Heuristic Move Pruning in Monte Carlo Tree Search for the Strategic Card Game Lords of War

Nick Sephton*, Peter I. Cowling*, Edward Powley†, and Nicholas H. Slaven‡
*York Centre for Complex Systems Analysis, Department of Computer Science, University of York, United Kingdom
Email: njs523@york.ac.uk, peter.cowling@york.ac.uk
†Orange Helicopter Games, York, United Kingdom.
Email: ed@orangehelicopter.com
‡Stainless Games, Isle of Wight, United Kingdom
Email: nick@stainlessgames.com

***Abstract*—Move pruning is a technique used in game tree search which incorporates heuristic knowledge to reduce the number of moves under consideration from a particular game state. This paper investigates Heuristic Move Pruning on the strategic card game *Lords of War*. We use heuristics to guide our pruning and experiment with different techniques of applying pruning and their relative effectiveness. We also present a technique of artificially rolling forward a game state in an attempt to more accurately determine which moves are appropriate to prune from the decision tree. We demonstrate that heuristic move pruning is effective in Lords of War, and also that artificially rolling forward the game state can increase the effectiveness of heuristic move pruning.**

I. Introduction

Monte Carlo Tree Search (MCTS) is a highly effective tree search technique which originated in 2006 [1], [2], [3]. It has seen success in many areas, and has seen specific success in contributing towards a strong player for Go, which was previously a very challenging target. MCTS is a tree search technique which uses guided forward simulations of a game to estimate the value of potential moves. Many enhancements to MCTS have been proposed, with varying degrees of effectiveness [4].

A number of these enhancements focus on *pruning* the MCTS tree, to remove sections of the tree from consideration. If poor sections of the tree are removed from consideration, the following tree search should converge more quickly upon a strong solution. Some methods of pruning also look to remove *trap states* [5], which are states that appear strong to a search agent, but are actually guaranteed losses, or simply very weak. Pruning is split into *Hard Pruning*, which permanently removes sections of the tree, and *Soft Pruning*, which temporarily removes sections of the tree, but then adds them back into the search at some later point.

In this paper we use heuristic hard pruning to reduce the branching factor of the search tree and demonstrate that this produces stronger play. We then combine heuristics to produce multi-heuristic agents which are played-off against the single heuristic agents to determine their relative playing strengths.

Finally, we investigate pruning using *State-Extrapolation*. For the initial heuristic pruning tests, we prune moves based on a heuristic evaluation of the game state after the move is made. When pruning a move using state-extrapolation, we move the game forward until just before the next opponent decision, then determine the suitability of the move by the heuristic evaluation of that state. This applies both in our chosen domain, Lords of War, and in many other domains where a player move consists of a series of linked decisions before the opponent has the opportunity to react. We show that this technique improves the strength of the search by allowing the heuristics to evaluate a more representative game state. We then compare State-Extrapolation across a selection of the strongest heuristics we have used, and examine its comparative effect on play strength for those agents.

The remainder of this paper is organised as follows. In section 2, we present a summary of related works on MCTS and associated move pruning techniques. Section 3 discusses both the game we selected for experimentation and the heuristic knowledge we have chosen to incorporate in our move pruning techniques. In section 4 we discuss our experimental methods and the exact trials that we performed to test our pruning techniques. Section 5 presents our results, and section 6 our conclusions and some notes on potential future work.

II. Related Work

A. Monte Carlo Tree Search (MCTS)

An early adaptation of more traditional tree search technologies, *Monte Carlo Tree Search (MCTS)* samples playouts to improve the specificity of the tree search and remove the need for knowledge about the game. While heuristic knowledge can greatly improve the performance of MCTS, regular operation without this knowledge can locate effective move decisions within the search space.

Since its inception in 2006 [1], [2], [3] a great deal of further research has continued on MCTS, and it has seen much success across many fields, specifically in the highly challenging game Go [6], which has traditionally proved very difficult for techniques such as minimax search.

MCTS is an incremental process which constructs a search tree representing the section of the decision space containing strong decisions. By running many thousands of playouts of the game from the root state and collating the rewards from terminal game states, the collected statistics provide a strong indication of the location of strong solutions. One of the strengths of MCTS is that the search tree grows in a

Algorithm 1 Basic MCTS Process Summary

```
function TREESEARCH(s_0)
    n_0 = new TreeNode(s_0)
    while t_i < t_max do
        n_1 ← treePolicy(n_0)
        r_1 ← defaultPolicy(n_1.s)
        backup(n_1, r_1)
    return bestChild(n_0).a
```

asymmetrical manner, using a tree policy which balances the effect from exploiting lines of play which are indicated to be strong versus exploring new areas of the decision space.

