

Genetic Algorithms for Evolving Computer Shogi

Ryan Cain

CPSC 490 - Fall 2020

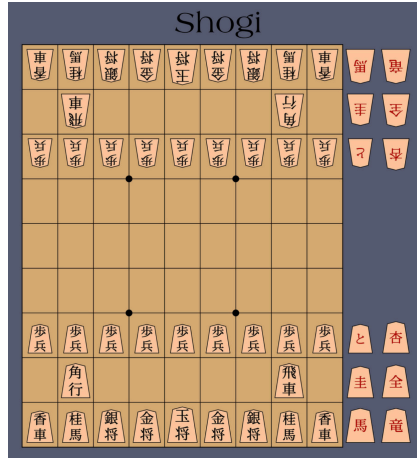


Figure 1: Traditional Shogi Board

1 Abstract

Shogi, also known as Japanese chess, is a two-player board game that sees origins of play as early as 1210. The game is played on a 9x9 grid (as opposed to an 8x8 grid for standard chess) with 8 unique pieces (*King, Rook, Bishop, Gold General, Silver General, Knight, Lance, and Pawn*). The aspects of the game that separate it from chess are the ability to promote multiple piece types (indicated by a red character on the reverse side of the piece) and the ability to drop captured pieces back into play. These rules result in a significantly larger branching factor than chess, leading to a total of 10^{226} possible play-outs of a shogi game, compared to just 10^{123} for chess[1]. This project approaches the problem by trying to use genetic algorithms (GAs) to evolve a set of weights used in a heuristic function to evaluate board positions. The methodology follows a paper that used similar approaches to achieve a grandmaster level of play for a chess program[7]. Based on the best evolved weights for its feature set, the final shogi heuristic function (H_f) found some success, correctly matching grandmaster moves 18% of the time in a 1-ply search. The target achieved in the model paper was 30%; however, given that the state-space complexity of shogi is orders of magnitude larger than chess, 18% accuracy is still a success. This is also the case when considering that the top performing shogi programs use heuristics comprised of hundreds of thousands of features[5], compared to the just 74 used in H_f .

Contents



1	Abstract	1
2	Background	3
2.1	Notation and Piece Movement	3
2.2	Computer Shogi	4
3	Introduction	4
3.1	Project Goals	5
4	Methodology	5
4.1	Feature Set	5
4.2	Heuristic Function	5
4.3	Evaluation	6
4.4	Genetic Algorithm	6
4.4.1	Weight Representation	6
4.4.2	Parameters	7
4.4.3	Optimizations	7
5	Experiments	8
6	Results and Discussion	8
6.1	Evolution	9
6.2	Piece Weights	9
6.3	Promotions	10
6.4	Drops	10
6.4.1	Missed Drops	11
6.4.2	Incorrect Drops	12
7	Conclusion	13
8	Future Research	14
A	Heuristic Features	15
A.1	Material	15
A.1.1	Balance**	15
A.1.2	Weight in Gold	15
A.1.3	Promotion Bonuses**	16
A.1.4	In-Hand Bonuses**	16
A.1.5	In-Hand Discount	16
A.1.6	In-Hand Count	16
A.2	King Control	17
A.2.1	King Safety	17

A.2.2 King Attacks	17
A.3 Controlled Squares*	18
A.4 Castle*	18
A.5 Shape	18
A.5.1 Bad Shape Penalties	18
A.5.2 Good Shape Bonuses	19
A.6 Mobility	20
A.7 Rook Position	20
A.8 Aggression Balance*	21
A.9 Blocked Flow Bonus*	21
A.10 Total Attacking*	22

B Evolved Parameters 23

2 Background

2.1 Notation and Piece Movement

Diagrams in this paper will use a single-letter western notation for individual pieces, with the only point of note being that *K* stands for King and *N* for Knight. Ownership is indicated by letter case, and promoted pieces will be shown in red. Unless otherwise specified, the board is always shown from black’s perspective – black’s pieces are capitalized and start moving up from the bottom half of the board, while white’s pieces are lowercase and start at the top half of the board. Notably, this is opposite the standard orientation of a chess board because in shogi, black (*Sen-te*, meaning "First Hand" in Japanese), always moves first. When appropriate, pieces off the board located below  and above  represent the pieces held in hand for black and white respectively. The starting arrangement of the board is shown in Figure 2 and a review of piece movements can be seen in Figure 3.



	9	8	7	6	5	4	3	2	1	
	l	n	s	g	k	g	s	n	l	a 
		r						b		b
	p	p	p	p	p	p	p	p	p	c
				•			•			d
										e
										f
	P	P	P	P	P	P	P	P	P	g
		B						R		h
	L	N	S	G	K	G	S	N	L	i

Figure 2: Starting position of shogi board with this papers notation

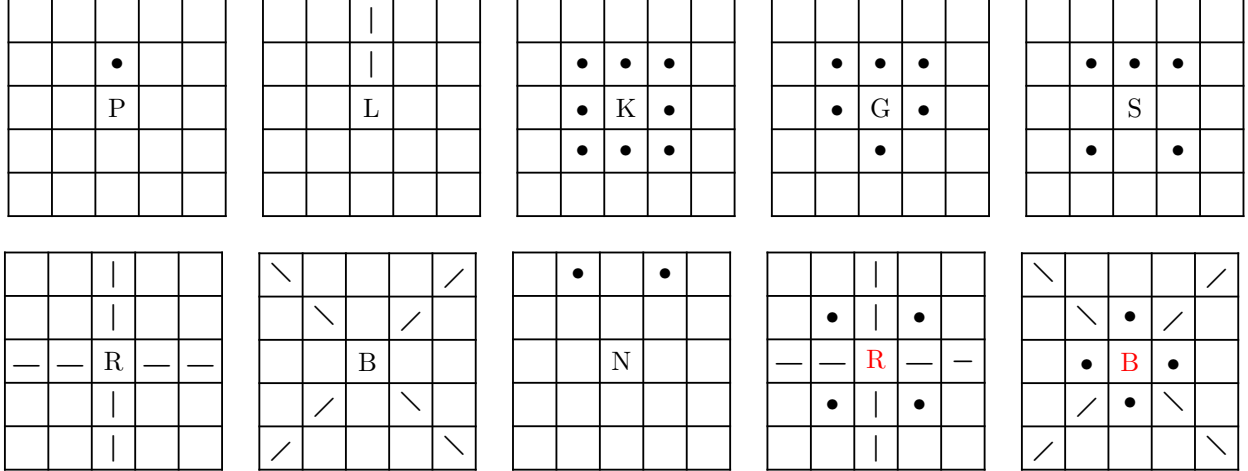


Figure 3: Shogi piece movements. The promoted pawn, lance, silver, and knight all move the same as gold.

2.2 Computer Shogi

Computer agents for shogi have been in development since the 1980s, with the first tournaments for human and computer agents being organized after the formation of the Computer Shogi Association of Japan (CSA) in 1991. Most of the successful computer agents since then have used a combination of heuristic evaluation and alpha-beta tree search that take advantage of considerable domain knowledge to further prune unnecessary branches. Popular chess agents such as Stockfish have even been abstracted to support shogi play at an extremely high level. Despite the strengths of these programs however, Google’s Deepmind project was recently able to beat even the top shogi computer agents developed over the past decades with just a few hours of training.

