CHAPTER 2
LITERATURE REVIEW

Computational intelligence techniques were combined with games for the first time in 1959, when Samuel applied a simple reinforcement learning algorithm to the board game Checkers [7]. Without human commands, through only playing itself and observing the sequences which won games and which sequences lost games, and using the very limited hardware available at the point in time, the algorithm was able to learn strategies good enough to beat its creator.

After the early success of Samuel, the research in the field remained silent for a long time. But as part of artificial intelligence research, a few researchers have worked on applying classical AI techniques, basically especially custom-made search algorithms, to board games such as Chess and Checkers. This direction of research ultimately led to the much-publicized victory of the IBM Deep Blue Chess computer over world Chess champion Gary Kasparov in 1997 [7]. Whereas this and other results are impressive in their own right, little or no computational intelligence is used in such research.

In the last few years, however, a significant mass of researchers have become interested in both computational intelligence and in games, in one way or another, for the small but rapidly growing research field of Computational Intelligence and Games to form. Much of the research in this field is published in a small number of conferences.

2.1 COMPUTER GAMES:
There is not one definition of what a game is, but ample. In fact, the philosopher Wittgenstein used the concept of a game in a number of thought-experiments designed to show that it was not possible to correctly define any concept in terms of sufficient and
necessary conditions. Concepts are implicitly defined by those things that they refer to, and which are related to each other through family likeness.

The legendary game designer Sid Meier defines a game as “a series of meaningful choices”. Others, such as Zalen and Zimmermann, emphasize conflicts as central to games: a game is “a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome”. Juul provides a more academic definition: “A game is a rule-based formal system with a variable and quantifiable outcome, where different outcomes are assigned different values, the player exerts effort in order to influence the outcome, the player feels attached to the outcome, and the consequences of the activity are optional and negotiable”.

In discussing these and other definitions, game designer Raph Koster remarks in his recent book that none of them contain the word “fun” [8]. As fun seems to be so vital to games, he then devotes the rest of the book to understanding what makes games fun. According to Koster, a game is fun to play because we learn the game as we play; we understand and learn the patterns underlying the game, and finally master how to play it [7]. This requires that the level of challenge always is approximately right, and that new patterns are always available to learn - games that are too simple or impossible to understand are boring. According to Koster, thus, a game is fun because it is a good teacher.

Another theorist who has tried to find the essence of games and why they are fun is Thomas Malone. He claims that the factors that make games fun can be organized into three categories: challenge, fantasy, and curiosity [9]. The first thing about challenge is that the existence of some sort of goal adds to the entertainment value. Further, this goal should not be too hard or too easy to achieve, and the player should not be too confident about what level of success he will achieve.

Games that include fantasy, according to Malone, “show or evoke images of physical objects or social situations not actually present”. The feeling of being somewhere else, doing something else. And for the third factor, curiosity, Malone claims that fun games
have an optimal level of informational complexity in that their environments are new and surprising but not completely beyond your understanding.

2.1.1 Abstract Representation of a Game:

An abstract representation of a game is a set of rules which allow a player to play the game lawfully, but which is more compacted than the general representation of the game. A well-known representation clearly lists the evolutions and outcomes for each state in the state-space of the game. In contrast, an abstract representation offers a generating function which defines the beneficiaries for each state in terms of a set of changes to the state’s internal structure. It also defines goals using predicates on states with internal structure. For example, a state-transition rule in an abstract representation of board game dictates that states which contain a discs or dice on a square have as successor’s states in which that square is empty, a dice is on another square relative to the first square, and all other squares are the same as in the first state. An abstract depiction of a game is one which is smaller than the game-tree generated by it.

The concept of abstract representation is basic to all work on games in AI. It is a prerequisite for game analysis. So, board representation is very much needed for the successful application of conceptual tools like strategies, heuristics, and so on to deliberate a competitive advantage on that game.