The basic MCTS algorithm is made up the steps below:

- Selection: On each iteration, the algorithm moves through the tree guided by the *tree policy* until it reaches a node which has unexpanded children or a terminal node.
- Expansion: If the selected node has unexplored moves, then one (or more) nodes are added to the tree to represent these moves.
- Simulation: The *default policy* guides a simulation of the game until a terminal state.
- Back-propagation: The simulation result is backed up through the ancestor nodes of the selected node, updating statistics until it reaches the root node.

B. Upper Confidence Bound applied to Trees (UCT)

The term *Upper Confidence Bound applied to Trees (UCT)* describes the use of MCTS with a default policy of random selection, and a specific tree policy named *UCB1*. UCB1 treats the choice of a child node as a multi-armed bandit problem [3], [7], and selects the child node that has the best expected reward as approximated by Monte Carlo simulations.

When the tree policy is required to determine which child node should be examined for expansion and simulation, it uses the UCB1 equation (equation 1) to make this determination. In equation 1, $\bar{X}_i$ is the average reward from currently examined node (i), C is the *Exploration Bias*, n is the number of visits to the parent node of i, and n_i is the number of visits to i.

$$UCB1 = \bar{X}_i + C\sqrt{\frac{2\ln n}{n_i}} \tag{1}$$

UCB1 balances exploration of new lines of play against exploitation of existing strong plays by scaling the number of visits to a given node against the rewards from that play through.

Kocsis and Szepesvári [3], [7] showed optimality of UCT in converging upon an optimal decision when given sufficient iterations. UCT will frequently find good move decisions even with a rather modest number of iterations.

C. Move Pruning

Move Pruning describes the process by which a number of branches of a game tree are removed from consideration (hard pruning) or are de-prioritised from consideration, but may be searched later (soft pruning). Move pruning has been shown to be a powerful approach when applied with traditional minimax techniques [8], and has shown some strength when applied with MCTS [9]. Heuristic pruning has been shown to be effective in a wide range of games, most notably in MOHEX [10], a 2009 world champion Hex player.

A very well-known pruning approach is *Alpha-beta Pruning* [11] is an enhancement to minimax search, which has long been the algorithm of choice for playing combinatorial games [12]. The process of Alpha-beta Pruning examines each state in the tree using a heuristic, and α (the minimum score that the maximising player can obtain) and β (the maximum score that the minimising player can obtain) are evaluated. If at any point β becomes smaller than α, the branch is "pruned", as it cannot be a branch of optimal play, and need not be explored. Alpha-beta pruning is not a heuristic technique, however application in combination with minimax search is recognised as near universal. Although alpha-beta pruning does not itself rely on heuristics, its efficiency (i.e. how much of the tree it prunes) depends on the order in which the moves are searched, and the heuristic knowledge is often applied to choose a good move ordering.

Progressive Unpruning [13] describes a process by which child nodes are added as normal to any node p in the MCTS tree until the number of visits n_p to a node equals a predefined threshold T. At this point, a large number of child nodes are pruned, and with each further game that plays through p, these moves are slowly "unpruned" (made re-available for selection & simulation). Progressive unpruning has been shown to be very effective, particularly when combined with Progressive Bias [13]. Progressive unpruning is very similar to an independently proposed scheme called *Progressive Widening* by Coulom [14]. These two techniques offer an advantage over standard hard pruning as they immediately provide the benefit of hard pruning, but then allow for the possibility that strong moves may have been accidentally pruned by "unpruning" moves later in the search. Given enough budget, the entire tree will eventually be considered by progressing unpruning.

Another strategy often used for implementing heuristic knowledge is *Progressive Bias* [13], which modifies the tree policy and therefore guides node selection. The guidance provided by the heuristic knowledge is slowly reduced as more iterations pass through the tree, providing strong guidance initially, and in the limit no heuristic guidance.

