Novel Threat-based AI Strategies that Incorporate Adaptive Data Structures for Multi-Player Board Games[*]

Spencer Polk[†] and B. John Oommen[‡]

Abstract

This paper considers the problem of designing novel techniques for *multi*-player game playing, in a range of board games and configurations. Compared to the well-known case of two-player game playing, multi-player game playing is a more complex problem with unique requirements. To address the unique challenges of this domain, we examine the potential of employing techniques inspired by Adaptive Data Structures (ADSs) to rank opponents based on their relative threats, and using this information to achieve gains in move ordering and tree pruning. We name our new technique the Threat-ADS heuristic. We examine the Threat-ADS' performance within a range of game models, employing a number of different, well-understood update mechanisms for ADSs. We then extend our analysis to specifically consider intermediate board states, which are more interesting than the initial board state, as we do not assume the availability of "Opening book" moves, and where substantial variation can exist, in terms of available moves and threatening opponents. We expand this analysis to include an exploration of the Threat-ADS heuristic's performance in deeper ply game trees, to confirm that it maintains its benefits even when lookahead is greater, and with an expanded examination of how the number of players present in the game impacts the performance of the Threat-ADS heuristic. We find that in nearly all environments, the Threat-ADS heuristic is able to produce meaningful, statistically significant improvements in tree pruning, demonstrating that it serves as a very reliable move ordering heuristic for multi-player game playing under a wide range of configurations, thus motivating the use of ADS-based techniques within the field of game playing.

Keywords: *Multi-Player Game Playing, Adaptive Data Structures, Move Ordering, Alpha-Beta Search*

1 Introduction

The problem of achieving robust game play against an intelligent opponent is very well-known within the field of AI, and has seen a substantial body of research, since the field's inception [17, 22]. A vast body of literature exists to address this problem, and powerful techniques, such as alpha-beta search, have been developed over

[*]The second author is grateful for the partial support provided by NSERC, the Natural Sciences and Engineering Research Council of Canada. *Both the authors are extremely grateful to the anonymous Referees of the initial version of this paper for their valuable comments. Their comments significantly improved the quality of this paper.* A preliminary version of this paper was presented at IEA/AIE'15, the 2015 International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, in Seoul, Korea, in June 2015. The paper won the *Best Paper* award of the conference.

[†]This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. E-mail: `andrewpolk@cmail.carleton.ca`.

[‡]Author's status: *Chancellor's Professor*, *Fellow: IEEE* and *Fellow: IAPR*. This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. The author is also an Adjunct Professor with University of Agder, Grimstad, Norway. E-mail: `oommen@scs.carleton.ca`.

time. However, the majority of research has been focused on the specific case of *two*-player games, such as Chess or Go [17, 25]. These are games in which there is a single perspective player, representing the AI player, and an opponent who seeks to minimize his score and win the game. The related, but more complex, field of *Multi*-Player Game Playing (MPGP), where a number of intelligent players vie against one another to be the single winner, such as the well-known strategy game Settlers of Catan, has seen comparatively much less focus in the literature [8, 19, 24, 25, 28]. Compared to the two-player case, multi-player games present a number of unique problems and complications, such as the fact that opponents may not always seek to minimize the perspective player, and may form temporary coalitions or alliances, complicating the strategic modeling of the game.

Due in part to limited research, in addition to the more complex problem posed by MGPG, the majority of MPGP algorithms have issues performing on a level comparable with their two-player counterparts [24, 26, 27]. In particular, a well-known problem that one encounters in multi-player environments is that of efficiently pruning the game tree. This is more challenging in the multi-player domain, due to the presence of multiple self-interested opponents both expanding the number of moves that must be searched, and complicating the Mini-Max paradigm upon which many powerful game playing techniques are based [25]. It is well known that efficient tree pruning can be achieved through good move ordering, or, techniques by which the best move is most likely to be searched first, and many well-known techniques, such as Killer Moves and the History heuristic, have been developed to achieve this [17, 20].

Formerly unrelated to the field of game playing, prior to our work, the field of Adaptive Data Structures (ADSs) deals with dynamically reorganizing elements in a data structure, in response to queries over time, to more efficiently handle them [3, 5, 6]. This is achieved through moving the most frequently accessed elements towards the head of the data structure, or the root in the case of a tree, so that these elements can be accessed more quickly, thus improving the data structures' amortized execution time [1]. A wide range of techniques to achieve this have been introduced by the field of ADSs, such as the Move-to-Front and Transposition rules for adaptive lists, and Splay Trees for binary search trees [1, 3].

Our critical observation is that ADSs are, in essence, providing a dynamic *ranking* mechanism for elements of the data structure. As there are many possible uses for ranking within game playing, such as the ranking of possible moves to achieve move ordering, as described above, there is a considerable motivation to investigate the applicability of ADS-based techniques in game playing. This is particularly relevant to MGPG, as the phenomenon of ranking the opponents is an intuitive concept that can be potentially achieved using ADSs, and as a method to improve existing multi-player techniques. The concept of applying techniques from the unrelated field of ADSs to game playing is a novel one, which has not been presented in the literature outside of this work.

In this work, we present a formal move ordering heuristic for a state-of-the-art multi-player game playing algorithm, the Best-Reply Search (BRS), which employs an ADS to dynamically rank opponents based on their relative threat levels. This information is then used to achieve improved move ordering, leading to a reduction in the BRS' game tree size. We name this new technique the Threat-ADS heuristic, and show that, in spite of a lightweight cost and relatively intuitive application of ADS-based mechanisms, it is capable of achieving statistically significant gains in tree pruning in a wide range of environments.

In particular, our experimental investigation examines its performance under a number of game models, an examination of a range of established, well-known ADS update mechanisms, as well as in searches of varying ply depths, to confirm that its benefits remain even in deeper trees. We furthermore explore the question of its applicability to both initial board positions, which are well known and orderly, and highly variable intermediate board positions, as well as the benefits it can obtain with differing numbers of opponents.

Some extremely preliminary results pertaining to this research are found in [11, 12, 14]. The remainder of the paper is laid out as follows. Section 2 details the specific challenges of multi-player game playing as opposed to two-player game playing, and describes, in detail, common techniques to address them. Section 3 describes the formerly unrelated field of ADSs, and a number of established mechanisms by which data structures can be efficiently and dynamically reorganized to match queries. Section 4 describes the Threat-ADS heuristic, including its motivation, development, and qualities, and Section 5 describes our experimental model, including game models, configuration, and the motivation behind and generation of intermediate board positions for testing. Sections 6 through 10 present and describe our results as they logically build on each other. Section 11 provides our analysis and discussion of these results, and Section 12 concludes the paper.

2 Multi-Player Game Strategies and Approaches

Compared to traditional two-player game playing, multi-player environments, through the addition of other, self-interested agents, introduce a range of new complications and challenges. These include:

- Any single player's gain does not necessarily generate an equal loss amongst all opponents.
- Temporary coalitions of players can arise, even in games with only a solitary winner.
- The board state can change more between each of the perspective player's moves.
- A single-valued heuristic is not always sufficient to correctly evaluate the game state.
- Established, highly-efficient tree pruning techniques, such as alpha-beta pruning, are not always applicable.

Despite these challenges, due to the historical success of Mini-Max with alpha-beta pruning in a wide variety of domains, substantial efforts have been dedicated to extending it to multi-player environments, with varying levels of success [8, 19, 24, 25]. In this section, we will detail a number of the more well-known of these techniques, specifically the Paranoid, Max-N, and Best-Reply Search strategies, and their respective benefits and drawbacks.

2.1 The Paranoid Algorithm

The intuitive extension of the Mini-Max technique to MPGP results in what is commonly termed the "Paranoid algorithm" in the literature [19, 24, 25]. This approach is the most intuitive one, and requires the fewest changes from the well-known Mini-Max algorithm. As in the Mini-Max approach, the Paranoid technique retains a single value at each node representing the heuristic value *h(x)* for the perspective player. As the game is now multi-player, rather than two player, each level of the game tree represents a different player's turn, and so there will be multiple opponent turns in between each of the perspective player's turns. The

Paranoid algorithm, being the simplest possible extension of Mini-Max to multi-player games, handles this by treating every opponent turn as a Min node [24]. Thus, for a three player game, the Paranoid algorithm could be referred to as Max-Min-Min, or for a four player game, Max-Min-Min-Min. A sample game tree for the Paranoid Algorithm is presented in Figure 1. The values indicate how "good" each node, or board position, is for the perspective player, derived from some arbitrary evaluation function.