Board game assumes that games are represented in a flat form, as a matrix or tree. It also assumes that players have no resource boundaries in terms of time or space, so that they can keep the complete game tree in memory, and can compute all the possible consequences of each move. Given these resolutions, and could conceive an algorithm which played the game perfectly in finite time, the game would be effectively trivial. This means that the finite two-player perfect-information games like Checkers and Reversi are considered trivial to this field.
However, when the resource limitations are taken into account of the computer board game players, it is obvious that a player could never preserve the entire game tree for a big and complex game in memory nor can consider all possible consequences of each action. Thus a player must consider potentials and outcomes selectively, and make conclusions based on less-than perfect information.

Thus Botvinnik for this reason placed games like chess into the class of inexact problems. As the player cannot in general see the exact influence of a move on the final goals of the game, it follows that her reasoning must be heuristic. That is, the reasoning must make direction to the structure encoded in an abstract illustration of the game. It means that the game analysis is significantly dependent on the existence of an abstract representation of the game.

The concepts for the game can be broadly categorized in two categories: game specific and game-assumptive concepts. In the context of chess, a game-specific search method might be to examine attacks against a king before other moves. Some game-specific heuristics might be that it is unfavorable to have one’s knight on the edge of the board, and favorable to have a king surrounded by pawns. A game specific strategy might be to build up an attack against enemy pawns which cannot be protected by other pawns.

These concepts are all game-specific, in that they refer to syntactic elements of one game, and they would be meaningless in the context of another game. For example, it does not make any sense to examine checks or attack weak pawns in the context of Go-Moku and Tic-Tac-Toe. In these games there is reference these pieces.

While some assumptions hold for one game but do not hold for the other game. These types of concepts are known as game assumptive concepts. When applied to games in which the same assumptions hold the concept retains its utility, but when applied to games for which they do not hold the utility is lost.
2.1.2 Types of Computer Games:

There is numerous taxonomy of computer games. One taxonomy is - computerized games, agent games and management games.

Computerized games are games which you don’t really need a computer to play, because the computation involved is minimal. Such games almost always have discrete state spaces and simple rule sets, and are usually a temporal. Examples we find are all the classic board games: Chess, Checkers, Go, Monopoly etc. We also find card games such as Poker and Bridge, and puzzles such as crosswords and Sudoku.

Due to the ease of implementing computerized games, they are well suitable for fair comparisons of different algorithms with each other, and with human players. This is particularly true for those games that have been widely analyzed and which have large and active communities of human players.

Agent games are games where the player controls an agent in a simulated world, and directly decides what the agent does at any point in time. Usually, such games happen in real time, adding an element of resource-constrained decision making; which are often games of imperfect information, as the player can not see the whole play-field or level.

In Management games, the player does not control a single alive agent, at least not most of the time. Rather, the task of the player is to develop strategies, distribute resources, schedule events. Management games are, like agent games, excessively complex to be played practically without computers.

2.1.3 Commercial versus Academic Game AI:
Commercial and academic game AI is two different fields. The two fields have different goals and use different types of algorithms [7]. The academic researchers use computer intelligence to try to beat games, whereas AI programmers in game companies use their techniques to try to make games more interesting. The main inspiration that drive AI
programmers in the game industry is to use AI that makes the game enjoyable, and keep the players playing, which ultimately drives the sales of the game.

Typically, game companies have a very practical view on how AI is implemented, and anything goes as long as it works in the game; the techniques used mostly try to create false impression of intelligence don’t need to have anything to do with what academic researchers would know as AI. Instead, majority of commercial games use finite state machines lists of conditional statements and the occasional some path finding algorithm. In general, no learning is present in either development or execution of the AI. Everything is hard-coded by the human game developers.

2.1.4 Game Programs:

In many games with comparatively simple logistics but complex strategy, computer programs have now surpassed the best human players by a huge margin. In each case, the formula for success has been the deep look-ahead search of the game tree using the highly efficient alpha-beta search algorithm, combined with a domain-specific evaluation function applied at the nominal leaf nodes of the search tree.