3 Introduction

The motivation for this project was to create the necessary tools for a fully functioning shogi agent. As discussed briefly above, there are many existing shogi programs which could have been used as a model for this project. However, most of these programs use a huge number of features and fairly complex methods for training their weights. Few of the models presented an easy entry point into developing a well-playing agent from scratch, so the focus of this project instead revolved around adapting training methods used in some simpler chess programs. As much as possible, this project followed the process discussed in the paper “Genetic Algorithms for Evolving Computer Chess Programs”[7].

The model paper utilized a three-step process for creating a competitive chess program capable of performing at or above grandmaster level. The first step of their process involved evolving a set of parameters for a heuristic function based on samples of grandmaster games. Next, the authors repeated the process to generate multiple candidate parameters that then compete in a co-evolution phase to produce a single best set of weights for the heuristic function. Lastly, they evolve a set of parameters used in common selective search algorithms. Given the original paper’s methodology, the following goals were determined at the start of the project.

3.1 Project Goals

1. Finalize a reasonable set of features to use in a shogi heuristic function.
2. Implement (refine an existing version of) a GA similar to the model paper that can efficiently evolve at least 10 sets of parameters to use in a heuristic function.
3. Implement a co-evolution algorithm to further evolve the top 10 candidate parameter sets via self-play.
4. Develop a selective search algorithm to use with the final heuristic function.
5. Evolve a set of parameters to help increase branch pruning for the algorithm in 4.

Goals 1 and 2 were met with reasonable amounts of success, and while the implementations of algorithms 3 and 4 was completed, the individual performances for the heuristic functions using evolved weights seemed too low to warrant their use. Unfortunately there was not time to meet goal 5. Thus, the focus of the following sections will be on items 1 and 2 and follow the methodology, optimizations, and choices made in order to reasonably meet these goals.

4 Methodology

This section examines the algorithms and optimizations used in the components of the overall genetic algorithm that evolves weights for different heuristic functions.

4.1 Feature Set

Features are descriptors of some aspect of a game state, such as the count of a given piece type, an indicator as to whether or not a given pawn structure is present, or the number of squares a piece is free to move without being attacked. Developing the feature set to use with the final heuristic function H_f took a considerable amount of trial and error, much more so than originally anticipated. Several initial features were later replaced or removed entirely based on the results of testing. Perhaps the most important fact of all the features though (and in line with the model chess paper) is that they are *static*. They only consider the board they are given and *not* any other information such as future moves or even more simply the type of the previous move (normal, upgrade, or drop). For a full list of the features used in testing and ultimately made their way into the final function H_f , see appendix [A](#).

4.2 Heuristic Function

Let us formally define the heuristic evaluation function used throughout the rest of the algorithms as $h(p, \mathbf{w})$, where p is a game position and \mathbf{w} is the feature weight vector containing the parameters to be adjusted. As in most chess programs, the function is a simple linear combination of weighted features. Using w_i as the i -th component of \mathbf{w} and $f_i(p)$ as the i -th feature value of position p , this can be represented as the following:

$$h(p, \mathbf{w}) = \sum_i w_i \cdot f_i(p) \tag{1}$$

4.3 Evaluation

The evaluation of an organism, or weight vector \mathbf{w} , is based on how well it works in the heuristic function $h(p, \mathbf{w})$ to select desired moves from a database of grandmaster games. A set of 5,000 game positions was used in training and the strength, or 'fitness' of a given set of weights was determined as the square of the number of grandmaster moves it could correctly predict in a 1-ply search. Given we are trying to find the set of weights \mathbf{W}^* that optimizes the number of correct guesses, the entire GA algorithm can be formally described as the following maximization:

$$\mathbf{W}^* = \max_{\mathbf{w}} \sum_{p \in \mathcal{P}} \left(\sum_{m \in \mathcal{M}_p} S[h(p_m, \mathbf{w}) - h(p_{m^*}, \mathbf{w})] \right)^2 \quad (2)$$

Here, p is a board position and \mathcal{P} is the set of 5,000 train positions labeled with grandmaster moves. p_m is the position resulting from making move m and m^* is considered the optimal move, in this case the labeled move that the grandmaster made from p . \mathcal{M}_p is the set of legal moves from position p and $S(x)$ is a simple step function that returns 1 if x is 0 and 0 otherwise. In practice though, especially with a randomly initialized set of weights \mathbf{w} , multiple positions can have the same heuristic score, so a slightly stricter version of S is used where something is only counted as correct if the move the heuristic chooses is *exactly* the same as the grandmaster's instead of just resulting in a board position with the same value.

4.4 Genetic Algorithm

Though equation (2) summarizes the mathematical model used to obtain a final set of heuristic weights, it is difficult if not impossible to directly compute \mathbf{W}^* . Instead, this project uses a genetic algorithm to approximate \mathbf{W}^* in a similar manner to the model chess paper.

4.4.1 Weight Representation

The weight vector \mathbf{w} is represented as a Gray encoded binary string (G), also called a chromosome, that is initialized randomly at the start of the algorithm. Gray encoding is a simple method for encoding binary numbers such that two successive decimal values only differ by a single bit. For example, binary representations of 1 and 2 are 001 and 010 respectively, which differ in the second and third bit position. A Gray encoding of 1 and 2 however yields the values 001 and 011, meaning the two bit strings only differ in the second bit position. Though not the only option for encoding \mathbf{w} , this property of Gray bit strings is useful in the context of GAs since it keeps small changes in a bit string closer in actual decimal value.

An individual weight w_i , considered a 'gene', is then assigned a fixed number of bits in G which are decomposed into base 10 to use in computing the value of $h(p, \mathbf{w})$. Bit widths are assigned according to the relative importance of the feature they are tied to. The final function H_f uses a bit width of 10 for major features (24 total) such as material or mobility, but a smaller bit width of 6 is used for weights tied to minor features (50 total), like pawn structure, etc. This leads to an overall chromosome length of 540.

It is worth noting that these decisions stray from those outlined in the model chess paper. The authors in the chess paper used a chromosome of 244 bits and had 4 major features (the piece value of knight, bishop, rook, and queen) each allocated with 16 bits, with other minor features (31 total) had 6 bits as well.

4.4.2 Parameters

The same set of parameters was used to drive the overall GA as in the model chess paper and is shown in table 1. However, in some of the experiments discussed later these parameters were changed slightly, but to no great affect on the overall results of evolution. Elitism was also used to ensure the best member of the population survived each generation.

