HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

INTELLIGENT OTHELLO

Course: Introduction to Artificial Intelligence
Supervisor: Assoc. Prof. Than Quang Khoat

Authors:	Student ID:
Nguyen Viet Anh	20225434
Pham Minh Hieu	20220062
Nguyen Trong Tam	20225527
Hoang Trung Khai	20225502
Trinh Duy Phong	20220065

Hanoi, December 2023

ABSTRACT

The realm of artificial intelligence has devoted significant focus to game playing, considering it one of the most demanding domains. Othello, in particular, has emerged as a game that seamlessly aligns with computer gaming strategies, sparking extensive research in this area.

Despite the creation of numerous computer-based Othello players that have outperformed human world champions, the specific interactions among various Othello heuristics remain unclear. Our research implements and examines various heuristics, in an attempt to make observations about the interplay between the heuristics, and how well each heuristic contributes as a whole. Identifying heuristics that contribute immensely to Othello game-play implies that more processor cycles could be allocated in that direction to enhance the quality of play. Due to the complexity of accurate calculations, most heuristics tend to approximate.

In the evaluation section, diverse experimental data is gathered and examined to offer insights into the optimal utilization of heuristics. While the focus of this paper is on Othello and its related heuristics, such analyses are applicable to other analogous games derived from Go.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 Othello

Game playing, a captivating facet of artificial intelligence research, has drawn considerable attention. Computers, exhibiting the ability to anticipate moves beyond the immediate future and strategically determine optimal next moves, emulate and, at times, outperform human intelligence. They've successfully navigated the intricate complexities of game playing. The importance of game playing arises out of the competition that exists between the human race and machines. It is a race to prove superior intelligence.

Othello [1], a classic board game with a rich history, has become a focal point of interest in both gaming and artificial intelligence. This strategic two-player game, also known as Reversi, involves players vying for control over the board by strategically placing their pieces. Othello's simplicity belies its complexity, making it an intriguing subject for research and analysis in the realm of artificial intelligence and game theory.

Othello has been an example where computers have exemplified great game play, beating human world champions . The primary reason being the small branching factor, allowing the computer to look ahead in abundance, thrashing human intuition and reasoning. As processors increase in speed and complexity, the ability of computers to reason beyond the current state increases, while human intelligence maintains a fairly static average performance over generations. This leads to an extending gap in the Othello-playing ability of humans and computers.

Aside from the minor branching elements, heuristic functions' accurate representation of the game's state is another reason for Othello's enormous success. When used carefully, heuristics in Othello have minimal drawbacks. Heuristic values are typically determined as a linear function of several separate heuristics. We implement various heuristics, along with several optimizations, and determine the contribution of each heuristic to the entire game play. We also ascertain how each heuristic interacts with the others during this process. Comprehending the function of every heuristic in the gameplay would facilitate the efficient utilization of additional processing cycles.

1.2 Game Play

1.2.1 Basic rules

Othello is an abstract strategy game played by two players on a board with 8 rows and 8 columns and a set of 64 identical pieces. The pieces are typically discs with a light and a dark face, each colour assigned to one of the players. Players take turns making a "move" which consists in placing one disc on the board with their assigned colour facing up and then flipping over all the opponent's discs that are "outflanked" by the disc that was just placed. The objective of the game is to have a majority of one's own coloured pieces showing at the end of the game.

Some basic rules of Othello: [2]

1. Black always moves first.
2. If on your turn you cannot outflank and flip at least one opposing disc, your turn is forfeited and your opponent moves again. However, if a move is available to you, you may not forfeit your turn.
3. Players may not skip over their own colour disc(s) to outflank an opposing disc.
4. Disc(s) may only be outflanked as a direct result of a move and must fall in the direct line of the disc placed down.
5. All discs outflanked in any one move must be flipped, even if it is to the player's advantage not to flip them at all.
6. Once a disc is placed on a square, it can never be moved to another square later in the game.
7. When it is no longer possible for either player to move, the game is over. Discs are counted and the player with the majority of their colour showing is the winner.
8. Note: It is possible for a game to end before all 64 squares are filled.

1.2.2 Pygame

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language.