1. In 1990, the checkers program CHINOOK earned the right to challenge the human world champion, Marion Tinsley, in a title match. CHINOOK lost narrowly in 1992, but won the return match in 1994, becoming the first computer program to win an official world championship in a game of skill [11].

The solution is likely to be achieved by combining three approaches [63][64].

• **Endgame databases:** The program has perfect information about all positions involving eight or fewer pieces on the board, a total of 443,748,401,247 positions, compressed into 6 GB for real-time decompression.

• **Middle-game databases:** Whenever the program is able to determine the game theoretic value of a position during a game, the position is added to the middle-
game database. In practice, positions with as many as 22 pieces on the board have been solved, i.e., CHINOOK has announced the result of the game as soon as move 5.

- **Verification of opening analysis:** Starting with published lines of play for each opening, or lines as played by the program, deep searches are used to try and solve positions as close to the start of the game as possible. These solved positions are added to the middle-game database. If the lines are stable and do not get refuted, the solution eventually reaches the opening position.

2. In 1997, the Othello program LOGISTELLO defeated the human world champion, Takeshi Murakami, in a six game exhibition match, winning all six games [12][38].

3. In 2000, the Lines of Action program MONA won the de-facto world championship, defeating all of the top human players, and winning every game it ever played against human opposition [62].

4. Hex is a relatively new game (invented in the 1940s), but its simplicity of rules gives rise to elegant and complicated play. In “A Hierarchical Approach to Computer Hex”, Vadim Anshelevich describes his program HEXY. The program uses a novel way of searching the tree, using a theorem-prover-like search to evaluate leaf nodes. HEXY is competitive with top human players, but mankind still holds a significant edge particularly on larger board sizes.

5. Shogi is often referred to as Japanese chess and, indeed, it shares many similarities with its Western counterpart. However, the ability to return captured pieces to the board results in a more tactical game with an increased branching factor. Hiroyuki Iida, Makoto Sakuta, and Jeff Rollason describe the state of the art for this game in their paper “Computer Shogi”. Programs have a long way to go before they can offer a serious challenge to a shogi grandmaster.
6. Draughts (10 * 10 checkers): Draughts programs are strong, but not yet a serious threat to the World Champion.

7. Amazons: Amazons is being touted as a challenge that nicely bridges the gap between chess and go. Like go, the object is to capture territory. Programs benefit from chess-like searches without the need for extensive knowledge, but suffer from the large branching factor. The best programs are still quite weak[61].

8. Building programs to play poker well is very challenging from the AI point of view. As Poker is imperfect information game, Programs must deal with incomplete knowledge - hidden opponent cards, multiple competing agents players per game, risk management betting, opponent modeling exploiting sub-optimal opponent play, and deception. In “The Challenge of Poker”, Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron describe their program POKI. The program plays well, but a considerable performance gap remains to be overcome.

9. Connect-Four is a four-in-a-row connection game played on a vertically-placed board. The standard game has a board of 6 rows and 7 columns. The game has been solved by Allen, using a brute-force approach [65], and simultaneously by Allis, using a knowledge based approach [66]. It is a first-player win. Allen’s program uses brute-force depth-first search with alpha-beta pruning, a transposition table, and killer-move heuristics. Moreover, he used expert knowledge to “tweak” the program. The game was weakly solved on a Sun-4/200-series workstation in about 300 hours. In contrast, Allis used nine strategic rules that identified potential threats of the opponent. He weakly solved the game, also on a Sun-4 in about 350 hours. The conditions of each rule were exactly formulated, as were the allowable combinations of the nine rules. These rules were implemented in the program VICTOR, which in addition exploited a
combination of straightforward depth-first search and conspiracy-number search [67].