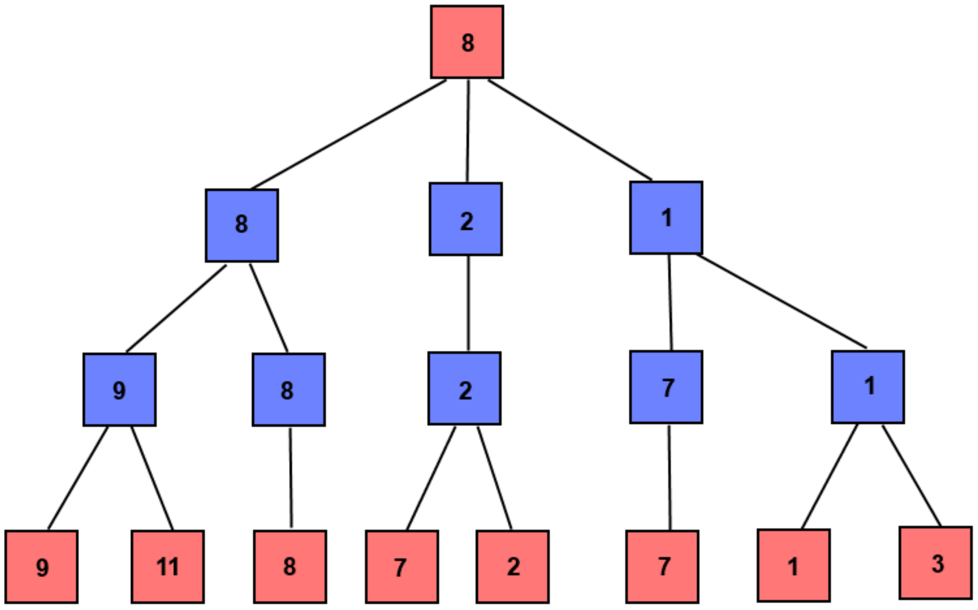

Figure 1: Sample Paranoid Tree in which the red nodes are MAX, and the blue nodes are MIN.

The Paranoid algorithm, naturally, treats all players as a *coalition* against the perspective player [25]. This is because each opponent, being treated as operating in a Min phase, makes choices based not on maximizing its own position in the game or removing other players from contention, but solely on the basis of minimizing the perspective player's score. Indeed, all opponents choose the most minimizing path. The algorithm even predicts that opponents will take moves operating under the assumption that *other* opponents will take actions leading to even greater minimization of the perspective player's score. Thus, the assumption is made that opponents will not only exclusively target the perspective player, but will, in fact, actively work together against him [25].

The Paranoid algorithm suffers from a glaring, intuitive drawback. While this assumption could be considered the "safest" approach to the game, it is, of course, unreasonable to operate on the belief that opponents in a multi-player game will solely work in a coalition, even to their own potential detriment. The Paranoid algorithm thus has a tendency to consider unrealistic game states, which could potentially lead to bad play, particularly in games with dramatic shifts in board position between moves [25].

However, as it maintains a single heuristic value, the Paranoid algorithm retains the benefits of alpha-beta pruning, which is not the case for all multi-player techniques, and thus can, in fact, outperform other, more realistic strategies due to achieving improved lookahead [19]. Despite this upside, it can only produce cuts at at boundaries between Max and Min nodes, and never between individual Min layers. Thus, less total pruning will occur in a Paranoid tree than in a Mini-Max tree.

2.2 The Max-N Algorithm

While the Paranoid algorithm is the most intuitive extension of Mini-Max to multi-player games, and the simplest to implement, a competing algorithm, called the Max-N algorithm, is the natural extension of Mini-Max principles to N-person games [8]. The basic philosophy of the Mini-Max algorithm, after all, is not based on minimizing a specific player's score, but instead on maximizing your own score [22]. For a two-player, zero-sum, combinatorial game such as Chess, for which the Mini-Max algorithm was originally developed, these two functions are naturally identical, and thus, thinking of it along those lines leads us to the Paranoid algorithm [24]. The Max-N algorithm, on the other hand, operates on the far more reasonable assumption that players will instead seek to maximize their own scores, without consideration for other opponents [8].

Rather than the heuristic function *h(x)* returning a single value, as is the case with the Mini-Max and Paranoid algorithms, the heuristic function for the Max-N algorithm, instead, returns a *tuple* of values of size N, where N is the number of players [8]. The N^{th} value, traditionally, corresponds to the N^{th} player, where the first player is the perspective player, and where subsequent opponents are numbered in their turn order, beginning from the perspective player. True to its name, at the i^{th} player's turn, he is assumed to choose the move that provides the maximum value in position i in the tuple, and, similar to the Mini-Max or Paranoid algorithms, this value is passed up the tree, until, eventually, a path is chosen for the perspective player at the root [8]. Figure 2 shows a sample Max-N tree after expansion of all the leaf nodes, with the values being passed up to the root. The values associated with each node, in this case, represent the tuple returned by the algorithm's heuristic function.

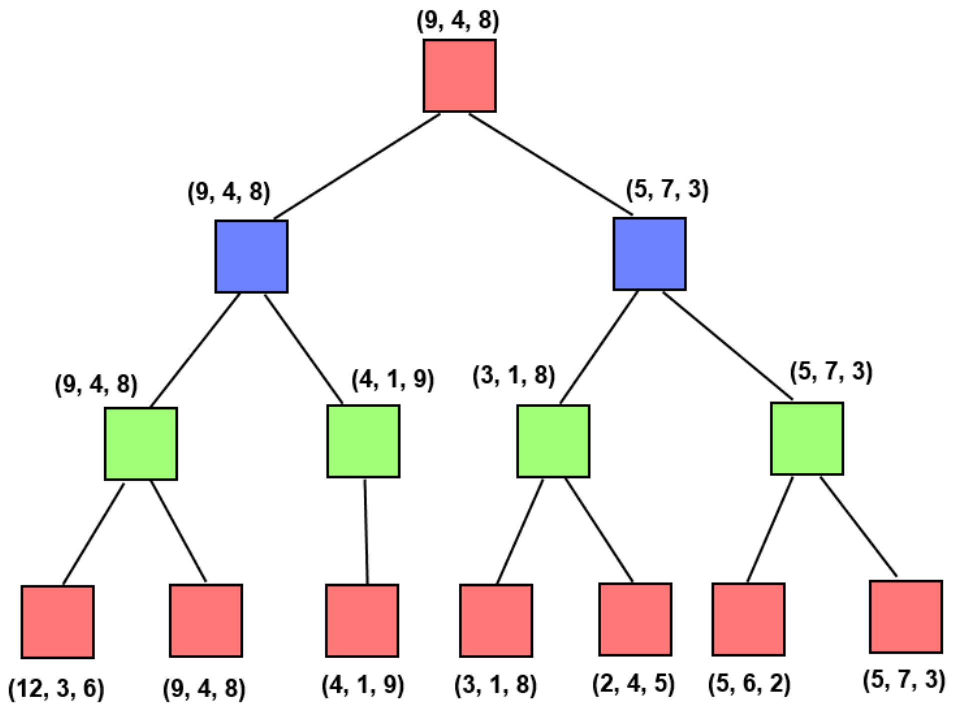

Figure 2: Sample Max-N Tree in which each color represents a different player.

The Max-N algorithm certainly operates using a more reasonable assumption in regards to how opponents will play. Assuming that the heuristic used is reasonable for the game in question, for a zero-sum N-player game, maximizing one's own score will translate to minimizing the opponents *as a group*, and thus the core of the Mini-Max philosophy remains intact [25]. This is the most clear benefit of using it over the Paranoid algorithm, as the more "relaxed" pressure on the perspective player allows him to take advantage of the various opportunities that may have been overlooked if the assumption of a "coalition of opponents" is made [24].

Unfortunately, despite a more realistic model of play, as the Max-N algorithm makes use of a tuple of values rather than a single integer value for the results of the heuristic function, and must find the heuristic value for all players at leaf nodes, it will, necessarily, be more computationally expensive than the Paranoid algorithm under an otherwise identical implementation. More importantly, the use of a tuple implies that alpha-beta pruning is unavailable to the Max-N algorithm, and equivalents that exist do not perform nearly as well [25].

2.3 Best-Reply Search

The Paranoid and Max-N algorithms remained the standard for deterministic MPGP for many years. However, recently, a new MPGP algorithm named the Best-Reply Search (BRS) has been introduced, which can, in some cases, significantly outperform both the Paranoid and Max-N algorithms [19]. In the case of the BRS, all opponents are again considered to operate as in a coalition, as in the Paranoid algorithm, but between each of the perspective player's turns, *it allows only a single opponent to act* [19]. The opponent who is allowed to act is the one who has the most minimizing move, in relation to the perspective player, at this point in time, or the “Best Reply”. In essence, the scheme pretends that all opponents are not simply a coalition, but that the coalition represents a single player with significantly more resources available than that perspective player. Figure 3 shows a single level of a BRS tree (only a single level is shown for space considerations, as the branching factor is considerably higher for opponent turns in BRS), where the minimum of all opponent turns is being selected.

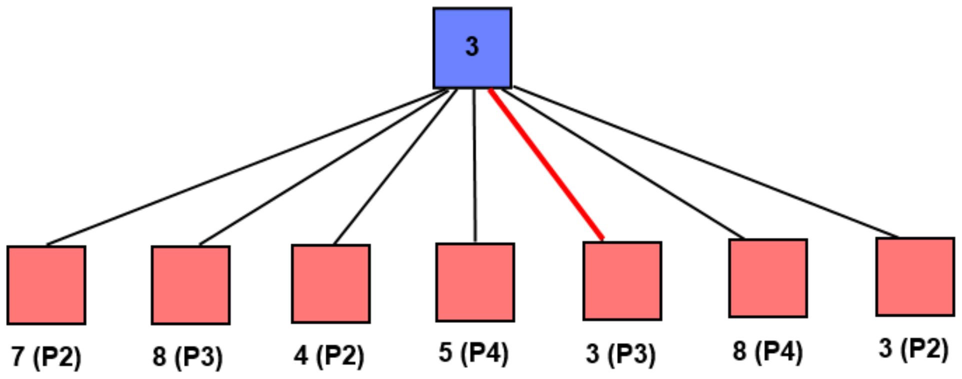

Figure 3: The operation of a single level of the Best-Reply Search. The scores that are reported have the opponent’s player number listed next to them (in parenthesis) to assist in the clarification.