In our research, we have already used Pygame to build up Othello with such features like Gameplay, specific buttons, flipping effect, and so on.

CHAPTER 2. THEORETICAL BASIS

2.1 Minimax algorithm

The minimax search [3] does a depth-first exploration of the entire game tree. The heuristic functions are utilized at the leaves to provide utility values. The utility values are then backed up all the way to the root. The manner in which the values are backed up to a node depends on whether the node is a min node or a max node. The max player has to make a move while at a max node, while the min player has to do so at the min node. The max player is the player who has to make the actual next move in the game, and has to maximize his/her utility value, while the min player does the reverse. A typical minimax tree is such that the min player and max player alternate. This need not necessarily hold throughout, since turns may be skipped by players in certain games, such as Othello.

2.2 Alpha-beta pruning

Alpha-beta search [4] is similar to minimax, except that efficient pruning is done when a branch is rendered useless. Such pruning tends to be rather effective and the search can proceed to great depths, allowing the computer to implement a relatively more powerful look ahead. Pruning is done when it becomes evident that exploring a branch any further will not have an impact on its ancestors.

2.3 Alpha-Beta pruning With Iterative Deepening

It becomes unclear as to the optimal depth to search up to, if the depth is to be defined statically. Hence, we implemented alpha-beta with iterative deepening. The algorithm sets depth to a reasonable initial value of three. Then the depth is increased and the search is conducted again. This is done till the timing constraints are not violated. Before searching with an increased depth, a naïve check is made to ensure that the search about to be spawned will not violate timing constraints. Using the time taken for the previous depth, we approximately calculate a new depth at which the next iteration can take place. This new depth would be such that it finishes, according to the approximation, before the allotted time. If it doesn't, the process is not preempted, and the execution exceeds by a few milliseconds.

The heuristic functions regulate the computer's capacity to accurately assess a state's suitability for a player. Several of factors influence whether a particular game state is beneficial to the player. In Othello, a player's advantage in a given position is determined by elements like mobility, stability, coin parity, and corners. The most logical method of determining a heuristic value is to combine the quantitative representations of the numerous significant variables in a linear fashion.

The utility value of a state is returned via two main functions. The initial utility function is a linear combination of various essential heuristic elements that are used to assess the condition. The utility value is determined by the second utility function by applying weights that are statically allocated to each square on the board. The utility value of a state can be obtained using either of these functions, which are covered in more detail in the subsections that follow.

3.1 Component-wise Heuristic function

This approach of determining the utility value makes use of a variety of heuristics and gives each one a unique weight. After establishing the mobility, coin parity, stability, and corners captured aspects of the configuration, the condition of an Othello game is assessed. For each of these, we have a heuristic function at our disposal. The return value of each heuristic is scaled from -100 to 100. To play the best game possible, we appropriately weigh these variables.

3.1.1 Coin Parity

This component of the utility function captures the difference in coins between the max player and min player. The return value is determined as follows:

$$\textbf{Coin Parity Heuristic Value} = 100 \cdot \frac{\textbf{Max Player Coins} - \textbf{Min Player Coins}}{\textbf{Max Player Coins} + \textbf{Min Player Coins}}$$

The most common tactic used by many early computer Othello players was to base their move on a greedy approach that attempted to maximize a player's coin total at any given time. Such approaches clearly and utterly failed. The maximum number of coins that a single move can flank is 18, suggesting that games can change hands extremely quickly. Since a complete exploration of the game tree would not be possible till the very end stages of the game, such a strategy does not incorporate the drastically dynamic nature of the game. It also doesn't explain why coins are unstable. It could be preferable to have a few stable coins rather than 10 unstable ones.

3.1.2 Mobility

An interesting tactic to employ is to restrict your opponent's mobility and to mobilize yourself. This ensures that the number of potential moves that your opponent has would drastically decrease, and your opponent would not get the opportunity to place coins that might allow him/her to gain control. Mobilizing yourself would imply a vast number of moves to choose from, hence indicating that you can exercise power and control the proceeding of the game.

