Space and Depth-related Enhancements of the History-ADS Strategy in Game Playing

Spencer Polk
School of Computer Science
Carleton University
Ottawa, Canada, K1S 5B6
Email: andrewpolk@cmail.carleton.ca

B. John Oommen
School of Computer Science
Carleton University
Ottawa, Canada, K1S 5B6
Email: oommen@scs.carleton.ca

***Abstract*—In the field of game playing, it is a well-known fact that powerful strategies, such as alpha-beta search, benefit strongly from proper *move ordering*. A popular metric of achieving this is the so-called "move history", that is, prioritizing moves that have performed well, earlier in the search. The literature reports a number of techniques, such as the Killer Moves and History heuristics, that employ such a philosophy. Inspired by techniques from the field of Adaptive Data Structures (ADSs), we[1] have previously introduced the History-ADS heuristic, which uses an adaptive list to record moves, and to improve move ordering based on move history. The History-ADS heuristic has been proven to produce substantial gains in tree pruning in a wide variety of cases. However, it made use of a relatively naive application of an unbounded, single adaptive list. In this work, we attempt to refine the History-ADS heuristic, by examining its performance by constraining the length of its adaptive list, and applying multiple ADSs for each level of the tree. Our results show that the vast majority of the savings from the History-ADS heuristic remain even with a very short list, which can be applied to mitigate the drawbacks of an unbound data structure. Although results for multiple ADSs did not outperform single ADSs, we show that they provide some insight into how similar techniques may be applied in the context of the History-ADS heuristic.**

***Keywords*—game playing; adaptive data structures; move ordering**

I. Introduction

The problem of achieving intelligent game playing against a human opponent is a particularly well-known and studied area, spanning decades of research [1], [2]. The extensive body of research dedicated towards this problem has led to the development of powerful techniques, such as the alpha-beta search, capable of achieving strong play in a wide variety of games, and is still widely employed today [1], [3], [4]. However, it is known that alpha-beta search performs best when moves are arranged from best to worst, achieving a much more efficient search [5]. This has led to the development of move ordering strategies, which attempt to approximate the best orderings available, such as the Killer Moves or History heuristics [5], [6], [7].

We previously introduced techniques to achieve move ordering, by employing techniques from the formerly unrelated field of *Adaptive Data Structures* (ADSs) [8], [9]. The field of ADSs attempts to address the problem of uneven query frequencies within a data structure, by developing techniques through which the ADSs' internal structure can converge to most efficiently handle queries over time [10], [11], [12]. By necessity, these methods must be highly efficient and incur very little cost, or losses due to overhead would cancel out gains in query efficiency, and the field of ADSs has developed a number of techniques to achieve this, such as the Move-to-Front and Transposition rules for adaptive lists [10]. In the research we embarked on, we considered if these efficient ranking mechanisms could be employed within the context of game trees.

Our previous investigations into the application of ADSs to game playing have led to the development of two families of heuristics. The older of these is the Threat-ADS, which employs ADSs to rank opponents in a multi-player game [8], and more recently, the History-ADS, which applies these strategies to rank moves in terms of their prior behaviours in the search [9]. However, the first demonstration of the History-ADS heuristic employed a relatively naive structure, and it is worthwhile to consider if this data structure can be refined or improved in some way, to improve performance and further investigate its qualities [9]. This paper concerns itself with investigating these potential refinements.

The remainder of the paper is laid out as follows. Section II discusses our previous work on applying the concepts of ADSs into game playing, and briefly describes the Threat-ADS and History-ADS heuristics. Section III describes the open questions from [9] that inspire our experiments presented in this work. Section IV describes our experiments, and Section V presents our results. Section VI holds our discussion and interpretation of these results. Section VII concludes the paper.

II. Previous Work

Techniques from the field of ADSs were designed with the intention of converging to a structure matching access frequencies [10], [11]. However, in doing so, they also provide a natural ranking of objects within the data structure, based on their relative access frequencies. By placing meaningful objects within an ADS, and querying them when an event occurs, we can employ the properties of the ADS to provide an efficient, dynamic ranking mechanism for those objects. Within the context of game playing, this could include opponents, possible moves, board positions, and/or many other possibilities. Our previous contributions, briefly explained below,

[1]The second author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway.

were concerned with applying this ranking strategy to enhance move ordering in a game-independent manner, and have led to substantial gains in pruning efficiency.