The immediate, glaring drawback of the BRS algorithm is that it considers illegal move states while searching. This is certainly a serious drawback, and in fact, limits the games to which the BRS can be applied [19]. The BRS can only be applied to those games where it is *meaningful* for players to act out of turn, and performs best when the board state does not change too dramatically in between turns [19]. An example of a game to which the BRS can *not* by applied is Bridge, because scoring in Bridge is based on “tricks”, and thus, allowing players to act out of order renders the game tree to be void of meaning. In a game where the game state changes significantly between turns, there is a serious risk of the BRS arriving at a model of the game which is significantly different from reality.

However, in cases where it can be applied, the BRS has many benefits over the Paranoid and Max-N algorithms, and often outperforms them quite dramatically [19]. As can be intuitively observed, when all opponents are considered to be a single entity, the game is again modeled as a two-player game, and played using the Mini-Max algorithm. Thus, alpha-beta pruning and all other two-player Mini-Max improvements can be applied with even less restrictions than those present in the case of the Paranoid algorithm. Also, as we showed earlier, alpha-beta pruning can achieve the fewest number of leaf nodes expanded when there are two players, and thus the BRS has the best node pruning of all multi-player Mini-Max style algorithms [19, 25].

As it simplifies things to work with a model analogous to a two-player game, the BRS also allows better look-ahead for the perspective player than either the Paranoid or the Max-N schemes. The benefits of this look-ahead are considered to be a significant factor in the performance of the BRS over the Paranoid and the Max-N in games such as Chinese Checkers [19].

3 The Improving Agent: Adaptive Data Structures

It is well-known in the study of data structures that the access frequencies of the elements in the structure are not uniform [5, 6]. For example, in a linked list consisting of five elements, A, B, C, D, and E, in that order, the corresponding access probabilities may be 20%, 5%, 10%, 40% and 25%. Using a traditional singly-linked list, this would pose a problem, as the two elements accessed the most frequently, D and E, could be located at the rear of the list, thus requiring a longer access time. We can intuitively see that another linked list, holding the same five elements, in the order D, E, A, C, B will achieve faster average performance. Thus, by restructuring the list, one can obtain an improved functionality for the data structure.

In the trivial example above, the reorganization is obvious, as the access probabilities are assumed to be known and stationary. However, in the real world, the access probabilities are, intuitively, not known when the structure is first created. The field of ADSs concerns itself with finding good resolutions to this problem [1, 3, 5, 6]. As the access probabilities are not known, the data structure must learn them as queries proceed, and *adapt* to this changing information by altering its internal structure to better serve future queries [5]. Individual types of ADSs provide *update mechanisms* that lead to this sort of behaviour for the specific data structure.

The method by which an ADS reorganizes its internal structure, in response to queries, or its update mechanism, must possess several qualities to be useful. Firstly, as ADSs attempt to reduce the amortized execution time of a large number of queries into the data structure, the update mechanism itself must be very efficient, or the time lost in reorganizing the data structure could potentially cancel out any gains. Secondly, as the intention is to arrange the elements from the most to the least frequently accessed, ADS update schemes provide an excellent, natural *ranking* mechanism, which can potentially be applied in other domains, such as game playing, provided that an efficient means to simulate queries exists. To our knowledge, the concept of employing an ADS as a ranking tool in game playing has not been presented in the literature to date, outside of our preliminary work.

An ADS may be of any type, typically a list or a tree, with the well-known Splay Tree being an example of the latter [1, 10]. However, in this work, we will be focusing only on adaptive *lists*, due to our intended application of their qualities to MPGP. The following subsection details the specifics of the update mechanisms that we will be examining in this work.

3.1 ADS Update Mechanisms

The schemes we involve in this work, and indeed the majority of well-known update mechanisms, have several commonalities, touched upon briefly in the previous section. Firstly, while no changes may be required, if the element accessed is at the head of the data structure already, in general, the update mechanism is executed each time an element is queried, moving it closer to the head in some way. Thus, as the goal of an ADS is to improve efficiency, these update mechanisms must be very low-cost. In general, they execute in $O(1)$ time, usually involving nothing more than the relocation of a single element within a linked list.

As mentioned above, while ADS varieties exist for a range of data structures, due to the small number of elements involved, we limit our inquiry to adaptive lists. Details of the update mechanisms we explore follow.

Move-to-Front: The Move-to-Front rule is one of the oldest and well-studied ADS update mechanisms, as well as one of the most intuitive [1, 5, 16, 23]. As its name suggests, when an element is accessed, it is moved from its current position to the head of the list. Given that it moves the accessed element to the head of the list with no regard for where it was previously located, usage of the Move-to-Front rule leads to dramatic changes in the structure of the list in a very short time span. However, if quick adaptation is desirable, as it can be in the case of a game where relative threats rapidly evolve, this can certainly be a desirable property, and in our work, it has performed particularly well.

Transposition: The most common competitor to the Move-to-Front rule, also studied extensively in the ADS literature, is the Transposition rule [1, 4, 5]. The Transposition rule dictates that when an element is accessed, it should be transposed one position towards the head, "swapping" its position with the element ahead of it. Compared to the Move-to-Front rule, the Transposition rule is slower to adapt, and leads to less dramatic changes in the list structure. This can be beneficial, however, as it is less susceptible to outlier queries, which can be relevant for the Threat-ADS if one opponent tends to remain the most threatening for a long period of time.

Move-Ahead-k: The Move-Ahead-k rule attempts to strike a compromise between the Move-to-Front and Transposition rules [6]. As its name suggests, when an element is accessed with a query, it is transposed k spaces forward, or to the front of the list if k spaces do not remain. Thus, the change is larger than for the Transposition rule, but not as dramatic as it is in the case of the Move-to-Front rule. Indeed, the Transposition rule could be defined as Move-Ahead-1, and the Move-to-Front rule could be defined as Move-Ahead-n, where n is the length of the list [6]. Given the small number of opponents in an average game, we will employ Move-Ahead-2.

POS(k): The POS(k) updating mechanism is yet another method which balances between the extremes of the Move-to-Front and Transposition rules, although it does so in a more complex fashion than Move-Ahead-k [6]. It has been proposed that, as a data structure approaches convergence, it would be best to switch from Move-to-Front to Transposition; the issue is that it is not easy to tell when convergence has occurred, and a series of queries can always be constructed that will "fool" attempts to do so with estimation [6]. The POS(k) mechanism was therefore developed to hybridize and balance between the two.

Under POS(k), for a value of k such that $1 \leq k \leq n$, where n is the length of the list, if an object in a position *less than* k is accessed, it is transposed with the element ahead of it in the list, identically to the Transposition rule [9]. However, if the object is in a position *greater than* k, it is instead brought to position k, thus behaving like the Move-to-Front rule, where position k is treated as the front of the list. Note, again, that we can express the Transposition and the Move-to-Front rules in terms of POS(k) since the former is the same as POS(n), and the Move-to-Front rule can be expressed as POS(1). Again, given the small number of opponents, we will focus on POS(2) in this work.

4 The Threat-ADS Strategy

One feature present in the context of multi-player games, which has no analogue in two-player games, is that of *opponent threat*. In the case of a two-player game, the perspective player is presented with a single opponent, and all minimizing actions must come from him. However, in a multi-player game, different opponents may be able to minimize the perspective player, in different ways, and to varying degrees. The concept of considering opponent threat, both to model and predict their future actions, as well as to prioritize opponents, is a known concept within the MPGP literature [26, 27, 30]. The phenomenon of opponent threat that we are concerned with in this work is that of *relative* opponent threats, that is, a ranking of opponents in terms of their capacity for minimization towards the perspective player.

It is in our attempt to achieve a highly efficient, dynamic means of determining relative opponent threats that we turn to techniques inspired by the field of ADSs. As discussed above, ADSs provide a natural mechanism by which efficient ranking can be achieved, and our strategy is to employ an ADS to rank opponents, in a manner that will reflect their relative threats. The application of ADS-based techniques to game playing is another novel concept that has not been presented in the literature, prior to our work. Given its efficiency in a broad range of use cases and its state-of-the-art nature, we focus our effort on the BRS technique, and develop a method by which an ADS may be applied, in conjunction with the BRS, to achieve move ordering based on relative opponent threats.

This move ordering technique, which we have named the Threat-ADS heuristic, is specified and described below.

4.1 Specification of the Threat-ADS Heuristic

Considering the execution of the BRS, we observe that, at each "Min" phase, the BRS determines which opponent has the most minimizing move. The phenomenon of having the most minimizing move against the perspective player, and also being characterized by possessing the relatively highest threat level against that player, are conceptually and intuitively linked. With that in mind, we query an ADS, which contains the identities of each opponent, with the identity of the opponent that is seen to possess the most minimizing move during a Min phase. This has the effect of advancing his position in the relative threat ranking, and allowing the ADS to "learn" a complete ranking over time.