Population Size:	100
Crossover Rate:	0.75
Mutation Rate:	0.005
Generations:	200

Table 1: GA parameters

4.4.3 Optimizations

A good deal of time was spent in the first half of this project attempting to efficiently calculate feature values given a static board representation. Even though the GA only evaluates an organism based on a number of a 1-ply searches, the large branching factor in shogi means that evaluation for each potential weight vector \mathbf{w} is quite costly. Given the specific set of 5,000 training positions used for this project, each organism had to be evaluated on 382,816 positions just to attain a single fitness value. For 100 organisms in 200 generations this expands to more than 7.5 billion positional evaluations, which again needs to be performed 10 times to obtain enough organisms for the co-evolution stage. Given the optimizations discussed below, the final time it took to produce a fully trained organism ready for co-evolution was reduced to just about 1.5 hours, slightly less than the model paper (A direct comparison is not quite fair however, since the paper is from 2014 and used slower hardware. Still though, the final function H_f uses over twice the number of features as the original paper on a game with a much larger branching factor, so these optimizations are non-trivial).

Legal Moves Cache The legal moves for each of the train positions is computed ahead of time by a python script and saved to a json file. This is then loaded into memory at run time and saves time of computing legal moves during the evaluation of each organism.

Feature Vector Cache This was the key optimization that allowed the algorithm to run in a reasonable amount of time. In hindsight it seems trivial, but it required a decent amount of work. Though each organism must be evaluated with a different set of weights, the actual computation of individual feature values of any given board position does not change from organism to organism. Thus, many expensive calculations can be saved during the evaluation of the first organism and re-used by later ones. The algorithm maintains a map from a string representation of a board position to a feature vector f_v that contains all the information necessary to compute the value for $h(p, \mathbf{w})$ given a \mathbf{w} . It is easy to do this in the case where there is bijection between the number of features and weights. However, it is a bit tricky to do efficiently for some of the more complicated features such as bonuses that needed to be added on top of another weight or for compound calculations like king attack scores (see A.2.2).

Multiprocessing Once one organism had been run through the full evaluation of train positions, C++ multiprocessing (MP) could be switched on in order to parallelize evaluations of the train positions. Python multiprocessing was also tried at the organism level, but the best results were seen by using multiple threads to speed up the process of all the individual 1-ply search operations that needed to be done.

Run times Run times at different points of the project were saved to give an idea of whether optimizations were worthwhile. Before implementing the feature vector cache, utilizing MP alone saw single organism evaluation times (for 5,000 train positions) of around 70 seconds. This meant it would take around two weeks to complete a full evolution with 200 generations of 100 organism – feasible yes, but highly impractical for testing. The final result of the above optimizations was that after the 7-8 minutes it took to cache the feature vectors during the evaluation of the first organism (this was done with MP turned off to help avoid tricky race conditions), subsequent evaluation times were reduced to mere milliseconds. So on average, a single generation of 100 organisms now only takes 20-40 seconds depending on the number of features being used, and a full run through of the algorithm takes just 1.5 hours.

5 Experiments

In total, about 70 experiments¹ were done which ran the full algorithm from start to finish. In most cases it was the combination or set of features that was changed from test to test; however, a lot of variation in the early results came from changing parameters like bit width and which features were allocated the larger or smaller number of bits. For example, it was not clear at first that piece bonuses (promotion and in-hand bonuses) should be allocated as many bits as those measuring raw piece value (See [A.1](#) for more detailed discussion). The assumption was that these bonuses should be relatively small compared to the underlying piece value, but in practice, tests that restricted the bit widths of these bonuses performed significantly worse. Tests were also done where all features had the same number of bits allocated but without much success. Other tests were carried out in a similar fashion to the chess paper, with a comparable number of features (33 vs. 35 used in the paper, i.e. before implementing more features like distance to king, in-hand bonuses, etc), 16 bits for material values, and 6 for all others.

6 Results and Discussion

The best heuristic function H_f was able to match grandmaster moves 17.72% of the time (886 correct out of 5,000). Excluding drop moves, which proved especially difficult for H_f to predict, accuracy was further increased to 22%. Though this falls short of the model paper’s target of 30%, many aspects of the final evolved parameters are still worthy of note. For the full set of finalized features used in H_f , see [appendix A](#), with their tuned weights shown in [appendix B](#).

¹Explaining the details of each is not practical, but all results are stored in the `/py/res/` project folder. Each file has a preamble detailing all of the parameters, major/minor feature distinctions, the results of the evolution, and the final population of trained weights.

6.1 Evolution

The figure below shows the results of a 200 generation evolution with 100 individuals for the best set of features as determined by the experiments. A few similar runs were done in order to make sure this result was consistent, and in some cases the algorithm was run with a slightly larger population, greater number of generations, or different crossover rate. The outcome of these runs was mostly the same, with the only difference being that more generations tended to give the population average more time to approach the level of the top performer, a result which reinforces the idea that changes to the feature set and bit widths influence the results the most, not GA parameters. In some cases however, longer evolutions appeared to produce more stable piece values, or at least the piece values relative to each other seemed more reasonable than in runs of just 200 generations.

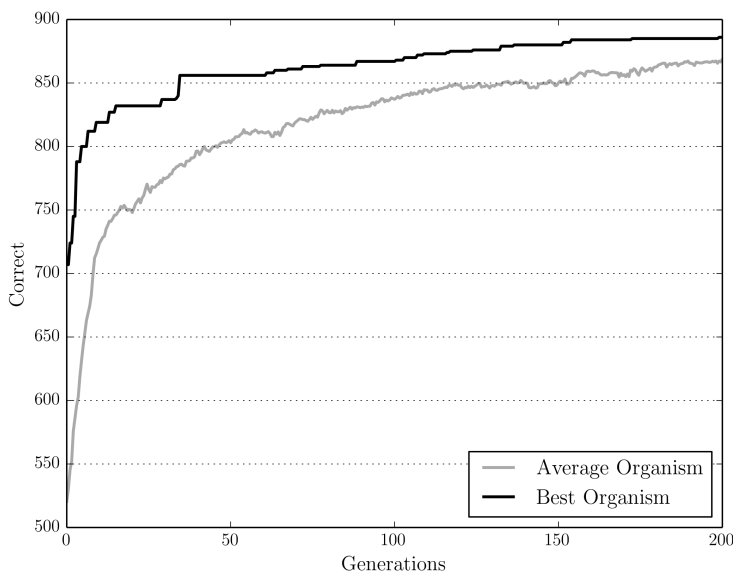


Figure 4: Number of positions solved (out of 5,000) for the best organism and the population average in each generation (total time for 200 generations \approx 1.5 hours) The Figure shows results of the best run out of over 50 tests.