A. The Threat-ADS Heuristic

Our first application of ADSs to game playing was the Threat-ADS heuristic, which employs a very small ADS that contains opponents, and is queried when one is found to be in a threatening position [8]. Naturally, the Threat-ADS heuristic is only applicable to multi-player games, and is based on the recent multi-player game playing algorithm, the Best-Reply Search (BRS) [7]. The BRS operates by simplifying an N-player game to a 2-player game, by grouping adversaries together into a single "super-opponent". It then performs a normal alpha-beta search, considering all moves of this "super-opponent" at each MIN node of the game tree, grouping opponent moves together. Naturally, this means that it will consider illegal game states in any game where each player takes a turn in order (the norm for N-player games), however its deeper search allows it to outperform competing algorithms in spite of this [7].

The Threat-ADS heuristic augments the BRS with an adaptive list holding each opponent, and, at each MIN node, gathering opponent moves in the order of the adaptive list. When an opponent is found to have the most threatening move at any given MIN node of the tree, the adaptive list is queried with the identity of that opponent. Thus, the most threatening opponent will trend towards the head of the list, and his moves will be investigated first. The Threat-ADS was found to produce noticeable, statistically significant improvements in tree pruning, accomplished for a very low cost, and in a manner applicable to the complete range of multi-player games to which the BRS applies [8]. This result has been found to hold for a range of ADS-based techniques, and at game searches at both initial board positions and at midgame states [13], [14].

B. The History-ADS Heuristic

Based on the success of the Threat-ADS heuristic, we considered it worthwhile to investigate alternative applications of ADSs to game playing. The well-known Killer Moves and History heuristics make use of *move history* when making decisions about how to rank moves [5]. These heuristics operate on the hypothesis that if a move is found to be good elsewhere in the tree (i.e. it produces a cut), it is likely to be good if encountered at another place, and should thus be examined first [5]. Based on this knowledge, the use of an ADS to rank moves, rather than opponents, was considered in [9], leading to the development of the History-ADS heuristic.

Observe that the History-ADS heuristic is a natural progression from the Threat-ADS heuristic. Again, it makes use of an adaptive list, which contains possible moves, usually indexed in terms of the game space where the move originates, and where it ends, similar to the History heuristic. When a move is found to produce a cut, the ADS is queried with its identity, and its position is changed depending on the ADS update mechanism being used, or it is added to the list if it is not already present. When exploring a new node, moves are examined in the order of the ADS. An example of how the History-ADS heuristic operates is presented in Figure 1.

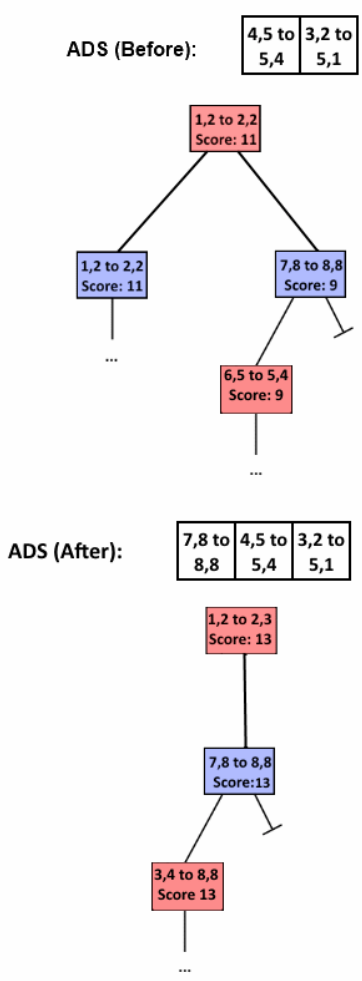

Fig. 1. An example of the History-ADS heuristic in operation. The move (7,8) to (8,8) produces a cut, and so it is moved to the head of the list, and informs the search later, in a different region of the game tree.

