December 9 | 12:00 pm | R | 6 | began merging our beta board rep with the rank 0 board rep, for speeding up eval |
| Sat December 9 | 6:00 pm | A,R,V | 6 | continued optimizing the board rep. wrote scraper for the scrimmage server and curated a small lookup table based on common moves. fixed a bug in the parallel search. |
| Sun December 10 | 6:00 pm | A,R | 6 | condensed representation for the transposition table, among other smaller heuristics. |
| Mon December 11 | 6:00 pm | everyone | 2 | worked on poster presentation. |
| Mon December 11 | 8:00 pm | I | 5 | wrote more complex script for generating a larger opening book. |
| Mon December 11 | 11:00 pm | V | 1.5 | exposing evaluation of positions with deep search to the program interface |
| Tues December 12 | 6:00 am | A | 1.5 | condensed `move_t` representation and unsuccessfully tried implementing ABDADA parallel search. |
| Tues December 12 | 6:00 pm | everyone | 3 | final attempt at larger opening book, implemented some smaller heuristics like stockfish eviction policy and unrolling. |
| Wed December 13 | 7:00 pm | everyone | 2 | final write-up |

# 1 Executive Summary

Our optimizations were generally split into three different categories: those that improved the speed of eval, those that improved the efficiency of search, and meta-optimizations. We found that optimizations to both eval and search proved to be equally important for the performance of our bot, while the meta-optimizations, which includes a small opening book and weight tuning, improved performance slightly.

# 2 Testing and Evaluation

## 2.1 Testing

For testing prior to the beta, we set up a simple testing framework with the goal of giving us some idea of how effective our optimizations were without having to submit them to the autotester.

Our testing framework included a simple correctness script that ran `perft` on two input binaries, and a small test suite of pre-generated fens, and compared the outputs. The purpose of this script was to test correctness for changes that were purely speed optimizations and did not change any actual logic in how moves were selected (e.g., changing the board representation).

We also set up a simple speed comparison script that compared the nodes searched per second for two input binaries. Our hope with this script was that we could easily identify when an optimization was "effective" by comparing this metric between two binaries.

Unfortunately, even with `awsrun`, there was lots of variance and it was quite difficult to use in practice. Therefore, between the beta and the final, we took inspiration from post 1275, and set up testing on Modal Labs to make our testing pipeline more efficient. Despite the scripts that we set up, testing still felt very inefficient, and it was always difficult to tell if an optimization was making the bot better. Thus, setting up Modal ended up being a critical part of our improvements after the beta, as it streamlined testing and made things much easier. We ended up using a few hundred CPU hours on testing alone; by utilizing the free credits given to new accounts, we luckily did not have to pay much for this compute.

## 2.2 Eval

After setting up Modal labs, the way that we evaluated two binaries was to play 200 games with `fis = 10 0.4`. From here on out, we notate performance improvements with e.g. [70 30], denoting that the optimization led to a 70% win rate of the bot with the improvement over the bot without the improvement. Performance improvements made before the beta will not have this metric attached.

# 3 Identification of Performance Bottlenecks

Based on perf reports, we identified that the following items were, often unnecessarily, degrading our performance (sorted roughly by decreasing performance impact)

- searching for the monarchs in `get_monarch` (25 % of all execution time)
- tracing the laser paths in the laser coverage heuristic (17 % of all execution time)

| Time spent | Function |
|---|---|
| 27.01% | laser_coverage |
| 23.40% | __memmove_avx_unaligned_erms |
| 7.84% | low_level_make_move |
| 5.21% | evaluateMove |
| 3.84% | get_move |
| 3.73% | scout_search |
| 3.33% | eval |
| 3.31% | generate_all |
| 2.44% | square_of |
| 2.33% | make_move |
| 2.20% | tt_hashtable_get |
| 1.88% | beam_of |
| 1.84% | abs_qi |
| 1.45% | rnk_of |
| 1.32% | get_monarch |