There are two types of mobility: prospective mobility and actual mobility. The amount of next moves a player has, considering the conditions of the game at that moment, is their actual mobility. The amount of probable moves the player may have during the following few moves is known as potential mobility. It should be noted that movements that are illegal at the moment but may become such in the near future are taken into consideration when determining potential mobility. Therefore, real mobility captures the player's current movement, whereas potential mobility records the player's long-term mobility. Potential mobility can make up for a shallow depth when measuring the mobility component of the game since it anticipates things on its own without the aid of searching strategies.

By looking over the board and counting the player's allowed moves, actual mobility is determined. The number of empty spaces adjacent to at least one of the opponent's coins is used to determine potential mobility. Although potential mobility is a very basic metric, it has shown to be quite useful. There is a trade-off between potential mobility calculation effectiveness and complexity. Because anticipating mobility is a challenging task, the more effective it must be, the more complex it would become. The routine would consume a large amount of processor time if it became overly complicated, time that could have been used to search the game tree. The calculation of the potential mobility heuristic value follows the same steps as the calculation of the actual mobility heuristic value.

$$\textbf{if } (\textbf{Max Player Mobility Value} + \textbf{Min Player Mobility Value}) \neq 0 :$$

$$\textbf{Mobility Heuristic Value} = \frac{100 \cdot (\textbf{Max Player Mobility Value} - \textbf{Min Player Mobility Value})}{\textbf{Max Player Mobility Value} + \textbf{Min Player Mobility Value}}$$

$$\textbf{else:}\quad \textbf{Mobility Heuristic Value} = 0$$

3.1.3 Corners Captured

Corners are the four squares a1, a8, h1, and h8. These squares are special in that the opponent cannot flank them once they are captured. In addition, they give the player's coins stability in the surroundings and let them be built around. Gaining these advantages would guarantee stability in the area, and stability heavily influences how things turn out in the end. A player's ability to win the game is highly correlated with how many corners they are able to capture. Of course, that is untrue because it is evident that capturing the majority of the corners does not guarantee victory. However, if most of the corners are taken, more stability can be constructed.

We gave weights to corners that were actually captured, probable corners, and improbable corners.

A player's possible corner is one that might be taken in the upcoming move, whereas an unlikely corner is one that is positioned so that it won't be taken anytime soon. The total of these variables yields the corner heuristic value for a player. The following formula is used to get the return value:

$$\textbf{if } (\textbf{Max Player Corner Value} + \textbf{Min Player Corner Value}) \neq 0 :$$

$$\textbf{Corner Heuristic Value} = \frac{100 \cdot (\textbf{Max Player Corner Value} - \textbf{Min Player Corner Value})}{\textbf{Max Player Corner Value} + \textbf{Min Player Corner Value}}$$

$$\textbf{else: Corner Heuristic Value} = 0$$

3.1.4 Stability

In Othello, coin stability is a crucial component. A coin's stability measure is a numerical representation of its susceptibility to being tricked. Coins can be categorized into three groups: semi-stable, unstable, and stable. Coins classified as stable are those that, in the game, cannot be flanked from their initial state at any point. Coins that could be flanked in the very next move are considered unstable. Coins that have the potential to be flanked in the future but do not immediately face the risk of being flanked in the next move are considered semi-stable. In nature, corners are always stable, and as you add to them, more coins become stable in the region.

Each of the three categories has a weight, which we add together to determine the player's final stability value. Typical weights could be 1 for stable coins, -1 for unstable coins and 0 for semi-stable coins.

$$\textbf{if } (\textbf{Max Player Stability Value} + \textbf{Min Player Stability Value}) \neq 0 :$$

$$\textbf{Stability Heuristic Value} = \frac{100 \cdot (\textbf{Max Player Stability Value} - \textbf{Min Player Stability Value})}{\textbf{Max Player Stability Value} + \textbf{Min Player Stability Value}}$$

$$\textbf{else: Stability Heuristic Value} = 0$$

3.2 Static Weights Heuristic Function

A static board of weights corresponding to each coin position, as seen in Table 3.1, is an alternative to using the utility function covered in Section 3.1. The weights of the squares that contain the player's coins are added up to determine the player's heuristic value.