With this ranking provided, we employ it by exploring the relevant moves, at each Min phase, in the order from the most to least threatening opponent. As a threatening opponent is more likely to provide the greatest minimization to the perspective player, this improves move ordering, and thus the savings from alpha-beta pruning. Algorithm 1 shows the complete, formal BRS algorithm, making use of the Threat-ADS heuristic.

We clarify this by means of an example in Figure 4. This figure shows, how the ADS updates, based on which opponent was found to have the most minimizing move, at a certain level of the tree. Here, opponent "P4" has the most minimizing move, and thus the ADS is updated by moving him to the head of the list.

Algorithm 1 BRS with Threat-ADS

```
Function BRS(node, depth, player)
 1: if node is terminal or depth $\leq$ 0 then
 2:   return  heuristic value of node
 3: else
 4:   if node is max then
 5:     $\alpha = -\infty$
 6:     for all child of node do
 7:       $\alpha = max(\alpha, BRS(child, depth - 1)$
 8:     end for
 9:   else
10:     $\alpha = \infty$
11:     for all opponents in game in ADS order do
12:       for all child nodes for opponent do
13:         $\alpha = min(\alpha, BRS(child, depth - 1)$
14:       end for
15:     end for
16:     Query the ADS with player information from move $\alpha$
17:   end if
18:   return $\alpha$
19: end if
End Function BRS with Threat-ADS
```

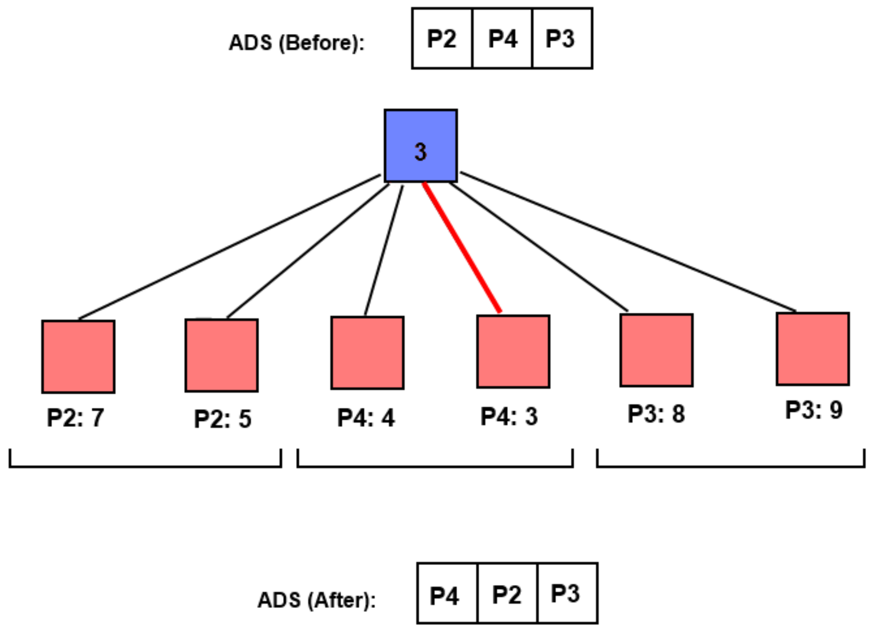

Figure 4: A demonstration of how the Threat-ADS heuristic operates over time.

4.2 Qualities of the Threat-ADS Heuristic

Before we continue, it is worthwhile to address a few questions about the operation of the Threat-ADS heuristic. First of all, we observe that, as is common with move-ordering heuristics, the use of the Threat-ADS heuristic does not in any way alter the *value* of the tree. Consequently, there can be no destructive influence on the decision making process that the BRS undergoes.

More importantly, although the BRS is subject to traditional move ordering techniques, it does not impose any *natural* ordering on the opponents or how their moves are collected. Therefore, no information can be lost when one uses the Threat-ADS heuristic to decide the order in which one collects moves from the opponent. It is possible that heuristics such as the History heuristic, or domain dependent heuristics, could compare with the "lightweight" Threat-ADS heuristic. However, in these cases, we can use the Threat-ADS heuristic to break ties within the former heuristic. This setup, of having an order in which heuristics are applied, and using a potentially weaker, more lightweight heuristic to break ties, is common in modern game playing engines. This was indeed, reported even in the introductory paper for the BRS [19].

The Threat-ADS heuristic adds the adaptive linked list to the BRS's memory footprint. Luckily, the list only has a length equal to the number of opponents, which is likely to be a very small constant (typically no more then seven). This can be intuitively derived, as our research concerns traditional board games (to which the BRS applies), and given that these games are designed to be played in person, the number of players cannot feasibly be a large value. While there are other varieties of multi-player games, such as commercial video games, where this fact may not be true, the BRS does not easily apply to them. The number of players may, in fact, decrease, as many multi-player games feature elimination of players as the game proceeds, with the ultimate remaining player usually being declared the Winner, in these cases. Modeling and operating a linked list of a length as large as even seven player identities, which could be simple integers, is trivial when dealing with a game tree in which millions of nodes may have to be remembered.

We furthermore observe that the vast majority of list-based ADSs feature very intuitive, constant-time update operations, generally no more than moving the position of the element concerned within the linked list. Even though this update will need to occur at every Min phase of the tree, this is incomparably negligible when we observe that even a simple heuristic is likely to contain far more constant time operations than a data movement operation on a linked list, and which must be performed at every leaf node. We thus conclude that using the Threat-ADS in the worst case, increases the BRS's runtime only very slightly, and does not influence its asymptotic runtime, although it could potentially decrease the runtime significantly.

Perhaps more importantly, the Threat-ADS heuristic does not require that the opponent moves are sorted. While one could naively sort a collection of moves in terms of the opponents' positions in the ADS, in fact, we can simply generate moves for each opponent in the order of their occurrences within the ADS. This is in contrast to a number of very powerful heuristics, such as the History heuristic, where sorting of moves is required, and where this operation can significantly negatively impact runtime [20].

From these facts, we can conclude that one loses essentially nothing in applying the Threat-ADS heuristic, to enhance the BRS.

5 Experiments

The arguments above describe how the Threat-ADS heuristic can be applied to the BRS, and that its application incurs a very low cost in terms of memory and execution time. We have also discussed how it has the potential to achieve a new kind of move ordering, yet unexplored in the literature, i.e., one that is based on multi-player specific qualities. However, in order to confirm that the use of the Threat-ADS is worthwhile, we must provide a demonstration and analysis of how it behaves in the context of real-world game trees. Given the difficulties inherent in the formal analysis of game-playing algorithms [25], we, instead, choose to verify its performance via experimentation.

What we are interested in learning is the improvement gained when using Threat-ADS, to a given search depth within a game tree. While the Threat-ADS is, as discussed above, a game-independent move ordering technique, if we were to only test it in the context of a single game, we would have no information regarding its performance in other games. Thus, we have decided to test it with a range of models, specifically the Virus Game, Chinese Checkers, and Focus (discussed in the following section). Chinese Checkers and Focus are chosen due to the fact that they are both acclaimed multi-player board games, and their use has been confirmed in the paper that originally reported the BRS [19]. They are also contrasted from each other in terms of the objective of the game, as the goal in Chinese Checkers is to race one's opponents to the end state, whereas in Focus it is to capture opponent pieces. The Virus Game, of our own creation, is chosen to serve as a territory control game, as opposed to a piece capturing or racing game, to contrast with these.

Since the Threat-ADS heuristic may be able to retain its knowledge in subsequent turns, we will allow the game to proceed for several turns from the start point. As the Threat-ADS heuristic does not influence the decisions of the BRS, but only its speed of execution, the end result of the game is not a fundamental concern in our experiments. Thus, we will not run the games to termination after this is done. Specifically, we will run the Virus Game for ten turns, Chinese Checkers for five, and Focus for three, which is consistent with our previously presented work.

It would seem intuitive to use CPU time as a measure for these experiments, as the purpose of tree pruning is to reduce the runtime of the Mini-Max algorithm. However, it has been observed in the literature that CPU time can be a problematic metric for these sorts of experiments, as it is prone to be influenced by the platform used and by the specific implementation [20]. We, therefore, make use of the Node Count (NC) measure, introduced by Schaeffer, and which has been found to be highly correlated with runtime [20]. Formally, the NC measure is defined as the sum of internal and leaf nodes at which computation takes place. This excludes those nodes that were considered, but never visited, as an alpha or beta cutoff occurred before that point, at their ancestor. As the NC measure deals only with how the tree is grown, it is not subject to these platform-specific influences.

More specifically, we will take, as our final measure, the sum of the NC over the turns which the game is played for. We run each of these trials fifty times, and report the average NC in each case. We employ the non-parametric Mann-Whitney test in order to determine statistical significance. We have furthermore included the Effect Size measure, expressed as the standardized mean difference between the two data sets, to make the degree of impact from the Threat-ADS more intuitive [2].

We first present an initial set of results (some of which were originally reported in [11, 12]), which serve to justify a deeper analysis of its qualities, in the context of 4-ply trees. We then explore the performance of a range of ADS update mechanisms, and an examination of its performance at 6-ply, in the context of the Virus Game. Focus and Chinese Checkers, unfortunately, have too large of a branching factor to be searched deeper than 4-ply in a reasonable amount of time. We then expand our inquiry to consider *intermediate* board positions, as is explained below and shown in the preliminary version of this paper, before extending to results for a wider range of players.