10. Qubic is a three-dimensional version of Tic-Tac-Toe, where the goal is to achieve a straight chain of four own stones in a 4 * 4 * 4 cube. No gravity conditions apply. In 1980 Patashnik weakly solved the game by combining the usual depth-first search with his own expert knowledge for ordering the moves. In the early 1990s Allis and Schoo wrote the program QBIG to solve Qubic. Initially they were not aware of Patashnik’s results. When they were notified in the last phase of the project, they decided to continue, since their main goal was not only the solution of Qubic, but also the testing of two new methods: threat-space search, developed for Go-Moku and Renju, and proof-number search, already tested in the domain of Connect-Four and Awari. Thus, Allis and Schoo weakly solved Qubic again [68], confirming Patashnik’s result that Qubic is a first-player win.

11. The Awari board contains two rows of six pits. The players, traditionally called *North* and *South*, each control one row. The game starts with four stones in each pit. An extra pit for each player is used to collect captured stones, but these are not used in sowing stones. A player to move chooses one of their non-empty pits and starts sowing. Whenever the sowing reaches the start pit again, it is ignored. If the last stone lands in an enemy pit which then contains 2 or 3 stones, the stones are captured. After the capture, the previous pit is examined, and when it too contains 2 or 3 stones, they are captured also, and so on. Capturing only occurs in enemy pits. The game terminates when a player has no moves left or when a player has collected 25 or more stones. The rules force each player to leave the opponent a move whenever possible. Figure 2.1 shows initial position of Awari.

In 2002, the ancient game of Awari was strongly solved, computing the exact minimax value for every reachable position in the game. Although the best
programs already played at a level far beyond any human player, the difference between super-human play and perfect play was shown to be huge.

![Image of Awari board](image)

Figure 2.1 Initial position of Awari

12. Another use of AI in games is in the game of backgammon. This was first used in the program BKG 9.8. In 1979, BKG 9.8 played a backgammon match against world champion Luigi Villa the day after he had won the world championship in Monte Carlo. The program won with a final score of 7 to 1. Despite the score, Villa played better than BKG 9.8. He played almost all the right moves, while the backgammon program only played 65 out of 73 correct moves.

Next in the 1980’s, Gerry Tesauro at IBM made a neural network program to play backgammon, called it Neurogammon. After training on data sets of expert games, it could assign weights to the pieces of knowledge. The program was good enough to win the 1989 Computer Olympiad. Tesauro’s next program used temporal difference learning, which means that instead of learning from games played by experts, it learns from self-played games. The program was called TD-Gammon (Temporal Difference-Gammon). TD-Gammon was one of the best backgammon players in the world.

13. Deep Blue is a machine programmed by IBM to play chess [10]. After six years of programming Deep Blue, the IBM team felt that they were ready to challenge the world champion - Gary Kasparov. In Game 1, 1996, Deep Blue started off with its first win. But Kasparov learned quickly. He won the match four to two and confidently proposed a rematch in 1997. Kasparov won the first
game in a breeze. But the next game, Kasparov said, “It played differently, more strongly, unlike a computer”. In the next three games, human and machine ended in a draw. Then finally, Deep Blue forced Kasparov into making a poor move. Kasparov then resigned. Deep Blue used Brute Force, but the search looked past the first few moves. It challenged Kasparov with 256 processors that could search about 200 million moves per second. Deep Blue analyzed possible outcomes of the game. Grandmasters coached the programmers at IBM to deepen Deep Blue’s book, its library about how to win. Kasparov cannot try to use the Brute Force approach. Instead, he learns what is important from experience, and relies on the human mind’s ability to recognize patterns.

Although Deep Blue was smart, it did no think in the same way that humans think. That is still many years and breakthroughs away. After the Deep Blue vs. Kasparov game IBM retired Deep Blue, and it never played again.

This led to a commonly held but early belief that chess programs had surpassed all human players. Several years later, the programs SHREDDER, FRITZ, and JUNIOR demonstrated performances on par with the best human players. In 2005, the program HYDRA convincingly defeated one of the strongest human players, Michael Adams, scoring five wins, zero losses, and one draw, providing a strong argument for the supremacy of chess programs.

