Enhancing *History*-based Move Ordering in Game Playing Using Adaptive Data Structures

Spencer Polk and B. John Oommen

School of Computer Science, Carleton University, Ottawa, Canada
andrewpolk@cmail.carleton.ca, oommen@scs.carleton.ca**

Abstract. This paper pioneers the avenue of enhancing a well-known paradigm in game playing, namely the use of *History*-based heuristics, with a totally-unrelated area of computer science, the field of Adaptive Data Structures (ADSs). It is a well-known fact that highly-regarded game playing strategies, such as alpha-beta search, benefit strongly from proper move ordering. From this perspective, the *History* heuristic is, probably, one of the most acclaimed move ordering techniques, that could enhance an alpha-beta search. Recently, the authors of this present paper have shown that techniques derived from the field of ADSs, which are concerned with query optimization in a data structure, can be applied to move ordering in multi-player games. This was accomplished by ranking opponent threat levels. The work presented in this paper seeks to extend the utility of ADS-based techniques to two-player and multi-player games, through the development of a new move ordering strategy that incorporates the *historical* advantages of the moves. The resultant technique, the History-ADS heuristic, has been found to produce substantial (i.e, even up to 70%) savings in a variety of two-player and multi-player games, at varying ply depths, and at both initial and midgame board states. As far as we know, results of this nature have not been reported in the literature before.

1 Introduction

The problem of intelligently playing a game against a human player using AI techniques has been studied extensively, leading to many well-known techniques, such as the alpha-beta search scheme, capable of achieving excellent performance in a wide variety of games [1]. The performance of alpha-beta search can be further improved by proper move ordering, which can lead to more efficient tree pruning, and thus permit deeper search or a farther look-ahead in the game [2, 3]. Specifically, the best pruning is obtained when the best move is searched first, thus leading to a wide range of move ordering heuristics, such as the Killer Moves and the History heuristic, that seek to obtain this arrangement in an environment where perfect information is naturally unavailable [3, 4].

** *Chancellor's Professor*; *Fellow: IEEE* and *Fellow: IAPR*. The second author is also an *Adjunct Professor* with the Dept. of ICT, University of Agder, Grimstad, Norway.

Originally unrelated to game playing, the field of ADSs deals with the dynamic optimization of a data structure based on the object access frequencies [5, 6]. ADSs attempt to solve the problem of the uneven accesses of objects by reorganizing them in response to queries. The field provides a wide range of methods to accomplish this, such as the Move-to-Front and Transposition rules for adaptive lists [7]. As the objective is to improve the performance of the data structure, these operations, by necessity, must be inexpensive to implement [7].

Previously, the authors of this present paper, demonstrated that techniques derived from ADSs had applications in game playing, particularly within the context of move ordering, leading to the development of the Threat-ADS heuristic [8] for multi-player games. The Threat-ADS heuristic operates within the context of the Best-Reply Search (BRS) for *multi*-player games, where opponents' moves are grouped together to form a "super-opponent", who minimizes the perspective player's gains [9]. The Threat-ADS places the opponents within an adaptive list, and groups opponent moves based on the list, achieving statistically significant improvements in terms of tree pruning in a wide variety of cases [8, 10]. Based on the success of the Threat-ADS, our submission is that it is worthwhile to investigate if ADSs can achieve improvements to move ordering using a metric other than opponent threats. This is the focus of this work.

The rest of the paper is laid out as follows. Section 2 describes, in more detail, the motivation for this research endeavour. Section 3 details the History-ADS, our new technique described in this work. Section 4 describes the experiments we have performed to demonstrate the History-ADS' capabilities. Section 5 presents the results from the experiments we have conducted, and Section 6 contains the discussion and analysis of the results we have obtained. Finally, Section 7 concludes the paper.

2 Motivation

The well-known and historically successful Killer Moves and History heuristics, among others, make use of the concept of move history to achieve move ordering [3]. Specifically, they are based on the hypothesis that if a move produced a cut earlier in the search of the game tree, it is likely to be a strong move if it is encountered again, and it should thus be searched first. The proven performance of these techniques shows that this hypothesis is valid, and we thus know that move history is a metric by which move ordering can be achieved.

The previous success of the Threat-ADS heuristic demonstrates that ADS-based techniques can provide tangible benefits, in terms of move ordering, within game tree searches. Given that it can achieve this using opponent threats, and move history is known to be a worthwhile metric for move ordering, it is reasonable to believe that ADSs can use move history as well as opponent threats. This can intuitively be achieved by having an ADS hold moves, rather than opponents, and update its structure when a move produces a cut, thereby keeping moves that frequently produce cuts near the head of the data structure.

An ADS-based technique using move history would provide a number of benefits including:

- Provide an inexpensive move-ordering strategy for a wide variety of games;
- Demonstrate another metric, aside from opponent threats, that ADSs can employ to achieve move ordering;
- Be applicable to two-player games, along with multi-player games, unlike the Threat-ADS heuristic;
- Potentially achieve greater savings in terms of tree pruning, compared to the Threat-ADS, given that many more possible moves exist in a game than opponents, in all but the most trivial of cases.