6.2 Piece Weights

The final set of piece values for H_f was determined as below. They have been converted from their raw value to their pawn weight in order to make comparisons easier. As noted in the discussion of material features in A.1, not all grandmasters or even top shogi programs agree on the valuation of individual pieces. Thus, it is expected that some of the values that H_f may not agree with conventional wisdom. In fact, the values here may even reflect the underlying mindset of the particular grandmasters used in the training sample. For example, given the values in Table 2, as well as in nearly all of the other experiments performed, H_f values the lance more so than the knight. Other shogi programs tend to favor the knight, however their values are often incredibly close if not equal [2]. Thus, this valuation attained in H_f could be indicative of

	P	N	L	B	S	G	R
Normal	1.00	2.09	3.40	5.30	6.75	7.87	8.86
Promoted	10.7	5.22	6.37	7.16	10.41	–	9.28
In-Hand	7.49	10.33	9.58	15.10	13.24	16.42	12.68

Table 2: Evolved piece weights in terms of number of pawns.

the underlying positions used in our training sample or the given grandmaster’s style. Amongst these piece weights, it is also promising to see the high valuation of the promoted pawn when compared to the promoted knight and lance. Similar valuation is often the case amongst grandmasters, and a more detailed description for the reasons to do so are described in appendix A.1.3. Also of note, in all cases except for the pawn, pieces in hand are valued the greatest. Again, this is good news seeing as other shogi programs tend to do the same given the role in-hand pieces play in mating the opponent [2].

The last standout feature of these values is a somewhat disappointing one – the bishop seems to be valued much lower than the silver and gold. This valuation is puzzling as most certainly the bishop should have a higher value than that of at least a silver. In fact, many other runs of the algorithm valued the bishop as the highest piece, or at least more comparable to the rook than shown here. Some explanation for this can be given to the fact that the organisms are initialized randomly, so even ones that perform similarly well towards the end of the algorithm may develop slightly different play styles and consequently value pieces differently. So this run of the algorithm could have learned to use more golds than bishops and continued to increase its fitness overtime by favoring that style. Another probable reason for such low bishop valuation is that the weights are being over-fitted for specific positions in the training set; however, performance of H_f is similar when run on the test data, so this is likely not the core of the issue.

Overall though, the valuations for the the different piece variations tends to be in line with high level play, which is a strong indicator that the algorithm is on the right track.

6.3 Promotions

Results for promotion accuracy were quite promising. H_f had a 53% accuracy when guessing grandmaster upgrade moves (186 of 352). On the surface this is a very solid result, but the reality is a bit more complicated. The reason the accuracy is so high is just a by-product of H_f ’s learned play style. Specifically, H_f develops a strong bias for upgrading, choosing to do so almost 1/5th of the time (in 964 of the 5,000 train positions). Most likely, this is due to the fact that the features covering material balance are allocated a larger number of bits than other features. However, since unique movements for pieces like the knight and lance are lost on an upgrade, most grandmasters hold off on upgrading these pieces unless absolutely necessary. The material feature set for H_f clearly prioritizes upgraded pieces more than regular ones though, the result of which is that that in any situation where an upgrade play is an option, H_f almost surely takes it.

6.4 Drops

Despite best efforts, drops continued to confound the many different heuristic functions. Sadly, this is a critical point of shogi and nearly 20% of the data set contains drop moves (922 of 5,000). Even the best organism had just 0.88% accuracy, correctly guessing only 8 of the 922 moves in which the grandmaster

played a drop. This number is shockingly low, but in some sense not entirely unexpected. For example, assuming that the feature set only contains a few features that partially help predict drop moves, one would expect that individuals over the course of the evolution would simply learn to ignore these moves, and any trait that might help with accurately predicting drop moves is then sidelined in favor of those that strongly predict the other types of moves. Thus, on the whole, evaluation becomes incredibly "drop-averse" unless there truly is no other option. This drop aversion is clear in the raw number of times H_f even attempts drop moves – it only chooses a drop move 96 times in total, which seems likely after the above discussion of how heavily it favors promotion moves. There may be a slight shimmer of hope for the feature set though, as this means that when the heuristic *does* choose a drop move, it does so with about 8% accuracy (8 of 96). Additionally, there is a bit more evidence to suggest that some aspect of the finalized feature set helps evaluate the strength of drop moves. For example, running the same GA on data for only the 922 drop moves from the training set produces weights that guesses the correct drop move 9% of the time. This performance is still sub-par and much worse than expected, but it is at least far better than random. Still, the logical conclusion is that a significant amount of domain knowledge about drops is either missing from the data set or cannot simply be described by the linear combination of such a limited number of static features.

6.4.1 Missed Drops

To help understand why predicting drops is so difficult, it is useful to examine some of the cases where the heuristic missed a drop move. Two such examples can be seen in Figure 5 (Note: For move notation, \times denotes capture, $*$ a drop, and $+$ an upgrade) with the grandmaster drop move being highlighted in gray. When applicable, the features being discussed also have links to their full definition in appendix A.

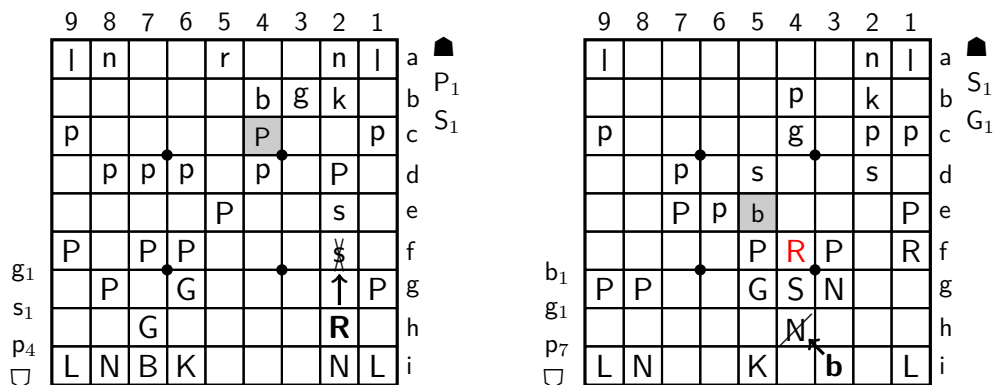


Figure 5: Heuristic plays **Rxf2** [$H_f = 618$] instead of **P*c4** [$H_f = -173$] for black (Left) and **bxh4+** [$H_f = 7088$] instead of **b*5e** [$H_f = 5081$] for white (Right)

For both cases in Figure 5, the H_f more highly values capture moves over drop moves. At first glance, the move the heuristic chooses for black in the left diagram seems to be a strong one. It captures white's silver on **f2** (A.1) and advances its rook closer towards the enemy camp while staying on the king's file (A.7). Of course, white can freely take black's rook via a silver on **e2** the following turn, and overall an exchange of silver for rook is a pretty bad one. However, positions are chosen based on a 1-ply search, so the heuristic has no access to the potential follow up moves, it merely looks at the static board state resulting from any

given move. When looking at the rest of the board too, there really are not too many other moves that black could make other than drops. A pawn here or there could be advanced, but those board positions will likely result in relatively low/similar heuristic scores. The positions resulting from drop moves are certainly weighted highly amongst such options simply because of the increase in material on the board, but clearly they are outweighed by the prospect of gaining a silver in hand. Even though we give a bonus for attacking the head of an enemy bishop (A.5.2) as is the result of the actual grandmaster move **P*c4**, this too seems to be overpowered by the weight of material gain. In the end, it seems that many of the features are working against a higher valuation of drop moves, and the result is that the heuristic chooses to blindly capture the silver instead of more highly valuing potential drops (It is also all the more obvious that the heuristic would do this when considering how highly it values a silver in hand. See 2). Still, it is sad to see the heuristic missing out on a very strong move, since placing a pawn on **c4** very effectively forces the enemy bishop into retreat.