As which moves are likely to be good for the maximizing and minimizing players are going to be different in many cases, the History-ADS heuristic employs two adaptive lists, one for each player in the case of a 2-player game, or one for the perspective player and the "super-opponent" if the BRS is being used. However, we observe that it somewhat naively manages the list, as it can grow to any arbitrary size, and is used equally in every MIN or MAX node of the tree. The possible issues with this, and how they may impact the performance, are described in the next section.

III. Open Questions

Given the large number of possible moves, which may produce a cut over the course of the game, and at different frequencies, there are a substantial number of possible refinements that could be made to the History-ADS heuristic's structure and application, compared to the naive approach described in [9]. However, in the context of this work, we will focus on two possible areas of refinement, that of limiting the length of the ADS, to reduce its memory footprint and traversal costs, and the possibility of employing a different ADS at each level of the game tree.

A. Limiting the Maximum Length of the ADS

Considering the first avenue of inquiry, the History-ADS heuristic updates its ADS whenever a move is found to produce a cut, using the identity of that move. Its position is then either changed according to the chosen ADS update mechanism if it is already in the list, or it is appended to the end of the list, after which its position is immediately updated. Thus, any element that produces a cut remains in the list, and it is never pruned, potentially leading to a very large data structure.

Retaining all the information pertaining to moves that have produced a cut is logically beneficial. However, it is possible, and in fact very reasonable to hypothesize, that the majority of savings do not come from moves which are near the tail of the list, but rather near the front. Therefore, if we provide a maximum size on the list, and only retain elements in those positions, it may be possible to noticeably curtail the size of the list, providing some guarantees on its memory performance, while maintaining the vast majority of savings provided by the History-ADS. The way in which we will accomplish this is by forgetting any element of the list that falls to position $N + 1$, if the maximum is N. Otherwise, the History-ADS heuristic's operation is unchanged. An example of such an ADS updating over several queries, operating with a bounded list, is presented in Figure 2.

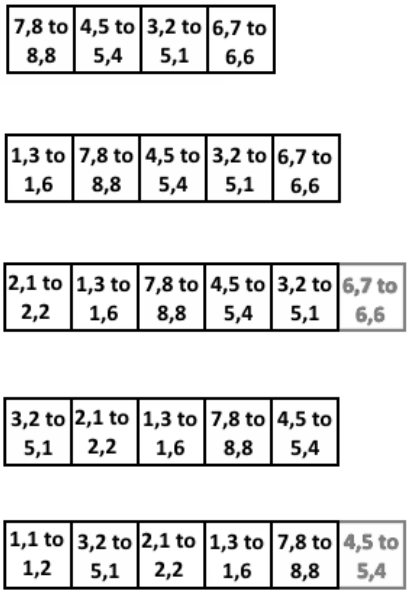

Fig. 2. An example of a History-ADS's list updating over several queries with a maximum length of 5. The ADS starts at length four, and is queried with the move (1,3) to (1,6), and it is moved to the front. It is then queried with (2,1) to (2,2), and as (6,7) to (6,6) is pushed to the sixth position, it is forgotten (highlighted in grey). The process continues as it is queried with (3,2) to (5,1), causing only an internal change, and finally (1,1) to (1,2), pushing (4,5) to (5,4) off the end of the list.

Beyond limiting the memory usage of the History-ADS heuristic, if the developer is attempting to avoid sorting moves by generating them in the order of the ADS, having to traverse a very long list to do this could defeat the purpose of omitting sorting. Thus, demonstrating that the History-ADS can achieve good results with a smaller list can assist in managing implementation concerns, as well.

B. Employing a Multi-Level ADS

Considering our second avenue of inquiry, as noted earlier, the History-ADS heuristic records moves that have produced a cut earlier in its adaptive list, which is used to order moves when they are encountered elsewhere in the tree. It performs this operation "blindly", without giving consideration to the location in the tree where the move produced a cut, relative to its current location. Thus, if moves are found to produce cuts at the lowest levels of the tree, they will be prioritized at the upper levels of the tree – later in the search.

While this may lead to improved savings, as certain moves may be very strong regardless of which level of the tree they occur on, there is a potential weakness in such a blind invocation. Consider the case where a move produces a cut close to the root of the tree, at node N. It is thus added to the adaptive list, and the search continues deeper into the tree, exploring it in a depth-first manner. Deeper in the tree, many moves are likely to produce cuts, and these will be added to the adaptive list ahead of the first move. When the search returns to the higher levels of the tree, and explores a neighbour of N, these moves will be prioritized first, over the move that produced a cut at its neighbour. However, intuitively the move that produced a cut at N, which is a more similar game state compared to those deeper in the tree, is likely to be stronger at the current node.