III. The Lords of War Game

A. Lords of War

Lords Of War[1,2] is a two-player strategic card game by publisher Black Box Games. A board is used for card placement, and the relative positions of the cards on the board are the main strategic interest. A player wins when they eliminate

[1] http://lords-of-war.com/

[2] http://boardgamegeek.com/boardgame/135215/lords-of-war-orcs-versus-dwarves

twenty of an opponent's cards, or they eliminate four of their opponent's *Command Cards*. Command cards are significantly more powerful than other cards, but placing them onto the board carries a risk that they may be eliminated.

The game board is 7×6 squares each of which can hold a single card. Cards have between 0 and 8 attacks, each with a strength value, and a directionality towards an orthogonal or diagonally adjacent square. Attacks from multiple cards can be combined to eliminate an opponent's card with a high defence value. Some cards also have ranged attacks which can eliminate (or contribute towards the elimination) of opponent's cards which are not adjacent. In regular play, cards can only be placed so as to attack enemy cards, however *Support Cards* also have additional placement rules allowing them to be placed next to friendly cards instead of attacking enemy cards.

On each player's turn, they are required to place exactly one card, then process combat to identify and remove eliminated cards, then they have a choice of either drawing a new card from their deck, or retreating a friendly unthreatened card from the board. The official Lords of War rulebook and a variety of other resources are available on the Lords of War website[3].

A normal game rarely extends beyond 50 turns, as most moves (particularly strong moves) result in a capture. Once an average human player has made 25 moves, they have probably captured more than 20 cards, and thus the game would have completed. Of course the games can end much sooner if command cards are placed carelessly or last much longer if players play cautiously. Games with MCTS agents last on average between 30 and 60 turns, depending on the nature of the agent.

Our experience with Lords of War has revealed that it commonly has a mid-game branching factor of 25-50, making move selection challenging. Previous work on Lords of War has studied the impact of parallelization on MCTS [15].

B. Heuristic Knowledge

We applied our own experience of Lords of War in the creation of several functions which may be powerful for examining a state. These functions (f_j) are applied to a specific game state S_i such that $f_i : S \rightarrow \mathbb{R}$, and are intended to form building blocks for construction of heuristics which will be used to measure fitness of a state (correlated to the probability that the assessing player will win from that state). Each function is performed including only cards of the active player unless the otherwise specified by the modifier *opp*, in which case the opponent's cards are considered instead (e.g. f_j^{opp}). The set B_i is the set of all the active player's cards on the board in state S_i (or the opponent's cards if *opp* is used). The set H_i, E_i and D_i are similarly the sets of cards in the players hand, eliminated pile and deck respectively. The following functions were used to simplify the expressions for the heuristics:

$$f_1(S_i) = |B_i|$$

$$f_2(S_i) = |E_i|$$

$$f_3(S_i) = \sum_{b \in B_i} (b.DefenceValue)$$

$$f_4(S_i) = |\{b \in B_i | b.Threat() > 0\}|$$

$$f_5(S_i) = |\{g \in E_i | g.IsCommand()\}|$$

$$f_6(S_i) = \sum_{b \in B_i} m_a(b)$$

$$f_7(S_i) = \sum_{b \in B_i} m_b(b)$$

$$f_8(S_i) = \sum_{b \in B_i} m_c(b)$$

$$f_9(S_i) = \sum_{b \in B_i} m_d(b)$$

$$f_{10}(S_i) = \sum_{b \in B_i} m_e(b)$$

$$f_{11}(S_i) = \sum_{b \in B_i} m_f(b)$$

To briefly explain these functions, $f_1(S_i)$ counts all the active player's cards, $f_2(S_i)$ counts all the active player's dead cards, $f_3(S_i)$ sums all defence values for all the active player's cards, $f_4(S_i)$ counts all squares threatened by the active player's cards, and $f_5(S_i)$ counts all the active player's dead commander cards. Functions $f_6(S_i) - f_{11}(S_i)$ refer use the heatmaps in Figure 1 to assign values to the active player's cards based on their position, and then sums those scores. The functions were then used to create the *State Evaluation Functions* listed below:

a) Simple Card Count (h_1): This heuristic was selected for testing partially because it was the simplest of the heuristics, but also because it appeared to be a very strong contender. h_1 assigns a weight of +1 for each card on the board and a weight of -1 for each card in the graveyard, negating these weights for opponent cards.