The game play is pushed to focus on corner capture by the static board, which subtly conveys the significance of every square on the board. The use of heuristics to determine a position's weight based

	a	b	c	d	e	f	g	h
1	100	-20	10	5	5	10	-20	100
2	-20	-50	-2	-2	-2	-2	-50	-20
3	10	-2	-1	-1	-1	-1	-2	10
4	5	-2	-1	-1	-1	-1	-2	5
5	5	-2	-1	-1	-1	-1	-2	5
6	10	-2	-1	-1	-1	-1	-2	10
7	-20	-50	-2	-2	-2	-2	-50	-20
8	100	-20	10	5	5	10	-20	100

Table 3.1: Shows the static weights assigned to each individual position in the board

on its stability, mobility, and other factors would be necessary if these weights were dynamically changed. This would suggest that the utility value computation process would resemble the one covered in Section 3.1.

$$\textbf{Utility Value} = \textbf{Max Player Utility Value} - \textbf{Min Player Utility Value}$$

CHAPTER 4. RESULT

We implemented the Othello game along with the various heuristics and search strategies in the Visual Studio Code. The search strategies implemented were the ones discussed in Chapter 2, while the heuristics implemented were the ones explained in Chapter 3.

The relative significance of the different heuristics is shown in this section. We ran several tests using our Othello game, enabling various heuristics with different weights.Though we implemented the three different search strategies mentioned in Chapter 3, we use the alpha-beta search strategy with a depth of 5 and 6, unless otherwise mentioned.

4.1 One-on-One Heuristic Comparison

4.1.1 Depth = 5

	Coins	Corner	Stability	Mobility
Coins		52-0 Corner	41-23 Stability	37-27 Mobility
Corners	45-19 Corner		35-29 Stability	44-20 Mobility
Stability	41-23 Coins	41-23 Corner		35-29 Mobility
Mobility	37-27 Mobility	48-16 Corner	41-23 Stability	

Table 4.1

	Coins Won	Coins Lost
Corner	259	107
Stability	192	189
Mobility	168	216
Coins	137	225

Table 4.2

4.1.2 Depth = 6

	Coins	Corner	Stability	Mobility
Coins		60-3 Corner	40-24 Stability	51-13 Coins
Corners	53-0 Corner		34-30 Corners	45-19 Corners
Stability	40-24 Coins	49-15 Corner		34-30 Mobility
Mobility	36-28 Mobility	50-13 Mobility	48-16 Stability	

Table 4.3

	Coins Won	Coins Lost
Corner	253	117
Stability	187	197
Mobility	168	215
Coins	146	218

Table 4.4

Table 4.1 and 4.3 show how each heuristic performed against the other heuristics in depth 5 and depth 6. Heuristics in column 1 played first and each cell specifies the score and the heuristic that won. A cell x,y represents the results of the game between heuristic x and heuristic y.

Table 4.2 and 4.4 show the number of coins won and lost by each of the heuristics on the whole, when it played against other heuristics.

The results of the games that were played are shown in Table 4.1. Table 4.2 displays the total number of coins won and lost by each heuristic, which corresponds to the results in Table 4.1. We played each heuristic against every other heuristic, and two such games were played for each pair, one with a different heuristic starting both times. The games were set up so that the heuristics listed down column 1 played first.

The tables clearly show that the corner heuristic is the most effective heuristic used alone. All heuristics are inferior to the corner heuristic. Stability and mobility are closely vying for second place, with each nearly matching the other.

The corner heuristic directs gameplay in a way that enhances the chances of capturing corners. The greater the number of corners captured, the more the control a player can exercise over the middle portions of the board, thus flanking a significant portion of the opponent's coins. Thus, this heuristic makes sure that corners are not easily given up, regardless of the role played by the other heuristics. This allows the corner heuristic to partially offset the benefits of the other heuristics when used in opposition to them.