Furthermore, by handling all moves equally, as there are many more nodes towards the bottom of the tree compared to the top, moves that are strong closer to the leaf nodes of the tree will receive many more updates and thus a higher ranking in the adaptive list. This will occur even though cuts near the top of the tree are comparatively more valuable. Both the Killer Moves and History heuristics employ mechanisms to mitigate these effects [5], [6]. Inspired by this, we augment the History-ADS heuristic with multiple ADSs, one for each level, and use them only within the contexts of their sibling nodes.

The use of multi-level ADSs may lead to a reduction in performance, given that learning can no longer be applied at different levels of the tree. But given the precedence set by the existing techniques in the literature and the potential benefits, we consider it a meaningful avenue of inquiry.

IV. Experimental Model

Given that we are interested in refining the History-ADS heuristic and determining its performance under these potential improvements, we have elected to perform a similar battery of experiments to those employed in [9], as well as those used to test the Threat-ADS heuristic in our earlier work. We are concerned with comparing the savings, in terms of tree pruning, of the History-ADS heuristic with a single, unlimited list, to its performance with both limitations on the length of the ADS, and a multi-level ADS. To do this, we will take the aggregate of the Node Count (NC) measure over a number of turns of the game, and average that over fifty trials in each case. The NC measure is defined as the number of nodes at which computation takes place in the game tree, and serves as an effective, platform-independent, measure of the performance of the underlying alpha-beta search [5].

In order to get a sense for how the History-ADS heuristic performs in a variety of domains, we have performed our experiments under a range of game models. Specifically, we employ the two-player games Checkers and Othello, and the multi-player games Focus and Chinese Checkers, all of which were used in [14]. The rules of these games are omitted for the sake of brevity, however the reader is referred to [8] for a detailed description of the multi-player games Focus and Chinese checkers. We consider the games Othello and Checkers to be well-known enough that a detailed explanation is not necessary.

In the case of Checkers, we found that the requirement that forces jumps when possible, often leads to a game with a very small branching factor, and an extremely high variability within our results. Thus, for our experiments, we choose to relax this rule, and do not require that a player must necessarily make an available jump. We shall refer to this game as "Relaxed Checkers". While Checkers has been recently solved [15], due to its prominence in popular culture and in the literature, it remains an excellent testing environment for a domain-independent technique.

In the work introducing the History-ADS heuristic in [9], we tested its performance using both the Move-to-Front and Transposition rules to update the ADS, which intuitively state that a queried element is moved to the head of the list, or transposed one element towards the head, respectively [10]. However, we found that the Move-to-Front rule outperformed the Transposition rule in all cases, and thus we omit the Transposition rule from our experiments in this work.

When considering limitations on the maximum length of the ADS, we choose the values of 5 and 20 as maximum sizes for the ADS. A maximum list length of 20 allows a good amount of information to be recalled, and many games will not encroach on substantially larger maximums over the course of their execution. On the other hand, the lower value of 5 is chosen because with a limit of 5, the History-ADS heuristic's adaptive list length is more in line with the very short list employed by the Threat-ADS. We perform experiments related to limiting the length of the ADS on all game models, with a 6-ply search depth for Relaxed Checkers and Othello, and a 4-ply search depth for Focus and Chinese Checkers, due to their explosive branching factors.

In the case of the multi-level ADS, we present results for both a multi-level ADS, and a multi-level ADS with a length limit of 5, combining the two concepts. We choose the length limit of 5 because we are already maintaining multiple ADSs, and so we can thus hold more information than a single ADS with a length that is limited to 5. Unlike the experiments dealing primarily with the maximum length of the ADS, we only perform experiments involving multi-level ADSs in cases where the ply depth is at least 6. The reason for this is that as it stands, the History-ADS heuristic maintains separate ADSs for each player, and pruning cannot take place at the top level (the root) or the bottom (the leaf nodes), implying that the multi-level ADS approach is identical to a single ADS for 4-ply trials. We therefore present results for both 6-ply and 8-ply Othello and Relaxed Checkers in this context.