$$h_1(S_i) = (f_1(S_i) - f_2(S_i)) - (f_1^{opp}(S_i) - f_2^{opp}(S_i))$$

b) Average Defence (h_2): This heuristic was selected for testing because it would appear that strong players often play defensively in Lords of War, and this heuristic would hopefully mimic that style of play. This heuristic measures the difference between player and opponent of the mean defence value of cards on the board. In the case when a player has no cards on the board, we assume a value of 0 for that player's contribution to the value of h_2.

$$h_2(S_i) = (f_3(S_i)/|B_i|) - (f_3^{opp}(S_i)/|B_i^{opp}|)$$

c) Threatened Area (h_3): This heuristic counts the number of empty or opponent occupied squares on the board that are directly threatened (under attack by adjacent card's

[3]http://www.lords-of-war.com/

non-ranged attacks) by active player cards. The same calculation is made for the opponent and subtracted from the total. This heuristic was selected so as to consider the positional elements of the game.

$$h_3(S_i) = f_4(S_i) - f_4^{opp}(S_i)$$

d) Simple Card Count with Dead Commander adjustment (h_4): This heuristic is similar to h_1, except command cards in the dead pile count for two cards instead of one. The adjustment to h_1 is due to our own play experience and understanding of the importance of command cards to the game (as it is possible that an AI may be too willing to lose its first 2-3 command cards in combat).

$$h_4(S_i) = (f_1(S_i) - f_2(S_i) - f_5(S_i)) - (f_1^{opp}(S_i) - f_2^{opp}(S_i) - f_5^{opp}(S_i))$$

e) Active Player Average Defence (h_5): This heuristic was a modification upon h_2 to remove the subtraction of an opponent score from the total. This was tested as a heuristic mainly because h_2 seemed like such a strong candidate, yet performed so weakly in tests.

$$h_5(S_i) = (f_3(S_i)/|B_i|)$$

f) Basic Heat Maps ($h_6 - h_{11}$): This set of heuristics is similar to h_1, except each card is assigned different values depending on its placement location, then these values are summed to create the state score. When the modifier "*opp*" is used, the heat maps are reflected about the horizontal axis to account for the opponent playing from the opposite side of the table (this is only of significance to m_a and m_b). The maps for these heuristics are shown in Figure 1. We would expect these heuristics to be poor when used in isolation, but perhaps stronger when combined with another heuristic which measures strategic strength, such as h_1 or h_5.

$$h_6(S_i) = f_6(S_i)$$
$$h_7(S_i) = f_7(S_i)$$
$$h_8(S_i) = f_8(S_i)$$
$$h_9(S_i) = f_9(S_i)$$
$$h_{10}(S_i) = f_{10}(S_i)$$
$$h_{11}(S_i) = f_{11}(S_i)$$

During pruning, we apply the appropriate heuristic (h_i) to the state that results from applying the move under examination to the current state. We then prune all except the top scoring moves. The number of moves that each heuristic selects is referred to as the *Hard Pruning Limit (HPL)*.

IV. Solution Methods

A. Single Heuristic Experimentation

During our experiments, values ranging from 1 to 35 were used for HPL. Each of the heuristics were run against Plain UCT using 10000 iterations. The following experiments were each repeated 500 times in each case, where UCT is plain UCT, and UCT(h_i[n]) is UCT using h_i for hard pruning with a HPL of n, and the value of i runs from 1 to 11 in each case.

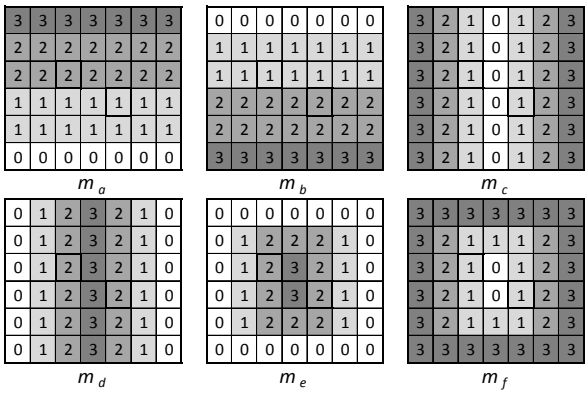

Fig. 1. Heat Maps

- UCT vs UCT(h_i[1])
- UCT vs UCT(h_i[2])
- UCT vs UCT(h_i[5])
- UCT vs UCT(h_i[10])
- UCT vs UCT(h_i[15])
- UCT vs UCT(h_i[20])
- UCT vs UCT(h_i[25])
- UCT vs UCT(h_i[30])

Due to the lack of success of h_2, an experiment with a negated value was attempted as well, but it was completely unsuccessful, winning 0 games in all tests. The clear trends shown in the results for the improvement of the results with an increase in HPL prompted further experimentation with higher pruning limits, so further experiments were run increasing the pruning limit until the Win% exhibited a decrease. Results of this experiment are given in section V-B.