Stability, with its classification of stable, semi-stable and unstable moves is able to guide the game in a good direction. It ensures that as many coins as possible are captured and promoted higher in the stability order. This strategy minimizes the opportunity for the opponent to flank coins and take over the game. Games played with stability as the primary concern, tend to play moves to capture corners and edges as fast as possible, and build upon these regions. This introduces stability in the region, which pervades through the board subsequently. Figure 4 shows how the stability heuristic captures corners and edges when it plays against the coin parity heuristic. Though the greedy coin parity heuristic starts well, by grabbing unstable coins, the stability heuristic prevails as it grabs stable coins and flanks the opponent's coins. Stability loses to the corner heuristic primarily because the corner heuristic is more hungry for corners, hence preventing the stability heuristic from capturing them. This means that stability cannot be built by the stability heuristic, because of the lack of its ability to grab the most stable regions of the board, the corners, against the corner heuristic.

Mobility, on the other hand, is effective mainly because it minimizes the number of moves the opponent has, while maximizing the other player's moves. This implies that the opponent is never able to take complete control over the game due to the lack of available moves. Mobility forces the opponent to choose from a constrained set of moves. The player using the mobility heuristic, on the other hand, would enjoy a wide variety of moves.

As expected, the greedy strategy of maximizing the number of coins does not perform too well. In Othello, it is easy to gain a lot of unstable coins in one move, but losing them is equally easy. The player with the most stable coins in the final stages of the game controls the board. Hence, though the ultimate goal is to maximize the number of coins, using that as a heuristic fails.

4.2 Heuristic Contributions

	E-Coins	E-Corners	E-Stability	E-Mobility
Everything go first	36-28 E-coins	50-14 Everything	34-30 E-stability	57-7 Everything
Everything go second	47-17 E-coins	35-29 Everything	38-26 Everything	40-24 Everything

Table 4.5: Depth = 5

	E-Coins	E-Corners	E-Stability	E-Mobility
Everything go first	34-30 E-coins	43-21 Everything	50-14 Everything	33-31 E-mobility
Everything go second	44-20 Everything	58-6 Everything	34-30 Everything	53-0 Everything

Table 4.6: Depth = 6

The two above tables depict the results for the games with a player with all heuristics enabled versus a player with all but one heuristic enabled. Such an experiment would depict the importance of the omitted heuristic, and would give an approximate idea of their impact on the game play, with optimal weight settings

We use the term Everything to represent the function that contains all the heuristic components with the equal weights. E-x is used to represent the function that calculates the heuristic value using all the heuristics in Everything except x.

We can easy notice that E-coins win the lowest number of match with only 1 game. This fickle behaviour can be attributed to the greedy nature of the coin parity heuristic. While using the coin parity heuristic, it is very easy to be caught in a local maxima trap, which is what happened in the game in which E-Coins won. Since the coin parity heuristic drove the Everything heuristic to a local maxima, the E-Coins heuristic was able to beat it easily. In the other case, where Everything won, either no such local maxima was encountered, or the rest of the heuristics managed to maneuver the computation safely away.

Corners, as expected, cause a degradation in game play quality when they are removed from the Everything heuristic. This is very obvious, since the corners heuristic ensures that corners, as starting stable positions are caught, and that the stability heuristic can build upon this. Corners appear to be a less powerful heuristic than stability when it is not a standalone heuristic, primarily because stability is better complemented by the mobility heuristic. Capturing corners, from our experience, increases mobility, hence the mobility heuristic aids the stability heuristic to capture corners and build stability. The mobility heuristic, however, does not aid the corner heuristic to build stability once the corners have been captured.

The mobility heuristic ensures that the opponent does not have too many moves to choose from, hence restricting the opponents control over the board. It has such a great impact on the game play, as demonstrated by the results. The Everything win most of the matchs, with one of them is very drastical. This suggests that increasing the weight of the mobility heuristic might enhance play quality, but that was not true during our experimentation, because high mobility downplayed other heuristics leading to bad moves.

4.3 Heuristic Combination

Based on the above results, now we will choose the rate for the components of the final heuristic function. Most of the rate satisfies the condition that the weight of the components is in the following descending order: Corner - Stability - Mobility - Coins Parity Here are some combinations we choose

	Corner	Coins	Mobility	Stability
Combine 1	6	1	2	3
Combine 2	10	4	5	7
Combine 3	7	6	9	4
Combine 4	9	3	4	5

Table 4.7: Coefficient of each heuristic combination

for the final heuristic. Note that the 3rd combination is in order Mobility ¿ Corner ¿ Coins Parity ¿ Stability because of the results in the section 2

	Combine 1	Combine 2	Combine 3	Combine 4
Combine 1	N/A	45-19 2 win	48-16 3 win	348-16 1 win
Combine 2	39-25 2 win	N/A	44-20 2 win	41-22 2 win
Combine 3	57-7 3 win	36-28 2 win	N/A	49-15 4 win
Combine 4	38-26 4 win	38-26 2 win	53-11 4 win	N/A

Table 4.8: Table show results after forcing every heuristic combination play against each other

Here is the result of the matches between the components. Note that the for each cell, the component in the left will go first.

