

# 6.172 Final Project

Yinzhan Xu, Haoran Xu, Yuzhou Gu, Chengkai Zhang

# Main results

Player Ranking				
Player	W	L	T	Elo
ckzhang-lemon510-xyzhan-yuzhougu	593	352	32	1514
budmonde-jefflu-parke-rsridhar	196	124	12	1309
liuaofei-rongil-werryju	243	287	16	1285
andrewhe-dzd123-hsteven-slv	846	493	53	1275
dansolo-earmstro-eewayhsu	114	69	8	1262
mkilgore-pnoyola-skamali	162	146	20	1248
minasyan-rthorn-rverkuil-tmansour	38	26	8	1248
akshayr-kimberli-vfazel	172	57	1	1233
abiswas-gcllang-kezike17-rcha	36	28	4	1231
ajays235-bkhadka-dgrullon-kezilu	778	293	29	1219
boki-jiahaoli-pgentili-sli2014	53	32	7	1217
akonradi-tomascg-tomasg-meganp	63	53	6	1208
cavery-dkogut-kevinzhu-lwang32	193	47	6	1208
cmozarmi-stewart-waack-wnoble	6	4	0	1207

Player Ranking				
Player	W	L	T	Elo
ckzhang-lemon510-xyzhan-yuzhougu	1141	541	53	1418
liuaofei-rongil-werryju	344	456	27	1338
budmonde-jefflu-parke-rsridhar	196	124	12	1309
andrewhe-dzd123-hsteven-slv	846	493	53	1275
dansolo-earmstro-eewayhsu	114	69	8	1262
minasyan-rthorn-rverkuil-tmansour	38	26	8	1248
mkilgore-pnoyola-skamali	162	146	20	1248
akshayr-kimberli-vfazel	172	57	1	1233
reference_plus	1080	1170	71	1226
ajays235-bkhadka-dgrullon-kezilu	781	295	30	1225
abiswas-gcllang-kezike17-rcha	39	33	4	1221
boki-jiahaoli-pgentili-sli2014	53	32	7	1217
akonradi-tomascg-tomasg-meganp	69	59	6	1211
cmozarmi-stewart-waack-wnoble	6	4	0	1207

Highest: 1514

Current: 1418

Average depth: Blitz: 7.6, Regular: 8.3

# Outline

- 1 Bottleneck Improvement
- 2 Openbook
- 3 Constant Optimization
- 4 Strategies without performance gains

# Outline

1 Bottleneck Improvement

2 Openbook

3 Constant Optimization

4 Strategies without performance gains

# Bottleneck Improvement - scout\_search

`scout_search` takes 32.55% of the time and does the following:

- \* Get a list of possible moves
- \* Sort the moves
- \* Check each move in order, do a recursive search if *necessary*

# Bottleneck Improvement - scout\_search

- \* Hash table move from pre-evaluation and killer moves are returned with high probability  
⇒ check them first before generating move list
- \* A move is ignored if the node is quiescent and there is no victim  
⇒ conservatively predict number of victims, exclude moves at quiescent node with no victim from move list
- \* About half of the moves have sort key of 0  
⇒ move them to end of move list directly and exclude from sorting procedure
- \* Maintaining node count introduces true sharing  
⇒ remove counting

# Outline

- 1 Bottleneck Improvement
- 2 Openbook
- 3 Constant Optimization
- 4 Strategies without performance gains

# Motivation of Openbook

Underlying Assumption:

- \* Good AI makes similar moves.
- \* Possible good moves for a given game state are limited.

Number of possible openings between two good AIs is reasonably small.

Openbook!



# Motivation of Openbook

Underlying Assumption:

- \* Good AI makes similar moves.
- \* Possible good moves for a given game state are limited.

Number of possible openings between two good AIs is reasonably small.

Openbook!

# Justification of Openbook

Validating the idea on real-world data:

- \* Downloaded the most recent 83000 games from Scrimmage.
- \* Training Set: about 39000 games.
- \* Test Set: about 44000 games.
- \* Consider the first 5 rounds of game.
- \* In Training Set, 782 (2%) openings occurred at least twice.
- \* In Testing Set, 2/3 of the openings falls into the 782 openings.

Hits 2/3 of the games with only 782 records!

# Justification of Openbook

Validating the idea on real-world data:

- \* Downloaded the most recent 83000 games from Scrimmage.
- \* Training Set: about 39000 games.
- \* Test Set: about 44000 games.
- \* Consider the first 5 rounds of game.
- \* In Training Set, 782 (2%) openings occurred at least twice.
- \* In Testing Set, 2/3 of the openings falls into the 782 openings.

Hits 2/3 of the games with only 782 records!

# Justification of Openbook

Validating the idea on real-world data:

- \* Downloaded the most recent 83000 games from Scrimmage.
- \* Training Set: about 39000 games.
- \* Test Set: about 44000 games.
- \* Consider the first 5 rounds of game.
- \* In Training Set, 782 (2%) openings occurred at least twice.
- \* In Testing Set, 2/3 of the openings falls into the 782 openings.

Hits 2/3 of the games with only 782 records!