Rather than only take measurements from the starting position of the game, we observe that midgame positions are often more interesting, due to higher variability and the lack of "opening book" moves. In order to also perform experiments from a midgame state, we play the game for a number of turns before taking measurements, in order to generate a reasonable (i.e., not random) midgame state. The exact methodology by which midgame states are generated is described in detail in [14]. We take measurements over 5 turns for Othello, Relaxed Checkers, and Chinese Checkers, and 3 turns for Focus. When generating midgame states, we run the game for 5 turns for Relaxed Checkers and Focus, and 10 turns for Othello and Chinese Checkers, before we begin aggregating the NC.

Statistical analysis of our results for significance employs the Mann-Whitney test, as we do not assume normalcy. All experiments were performed on an Intel i5 3.4 GHz processor. Our results are presented in the next section.

V. Results

We present our results in the following sections, first for ADSs with a limited length, and then for multi-level ADSs.

A. Results for ADSs with Limited Length

Table I presents our results for Othello. We observed that in all cases, the use of the History-ADS heuristic, with or without a limit on the length of the ADS, produced substantial savings in terms of NC. This effect was statistically significant in all cases. By applying a very strict limit of 5 to the length of the ADS, savings were reduced from 26% to 22% when measurements were taken from the initial board state, and 34% to 26% in the midgame case. While applying a limit reduced savings, the loss was not statistically significant in all cases.

TABLE I. Results for Othello with a Varying Maximum on the Length of the ADS

ADS Size Limit	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	5061	20,100
Unlimited	3727	13,300
20	3779	13,900
5	3961	14,800

Consider Table II, which holds our results for Relaxed Checkers. As with Othello, large reductions in NC were observed in all cases where the History-ADS was used. Savings were reduced, comparing the unlimited length to a maximum of 5 as before, from 48% to 44% in the initial case, and 46% to 39% in the midgame case, although in the initial case this change was statistically significant.

TABLE II. Results for Checkers with a Varying Maximum on the Length of the ADS

ADS Size Limit	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	78,600	64,000
Unlimited	41,000	34,400
20	42,700	36,700
5	44,700	39,500

Our results for the multi-player game Focus are presented in Table III, where we observed very large savings in terms of NC in all cases. Again comparing an unlimited list to a maximum length of 5, savings were reduced from 70% to 68%

TABLE III. RESULTS FOR FOCUS WITH A VARYING MAXIMUM ON THE LENGTH OF THE ADS

ADS Size Limit	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	6,970,000	14,200,000
Unlimited	2,150,000	3,120,000
20	2,210,000	3,310,000
5	2,230,000	3,370,000

from the initial board state, and 79% to 76% from a midgame state, and the change was not statistically relevant in any case.

Lastly, Table IV shows our results for Chinese Checkers. We observed a similar pattern as with the other game models, with strong performance from the History-ADS heuristic. Comparing an unlimited list to a maximum length of 5, as before, savings reduce from 62% to 60% in the initial case, and 61% to 60%. As one would expect, this very small change was not statistically significant.

TABLE IV. RESULTS FOR CHINESE CHECKERS WITH A VARYING MAXIMUM ON THE LENGTH OF THE ADS

ADS Size Limit	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	3,370,000	8,260,000
Unlimited	1,280,000	3,200,000
20	1,330,000	3,290,000
5	1,360,000	3,340,000

B. Results for Multi-Level ADSs

Table V demonstrates our results for 6-ply Othello, comparing single-level and multi-level ADS approaches. As with our previous results, we observed savings in all cases when the History-ADS was used, although in this case, the multi-level ADS with a maximum length of 5 fell slightly outside 95% certainty in the case of the initial board state. We furthermore found that the multi-level ADS performed worse than the single-level ADS, and worse still when its maximum length was limited. Savings reduced from 26% to 15% in the initial board state case, and 34% to 20% in the midgame case, although these differences were not statistically significant.