5.1 Game Models

In this section, we will briefly detail the three game models employed in our experiments, to give the reader a better sense for the objectives and play of the games in question, and to contrast them against each other.

Virus Game: The Virus Game is a multi-player board game of our own creation, modeled after similar experimental games from previous works, based on a biological metaphor [15]. The Virus Game is, in essence, a "territory control" game where players vie for control of squares on a game board of configurable size (in this work, we use a 5x5 board). During his turn, a player may "infect" a square adjacent to one he controls, at which point he claims that square, and each square adjacent to it. The Virus Game is designed primarily as a highly-configurable testing environment, rather than a tactically interesting game, as it is easy for players to cancel each others' moves; however, it shares many elements in common with more complex games. A possible starting position, and an intermediate state of the Virus Game, are shown in Figure 5.

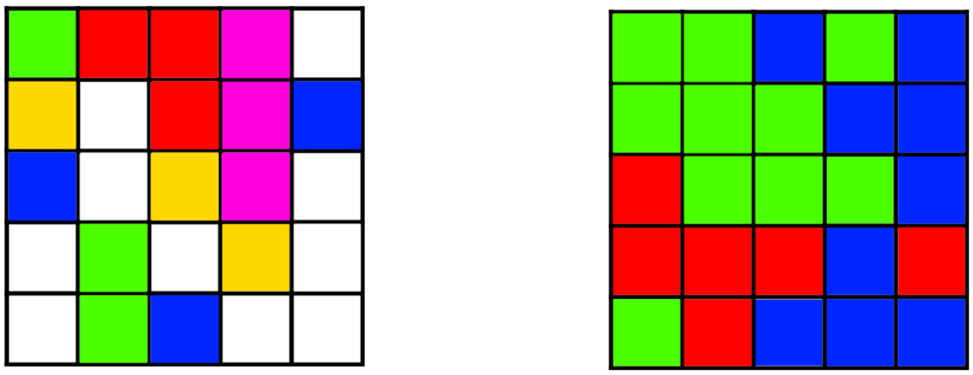

Figure 5: The Virus Game at its initial state, and ten turns into the game. Observe that two players have been eliminated, and the pieces are more closely grouped together.

Focus: Focus is a board game based on piece capturing, designed to be played by two to four players, on an 8x8 board, with the three squares in each corner omitted. The game was originally developed by Sackson in 1969 and released since under many different names [18]. Unlike most other games of its type, Focus allows pieces to be "stacked" on top of each other. The player whose piece is on top of the stack is said to control it, and during his turn, may move one stack he controls, vertically or horizontally, by a number of squares equal to the height of the stack. When a stack is placed on top of another one, they are merged, and all pieces more than five from the top of the stack are removed from the board. If a player captures his own piece, he may place it back on the board in any position, rather than moving a stack. Starting positions for Focus are pre-determined to insure a fair board state. The starting positions for Focus are shown in Figure 6.

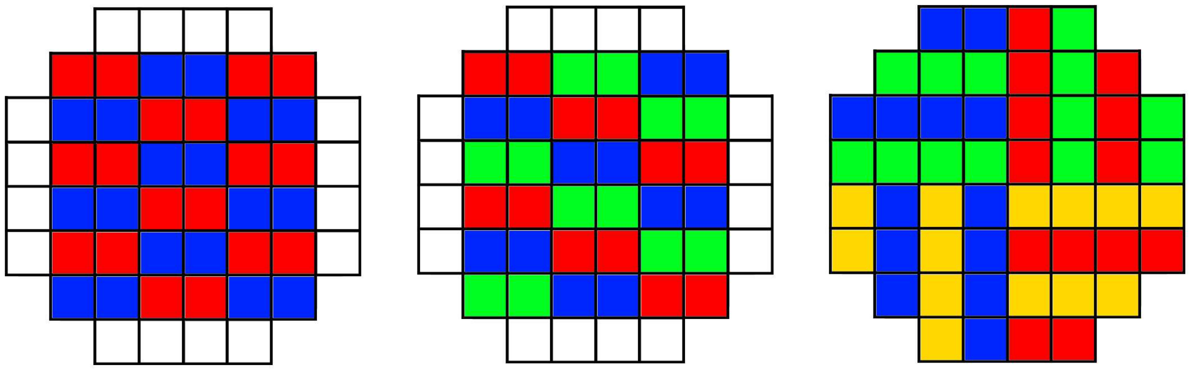

Figure 6: The two, three, and four player starting positions for Focus.

Chinese Checkers: Chinese Checkers is a well-known multi-player board game, played by between two and six players, omitting five players, as it would give one an unfair advantage. The game is played on a star-shaped board, and the objective is to move all of one's pieces to the opposite corner from one's starting position. On his turn, a player may move one of his pieces to one of the adjacent six positions, or "jump" an adjacent piece, which could be either his or his opponent's. As in draughts, jumps may be chained together as many times as possible, allowing substantial distances to be covered in a single move. The possible starting positions for Chinese Checkers are shown in Figure 7.

5.2 Intermediate Board States

Rather than only considering the Threat-ADS heuristic's performance at initial board positions, it is also worthwhile to investigate its performance as an expedient strategy for search from an intermediate, or mid-game, board state. Compared to searching from the initial board state, considering intermediate states is a more difficult and interesting problem, due to a variety of reasons. These include:

- "Opening book" moves are not available at an intermediate board state [7].
- In all non-trivial games, the variability in possible intermediate board states is almost always exponentially large.
- Unlike two-player games, in multi-player games, the number of opponents who pose an adversarial threat in an intermediate board state could be larger than the number that the perspective player faces at the start of the game.
- In some cases, at an intermediate board state, the number of adversarial opponents could have reduced, due to player eliminations.
- Apart from the number of opponents, the identity of the opponents that could potentially threaten the perspective player could change in the intermediate board sate.

To explain the above issues, it is clear that if a game is well known, a game playing engine will, generally, make use of an opening book of strong, known moves that are derived from either expert knowledge, or trained from previous experience with the game [7]. In the case of intermediate game states, however, such

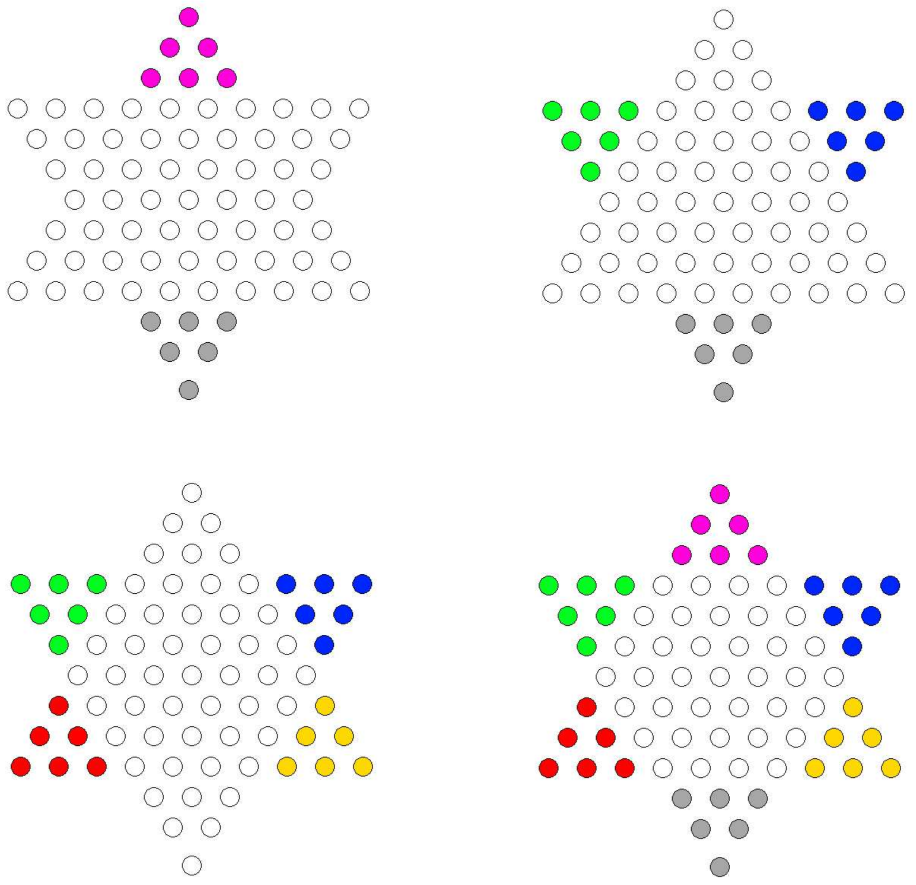

Figure 7: The two, three, four, and six player starting positions for Chinese Checkers.

well-known opening moves are unavailable, and thus the efficiency of the game search is more important. Furthermore, the opening stages of the game have a relatively small degree of variability, compared to the vast number of possible intermediate board states. Thus, when analyzing the performance of move ordering, addressing intermediate board states provides much more powerful results related to its performance.

The problem is accentuated in the multi-player case. In the case of a multi-player game, there are other considerations at work aside from the variability of the intermediate board states. It is typical that in a multi-player game, the winner is determined by the last player remaining in the game, and thus, over time, it is possible that players will be eliminated from play. Considering that the Threat-ADS makes gains by prioritizing the most threatening player, it is possible that the removal of players from contention will have an impact on its performance. Furthermore, depending on the qualities of the game, different opponents may be more threatening at different stages of play. For example, if one opponent has performed very well thus far, he may be more capable of minimizing the perspective player's score and pose a greater threat than he posed earlier in the game.