The approach has not been successful for the game of Go, however, owing to the high branching factor and vast search space, and the fact that goals and sub-goals are very difficult to assess with heuristic evaluation.

2.2 SOLVING GAMES:

Saying that a game is solved usually indicates in general, that a property with regard to the outcome of the game has been determined. Even for two-player, zero-sum games with perfect information, at least three different types of definitions could be intended, which are ultra-weakly solved, weakly solved and strongly solved.
Ultra weakly solved game indicates that at the start of the game, it is known what the outcome of the game would be with optimal play by both sides. It is not necessarily known how either player can achieve the optimal outcome [16] [17].

Weakly solved game indicates that the player need to be able to achieve a draw, in every game played. It is not necessary for player to win against a non-optimally playing opponent, when player is given a winning opportunity [15].

Strongly solved game demands a strategy not just from the initial positions, but from all possible legal positions. Thus, against a non-optimally playing opponent, the strategy should be such that each mistake must be capitalized upon [15].

There also exists ordering among three definitions of solved games. Any strongly-solved game is also weakly solved, while a weakly-solved game is also ultra-weakly solved. When any program claims to have solved any game, it must be compared against a rated opponent. A program which has ultra-weakly solved a game does not guarantee being capable of playing the game at all. A program which has weakly solved a game will at least draw every match it plays. The definite performance level, i.e., ensuring that no single match is lost, can be declared as solved game.

2.2.1 Methods for Solving Games:
This section gives an overview of methods developed in the course of solving games. We describe them briefly, and refer the reader to relevant references for more details. Section 2.2.1.1 concentrates on brute-force methods, while in Section 2.2.2.2 the knowledge-based methods are discussed.

2.2.1.1 Brute-force Methods
Brute-force methods have been important tools helpful in solving games. Many solving programs use basic brute-force methods such as $\alpha$-$\beta$ and their enhancements in some way or another. Two methods which have their application especially in solving games are the construction of databases by retrograde analysis and enhanced transposition-table methods. They are briefly discussed below.
**Retrograde analysis:**

Retrograde analysis is a method in which for each position of some specific game or endgame the number of moves towards the best reachable goal is stored. For instance, in Chess, assuming perfect counterplay, the number of moves to be played by the stronger side up to mate or conversion is stored. Checkers databases sometimes only contain an indication of won, drawn, or lost per position. A database is constructed by starting in terminal positions and then working backwards [70]. Once database is constructed, perfect play is guaranteed: the stronger side chooses a move with the shortest distance-to-mate and the weaker side opts for moves with the longest distance-to-mate. Perfect play by a computer in a position which is game-theoretically drawn or lost does not guarantee the best performance against imperfect opponents, as was demonstrated by Jansen [71].

Nowadays the use of retrograde analysis is commonplace for the construction of endgame databases. It has deepened the understanding of such endgames considerably, and resulted in notions as max-to-mate, max-to-conversion, max-to-zeroing-move, and max-to-the- rule [51].

**Enhanced transposition-table methods:**

The traditional transposition tables used in game-playing programs normally exploit the DEEP replacement scheme, i.e., when two different positions compete for the same entry in the table, the old position is overwritten by the newer one provided that the latter is searched at least as deep as the former. Research on this and other replacement schemes showed that the DEEP scheme is not the most efficient one [72].