B. Multi-Heuristic Experimentation

In later tests, multiple heuristics were used in combination. An AI using multiple heuristics would fulfil the HPL from each heuristic in turn, then combine the obtained results, removing any duplicate entries. This results in a list of top moves which numbers between HPL and $(n \times HPL)$, where n is the number of heuristics being used in the agent.

Our strongest single heuristic (h_4), was combined with each of the other heuristics in an attempt to create a strong multi-heuristic agent. These new agents were then played against the original h_4 to determine their strength against our strongest single heuristic agent. Results of this experiment are given in section V-C.

C. State-Extrapolation

In all previous experiments, we have pruned moves based on the score obtained from the state following that move. In these State-Extrapolation experiments, we artificially roll forward the game state to some forward point in the game

and prune based on the state at that point. This has the effect of running through the combat step (when ranged attacks are assigned, and dead cards are removed from the board), and thus should provide a better estimation of the strength of the move.

There are multiple ways in which we can roll forward. The simplest way is to randomly select moves until some point in the future, most logically the opponent's next move. We can also look to perform a searches over the sub-tree from the remainder of the turn. In this paper, we experiment with randomly rolling the state forward until the opponent's next move.

We expect State-Extrapolation to be a strong technique, as it should give a more accurate representation of the actual game state. For example, rolling forward past the end of combat step would allow us to observe clearly which cards will be removed, and thus the actual layout of the game board.

Strength of play in Lords of War is closely tied to positional elements of card placement. As such, the deployment move is the most strategically important move. The selection of whether to remove a card from the battlefield or draw a new card is comparatively simple, however it is occasionally complex. The simplest choice is that of selecting a target for a ranged attack. The target is normally obvious, as the situation in which more than one destroyable target is available is uncommon, and when there is an available target which can be destroyed with a ranged attack, performing that ranged attack is almost always the stronger decision. If there is no such target to destroy with a ranged attack, then the move selected is irrelevant and there is little point wasting time selecting a move.

V. Results

A. Game Engine

Lords of War and our experimental MCTS engine were implemented in C++. All experiments were run on a cluster of seven PCs, of various specifications. The full game of Lords of War features hidden information, as players draw a private hand of cards from a shuffled deck. We ignore this aspect of the game in these experiments, assuming that both hands and decks are fully visible to both players. The game is still highly playable by human players in this form, and plays rather similarly to the normal hidden information game.

B. Single Heuristic Results

The results for the initial heuristic tests are shown in Figure 2, 3 & 4.

A Hard Pruning Limit of below 5 seems poor for all heuristics tested, with all such agents consistently losing to Plain UCT. If a heuristic was a poor indicator of move strength, we would expect to see a slow and roughly linear increase in strength as the HPL rises, which should come to a halt at approximately 50% win rate. This is due to a high HPL being equivalent to using no heuristic at all, as no moves will be pruned by either method. We can see this behaviour in the agent using h_3, and all the heat map heuristics. This is consistent with our belief that the heat maps in isolation would be poor pruning heuristics.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}
1	6.0%	0.0%	2.4%	11.6%	6.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
2	22.0%	0.0%	7.8%	24.8%	15.8%	0.8%	2.2%	0.6%	1.2%	3.2%	2.8%
5	58.0%	0.0%	18.8%	60.8%	50.0%	3.8%	3.8%	4.0%	3.4%	0.0%	0.0%
10	77.6%	0.0%	35.6%	82.6%	77.6%	13.8%	14.4%	17.0%	17.0%	6.8%	11.2%
15	86.8%	0.2%	47.8%	90.0%	81.4%	30.8%	25.8%	29.6%	28.8%	14.2%	16.6%
20	85.4%	0.0%	47.6%	88.6%	83.6%	25.6%	31.8%	28.6%	33.6%	29.0%	33.4%
25	80.0%	0.0%	58.6%	83.2%	83.6%	39.6%	41.0%	35.4%	34.2%	44.2%	27.8%
30	79.4%	0.8%	56.0%	79.0%	82.8%	38.0%	38.8%	42.0%	40.4%	50.0%	41.6%