Generation of Intermediate Board Positions: During turns within which measurement is taking place, other than the perspective player, all opponents made random moves, to cut down on experiment runtime, as we are interested in tree pruning rather than the final state of the game. However, this was not considered to be a valid way to generate intermediate starting board positions, as the intermediate positions would be unrealistic if one player was acting intelligently, and the opponents were acting at random. The perspective player would likely be in an unrealistically advantageous position, compared to his opponents, and their relative threats could thus be heavily impacted.

To address this concern, we progressed the game to an intermediate position by having each player use a simple 2-ply BRS for a pre-specified number of turns, and then switching to a 4-ply BRS for the perspective player, and random behaviour for the opponents, while measurements were taken. The number of turns we advanced into the game in this way was fifteen for the Virus Game, ten for Chinese Checkers, and, given its short duration, three for Focus. This has the advantage of generating a wide range of feasible intermediate board positions, thus insuring the breadth of the experimental results.

6 Proof-of-Concept Results

Table 1 contains our initial, proof-of-concept results [11]. Specifically, it contains initial results for the four-player Virus Game, four-player Focus, and six-player Chinese Checkers, as an initial demonstration of the viability of the Threat-ADS heuristic.

We can see that, considering each game model, the use of the lightweight Threat-ADS heuristic produced a statistically significant reduction in tree size. The degree of the savings varied between each case, with the best being a savings of 12%, in the case of Chinese Checkers. The Effect Size metric varied between 0.49 and 0.83, well within the range of a medium-to-large effect, according to the standard interpretation of the metric [2, 29].

Table 1: Proof-of-Concept Results of the Threat-ADS Heuristic on the Benchmark Games.

Game Model	Threat-ADS?	Avg. NC	Std. Dev	P-Value	Effect Size
Virus Game	No	264,000	$32,000$	-	-
Virus Game	Yes	237,000	$29,000$	3.0×10^{-5}	0.84
Focus	No	6,859,000	$695,000$	-	-
Focus	Yes	6,443,000	$805,000$	1.9×10^{-3}	0.61
Chinese Checkers	No	3,485,000	$850,000$	-	-
Chinese Checkers	Yes	3,070,000	$836,000$	0.014	0.49

Based on these strong initial results, in a range of very different multi-player games, we expand our inquiry to consider a wider range of ADS update mechanisms in the following section.

7 Results for a Variety of Update Mechanisms

Table 2 shows our results for the Virus Game. As Move-Ahead-2 is identical to Move-to-Front when three opponents are present, and POS(k) is identical to the Transposition rule in the same circumstances, we employ the five-player Virus Game in this case, so that the update mechanisms differ. From this table, we see that invoking any of the update mechanisms mentioned in Section 3.1 produces a similar reduction in tree size. However, we do observe a slightly smaller benefit if the Transposition rule is used. The best scenario was for the POS(K) rule where the improvement is 11%.

Table 2: Results for the Virus Game from an Initial Game Position for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	266,000	$30,000$	-	-
Move-to-Front	242,000	$36,000$	1.2×10^{-4}	0.79
Transposition	250,000	$30,000$	6.7×10^{-3}	0.53
Move-Ahead-k	241,000	$40,000$	1.1×10^{-4}	0.80
POS(k)	237,000	$29,000$	$< 1.0 \times 10^{-5}$	0.96

Table 3 shows our results for Focus. Again, all the update mechanisms structures do well, and roughly display the same performance. As Focus is at most a four-player game, we see identical values for the Move-Ahead-k and Move-to-Front, as well as for the POS(k) and Transposition, whose results have been included in the interest of completeness, and to maintain consistency and readability between the tables (marked with a "*"). The best scheme was the Transposition rule, with a 7% reduction in tree size.

Table 3: Results for Focus from an Initial Game Position for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	6,859,000	$695,000$	-	-
Move-to-Front	6,443,000	$805,000$	1.9×10^{-3}	0.61
Transposition	6,376,000	$572,000$	2.2×10^{-3}	0.70
Move-Ahead-k*	6,443,000	$805,000$	1.9×10^{-3}	0.61
POS(k)*	6,376,000	$572,000$	2.2×10^{-3}	0.70

We display our results for the four-player Chinese Checkers case in Table 4, where we observe a slightly better performance from the Move-to-Front update mechanism, compared to the Transposition rule. However, we emphasize again that all techniques generated an improvement. The best strategy was the Move-to-Front, generating a 14% reduction in tree size.

Table 4: Results for the 4-player Chinese Checkers from an Initial Game Position for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	1,380,000	$379,000$	-	-
Move-to-Front	1,192,000	$337,000$	5.4×10^{-3}	0.50
Transposition	1,248,000	$398,000$	0.02	0.35
Move-Ahead-k*	1,192,000	$337,000$	5.4×10^{-3}	0.50
POS(k)*	1,248,000	$398,000$	0.02	0.35

Consider Table 5, where we present our results for six-player Chinese Checkers. Again, each technique generates an improvement. However, this time the improvement from using the Move-to-Front was highest at a 12% improvement, and it is noticeably superior to the Transposition rule and POS(k).

Table 5: Results for the 6-player Chinese Checkers scenario from an Initial Game Position for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	3,485,000	$850,000$	-	-
Move-to-Front	3,070,000	$836,000$	0.01	0.49
Transposition	3,267,000	$925,000$	0.07	0.26
Move-Ahead-k*	3,149,000	$889,000$	0.04	0.40
POS(k)*	3,260,000	$927,000$	0.08	0.26

As we have observed very consistent performance from the Threat-ADS under a range of update mechanisms, we now examine its performance in intermediate board positions, as described earlier.

8 Results for Intermediate Board States

Consider Table 6, where we present our results for the Virus Game at an intermediate board state. We observe that, as was the case in the initial board state, all ADSs produced a statistically significant improvement in pruning, ranging from a 5% – 10% reduction in NC. The Move-to-Front and Transposition performed the best, and equally well, in this case. The Effect Sizes ranged between approximately 0.5 and 0.9, with the majority near 0.8.

Table 6: Results for the 4-ply Virus Game at an Intermediate Board State for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	303,000	$35,000$	-	-
Move-to-Front	275,000	$40,000$	2.5×10^{-4}	0.82
Transposition	275,000	$37,000$	6.0×10^{-5}	0.87
Move-Ahead-k	280,000	$30,000$	1.4×10^{-4}	0.72
POS(k)	286,000	$40,000$	5.7×10^{-3}	0.56

Table 7 shows our results for Focus at an intermediate board state. In this case, again, all adaptive update mechanisms produced a statistically significant improvement in tree pruning, which was a very large reduction of roughly 20% for all cases. However, here the Effect Size lingered around 0.45, given the much-larger variance of the datasets.

Our results for four-player Chinese Checkers, at an intermediate board state, are shown in Table 8. Here we saw a statistically significant improvement from the Move-to-Front/Move-Ahead-k rule (a 13% reduction in tree size). However, while the average NC was reduced, the Transposition/POS(k) fell outside 95% certainty.

Table 7: Results for Focus at an Intermediate Board State for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	16,767,000	$7,161,000$	-	-
Move-to-Front	13,592,000	$5,240,000$	0.01	0.44
Transposition	13,671,000	$4,240,000$	0.03	0.43
Move-Ahead-k*	13,592,000*	$5,240,000$	0.01	0.44
POS(k)*	13,671,000*	$4,240,000$	0.03	0.43

Table 8: Results for four-player Chinese Checkers at an Intermediate Board State for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	3,458,000	$905,000$	-	-
Move-to-Front	3,024,000	$816,000$	0.01	0.46
Transposition	3,206,000	$798,000$	0.12	0.26
Move-Ahead-k*	3,024,000*	$816,000$	0.01	0.46
POS(k)*	3,206,000*	$798,000$	0.12	0.26

Table 9 shows our results for six-player Chinese Checkers, again at the intermediate state. In this case, Threat-ADS had a smaller impact than the others. However, we do observe that the average NC is lower when Threat-ADS is employed in all cases. The best performance was obtained from the Transposition rule, with an 8% reduction in tree size, which fell within 90% certainty.

Table 9: Results for six-player Chinese Checkers at an Intermediate Board State for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	8,168,000	$2,560,000$	-	-
Move-to-Front	7,677,000	$2,280,000$	0.18	0.30
Transposition	7,494,000	$1,670,000$	0.09	0.26
Move-Ahead-k	7,644,000	$1,830,000$	0.21	0.20
POS(k)	7,975,000	$2,190,000$	0.44	0.08

Having verified the Threat-ADS heuristic's performance in intermediate board positions, we examine its performance at deeper ply depths in the next section, at both initial and intermediate board states.

9 Results for Deeper Ply Depths

Table 10 presents our results for the Virus Game, from an initial board position, at a 6-ply search depth. We observe that the best performance was found from the Move-Ahead-2 technique, with a 15% reduction in tree size. However, consistent with previous results, all update mechanisms performed roughly equally well, and compared to the 4-ply case, with a higher degree of impact.