The position on the right of Figure 5 further demonstrates the material bias of the heuristic function H_f . The grandmaster drops a bishop to **5e** for white, a strong move that leaves the piece with much mobility (A.6) and potential for future attacks (A.10). It is promising to see that H_f values this resulting position very highly, but instead it chooses an obviously poor move, bishop capture and upgrade to **4h**. Again though, it is not difficult to see why. Though white will clearly lose the promoted bishop on black's following turn, H_f only knows the current board and sees the superficial strength of such positioning based on its feature set. Mainly, it values the promotion of the bishop (A.1) as well as the fact that it is positioned to attack the king square, two squares adjacent the king, and three other pieces, the gold, silver, and knight, positioned in row **g** (A.2.2). In hindsight, it might have been worthwhile to experiment with greater penalties specifically for positions in which the major pieces, rook and bishop, are being attacked in order to prevent moves like in Figure 5. However, at the time features like A.10, A.6, A.7, and A.5.2 were thought to cover these cases.

6.4.2 Incorrect Drops

Lastly, consider the following positions in Figure 6, where the heuristic did in fact choose to play a drop, it just did so incorrectly. Admittedly though, this example is hand picked to showcase when H_f was somewhat close to guessing the correct drop. A few such cases exist, but as discussed in the previous two examples, the overwhelming majority of the time H_f chooses to ignore drop moves all together and play futile capture or upgrade moves instead. Still, it is useful to consider the cases where H_f was somewhat on the right track.

The board positions on each side of Figure 6 are the same, with the move made by H_f shown on the left in gray and that of the grandmaster on the right board. To give the position in the figure context, consider the opponent (white)'s previous move, which was to advance its rook to **h2**, placing black in check. These kinds of long range checks are very common in both chess and shogi, but given the drop rule, these forms of attack are often stopped easily in shogi by dropping a piece in between the attacker and the king. In fact, they are so common that a feature counting blocked flow (See A.9) was created in order to encourage this kind of behavior. The fact that H_f plays a drop here at all should in some sense then, be seen as a success. However, it is very clear why the move is sub optimal. H_f playing **G*h3** for black temporarily blocks the attack of white's rook, but **h3** itself is unprotected, so white can easily just capture the gold and place black back in check. However, consider a view of the board through the lens of H_f . This move is valued higher than other moves because it has "checked the boxes" of a lot of valuable features – it reduces the number of

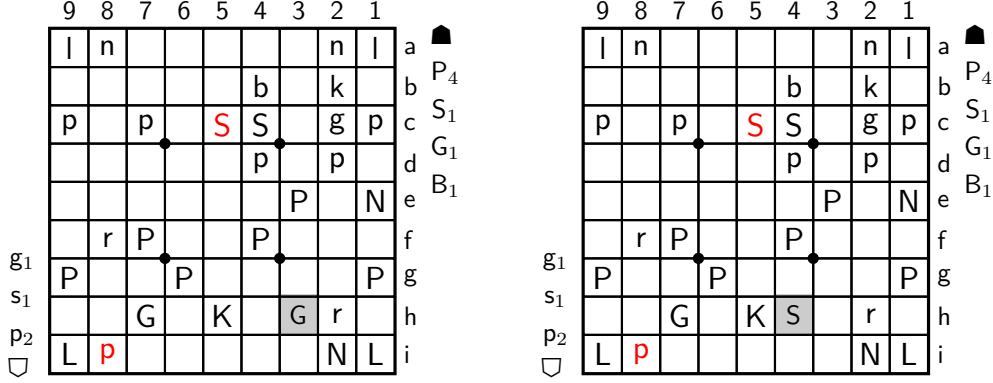


Figure 6: Same board position, with heuristic choosing to play **G*h3** [$H_f = 3256$] (Left) instead of **S*h4** [$H_f = 2121$] (Right) for black.

threats to its king (A.2.1), blocks the flow and reduces the mobility of the opponent rook (A.9, A.7), adds a gold to the board (A.1), as well as protects two squares directly adjacent its king (A.2.1).

The grandmaster’s move, **S*h4**, is much a better counter to the check, and interestingly, it also checks many of the same boxes for H_f . However, to H_f , the resulting position not only leaves the opponent rook with greater mobility (A.7), it also leaves more squares in its own camp (bottom three rows) unprotected (A.3), gives fewer escape routes to its king (A.2.1), and places a piece more often used for attacking too close to its king (A.2.1).² In the end, it seems that the weight of these features is masked by the others mentioned above, and though the heuristic was close to guessing the best move, it is clear that balance is off between features aimed at encouraging drop moves and those designed to cover other aspects of the game.

7 Conclusion

Though the final heuristic function H_f had only an 18% accuracy in guessing grandmaster moves in a 1-ply search, these results should still be taken as a success. The target accuracy achieved in the original chess paper was 32%; however, chess is a much simpler game. Shogi’s explosively large branching factor means that there are many more facets of the game that need to be accounted for in a heuristic’s feature set. Furthermore, as discussed in this paper, certain aspects of shogi, particularly drop moves, make it difficult to design a truly static set of features to use in evaluation. Not only that, but many high performing shogi programs actually rely on a set of hundreds of thousands, if not millions of individual features to evaluate board positions[5]. Thus, the fact that over half of the original paper’s accuracy was matched with a comparable number of features in game with a far larger state-space complexity should certainly be taken as evidence for the strength of GA-based models in developing heuristic evaluation functions. Additionally, these results show that established GA training methods perform surprisingly well, even when the search space of the original problem is expanded dramatically.

²The final evolved weight for the “distance to king friendly silver” feature is 42 as shown in 4. This value is relatively high, given that the max a weight can be with 6 bits is just 63. A high value here means that the piece type is valued as an attacking piece. For example, a silver just one square away from its king will add 42 to H_f , but one two squares away will add 84 to H_f , 3 squares away 126, and so on. Thus, H_f sees positions having the silver piece further away from the king (i.e. attacking the other king or holding a central position) as more valuable

Lastly, these results can also be taken as a success in the context of the original project goals (3.1). A large number of reasonable/candidate features were developed to help in the static evaluation of board positions, an efficient GA methodology for evolving the weights of any given feature set was implemented in Python and C++, and the framework for parameter co-evolution and selective search using a Negamax were also created. Though the latter of these algorithms were not run and there was not enough time to write the third set GAs designed to increase tree pruning, a good deal of effort was spent investigating and achieving the first set of goals. Overall, the results were promising and leave ample room for further investigation, a few ideas of which are discussed below.

8 Future Research

Given the results of the current feature set, there is definitely room for further research. Most of the ideas discussed below are based off of observations made during the different stages of research, testing, and experimentation.

King Control There was not as much time to test more complicated models for king safety/threats such as the king attack function described in A.2.2. For example, modifying the function to only consider safe attacks could likely help prevent many of the poor moves made in Figures 5 and 6. Additionally, increasing the size of the "king zone" to be two squares adjacent to the king in any direction might have positive results as it seems many other top programs do this.