**2.2.1.2 Knowledge-based Methods**

Next to brute-force methods it is often beneficial to incorporate knowledge-based methods in game-solving programs. Their main advantage is that it provides an appropriate move ordering or selection in the search trees. Some of the methods are:
Threat-space search and $\lambda$-search:

Allis generalized the idea of threat-space search to a method called dependency-based search. Threat-space search investigates whether by a sequence of threats, to which the opponent at any time has only a limited set of replies, a win can be forced. Since the opponent effectively has no real choices, this search algorithm represents the application of single-agent search to two-player games. A recent successor of threat-space search, called $\lambda$-search, has been proposed by Thomsen [73]. This method uses null moves combined with different orders of threat sequences, called $\lambda$-trees. Thomsen introduces $\lambda_1$-moves, which threaten to end the game or reach a specified sub goal immediately, followed by $\lambda_2$-moves threatening a winning $\lambda_1$-sequence, and so on. The method behaves as a goal-directed searcher, with a favorable tree size relative to standard $\alpha$-$\beta$ trees. It can be combined with any search method to search the $\lambda$-trees. As a relevant example Thomsen mentions proof-number search. A combination of null moves and proof numbers seems a promising method for solving Go endgames.

Proof-number search:

Proof-number search [74] is a best-first search, in which the cost function used in deciding which node to expand next is given by the minimum number of nodes that have to be expanded to prove the goal. As such it is a successor of conspiracy-number search. Proof-number search is appropriate in cases where the goal is a well-defined predicate, such as proving a game to be a first-player win.

Depth-first proof-number search:

The Tsume-Shogi-solving program SEO is based on a newly-developed iterative deepening depth-first version of proof-number search, named PN* [75].
Pattern search:
Pattern search, introduced by Van Rijswijck [76], is a game-tree search enhancement and Allis’ threat-space search. It applies to games where immovable counters are placed on the board, and was developed for the game of Hex. The method is able to prove wins and losses while searching a tree smaller than the minimal proof tree of the game, by proving the result for several moves at once in lost positions.

Pattern search concentrates on finding threat patterns. A threat pattern is a set $\Psi$ of empty cells in a position $P$, with the property that the game-theoretic value of $P$ is unaltered when the winning side is restricted to playing moves within $\Psi$. A threat pattern is not unique for a position, since adding an empty cell to a threat pattern always creates another valid threat pattern. A threat pattern can be considered as representing the applicable area on the board, an area that human players commonly identify when analyzing a position. The patterns can be calculated recursively.

2.3 Research work on Go-Moku:

Many people have tried to apply artificial intelligence techniques in the field of game playing. Most of the people have tried to apply normal AI techniques such as mini-max and alpha beta cut off on Go-Moku board game. L. Victor Allis, in his Ph D thesis titled, “Searching for Solutions in Games and Artificial Intelligence”, applied db-search and PN-search methods on Go-Moku [18]. It uses PN search, DB search, database creation and database lookup; database is used as transposition table.

The rules of Go-Moku and Renju differ [118]. Even for the game of Go-Moku there are different versions. For instance, in free-style Go-Moku an over line of six-in-a-row wins, whereas in standard Go-Moku it does not win. Both versions have the property that the same rules hold for both players, i.e., symmetric ruling. In Renju, additional restrictions are imposed on the first player, making the game asymmetrical: when making an over line or a so-called double three or double four Black loses the game immediately, whereas White is allowed to make an over line, double three or double four.
Consequently, most professional players favored the game of Renju, in which several restrictions were forced on the first player. Still it was usually believed that Renju also is a first-player win. This assumption has been verified by Wágner and Virág [69], who weakly solved Renju by a search process that used transposition tables, threat-sequence search, and expert knowledge for no-threat moves. The program took advantage of an iterative-deepening search based on threat sequences up to 17 plies.

Marco Kunze and Sebastian Nowozin in their study “An AI for Gomoku/Wuziqi – and more...” also tried to use alpha-beta cut-off procedure in order to reduce the search area [19].

There is very little mention of research using genetic algorithm on Go-Moku.

2.4 RESEARCH WORK ON OTHELLO:

Following is the summary of research work carried out in the game Othello.

IAGO:

IAGO was developed in the early 1980s, and was the first remarkable program which played Othello at a non-trivial level. Its evaluation function was constructed in a similar way to that of pre-Samuel AIs in that it was entirely hand crafted, and had no self-learning abilities [12].