Fig. 2. Win% of single heuristic agents vs Plain UCT at varying HPL

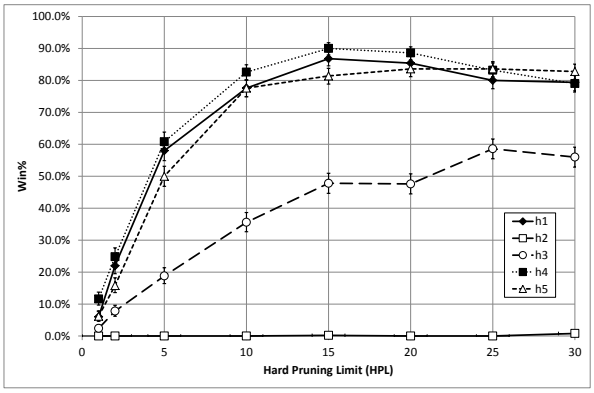

Fig. 3. h_1 - h_5 vs. Plain UCT player

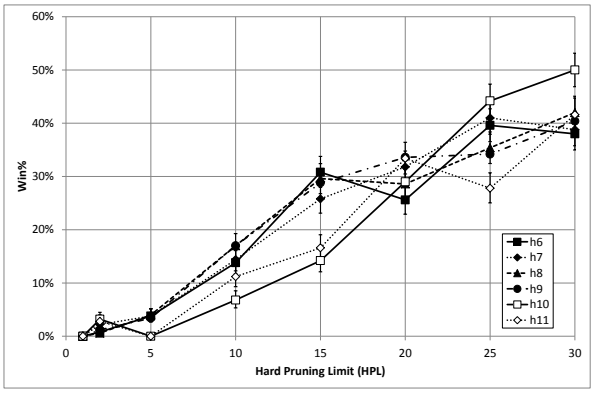

Fig. 4. h_6 - h_{11} vs. Plain UCT player

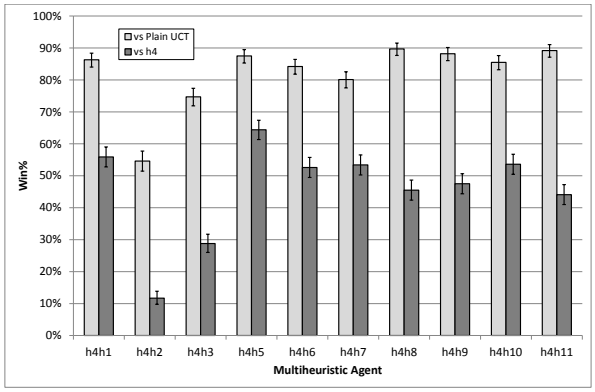

Fig. 5. Win% of multi-heuristic agents (HPL 15)

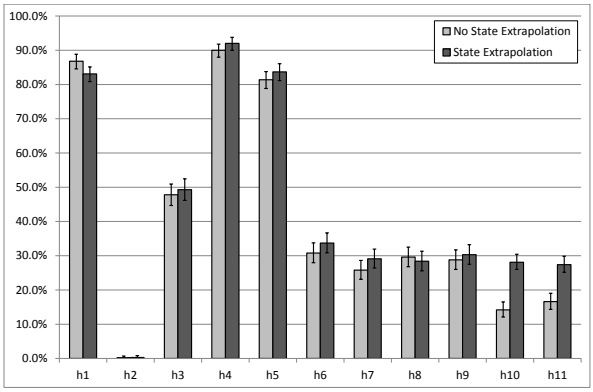

Fig. 6. Effects of applying State-extrapolation to heuristic agents

The strongest single heuristic results appear between a HPL of somewhere between 15 and 25, with the strongest individual result being h_4 at a HPL of 15. h_1 and h_4 are very similar heuristics. However as h_4 performs better, we can see that including an adjustment for command cards within the heuristic has increased its effectiveness. This could be considered for future heuristics, and may increase their play strength.