Better Feature Set The fact that 18% accuracy could be achieved with the current feature set is a strong indicator that there is hope for these methods in shogi. It is very likely that a better designed feature set could achieve results closer to the chess paper. Spending more time systematically testing the affects of each feature (as well as its interaction with others) could help determine which features should be kept and which should be removed or added.

Relative Piece Value Tables Mentioning the "hundreds of thousands" of features in other shogi programs is a reference to their piece value tables. Such tables estimate the value of a piece based on it position relative to either king. For example, a silver would have a unique value for all of it's possible locations relative to both its own king and the enemy's. With 81 possible locations of a king on the board, 80 locations a piece can be relative to that king, 13 unique piece types, and two kings in play, this leads to $81 \cdot 80 \cdot 13 \cdot 2 = 168,480$ individual piece values that have to be evolved. The number grows to the millions when considering the values of pieces relative to pieces other than the king as well, such as bishop or rook. This large number of features is likely what helps agents perform much better in deciding proper drop and upgrade moves. A paper shows that this number of weights can be trained effectively[6]; however, it would be interesting to see if a fewer number of weights could achieve similar results. For example, taking the weights produced in that paper and modeling relative piece value as some non-linear function of its distance to other pieces.

Non-static Features All of the features described look only at the static board representation, i.e. it has no idea which piece was actually moved to get to that position. It might be interesting to try the same

techniques but give the heuristic access to information about the move it is making. This would allow normal moves, upgrade moves, and drop moves to be evaluated differently, even with the same set of weights.

Multiple Training Sets Perhaps better results could be achieved by evolving three separate evaluation functions – one on a data set of drop moves, one on normal moves, and one on upgrade moves. This could give a similar effect of using a non-static feature set as described above, but some work would have to be done in figuring out a method for determining which move to make amongst the best chosen drop, upgrade, and normal move.

A Heuristic Features

This is an appendix describing all of the features considered and or used throughout the course of developing different shogi heuristic functions. These features are always evaluated from the perspective of the *root* node, which is used interchangeably with "player" or "player one" in the following definitions. Features marked with an asterisk (*) were used in the final evaluation function (H_f) that saw the best results. Unless otherwise specified, a single weight for each of these features was evolved in the GA, and those with a double asterisk (**) were considered as major in H_f , meaning they were assigned 10-12 bits instead of just 6.

A.1 Material

In any heuristic function, albeit chess or shogi, the raw piece values are arguably the most important feature. Shogi presents an added twist to this concept in that pieces can both be upgraded and replayed later in the game. Additionally, even amongst grandmasters there is dispute over the individual piece values to an extent not seen in chess[4]. The various approaches to handling these challenges are discussed below.

A.1.1 Balance**

Traditionally, each player's score is calculated as the sum over all counts of the pieces *on the board* times their weighted value – for H_f this is the value as determined by the genetic algorithm. Material balance is then defined as the difference between these scores for player one and player two.

A.1.2 Weight in Gold

This feature was a first attempt at handling promotions. Since the promoted pawn, lance, knight, and silver all gain the same range of movement as a gold general, they are each assigned the gold's value instead of having their own. The domain assumption here was that there should be no difference in the interpretation of their value if they all move the same on the board. This idea was extended to the promoted values of the rook and bishop as well. Since promoting these two pieces effectively grants them the same range of motion (such that they gain movement similar to a king – additional 4 corners for rook and 4 points of cross for bishop, see 3), it seemed reasonable to evolve a single bonus value for their promotion instead of separate ones

A.1.3 Promotion Bonuses**

Despite the benefits of the above approach, mainly being that there are fewer weights to evolve, better results were seen when assigning a unique promotion bonus to each of the pieces. The reason for this has to do with mid-late game interactions between promoted pieces and shogi’s drop rule. In practice, promotion bonuses should be somewhat inversely proportional to the underlying piece value. This is because if that piece is captured, the opponent only gains the ability to initially replay that piece in its un-promoted form. For example, although a promoted a pawn and knight may both move identically to a gold general, players often play more aggressively with the promoted pawn. It is far more advantageous to sacrifice the promoted pawn than the knight because the opponent only receives an un-promoted pawn as opposed to a knight to use at a later point in the game.

With this consideration the final H_f has unique promotion bonuses for all 6 of the pieces eligible for promotions. These are not unique weights for the value of each of the promoted pieces (although that was also tested), but instead are linked to the underlying value piece. So the total contribution to the final value of H_f is:

$$\sum_{p \in P} n_p \cdot (w_p + b_p) \quad (3)$$

where n_p is the number of piece(s) p on the board and w_p, b_p are the weight and promotion bonus for piece p respectively. As described in the balance section above, H_f uses the difference between this score for player one and player two.

A.1.4 In-Hand Bonuses**

By the same reasoning as described above, pieces in hand should not be weighted exactly the same as pieces on the board. A small bonus is given to the value of each piece in hand (linked to the underlying piece) and is added to H_f by the same process as equation 3.

A.1.5 In-Hand Discount

Another model for the value of pieces in hand was tried that discounted the weight of each piece. The thought process was that although having pieces in hand is undoubtedly a benefit, they are not worth as much as pieces actually in play on the board. Rather, it is an advantage to have the piece for use at a later point in the game, but those pieces cannot be used all at once and take additional time to develop. The contribution to H_f was given by the following function, where w_p, n_p is the weight/count for piece p , N is the total number of pieces in hand, and the exponent i was the value evolved by the GA.

$$\left(\frac{1}{N}\right)^i \cdot \sum_{p \in P} w_p \cdot n_p \quad (4)$$

A.1.6 In-Hand Count

A naive feature that simply counted the number of pieces in had was also tried at first, but not surprisingly this did not yield very good results and was thrown out very early.

A.2 King Control

The next most important set of features behind material value are those that measure the safety of a player's own king versus their opponent's. Several methods were applied in testing.

A.2.1 King Safety

Threats Penalty** A penalty based on the number of enemy pieces attacking the player's king as well as the up to 8 squares surrounding it.

Defenders* Weight assigned to the number of friendly pieces that are defending the player's king square as well as the up to 8 surrounding squares.

Escape Routes* Number of squares adjacent to the player's king that are safe to move to, i.e. no enemy pieces currently attack them.

Distance to Player's King* This feature was initially implemented in order to try and help improve the heuristic's ability to guess drop moves by giving different bonuses for pieces placed nearby, but not necessarily protecting the king directly. Thus, one feature for each of the 13 unique pieces, each with their own weight, are assigned for the euclidean distance between a piece p and its king.

A.2.2 King Attacks

Distance To Opponent's King* 13 features are added exactly the same as above, except the distance is measured from the current player's pieces to the enemy king square. These features, when combined with those measuring distance to one's own king, proved to be strong predictors of the grandmaster moves.

King Attacks Safe Number of safe (backed up by another piece) attacks in the enemy's king zone. This feature was very strong in many tests but was weakened by addition of the feature measuring distance to opponent's king. Ultimately, models with the distance to king's feature performed better overall, despite this feature initially standing out.

