**Move Performance Optimisation for Large Search Space using Decision Tree Learning with the Minimax Algorithm and Alpha-Beta Pruning**

**Khong Wai Foo Andrew,** **Sun Yick William, Samuel Winterton, Kim Tae Hyung**School of Computer Science & Software Engineering  
The University of Western Australia  
35 Stirling Highway, Crawley WA 6009  
[khongw01@student.uwa.edu.au](mailto:khongw01@student.uwa.edu.au1), [suny05@student.uwa.edu.au](mailto:suny05@student.uwa.edu.au2), [wintes02@student.uwa.edu.au](mailto:wintes02@student.uwa.edu.au3), [kimt01@student.uwa.edu.au](mailto:kimt01@student.uwa.edu.au4)

**Abstract – The Minimax algorithm with alpha-beta pruning (or simply alpha-beta pruning) is commonly used in machine playing of two-player games such as Tic-tac-toe and chess.**(1)

**In this paper, we show that an agent using the alpha-beta pruning algorithm will not win the game or wins the game with longer time duration than it should take with a large moves space in the MixMeta4 environment. A decision tree learning technique will then be used to decide the move that will lean towards winning the game in the fastest possible way. Finally data gathered from the decision tree will be compared with just the alpha-beta pruning algorithm to determine whether there is an improvement in the agent’s play.**

**[NEED TO INCLUDE WHAT WE FOUND HERE]**

*Keywords:* Minimax, Alpha-Beta Pruning, MixMeta4, Decision Tree, Learning

**1 Introduction**

The Minimax algorithm with alpha-beta pruning or simply just alpha-beta pruning algorithm is more commonly used in machine playing games than the naïve Minimax algorithm, generally performing better by pruning away search paths and thus reducing the size of the search space.

In the MixMeta4 environment, an agent that utilises the alpha-beta pruning algorithm is expected to win a game against an agent that simply chooses random moves. However, when played against more intelligent agents such as Hal, its endgame performance is lacking, often producing moves backwards or away from the opposition, losing the game.

In this paper, we investigate the effect of allowing the agent to learn better moves from playing agents such as Hal, where the result probabilities are gathered using Decision Trees and stored for future games.

Therefore our hypothesis is that an Agent that uses the Decision Tree learning technique will make beneficial moves that will improve its game playing performance over an agent that simply uses only Alpha-Beta Pruning.

In section 2 we describe our method for capturing *good* moves using Decision Tree learning, and in section 3 show the effect of applying this *learnt knowledge* against agents such as Hal that have historically performed well against a Alpha-Beta Pruning agent.

Finally we discuss in a more general sense what implications our results have for not only game playing, but also machine learning.

**2 Method**

**2.1 Decision Tree Training Sets**

To effectively "teach" our Agent what the best move is given the current state of the environment, we asked it to "learn" decision trees, something known as Decision Tree Induction.

We started by building a training set; a database of Attributes and their associated Goals.

Each node within the tree is called an Attribute, and can be thought of as the input. Each resulting decision is called the Goal.

To maintain a feasible experiment we chose relatively simple, but pertinent Attributes to compute, such as if our Agent could take a piece or be taken by a piece.

The Goals were provided by a human "expert" or an Agent known to be better than Alpha-Beta pruning playing games. We captured the result of the Agent performing each combination of Attributes.

Originally we hoped to provide all Goals via human experts, but due to the volume of data, generated Goals from known better performing players were used.

Each training set was built from data captured from multiple games and stored within a training set directory.

**2.2 The jaDTi Library**

Once our training set was complete, we then fed this raw data into the jaDTi library (2) to produce our actual Decision Trees.

We used the jaDTi library in interest of time, and because we didn’t want to “reinvent the wheel.” Although we did actually first attempt to write a Decision Tree library ourselves, the jaDTi library proved to produce the results we were looking for.

To feed data into the library, we first had to write a utility to convert our training set of raw data files into a Java Database readable by the library.

Using jaDTi also allowed us to visualise the Decision Trees built as shown below.

[INSERT DECISION TREE FIGURE HERE]

Once the database was built, upon initialisation of our Agent we run jaDTi over the data, generating actual Decision Tree objects. At this stage the Agent has “learnt” the moves of the “expert.”

These Decision Trees are then used during game play to choose the best move to play when the Agent is presented with a similar set of Attributes during the game.

**2.3 Generalisation of Moves**

As it would be infeasible to capture data for all possible moves, and therefore build Decision Trees for every game state, our Agent had to fall back on a generalisation where it had insufficient “knowledge.”

As we only wanted to test the effect of Decision Tree learning, we generalise by falling back on just Alpha-Beta pruning.

**3 Results**

**4 Discussion  
4.1 Alpha-Beta Pruning  
4.2 Game Playing  
4.3 Decision Trees  
4.4 Machine Learning  
  
5 Conclusion**

**References**

1. Alpha-Beta pruning. *Wikipedia.* [Online] [Cited: 13 May 2009.] http://en.wikipedia.org/wiki/Alpha-beta\_pruning.

2. **François, Jean-Marc.** *jaDTi - Decision Trees: a Java implementation.* [Online] [Cited: 14 May 2009.] http://www.run.montefiore.ulg.ac.be/~francois/software/jaDTi/.

3. **Laramée, François Dominic.** Chess Programming Part VI: Evaluation Functions. *GameDev.* [Online] [Cited: 12 May 2009.] http://www.gamedev.net/reference/articles/article1208.asp.