TABLE V. RESULTS FOR 6-PLY OTHELLO WITH A MULTI-LEVEL ADS

ADS Type	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	5061	20,100
Single-Level	3727	13,300
Multi-Level	4110	14,700
Multi-Level (length 5)	4305	16,000

Our results for 8-ply Othello are presented in Table VI. In this case, we see a slight deviation from the general pattern, where, in the midgame case, the multi-level ADS with a maximum length of 5 actually performed better than the unlimited multi-level ADS, although the difference is statistically insignificant. Savings were reduced from 39% to 34% in the initial case, and 49% to 40% in the midgame case. The change was within 95% certainty in the case of the initial board state.

Table VII presents our results for 6-ply Relaxed Checkers. Again, in the midgame case we see a deviation where the limited length multi-level ADS performed slightly better, although again by an insignificant margin. Savings were reduced by 48% to 40% in the initial board state case, and 46% to 41% in the midgame case. This reduction was significant in both cases.

TABLE VI. RESULTS FOR 8-PLY OTHELLO WITH A MULTI-LEVEL ADS

ADS Type	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	38,800	182,000
Single-Level	23,600	92,900
Multi-Level	24,900	110,000
Multi-Level (length 5)	25,600	105,000

TABLE VII. RESULTS FOR 6-PLY RELAXED CHECKERS WITH A MULTI-LEVEL ADS

ADS Type	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	78,600	64,000
Single-Level	41,000	34,400
Multi-Level	45,300	38,100
Multi-Level (length 5)	46,900	37,800

Finally, Table VIII shows our results for 8-ply Relaxed Checkers. In this case, savings were reduced from 61% to 54% when measurements were taken from the initial board state, and from 66% to 53% in the midgame case. This difference was significant in the midgame case.

TABLE VIII. RESULTS FOR 8-PLY RELAXED CHECKERS WITH A MULTI-LEVEL ADS

ADS Type	Avg. NC (Initial)	Avg. NC (Midgame)
No ADS	910,000	859,000
Single-Level	358,000	293,000
Multi-Level	394,000	350,000
Multi-Level (length 5)	416,000	404,000

VI. DISCUSSION

Our results presented in the previous section demonstrate that in almost all cases, without considering the configuration employed, the History-ADS heuristic is able to produce very large, statistically significant gains in tree pruning, in some cases over 75%. These results are consistent with our previous observations, reported in [9]. As was observed in that work, we found that, in general, larger savings were correlated to larger search trees, both in terms of branching factor and ply-depth. These observations reinforce our results from [9].

We found that when limiting the maximum size of the ADS, while there was some reduction in performance, as was expected, the loss was very slight in most cases. This confirms our hypothesis that elements near the head of the list tend to remain there, and provide the majority of the move ordering benefits, with diminishing returns as the list gets longer. The fact that the majority of savings are still maintained in all cases even when the list is limited to have at most five elements, successfully addresses one of the concerns we had with the History-ADS heuristic as originally presented, namely that the length of the adaptive list can potentially be quite large.

The trend was consistent across all cases, with the limit of 20 performing marginally worse than an unbound list, and the limit of 5 doing a little worse. The degree of reduction in performance varied between game models, with a maximum

34% to 26% in the case of Othello, with a maximum list size of 5. This demonstrates that even in the worst case, the majority of the savings from the History-ADS heuristic remain in a very short list, compared to one of an unbound length. Furthermore, while the reduction was consistent across all cases, it was only statistically significant in one case, which was when considering Relaxed Checkers from an initial board position. In all other cases, including in the worst case mentioned above, this reduction in savings falls outside 95% certainty.

The multi-level ADS approach was found to do worse than the single ADS approach, suggesting that savings that may be gained for prioritizing moves that produced a cut on the current level of the tree, if they exist, are offset by the inability to apply what the algorithm has learned to other levels of the tree. We can thus conclude that the absolutist approach of having a separate ADS at each level of the tree is likely not the optimal way to address concerns of overvaluing moves that are strong near the bottom of the tree. However, the multi-level ADS approach may not be completely useless. If the History-ADS heuristic is employed alongside other, perhaps domain-specific move ordering heuristics, then prioritizing only the best moves at each level may be as effective. This approach therefore begs for more investigation in the future.