	Coins Won	Coins Lost
Combine 2	243	140
Combine 4	223	179
Combine 3	179	205
Combine 1	141	243

Table 4.9

With this table, we can see that the 1st combination is the weakest because in that combination, it almost ignores the importance of three heuristics other than Corner. The 3rd one is not very strong, which show that the order Mobility - Corner - Coins Parity - Stability is not effective. The 2nd and 4th combinations have nearly the same efficiencies to the remainders, but the 2nd one is a little stronger as can be seen from the result of their fight so we choose it as the final distribution for the evaluate function.

4.4 Component-wise Heuristic Vs Static Board Heuristic

The component-wise heuristic is basically a mix of the four heuristics. The static board heuristic is the one in the section 3.2, where board positions are assigned certain static weights. Two games were played between- the two. The game with the static board heuristic starting first had a result of 38-26 with the component-wise heuristic winning. While the other game, with the component-wise heuristic playing first, had a score of 41-22 with the component-wise heuristic winning again. The main reason for the defeat of the static board heuristic is due to its lack of ability to dynamically change weights to represent the current state of the game. The component-wise heuristic captures that state of the game and suitably modifies the weight in order to guide the game in the right direction. For example, the static board heuristic does not take into account the stability of the current state of the game before making the next move. The static board heuristic takes a rather narrow-minded view of the game by claiming that certain squares are always good positions to play coins at, irrespective of the current state of the game.

4.5 Summary

The results imply that when combined with the corner, coin parity, and mobility heuristics, stability is a very effective heuristic. Currently, the actual stability value can be approximated by applying a standard stability heuristic. This is due to the complexity involved. However, dedicating additional processing power to this task would undoubtedly yield financial benefits.

CHAPTER 5. CONCLUSION

5.1 Conclusion

It will always remain the case that one of the most appealing areas of artificial intelligence research is game playing. It is one of the aspects of artificial intelligence that the common person can observes and interacts with.

We assessed which heuristics are most useful for improving Othello game play. This research attempted to clarify how different heuristics that are used to explain an Othello game's state interact with one another. We examined their importance as well. We discovered that significantly increasing the stability heuristic's accuracy would improve game play. It was also notable that stability was a key factor in the component-wise heuristic function, even though corners was the most effective standalone heuristic. This study would allow one to identify the most important aspects, and enable more processor power to be thrown into it in order to increase the accuracy of the heuristic.

5.2 Future Work

Learning strategies would be incorporated into the system in future work. Such strategies tend to be very powerful because they can leverage enormous volumes of pre-existing data and avoid common mistakes made by deterministic algorithms. We can also improve our framework further to train a skilled Othello player who can later participate in competitions. That would necessitate careful weight adjustment, optimized lookup tables, and potent learning techniques.

REFERENCE

An analysis of heuristic in Othello, Vaishnavi S., Muthukaruppan A.

[1]Wikipedia, “Othello”. [Online]. Available: `https://en.wikipedia.org/wiki/Othello`

[2]World Othello Federation, “OFFICIAL RULES FOR THE GAME OTHELLO”. [Online]. Available: `https://www.worldothello.org/about/about-othello/othello-rules/official-rules/english`

[3]Alexander Katz and Eli Ross, “Minimax”. [Online]. Available: `https://brilliant.org/wiki/minimax/#:~:text=In%20game%20theory%2C%20minimax%20is,payoff%20as%20large%20as%20possible.`

[4]Great Learning Team, “Alpha Beta Pruning in AI”. [Online]. Available: `https://www.mygreatlearning.com/blog/alpha-beta-pruning-in-ai/`