Table 10: Results for the 6-ply Virus Game from an Initial Game Position for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	11,760,000	$1,430,000$	-	-
Move-to-Front	10,030,000	$1,530,000$	$< 1.0 \times 10^{-5}$	1.21
Transposition	10,250,000	$1,620,000$	$< 1.0 \times 10^{-5}$	1.05
Move-Ahead-k*	10,010,000	$1,430,000$	$< 1.0 \times 10^{-5}$	1.22
POS(k)*	10,200000	$1,260,000$	$< 1.0 \times 10^{-5}$	1.09

Table 11 holds our results for the 6-ply Virus Game at an intermediate board position. The best performance was again from Move-Ahead-k tied with the Transposition rule in this case, with a 10% reduction in tree size.

Table 11: Results for the 6-ply Virus Game from an Intermediate Game Position for the Various ADSs Examined.

Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
None	12,470,000	$1,710,000$	-	-
Move-to-Front	11,410,000	$1,320,000$	1.5×10^{-3}	0.65
Transposition	11,280,000	$1,130,000$	8.4×10^{-4}	0.70
Move-Ahead-k*	11,280,000	$1,130,000$	5.2×10^{-4}	0.69
POS(k)*	11,300,000	$1,710,000$	1.2×10^{-3}	0.68

Extending our analysis from the preliminary version of this paper to a new domain, we present our results for a variable number of players in the next section.

10 Results for a Variable Number of Players

Table 12 presents our results for the Virus Game with a range of players, where measurements were taken from the initial board state, when the ADS used was the Move-to-Front rule. We observe that the Threat-ADS is capable of producing a consistent reduction in tree size in all cases, with a slightly lower impact in the case of three players. Our best results are in the four player case, with a 10% reduction in the tree size.

Table 13 showcases our results for the Virus Game, considering a range of players, at an intermediate board state. Unlike in the case of the initial board position, here the three player case did as well as the others, with our best results coming from the six player case, which resulted in a 10% reduction in tree size, again.

Consider Table 14, where we present our results for Focus for both three and four player games (rules do not exist for a larger number of players in Focus), from the initial board state. We see similar performance

Table 12: Results for the Virus Game with a Range of Players at an Initial Board State where the ADS Used was the MTF Rule.

Number of Players	Threat-ADS?	Avg. NC	Std. Dev	P-Value	Effect Size
Three	No	219,000	$34,000$	-	-
Three	Yes	201,000	$25,000$	1.5×10^{-4}	0.50
Four	No	264,000	$32,000$	-	-
Four	Yes	237,000	$29,000$	3.0×10^{-5}	0.84
Five	No	266,000	$30,000$	-	-
Five	Yes	242,000	$31,000$	9.0×10^{-5}	0.79
Six	No	284,000	$34,000$	-	-
Six	Yes	264,000	$33,000$	2.3×10^{-3}	0.79

Table 13: Results for the Virus Game with a Range of Players at an Intermediate Board State where the ADS Used was the MTF Rule.

Number of Players	Threat-ADS?	Avg. NC	Std. Dev	P-Value	Effect Size
Three	No	303,000	$37,000$	-	-
Three	Yes	275,000	$30,000$	$< 1.0 \times 10^{-5}$	0.77
Four	No	307,000	$38,000$	-	-
Four	Yes	287,000	$27,000$	6.6×10^{-4}	0.53
Five	No	306,000	$35,000$	-	-
Five	Yes	277,0000	$40,000$	2.5×10^{-4}	0.82
Six	No	299,000	$35,000$	-	-
Six	Yes	268,000	$33,000$	$< 1.0 \times 10^{-5}$	0.89

from the Threat-ADS in both cases, with slightly greater improvements in the four player case, leading to a 6% reduction in tree size.

Table 14: Results for Focus with a Range of Players at an Initial Board State where the ADS Used was the MTF Rule.

Number of Players	Threat-ADS?	Avg. NC	Std. Dev	P-Value	Effect Size
Three	No	5,290,000	$499,000$	-	-
Three	Yes	5,000,000	$435,000$	1.5×10^{-3}	0.60
Four	No	6,860,000	$695,000$	-	-
Four	Yes	6,440,000	$805,000$	1.9×10^{-3}	0.61

Contrasting the above, Table 15 shows our results for Focus for both three and four player games at an intermediate board position. The four player case again performed better, with a 19% reduction in tree size.

Table 16 holds our results for Chinese Checkers with a range of players. Under normal rules, Chinese

Table 15: Results for Focus with a Range of Players at an Intermediate Board State where the ADS Used was the MTF Rule.

Number of Players	Threat-ADS?	Avg. NC	Std. Dev	P-Value	Effect Size
Three	No	15,160,000	$3,370,000$	-	-
Three	Yes	13,180,000	$2,550,000$	1.8×10^{-3}	0.59
Four	No	16,770,0000	$7,161,000$	-	-
Four	Yes	13,590,000	$5,235,000$	0.01	0.44

Checkers cannot be played fairly with five players, as one player would have no opponent across from him - which is an incredible advantage. However, as a means to test the performance of the Threat-ADS, we still consider it meaningful and interesting, and thus include both those cases where the perspective player has the advantage, and when he instead has an empty space next to him. Threat-ADS generated the greatest reduction in the five player case when the perspective player had a disadvantage, with a 20% reduction in tree size, although Threat-ADS produced no improvements in the three player case. This will be discussed in detail in the next section.

Table 16: Results for the Chinese Checkers with a Range of Players at an Initial Board State where the ADS Used was the MTF Rule.

Number of Players	Threat-ADS?	Avg. NC	Std. Dev	P-Value	Effect Size
Three	No	732,000	$182,000$	-	-
Three	Yes	745,000	$148,000$	0.18	0.07
Four	No	1,380,000	$379,000$	-	-
Four	Yes	1,190,000	$337,000$	5.4×10^{-3}	0.50
Five (Adv)	No	2,420,000	$589,000$	-	-
Five (Adv)	Yes	2,190,000	$872,000$	9.3×10^{-3}	0.38
Five (Dis)	No	2,500,000	$896,000$	-	-
Five (Dis)	Yes	2,010,000	$596,000$	0.011	0.45
Six	No	3,490,000	$850,000$	-	-
Six	Yes	3,070,000	$836,000$	0.014	0.49

Lastly, Table 17 holds our results for Chinese Checkers at an intermediate board state, with a range of players. We observe similar results, compared to the above table, however the four player case performed best, with a 13% reduction in tree size, and the disadvantaged five player case did not generate as drastic a reduction in tree size. Again, the three player case led to no improvement. These patterns of behaviour will be explored in the next section.

Table 17: Results for the Chinese Checkers with a Range of Players at an Intermediate Board State where the ADS Used was the MTF Rule.

Number of Players	Threat-ADS?	Avg. NC	Std. Dev	P-Value	Effect Size
Three	No	1,250,000	$400,000$	-	-
Three	Yes	1,260,000	$378,000$	0.42	0.02
Four	No	3,490,000	$905,000$	-	-
Four	Yes	3,020,000	$816,000$	0.01	0.46
Five (Adv)	No	6,120,000	$1,810,000$	-	-
Five (Adv)	Yes	5,470,000	$1,570,000$	0.017	0.36
Five (Dis)	No	7,170,000	$896,000$	-	-
Five (Dis)	Yes	6,490,000	$596,000$	0.011	0.45
Six	No	8,170,000	$2,460,000$	-	-
Six	Yes	7,680,000	$2,040,000$	0.099	0.27

11 Discussion

Our results presented in this work clearly demonstrate that the Threat-ADS is able to generate statistically significant savings, in terms of tree size, in a wide variety of cases, with a range of ADS update mechanisms, and with a variable number of players involved in the game. We have also demonstrated that the Threat-ADS heuristic is able to achieve consistent performance in the case of intermediate board positions, as well as maintaining or achieving even better performance in the case of deeper ply depths, at least in the context of the Virus Game.

We found that the majority of update mechanisms performed roughly equally, with only small changes, which can be statistically ascribed to chance. The exception is in the case of Chinese Checkers, in the six-player scenario, where only the Move-to-Front and Move-Ahead-k techniques produced a statistically significant change in tree size. These techniques also performed noticeably better in the four-player scenario, especially at an intermediate board state (they are semantically identical in this case). Also, compared to the Transposition rule, the Move-to-Front produces a much greater change in the list's structure, it may be that Chinese Checkers responds better to this increased sensitivity. However, given that POS(k) can produce large changes as well, and the Transposition rule performed slightly better in the case of intermediate board positions, we cannot confidently conclude this from our results. What is apparent is that it is the use of an adaptive list in general, more than the specific update mechanism, that is responsible for the Threat-ADS heuristic's benefits.

The Threat-ADS' performance remains strong in the context of intermediate board states, in the vast majority of cases, or can actually outperform itself compared to the initial board position, despite the difficulties of intermediate board positions described earlier. These results demonstrate that the Threat-ADS can indeed serve as an effective searching strategy for the problem of intermediate board states. Given that the games explored vary substantially from each other, being based on piece capturing, racing, and territory control for Focus, Chinese Checkers, and the Virus Game, respectively, we hypothesize that the Threat-ADS will continue to perform well in other multi-player games too.

When considering the Virus Game, the behaviour and benefits of the Threat-ADS were extremely consistent across all cases, with a generally 10% reduction in tree size. Slightly weaker performance was observed in the three player Virus Game, and only when measurements were taken from an initial board position. Given that the Virus Game begins with each player controlling a few spaces on a largely empty board, we suspect that this is caused by players simply being too far from each other to substantially threaten one another, before a number of turns have passed. This is supported by the fact that the three player Virus Game's results were consistent with all other cases when an intermediate board state is considered.