Figure 1: Profiling after monarchs optimization, laser_coverage took a lot of time, and had little weight in the evaluation

| Time spent | Function |
|---|---|
| 31.28% | eval |
| 20.11% | fen_to_pos |
| 17.32% | abs_qi |
| 4.47% | mface |
| 4.47% | rel_qi |
| 4.47% | square_of |
| 3.35% | fil_of |
| 2.79% | compute_zob_key |
| 2.23% | rnk_of |
| 1.68% | dir_of |
| 1.68% | find_monarchs |
| 1.68% | mcede |
| 1.12% | read |
| 1.12% | get_monarch |

Figure 2: Profiling eval after removing laser_coverage heuristic; abs_qi was observed to be the bottleneck

- copying/moving `position_t` objects (which representing the board state) (up to 15 % of all execution time)
- search through the board history in `is_draw`
- precalculating the absolute and relative qi heuristics
- `sort_insertion` in `searchPV` and `scout_search`
- function call overhead for simple functions such as `square_of`, `rnk_of`, or `fil_of`
- mutex operations around accesses to `node_count_serial`

# 4 Optimizations to Eval

Based on the performance bottlenecks that we identified, we made the following optimizations related to position evaluation for the bot:

- removing the laser coverage heuristic (we don't calculate laser coverage and we keep the original weights for all the other heuristics)
- decreasing the size of the `position_t` struct by storing a `piece_t` in a single `uint8_t` by using bitfields for the type, color, and rotation of the piece. we also replaced the original 16-by-16 board by a 10-by-10 board.
- decreasing the size of the `move_t` struct by storing it in a `uint32_t`. [60 40]
- early termination of the history search in `is_draw` once
  - enough repetitions of the current positions are found for a draw to be declared (return `true`), or
  - a move which removes a piece from the board is encountered (return `false`)
- stored a bit board for pawns and monarchs. this made metrics like `abs_qi`, `rel_qi`, among others, trivial to compute, which ended up giving us significant speedup [70 30]. we give credit to the rank 0 beta code for giving us major inspiration on more efficient ways to optimize this; we had a version of this that was much more inefficient before the beta, and we ended up implementing many of the optimizations that were present in rank 0 beta to help give us more speedup.
- changing `sort_insertion` to first move all moves with key 0 to the end of the array and then sort the remaining moves using `qsort` from the standard library. for very small arrays, we performed a manual insertion sort instead of calling `qsort`. [60 40]
- moved functions such as `square_of`, `rnk_of`, `fil_of`, `qi_at`, `centrality` into lookup tables. [60 40]
- various inlining and unrolling of loops inside of the main `eval` function. [55 45]

# 5 Optimizations to Search

- parallelized both `search_scout` and `search_pv` with the young siblings wait algorithm. we were able to do some fine tuning by adjusting the number of branches to search serially in both methods before running the rest in parallel. we ultimately found that 5 serial searches for `search_scout` and 6 serial seraches for `search_pv` gave us the best performance. [70 30]
- before getting all moves inside of both `search_scout`, `search_pv`, we first (serially) explored the move in the hashtable, betting on the fact that this hashtable move would lead to pruning with high frequency, thus allowing us to skip move generation. [60 40]
- made functions specific to either `search_scout` or `search_pv`, which were originally in `search_common`. our reasoning was that it was quite easy to copy functions into either the scout search or pv search files and save the overhead from functions that head to check search type. [55 45]

- reduced the size of the transposition table and set the cache size to be 4-way associative. we reduced the size by condensing the `ttRec_t` type into a `uint64_t`, which was possible because various types inside of the struct could be modified from `int` to smaller types like `uint8_t`. after grid searching, we found the 4-way cache to give the best performance. overall, our optimizations to the transposition table led to [60 40].
- this change was inspired by the poster presentation given by `elo_engineers`: we slightly modified the metric that was being used to evict entries from the transposition table, to be the one used by stockfish. [55 45]