The most effective heuristics seem to be h_1 and h_4, with h_5 being a strong third. It is surprising that h_2 performed so poorly given that h_5 performed so well. This is possibly due to h_2 being highly susceptible to strong play from an opponent, or possibly that the state with no cards on the board is stronger than the heuristic indicates (for example, having no cards on the board while your opponent only has one is not as poor a position as h_2 would indicate, since it means that you have a target for attack where your opponent has none.)

C. Multi-heuristic Results

The results of the experimentation with multi-heuristic agents are displayed in Figure 5. The combination of h_4 with any of the heat map heuristics causes a strong improvement in performance, and while none clearly exceed the original performance of h_4 against plain UCT, they perform at about the same level. This is likely due to the moves being selected by h_4 being responsible for most of the strong decisions. When each of these agents are played against h_4, the agent h_4h_5 performs the best, suggesting that h_5 is contributing towards the success of h_4. Of the multi-heuristic agents using heat maps, the agents using h_4h_6, h_4h_7 and h_4h_{10} show the best performance. This confirms our experience that playing near the front or back of the board is strong, but suggests that playing in the centre of the board is stronger than playing at the edges. This may indicate that controlling the centre of the board is more important than the benefit of playing your cards against the edge in order to protect their weaker sides. This difference in performance may also be due to human players trying to place blank card sides (sides with no attack value) against the board edges, where as no such consideration is included in the agents.

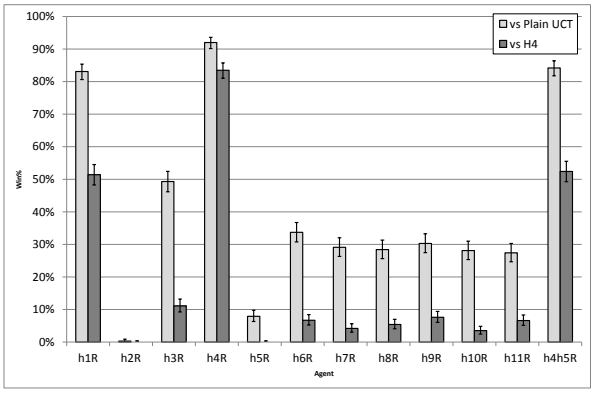

Fig. 7. Win% of agents using state-extrapolation

D. State Extrapolation

We can see from comparison of the original agents versus those using state-extrapolation that there is little difference in win% in most cases (see Figure 6). However the difference in two specific cases is significant, those of agents using h_{10} and h_{11}, and for all but one heuristic state extrapolation gives slightly stronger results.

Heuristics h_{10} and h_{11} use heat maps which are exact opposites of each other (see Figure 1). Our experience of Lords of War is that the strength of a move can be closely associated to proximity to a board edge. The difference in effect upon these two heat maps and the other heat maps ($h_6 - h_9$) can likely be attributed to this difference.

Figure 7 shows us that using state extrapolation has strengthened the h_4R agent (where R denotes the use of state extrapolation), displaying a win rate of approximately 80% against our previous strongest single heuristic agent, h_4. It is also worth noting that h_1R wins approximately 50% of games against h_4.

VI. Conclusions & Future Work

A. Conclusions

In this paper, we experimented with heuristics in two general categories; heuristics that drew statistics from the cards in the game, and heuristics that used heat maps to prioritise card placement in specific positions. Overall the first category proved the most effective, particularly the simplest heuristics that merely totalled number of cards. The heat map heuristics were generally ineffective, however they did show the largest relative improvement from state-extrapolation.

While state-extrapolation did have an effect upon playing strength in some cases, it was only effective in improving agent strength in certain cases, most notably h_{10} & h_{11}. This is possibly due to the fact that placing cards around the edges and/or the centre is strategically important (as our own experience would suggest), however the strategic impact is not always immediately apparent from the game state halfway through a player's move decisions. As discussed earlier, the heat maps alone were not expected to create strong agents, and the application of state extrapolation to h_{10} & h_{11} may have revealed that placing around the edges or centre is a strong move, and thus that these two maps are actually superior to the other heat maps.

B. Future Work

It would be of interest to look at other methods of performing state extrapolation, more specifically other methods of searching the sub-tree that is traversed before the state is analysed. In other games where this sub-tree is not as simple, more advanced techniques may be appropriate to ensure reasonable decisions are being made.