Despite the presence of multiple ADSs, limiting the size of the list to 5 reduced improvements even more, in the majority of cases. Observing the internal functioning of the search, the reason for this appears to be that the limit is only a factor at the lowest levels of the tree, where many more moves produce cuts and the limit impacts the ADS heavily. As opposed to this, levels closer to the root do not require as much space. This is especially visible in the case of 6-ply Othello from the midgame case, where a multi-level ADS with a maximum size of 5 did not, in fact, produce a statistically significant improvement in tree pruning, which is the only case in which the History-ADS heuristic has failed to achieve this. Overall, however, in most cases, the performance of the multi-level ADS was close to the original version.

VII. Conclusions

The results presented in this paper reinforce our conclusions from our earlier work regarding the History-ADS heuristic. In nearly every case, we found that the use of the History-ADS heuristic, regardless of its configuration, was able to achieve noticeable, statistically significant savings, at times over a 75% reduction in tree size.

When the maximum length of the ADS is restricted, while some reduction in performance was observed, as can be expected, the vast majority of savings remain. This is maintained even if the list is restricted to a very short length of 5. This is a particularly valuable observation, as, to summarize earlier discussion, one of the main concerns with the History-ADS heuristic was that its list size was unbound, potentially leading to a negative impact on its real-world performance. With a maximum size of 5, however, its size is on the order of the very lightweight Threat-ADS heuristic, and these concerns are successfully mitigated.

While the use of a multi-level ADS achieved a similar level of savings compared to a single ADS, it was outperformed, in all cases, by the single ADS. However, given the strong basis in the literature of techniques that consider moves based on the level of the tree where they are found, such as with the Killer Moves and History heuristics, we believe that it is worthwhile to investigate this area further. Work is currently ongoing on methods to prioritize learning at the level of the tree where it was acquired, while not completely excluding information obtained elsewhere.

References

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, pp. 161–201. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 3rd ed., 2009.

[2] C. E. Shannon, "Programming a computer for playing Chess," *Philosophical Magazine*, vol. 41, pp. 256–275, 1950.

[3] G. M. Baudet, "An analysis of the full alpha-beta pruning algorithm," in *Proceedings of the tenth annual ACM symposium on Theory of computing*, pp. 296–313, 1978.

[4] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, pp. 293–326, 1975.

[5] J. Schaeffer, "The history heuristic and alpha-beta search enhancements in practice," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 1203–1212, 1989.

[6] S. Akl and M. Newborn, "The principal continuation and the killer heuristic," in *Proceedings of ACM'77 the 1977 Annual Conference*, pp. 466–473, 1977.

[7] M. P. D. Schadd and M. H. M. Winands, "Best Reply Search for multiplayer games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, pp. 57–66, 2011.

[8] S. Polk and B. J. Oommen, "On applying adaptive data structures to multi-player game playing," in *In Proceedings of AI'2013, the Thirty-Third SGAI Conference on Artificial Intelligence*, pp. 125–138, 2013.

[9] S. Polk and B. J. Oommen, "Enhancing history-based move ordering in game playing using adaptive data structures," in *To Appear in Proceedings of ICCCI'2015, the 7th International Conference on Computational Collective Intelligence Technologies and Applications*, 2015.

[10] S. Albers and J. Westbrook, "Self-organizing data structures," in *Online Algorithms*, pp. 13–51, 1998.

[11] T. H. Corman, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, pp. 302–320. Upper Saddle River, NJ, USA: MIT Press, 3rd ed., 2009.

[12] G. H. Gonnet, J. I. Munro, and H. Suwanda, "Towards self-organizing linear search," in *Proceedings of FOCS'79, the 1979 Annual Symposium on Foundations of Computer Science*, pp. 169–171, 1979.

[13] S. Polk and B. J. Oommen, "On enhancing recent multi-player game playing strategies using a spectrum of adaptive data structures," in *In Proceedings of TAAI'2013, the 2013 Conference on Technologies and Applications of Artificial Intelligence*, 2013.

[14] S. Polk and B. J. Oommen, "Novel AI strategies for multi-player games at intermediate board states," in *Proceedings of IEA/AIE'2015, the Twenty-Eighth International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems*, pp. 33–42, 2015.

[15] J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P. Lu, and S. Sutphen, "Checkers is solved," *Science*, vol. 14, pp. 1518–1522, 2007.