# 6 Meta-optimizations

- We implemented a small lookup table that we created by scraping the most recent 1000 games from the server, and adding the most commonly winning moves from the first 6 depths into a lookup table. The final table did not end up having that many positions, but still led to around [60 40] improvement. We tried to make a much more extensive opening table, but were unfortunately not able to make it work (see "failed optimizations").
- We retrained weights after significantly modifying our search logic. After running 1.2 million games using the data generation script provided, we trained new weights based on the data from the games, where we intentionally left out the `laser_coverage` statistic as to fine-tune more effectively. After training new weights multiple times on the same data, we found that there were a few local minima that the new weights fell into, including a local minima that seemed to be similar to the original weights provided by the staff. After trying all of these local minima, we found that the local minima corresponding to the weights that seemed similar to the original staff weights worked the best. Thus, we ended up with weights that were numerically similar to the staff weights, maybe off by at most `0.1` in all metrics (the only exception being `laser_coverage`, which we set to 0, since that is how we retrained the weights). [60 40].

# 7 Failed optimizations

- we attempted to memoize the `abs_qi` metric by precomputing and then only making point updates inside of `low_level_make_move`. even though this seemed like it should obviously improve performance, we unfortunately did not see much speedup. we're not really sure why; one guess is that the weight for the `abs_qi` metric is small, so maybe it did not matter too much? (we're not convinced that this is a valid reason, but we weren't able to come up with any other explanation)
- we attempted to create a very large opening book by creating a script that generated strong opening moves given a fixed branching factor $b$. unfortunately, despite searching with `go depth 10` for branching factor $b = 4$, the opening book we came up did not beat the opening book generated just by looking at common moves. we feel we might have been able to come up with a stronger open book by combining these two strategies, i.e., running a deep eval on common moves, which is one way we might have been able to further improve our bot.

- inside of search, we currently search the hash table move serially before generating the move list. we also tried searching the killer move table moves `killer_a` and `killer_b` before generating the move list, since killer moves would also end up at the front of the move list after sorting (behind the hash table move). this degraded the performance of the search considerably; we're not too sure why.

# 8 Team Dynamics

We didn't always stick to a strict division-of-labor plan, but instead started working on whatever seemed most important as the next step after checking that nobody else has already done it. Sometimes, two people tried doing the same thing when it was unusually complicated and then we combined our ideas.

# 9 Completeness, Expected Performance

**Player Ranking**

| Player | W | L | T | Elo |
|---|---|---|---|---|
| AVAnishingKnight | 215 | 126 | 13 | 1511 |
| bot6 | 387 | 436 | 27 | 1497 |
| reference_2 | 1180 | 381 | 70 | 1488 |
| duckgm1 | 558 | 580 | 62 | 1455 |
| AmarBot | 269 | 227 | 15 | 1393 |
| kabab | 486 | 255 | 34 | 1387 |
| VMware-Horizon-Client | 235 | 243 | 17 | 1354 |
| Republic-of-GAMERS | 171 | 138 | 25 | 1338 |
| CacheMeOutside | 367 | 299 | 14 | 1333 |
| koalas | 295 | 287 | 47 | 1328 |
| Project-4-Rank-0 | 346 | 343 | 41 | 1287 |
| EmAIL | 77 | 56 | 5 | 1275 |
| elo-engineers | 132 | 147 | 10 | 1263 |
| --- | 401 | 342 | 27 | 1261 |

We expect that our code runs without issues, i.e., fully compiles both serially and parallel, and makes legal moves. Based on tests in the scrimmage server, we expect our win rate against `reference_1` to be close to 100%.

# 10 Acknowledgment

- We acknowledge the staff for their help on Piazza and during office hours. We are grateful for their hard work on making this class run as smoothly as possible.
- Post 1275 about Modal helped us improve our workflow significantly.

- We acknowledge the Chess Programming Wiki website for explaining the relevant chess bot programming concepts, many of which are relevant to Leiserchess too.