King Zone Attack Function Many chess programs, including those modified to play shogi such as Stockfish, have some function to measure king threat/safety. The common idea is to weight attacks of individual piece types differently and to also give a unique weight for the total number of attacks in the king zone. A simple feature was designed to take the difference between these two values for player one and player two. So with w_n being the weight of n unique attacks in the king zone KZ , aw_p the attack weight for piece p , and n_p the number of squares that piece p attacks in the king zone, the total contribution to H_f would be the difference for player one and player two of the following:

$$\frac{w_n}{100} \cdot \sum_{p \in KZ} aw_p \cdot n_p \quad (5)$$

Two approaches with this function were tested, first where both w_n (for 1-7 total attacks in the king zone) and each aw_p were evolved by the GA. In order to reduce the number of weights to be evolved, a separate

version was also tested using the following predetermined values of w_n in table 3. Though the results were

n	1	2	3	4	5	6	7
w_n	0	50	75	88	94	97	99

Table 3: Set attack weights based on n attackers

varied in testing, this feature is very promising and some form of it should likely be used in the algorithm in place of some of the other king features; however, there was not enough time to get it properly balanced. Stockfish and other programs are known to consider squares beyond just those adjacent to the king (perhaps even more important in shogi given the board is larger) and they also change the piece/number of attack weights based on king position and whether it is early, mid, or late game.

A.3 Controlled Squares*

Enemy-Camp Attack* Number of squares in the enemy camp (three rows closest one’s opponent) which have a greater number of the player’s pieces attacking them than the enemy has defending them.

Home-Camp Vulnerability* Number of squares in a player’s home camp (three rows closest to a player) which have a greater number of pieces protected by one’s own pieces than are attacked by the enemy.

A.4 Castle*

H_f uses a dictionary of 44 castle positions to determine how close a player’s pieces are to a given castle formation. ”Closeness” is measured as the number of the player’s pieces that are in the correct position for the castle *if and only if* the player’s king is also in the correct spot. This feature returns the number of pieces correctly positioned in the closest matching castle formation from the dictionary.

A.5 Shape

Good shape and bad shape are probably some of the most difficult features to encode. Many of the decisions for these features was informed by the book *Better Moves for Better Shogi* [3]. However, many features proved less useful as predictors of grandmaster moves than one would have hoped. Features that better capture good and bad board shape would likely contribute to better prediction of more complicated drop moves.

A.5.1 Bad Shape Penalties

Gold Ahead Silver* It is generally considered a disadvantage to have a gold piece directly in front of a silver piece since the silver piece cannot move one square backwards, thus leaving more openings for an opponent as shown in Figure 7. In some situations this formation is considered an advantage, but overwhelmingly it seems to be a disadvantage, so a minor penalty is given to positions with this shape. The feature simply counts the number of gold’s (or those that move as gold) positioned ahead of silvers.

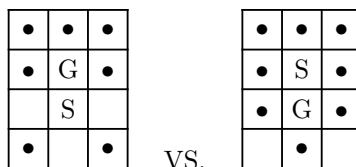


Figure 7: Gold in front of silver leaves more openings

Gold Adjacent Rook* It is also considered bad shape to have a gold directly adjacent to a rook. This is obvious because not only does the gold restrict the rook’s side to side movement, the gold piece also attacks the square ahead of the rook, which is an unnecessary defense given the rook attacks all square ahead of it anyways.

Boxed in Bishop Penalty given based on the number of corners of bishop movement which are blocked by friendly pieces. This feature was scrapped in favor of overall bishop mobility (A.6).

Piece Ahead of Pawn* Apart from some cases such as reclining silver (A.5.2), it is generally advised to avoid having pieces in front of pawns. This is because unlike traditional chess, pawns cannot capture adjacent squares, only forward, so although the pawn protects the piece at its head, consequently the pawns mobility is completely stifled. In retrospect, this feature may have prohibited guessing certain drop moves into the enemy camp where this formation becomes the only valid option, but not enough tests were done to say for certain.

A.5.2 Good Shape Bonuses

Bishop Heads* As in chess, bishops are vulnerable to head-on attacks as they cannot move forward (the promoted bishop in shogi can move forward however). This feature counts the number of normal bishops (1, 2, or 0 are the only possibilities) who have the squares directly in front of them protected by a friendly piece. Along the same lines, it also counts the number of the enemy’s bishop heads currently being attacked.

Reclining Silver* This feature is designed to give a player a bonus if determined that the player is using a formation known as ”reclining silver.” Such a formation is considered strong since the pawns compliment a forward advance of the silver. The formation gets its name because it looks like the silver piece is sitting on a chair of pawns as seen in Figure 8.

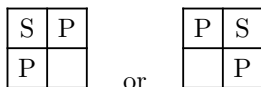


Figure 8: Left and right orientations of reclining silver

Vanguard Pawns / Claimed Files* A player is said to have ’claimed a file (column)’ (or owns a vanguard pawn/position) if they have a pawn in the fifth rank (row) that is defended from behind. This feature simply counts the number of claimed files that the given player has. As with the other features trying to capture

shape, there are situations where a file is not truly 'claimed' just by having these two conditions met. As an example in the early game, a claimed file by this definition can happen when a rook supports a pawn from behind, but if kept in this position too long will not allow the rook to develop fully.

Adjacent Silvers* Having two silvers directly next to each other is considered a strong position since they fill each other's gaps in backward movement as shown below in Figure 9. This feature is a binary indicator of whether or not the player holds such a position(s)

	S	S	
●	●	●	●

Figure 9: Adjacent Silvers Protect Gaps

Adjacent Golds* As with silvers, adjacent golds are considered strong and this feature determines if adjacent golds are present or not.

A.6 Mobility

The bishop and rook are the two strongest pieces in shogi, with players tending to side to a play style that heavily utilizes one or the other. For example, many openings that use the rook are considered "Ranging Rook" formations and those that use the bishop "Ranging Bishop". Measuring their mobility is key to evaluating good board position, so these features are allocated the same bit width as the material features.

Bishop** This feature is the difference between player one and player two's bishop mobility. Mobility itself is measured as the number of safe squares the bishop can freely move to.

Rook** Same measure of mobility as above, only for player's rooks.

A.7 Rook Position

It seemed logical to lean towards one playing style and given that there was a focus in the computer chess paper on rook movements[7], several features were developed to capture good and bad rook positioning.

Rook Enemy Camp* Simple feature to count the number of rook(s) or promoted rook(s) that are within the enemy camp (closest three rows to opponent).

Rook Attack King File* Count the number of rook(s) or promoted rook(s) that are positioned on the same file (column) as the enemy king.

Rook Attack King Adjacent File* Count the number of rook(s) or promoted rook(s) that are position on the file *next* to the enemy king.

Rook Attack King Adjacent File 9/8/2/1* Same as the feature above, but only counted if the king is 'trapped' in the either of the edges, files 9 and 8 on the left or 2 and 1 on the right.