When the Virus Game is played to a depth of 6-ply, we observed that performance did not decrease due to the deeper search depth, and in fact, in the initial board position case, it performed noticeably better. While the same strong difference was not observed when results were recorded from an intermediate board state, the Threat-ADS heuristic did not perform any worse than it did in a 4-ply game tree. This is a powerful result, as it is known that certain well-known move ordering techniques, such as the History Heuristic, decay in efficacy in very deep trees [21].

In the case of Focus, we observe that the savings from the Threat-ADS were actually substantially larger when intermediate board states were considered, with a 20% average reduction as opposed to a 7% reduction in tree size. We suspect the reason for this is that, after some time is passed, it is likely that one or more of the opponents will have performed well in the opening moves of the game, and thus pose a greater threat to the perspective player. By prioritizing that opponent, the Threat-ADS is therefore able to make substantial savings, whereas at the beginning of the game, all players start at equal footing. This demonstrates that, in fact, the benefits of the Threat-ADS can improve over the course of the game, generating greater savings as time progresses.

The game of Chinese Checkers generated the most variable performance from the Threat-ADS heuristic, with differing numbers of players, in particular, having a strong impact on the savings it was capable of achieving. Specifically, when four or five players were present, the Threat-ADS performed very well, however in the three and six player cases, its savings were noticeably less. The Threat-ADS was only able to produce gains outside 95% certainty in the case of six player Chinese Checkers, at an intermediate board state, however it was still capable of producing reductions in NC. We suspect, therefore, this is in part due to the high variance in intermediate board states in Chinese Checkers, and the fact that many different players' pieces are located throughout the center of the board in the midgame, reducing the difference in threats between opponents.

In the case of three player Chinese Checkers, no benefit for the Threat-ADS was observed at all. However, this fits with intuitive expectations, as players begin play some distance from each other in the three player case, and with no opponent adjacent to or across from the player, each of the two opponents has an equal likelihood of minimizing the perspective player. By contrast, the Threat-ADS heuristic produced stronger results for four player Chinese Checkers, where a single opponent is adjacent to the perspective player and thus can more easily hamper his movements, supporting this reasoning.

The Effect Size provides an easily-understood metric for demonstrating the degree of impact that a new technique has, in a way that is immediately obvious to the reader. While these rules are not universal, an Effect Size of 0.2 is considered to be small, 0.5 to be medium, and 0.8 to be large [2]. Given that our best-performing cases are normally between 0.5 and 0.8 or larger, we conclude that Threat-ADS' contribution to tree pruning is not coincidental, or a result of over-sampling.

Although the savings from the Threat-ADS heuristic may appear minimal in comparison to techniques that employ far larger data structures and mechanisms, such as the History heuristic, the Threat-ADS heuristic can achieve its benefits with a very small costs, and furthermore does not require that moves be sorted, as was discussed earlier. Lastly, given that the Threat-ADS heuristic presents a different, and novel, form of move ordering (that of employing relative opponent threats to achieve ordering), compared to existing techniques, it can easily be combined with existing strategies.

12 Conclusions, Contributions and Future Work

In this paper, we introduced the novel approach of employing techniques inspired from the formerly-unrelated field of ADSs to game playing. The proposed scheme, which we have named the Threat-ADS heuristic, achieves move ordering in a multi-player environment, by employing the qualities of ADSs to rank opponents, based on their relative threats. A wide range of experiments in this work have demonstrated that the Threat-ADS heuristic is capable of achieving significant gains in terms of tree pruning in a very wide range of cases, including consideration of multiple ADS update mechanisms, initial and intermediate board states, game trees of varying search depths, and variation on the number of players in the game.

Our results indicate that, considering a number of well-known ADS update mechanisms, similar performance gains were observed independent of the strategy employed, suggesting that it is the use of an adaptive list in the first case that leads to the majority of gains from the Threat-ADS heuristic, and the specific update mechanism is less critical. Furthermore, we observed that the Threat-ADS heuristic tends to perform to the same degree of efficacy in intermediate states, as it does in the corresponding initial board position case, and in some cases actually performs better later in the game, as we observed with Focus. Similarly, it can achieve better performance in deeper, more complex trees, as opposed to shallow ones.

When considering variations on the number of players, we found the most important factor in whether the Threat-ADS performs any differently is not, in fact, the number of available threats, but rather differences in board positions for varying number of players. This is most clear in the case of Chinese Checkers, as certain player arrangements allow for specific players to minimize each other more or less than others, and this is not the case all the time. Compared to the Virus Game, where the game is played similarly with any number of players, we found that the Threat-ADS performed equally well in almost all cases, in both initial and intermediate states. We can thus conclude that the Threat-ADS heuristic remains expedient for multi-player games with a wide range of opponents.

The Virus Game, Focus, and Chinese Checkers are different enough from each other, in terms of rules and strategies, that one can reasonably hypothesize that the Threat-ADS will remain viable under other game models. However, we have currently not explored its function in a very wide set of games, and thus exploration of a larger set of games is a potential future research direction. We also have not explored the performance of the Threat-ADS heuristic in the context of very deep game trees, due to limitations on hardware resources, and its viability as an expedient strategy in that environment remains unproven. Given its achievements under a range of parameters, however, we believe this is another worthwhile avenue of future research.

More than a single move ordering strategy, what the Threat-ADS heuristic's consistent results, with its inexpensive cost, demonstrates is the applicability and potential benefits of ADS-based techniques to game playing as a whole. Indeed, we are currently working on ADS-based move ordering strategies for two-player games based on this work, and preliminary results show a great deal of promise [13]. Our sincere hope is that this work will inspire others and provide a basis for further exploration of ADS-based techniques within area of game playing.

References

[1] S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms*, pages 13–51, 1998.

[2] R. Coe. It's the effect size, stupid: What effect size is and why it is important. In *Annual Conference of the British Educational Research Association*, University of Exeter, Exeter, Devon, 2002.

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

[4] V. Estivill-Castro. Move-To-End is best for double-linked lists. In *Proceedings of the Fourth International Conference on Computing and Information*, pages 84–87, 1992.

[5] 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*, pages 169–171, 1979.

[6] J. H. Hester and D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17:285–311, 1985.

[7] M. Levene and J. Bar-Ilan. Comparing typical opening move choices made by humans and chess engines. *Computing Research Repository*, 2006.

[8] C. Luckhardt and K. Irani. An algorithmic solution of n-person games. In *Proceedings of the AAAI'86*, pages 158–162, 1986.

[9] B. J. Oommen and D. C. Y. Ma. Deterministic learning automata solutions to the equipartitioning problem. *IEEE Transactions on Computers*, 37, 1988.

[10] S. Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, 2008.

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

[12] 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*, pages 164–169, 2013.

[13] S. Polk and B. J. Oommen. Enhancing history-based move ordering in game playing using adaptive data structures. In *Proceedings of ICCCI'2015, the 7th International Conference on Computational Collective Intelligence Technologies and Applications*, pages 225–235, 2015.

[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*, pages 33–42, 2015.

[15] P. Rendell. A universal Turing machine in Conway's Game of Life. In *Proceedings of HPCS'11, the 2011 International Conference on High Performance Computing and Simulation*, pages 764–772, 2011.

[16] R. L. Rivest. On self-organizing sequential search heuristics. In *Proceedings of the 1974 IEEE Symposium on Switching and Automata Theory*, pages 63–67, 1974.

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

[18] S. Sacksin. *A Gamut of Games*. Random House, 1969.

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

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

[21] E. Schrder. Move ordering in rebel. Discussion of move ordering techniques used in REBEL, a powerful chess engine, 2007.

[22] C. E. Shannon. Programming a computer for playing Chess. *Philosophical Magazine*, 41:256–275, 1950.

[23] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[24] N. Sturtevant. A comparison of algorithms for multi-player games. In *Proceedings of the Third International Conference on Computers and Games*, pages 108–122, 2002.

[25] N. Sturtevant. *Multi-Player Games: Algorithms and Approaches*. PhD thesis, University of California, 2003.

[26] N. Sturtevant and M. Bowling. Robust game play against unknown opponents. In *Proceedings of AAMAS'06, the 2006 International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 713–719, 2006.

[27] N. Sturtevant, M. Zinkevich, and M. Bowling. Prob-Maxn: Playing n-player games with opponent models. In *Proceedings of AAAI'06, the 2006 National Conference on Artificial Intelligence*, pages 1057–1063, 2006.

[28] I. Szita, C. Guillame, and P. Spronck. Monte-carlo tree search in settlers of catan. In *Proceedings of ACG'09, the 2009 Conference on Advances in Computer Games*, pages 21–32, 2009.

[29] A. Turner and J Miller. The importance of topology evolution in neuroevolution: A case study using cartesian genetic programming of artificial neural networks. In *In Proceedings of AI'2013, the Thirty-Third SGAI Conference on Artificial Intelligence*, pages 213–226, 2013.

[30] I. Zuckerman, A. Felner, and S. Kraus. Mixing search strategies for multi-player games. In *Proceedings of IJCAI'09, the Twenty-first International Joint Conferences on Artificial Intelligence*, pages 646–651, 2009.