We would expect heuristics which considered availability of squares, specifically those around the edges of the board, would be good candidates for creating a strong agent, and it would be of interest to explore such heuristics in a future paper. The possibility of evolving heat maps rather than designing them by hand would also be of interest [16].

It would be of interest to investigate the manner in which moves are selected by heuristics, particularly in multi-heuristic agents. Perhaps a move could be prioritised if it was selected by multiple heuristics, or perhaps moves that are only selected by a single heuristic could be soft-pruned until later stages of the search. Also, examining the total number of moves returned by multi-heuristic agents (and the difference from the maximum of $n \times HPL$) could be interesting.

The Application of progressive techniques to heuristic agents in Lords of War would also be of interest, as it is entirely possible that the success of certain agents is being limited by regular exclusion of promising moves, which would be otherwise reintroduced at a later point in the search by a progressive technique.

Acknowledgements

The work displayed here was supported by EPSRC (http://www.epsrc.ac.uk/), the LSCITS program at the University of York (http://lscits.cs.bris.ac.uk/), and Stainless Games Ltd (http://www.stainlessgames.com/).

We thank Black Box Games for their support in working with their game Lords of War.

References

[1] G. M. J.-B. Chaslot, J.-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik, "Monte-Carlo Strategies for Computer Go," in *Proc. BeNeLux Conf. Artif. Intell.*, Namur, Belgium, 2006, pp. 83–91. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.8924\&rep=rep1\&type=pdf

[2] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Proc. 5th Int. Conf. Comput. and Games, LNCS 4630*, Turin, Italy, 2007, pp. 72–83. [Online]. Available: http://portal.acm.org/citation.cfm?id=1777826.1777833

[3] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *Euro. Conf. Mach. Learn.*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Germany: Springer, 2006, pp. 282–293. [Online]. Available: http://www.springerlink.com/index/d232253353517276.pdf

[4] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6145622

[5] R. Ramanujan, A. Sabharwal, and B. Selman, "On Adversarial Search Spaces and Sampling-Based Planning," in *Proc. 20th Int. Conf. Automat. Plan. Sched.*, Toronto, Canada, 2010, pp. 242–245.

[6] S. Gelly and Y. Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," in *Proc. Adv. Neur. Inform. Process. Syst.*, Vancouver, Canada, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.7504\&rep=rep1\&type=pdf

[7] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo Search," Univ. Tartu, Estonia, Tech. Rep. 1, 2006. [Online]. Available: http://www.sztaki.hu/~szcsaba/papers/cg06-ext.pdf

[8] B. Bouzy, "Move Pruning Techniques for Monte-Carlo Go," in *Proc. Adv. Comput. Games, LNCS 4250*, Taipei, Taiwan, 2005, pp. 104–119. [Online]. Available: http://www.springerlink.com/index/q5351066r8h62285.pdf

[9] J. A. M. Nijssen and M. H. M. Winands, "Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 4, pp. 282–294, Dec. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6266709

[10] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 251–258, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5551182

[11] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 2009. [Online]. Available: http://books.google.com/books?hl=en\&lr=\&id=8jZBksh-bUMC\&pgis=1

[12] R. Ramanujan, A. Sabharwal, and B. Selman, "On the Behavior of UCT in Synthetic Search Spaces," in *Proc. 21st Int. Conf. Automat. Plan. Sched.*, Freiburg, Germany, 2011. [Online]. Available: http://www.informatik.uni-freiburg.de/~icaps11/proceedings/mcts/ramanujan-et-al.pdf

[13] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Math. Nat. Comput.*, vol. 4, no. 3, pp. 343–357, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.9239\&rep=rep1\&type=pdf

[14] R. Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go," *Int. Comp. Games Assoc. J.*, vol. 30, no. 4, pp. 198–208, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.5614\&rep=rep1\&type=pdf

[15] N. Sephton, P. I. Cowling, E. J. Powley, D. Whitehouse, and N. H. Slaven, "Parallelization of Information Set Monte Carlo Tree Search," in *IEEE Congress on Evolutionary Computation (to appear)*, 2014.

[16] D. Robles, P. Rohlfshagen, and S. M. Lucas, "Learning Non-Random Moves for Playing Othello: Improving Monte Carlo Tree Search," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, South Korea, 2011, pp. 305–312.