Rook Open Semi Open File* A file is considered 'open' if it has no friendly or enemy pieces on it. A semi-open file is one that has only one enemy piece (usually a pawn) on it. Thus H_f uses two features, one that counts the number of rook(s) or promoted rook(s) on completely open files and those on semi-open files.

A.8 Aggression Balance*

This feature takes a simple overview of how far forward each player has pushed into the other's camp. Each piece for player one and player two is assigned a score equal to the number of rows it has successfully advanced. The difference between the sum of these values for the two players is the value added to H_f .

A.9 Blocked Flow Bonus*

This feature was also added in an attempt to better predict drop moves. It is common practice to block the long range mobility of the major pieces, considered "flow" moves. Often, the drop of a simple pawn will foil the ranging attack of a bishop or rook. Though the difference in mobility of these pieces is already captured by (A.7, A.6), this feature gives a bonus for plays or positions that actively block the opponent. Blocked flow is computed as the number of squares of mobility the opponent is 'missing out on' based on the position of player one's pieces. The blocked flow is only counted however, if the blocking piece itself is on a safe square. For example, imagine we are calculating how much black is blocking white's flow moves in Figure 10 below. Assuming black's gold [G, f6] and pawn [P, e5] are protected, together they block a total of 4 squares (indicated by X) of mobility for the enemy rook and promoted bishop. Note that squares h5 and i5 are not counted as blocked. Although those squares are blocked for white's rook on a5 by black's pawn on e5, white's promoted bishop on h4 can still reach them.

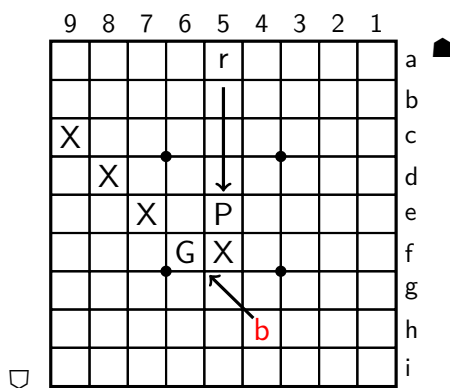


Figure 10: Example of Blocked Flow

A.10 Total Attacking*

This is simply counted as the difference in the number of pieces that player one attacks on the board compared to player two.

B Evolved Parameters

PAWN_VALUE	100	LANCE_VALUE	340
KNIGHT_VALUE	209	SILVER_VALUE	675
BISHOP_VALUE	530	ROOK_VALUE	886
GOLD_VALUE	787	PROMOTED_PAWN_BONUS	970
PROMOTED_LANCE_BONUS	297	PROMOTED_KNIGHT_BONUS	313
PROMOTED_SILVER_BONUS	366	PROMOTED_BISHOP_BONUS	188
PROMOTED_ROOK_BONUS	42	PAWN_IN_HAND_BONUS	649
LANCE_IN_HAND_BONUS	618	KNIGHT_IN_HAND_BONUS	824
SILVER_IN_HAND_BONUS	649	BISHOP_IN_HAND_BONUS	980
ROOK_IN_HAND_BONUS	382	GOLD_IN_HAND_BONUS	855
PLAYER_KING_THREAT_PENALTY	228	BISHOP_MOBILITY	75
ROOK_MOBILITY	33	ENEMY_KING_ATTACKS	10
BISHOP_HEAD_PROTECTED	39	BISHOP_HEAD_ATTACK	5
PLAYER_KING_DEFENDERS	31	PLAYER_KING_ESCAPE_ROUTES	56
IN_CAMP_VULNERABILITY_PENALTY	9	OUT_CAMP_ATTACK	22
CASTLE_FORMATION	25	GOLD_AHEAD_SILVER_PENALTY	1
GOLD_ADJACENT_ROOK_PENALTY	2	BOXED_IN_BISHOP_PENALTY	60
PIECE_AHEAD_OF_PAWN_PENALTY	59	RECLINING_SILVER	60
CLAIMED_FILES	10	ADJACENT_SILVERS	54
ADJACENT_GOLDS	58	ROOK_ENEMY_CAMP	22
ROOK_ATTACK_KING_FILE	35	ROOK_ATTACK_KING_ADJ_FILE	20
ROOK_ATTACK_KING_ADJ_FILE_9821	49	ROOK_OPEN_FILE	20
ROOK_SEMI_OPEN_FILE	7	BLOCKED_FLOW_SAFE	24
AGGRESSION_BALANCE	58	TOTAL_ATTACKING	44
DTK_FRIENDLY_PAWN	44	DTK_ENEMY_PAWN	58
DTK_FRIENDLY_LANCE	21	DTK_ENEMY_LANCE	23
DTK_FRIENDLY_KNIGHT	57	DTK_ENEMY_KNIGHT	14
DTK_FRIENDLY_SILVER	42	DTK_ENEMY_SILVER	19
DTK_FRIENDLY_BISHOP	6	DTK_ENEMY_BISHOP	45
DTK_FRIENDLY_ROOK	22	DTK_ENEMY_ROOK	29
DTK_FRIENDLY_GOLD	26	DTK_ENEMY_GOLD	9
DTK_FRIENDLY_PROMOTED_PAWN	13	DTK_ENEMY_PROMOTED_PAWN	8
DTK_FRIENDLY_PROMOTED_LANCE	11	DTK_ENEMY_PROMOTED_LANCE	0
DTK_FRIENDLY_PROMOTED_KNIGHT	31	DTK_ENEMY_PROMOTED_KNIGHT	50
DTK_FRIENDLY_PROMOTED_SILVER	10	DTK_ENEMY_PROMOTED_SILVER	21
DTK_FRIENDLY_PROMOTED_BISHOP	16	DTK_ENEMY_PROMOTED_BISHOP	14
DTK_FRIENDLY_PROMOTED_ROOK	42	DTK_ENEMY_PROMOTED_ROOK	6

Table 4: Average evolved parameters of the evaluation function of the best individual from the population after roughly 70 runs.

References

- [1] Computer shogi complexity.
- [2] Yamamoto Masahito, Suzuki Keiji , Ohuchi Azuma. An acquisition of evaluation function for shogi by learning self-play. *International Transactions in Operational Research*, 8(3):305–315, 2001.
- [3] Aono Teruichi, John Fairbairn. *Better Moves for Better Shogi*. Ishi Press International, 1983.
- [4] Reijer Grimbergen. Plausible move generation using move merit analysis with cut-off thresholds in shogi. *Computer and games. Springer.*, page 315–345, 2001.
- [5] Yoshikuni Sato, Daisuke Takahashi, Reijer Grimbergen. A shogi program based on monte-carlo tree search. *International Computer Games Association*.
- [6] Hoki K. Optimal control of minimax search results to learn positional evaluation. *The 11th Game Programming Workshop*, 2006.
- [7] Eli (Omid) David, H. Jaap van den Herik, Moshe Koppel, Nathan S. Netanyahu. Genetic algorithms for evolving computer chess programs. *IEEE Transactions on Evolutionary Computation*, 18(5):779–789, 2014.