# Justification of Openbook

Validating the idea on real-world data:

- \* Downloaded the most recent 83000 games from Scrimmage.
- \* Training Set: about 39000 games.
- \* Test Set: about 44000 games.
- \* Consider the first 5 rounds of game.
- \* In Training Set, 782 (2%) openings occurred at least twice.
- \* In Testing Set, 2/3 of the openings falls into the 782 openings.

Hits 2/3 of the games with only 782 records!

# Advantage of Openbook

Openbook offers two main advantages:

- \* We can search very deep for a good move in openbook.  
So for the first several moves, our choice is very optimized.
- \* And those moves take no time at all!

If tested using default timing strategy:

- \* Hitting 5 rounds: 20s advantage in Regular, 8s advantage in Blitz.
- \* Hitting 10 rounds: 38s advantage in Regular, 15s advantage in Blitz.

# Advantage of Openbook

Openbook offers two main advantages:

- \* We can search very deep for a good move in openbook.  
So for the first several moves, our choice is very optimized.
- \* And those moves take no time at all!

If tested using default timing strategy:

- \* Hitting 5 rounds: 20s advantage in Regular, 8s advantage in Blitz.
- \* Hitting 10 rounds: 38s advantage in Regular, 15s advantage in Blitz.

# Calculating Openbook

Generating popular openings:

- \* Used MySQL to manage data set for its convenience and power, and easiness to interact with web applications.
- \* Downloaded the most recent 83000 games from Scrimmage.
- \* Extracted frequent openings, and store them into MySQL.
- \* Search depth varies from 9 to 11 for each opening move.
- \* Openings with higher # of occurrences are calculated with deeper depth, for a possibly better move.



# Calculating Openbook

More than 100000 openings generated.

Impossible to calculate all of them with a single machine!

Distributed computing!

- \* LAMP (Linux+Apache+MySQL+PHP) web server to distribute down tasks and collect up results.
- \* Clients use WGET to interact with web server.
- \* 150+ CPUs in Microsoft Azure.
- \* 15000+ CPU Hours in total.

# Calculating Openbook

More than 100000 openings generated.

Impossible to calculate all of them with a single machine!

Distributed computing!

- \* LAMP (Linux+Apache+MySQL+PHP) web server to distribute down tasks and collect up results.
- \* Clients use WGET to interact with web server.
- \* 150+ CPUs in Microsoft Azure.
- \* 15000+ CPU Hours in total.

# Calculating Openbook

Screenshot of our web server:

## 6172 Final Project

---

Below is current progress.

Depth	Total	Calculating	Completed
9	153857	28	49144
10	64502	290	54012
11	1553	13	1540

# Openbook Test Results

Original Version: 50% winrate against ReferencePlus.

Openbook VS Original: 61% winrate.

However..

Openbook VS ReferencePlus: only 45% winrate!

# Openbook Test Results

Original Version: 50% winrate against ReferencePlus.

Openbook VS Original: 61% winrate.

However..

Openbook VS ReferencePlus: only 45% winrate!

# Openbook Test Analysis

What might have happened?

- \* The opening patterns of ReferencePlus are not captured in the data (at the time we capture data ReferencePlus is still not available).
- \* A deeper search doesn't guaranteed a better move, but just gives a good move with higher probability. An unlucky bad move in the hotspot of openbook might actually degrade performance.

Experiments shows **both** explanations are correct.

# Boosting Openbook

Addressing the problem caused by missing opening patterns:

Add the games played against ReferencePlus into Openbook!

- \* Before Boosting: 45% winrate.
- \* After Round 1 Boosting (1500 games): 50% winrate.
- \* After Round 2 Boosting (1500 games): 56% winrate.
- \* After Round 3 Boosting (1500 games): 61% winrate.

Steady increase in winrate!

# Boosting Openbook

Addressing the problem caused by popular bad move:

The bot has a largely different winrate between moving first (30%) and moving second (60%).

- \* Might the opening move “h4g5” actually be a bad move?
- \* Rotate the King in the first move!
- \* Now the game is very similar to as if we were moving second.

Amazing winrate increase: from 45% to 60%!

Together with boosted openbook: 69% winrate against ReferencePlus!



# Openbook Summary

In the end, our openbook:

- \* Contains about 200000 game states arose from 140000 games.
- \* Almost always hits 6 rounds.
- \* With good probability hits 7 or 8 rounds.
- \* Can sometime even reach 10 rounds or more.

# Openbook Final Results against Refplus

remember to add in final result against refplus..

# Outline

- 1 Bottleneck Improvement
- 2 Openbook
- 3 Constant Optimization**
- 4 Strategies without performance gains

# Constant Optimization

- \* Use `uint64_t` to store cells that are lasered
- \* Use two bitmaps to store occupied positions, one for each color
- \* Change `ARR_SIZE` to 10
- \* Precompute and use constant tables to save repeated computation in `pcentral` and remove divisions
- \* Pack `victims_t` in `int16_t` since storing all victims is unnecessary
- \* Change the set in transposition table to be 4-way set-associative

# Outline

- 1 Bottleneck Improvement
- 2 Openbook
- 3 Constant Optimization
- 4 Strategies without performance gains**

# Strategies without performance gains

- \* Closebook: rarely used
- \* Range tree instead of sorting in `scout_search`