BILL:

BILL was the first program to beat IAGO—in fact, it did so in impressive fashion, beating IAGO in every match played, with only 20% of the thinking time [12]. Unlike IAGO, in addition to its evaluation function, BILL had a built-in learning function: it stored every game position it encountered, and assigned them as winning or losing, according to the outcome of the game. Using these, it learned to recognize patterns of winning and losing positions, and altered its evaluation function accordingly. One
drawback of using this method was that no non-expert games were used to train BILL, and therefore against a trainee player BILL did not have such an edge.

**Logistello:**

Logistello was released in 1994, and is currently one of the strongest Othello playing programs [12]. Whilst it is self-learning, it does not identify features which it considers to be important—these are still to be specified by its programmer. However, Logistello is able to decide how combinations of features are important—that is, unlike BILL, which treats every feature separately. Additionally, Logistello uses both an endgame database to ensure a perfect endgame, and an opening book to influence the chances into its favor from the beginning.

**Moriarty and Miikkulainen:**

Moriarty and Miikkulainen developed an Othello player in a similar manner to Anaconda [13]. No strategies or tactics were provided to the program. The program was unusual in the sense that it did not expand the game tree at all, beyond looking at the currently legal moves, and despite this, it independently developed an “advanced mobility strategy”, which Moriarty and Miikkulainen state was discovered only once, and not, like many strategies, discovered in several places independently. They go on to state that the rediscovery of this method was “a significant demonstration of the potential power of neuro-evolution” and that mobility strategies are notoriously difficult to learn, particularly as they involve counterintuitive aspects, such as minimizing your own piece count during the middle-game. This, however, restricts the legal moves available to the opponent, thus providing an advantage as they are forced to make bad moves.

It is stated in this paper that their population of neural networks was evolved initially against a random moving opponent, and then later against players afforded with alpha-beta search capability. The neural network population then evolved to exploit their initial material disadvantage and discovered the mobility strategy.
Michael Buro:

In a paper titled “Improving heuristic mini-max search by supervised learning“, the author showed the modification of min-max technique to play the game of Othello. The author proposed a new selective heuristic technique called as PROBCUT [14]. It uses a minimum window for a shallow search called $D'$ and $D$ as the depth of deeper search. The difference $D-D'$ is in proportion to the depth of the cut sub tree. On the other hand, if the difference is too large, the numbers of cuts will be reduced since the variance of the error will be also large. In the PROBCUT implementation in the Othello program LOGISTELLO, the author chooses $D'=4$ and $D=8$ evaluation pairs $(V_{D'}, V_D)$ generated by non-selective searches. After that the best $t$ is determined using a tournament between versions of the selective program with different cut thresholds and non-selective version. The PROBCUT extension can increase some game programs’ playing strengths considerably. In the Othello game LOGISTELLO, Buro reports that the PROBCUT enhanced version defeats the brute-force version with a winning percentage of 74%. This tournament uses 35 balanced opening positions as starting positions and all program versions are with quiescence search and iterative deepening.

The technique can prune probably inappropriate sub-trees with a prearranged confidence. The Tournament results indicate a substantial playing strength improvement compared to full-width $\alpha-\beta$ search.

Multi-PROBCUT generalizes the PROBCUT procedure to prune even more unpromising sub trees by using additional checks and cut thresholds. Multi-PROBCUT refines the PROBCUT procedure in three ways:

1. Multi-PROBCUT allows cutting irrelevant sub trees recursively at several heights instead of only at one specific height.
2. By performing several check searches of increasing depth, Multi-PROBCUT can detect extremely bad moves at very shallow check searches.
3. Multi-PROBCUT optimizes the cut thresholds separately for different game stages instead of using a constant cut threshold for the whole game.

Buro’s experiments in Othello game program show that Multi-PROBCUT outperforms PROBCUT. The winning percentage of the Multi-PROBCUT version of LOGISTELLO playing against the PROBCUT version was 72% in a tournament of 140 games of 30 minutes per move.