The development of a heuristic that accomplishes this, which we have named the History-ADS heuristic, is described in the next section.

3 The History-ADS Heuristic

3.1 Developing the History-ADS Heuristic

The development of the History-ADS heuristic is achieved by explaining the analogous operations of the Threat-ADS heuristic [8]. First of all, we must understand how to update the ADS at an appropriate time, i.e., through querying it with the identity of a move that has led to a cut. The identity of such a move will vary depending on the game, however it would normally be represented in the same way as it is in the context of the Killer Moves or the History heuristics. When the ADS is queried with a move, the move is repositioned according to its update mechanism, or added to the data structure if it is not already present. This is analogous to querying the ADS with the most threatening opponent within the context of the Threat-ADS. Then, at each Max and Min node, we must somehow order the moves based on its order within the ADS.

Fortunately, within the perspective of alpha-beta search, there is a very intuitive location to query the ADS, which is where an alpha or a beta cutoff occurs, before terminating that branch of the search. To actually accomplish the move ordering, when we expand a node and gather the available moves, we explore them in the order proposed by the ADS, as alluded to before.

The last issue that must be considered is that a move that produces a cut on a Max node may not be likely to produce a cut on a Min node, and vice versa. This would occur, for example, if the perspective player and the opponent do not have analogous moves, such as in Chess or Checkers, or if they are some distance from each other. We thus employ two list-based ADSs within the History-ADS heuristic, one of which is used on Max nodes, while the other is used on Min nodes. This parallels how the History heuristic maintains separate values for each player [3].

To clarify matters, a simple example of how the History-ADS updates its data structures and applies that knowledge to achieve move ordering is provided in Figure 1. The formal execution of the enhanced alpha-beta search is provided in Algorithm 1.1. The reader must observe that the enhanced algorithm merely adds a couple of extra lines of pseudo-code to the original alpha-beta search scheme. The efficiency of this inclusion and the marginal additional time footprint encountered, is clear.

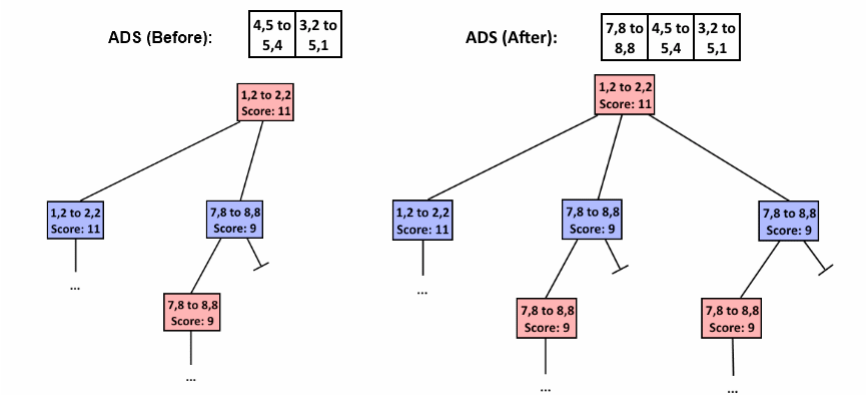

Fig. 1. A demonstration of how an ADS can be used to manage move history over time. The move (7,8) to (8,8) produces a cut, and so it is moved to the head of the list, and informs the search later.

Algorithm 1.1 Mini-Max with History-ADS

Function BRS(node, depth, player)

if node is terminal or depth ≤ 0 **then**
  return heuristic value of node
else
  if node is max then
    for all child of node in order of MaxADS do
      $\alpha = max(\alpha, minimax(child, depth - 1)$
      if $\beta \leq \alpha$ **then**
        break (Beta cutoff)
        query MaxADS with cutoff move
      end if
    end for
    return α
  else
    for all child of node in order of MinADS do
      $\beta = min(\alpha, minimax(child, depth - 1)$
      if $\beta \leq \alpha$ **then**
        break (Alpha cutoff)
        query MinADS with cutoff move
      end if
    end for
    return β
  end if
end if

End Function Mini-Max with History-ADS

3.2 Features of the History-ADS Heuristic

As the reader will observe, the History-ADS heuristic is very similar, in terms of its design, construction and implementation, to the Threat-ADS heuristic. Like the Threat-ADS heuristic, it does not in any way alter the final value of the tree, a trait common to many schemes that enhance tree pruning. As in the case of the Threat-ADS, the ADSs are added to the algorithm's memory footprint, and their update mechanisms must be considered with regard to its running time.

Compared to the Threat-ADS, the History-ADS can be expected to employ a much larger data structure, as there will be many more possible moves than total opponents in any non-trivial game. One must further observe that, within its ADSs, the History-ADS remembers any move that produces a cut, for the entire search, even if it never produces a cut again and lingers near the end of the list. However, as duplicate moves are never added to the list, the maximum possible length is equal to the number of viable moves, and thus the History-ADS heuristic's memory requirements are bound by those of the highly-regarded History heuristic. Furthermore, as it only maintains memory on moves that have, indeed, produced a cut, it is very likely that the size of the data structures will be far less than the number of possible moves.

However, unlike the History heuristic, which requires moves to be sorted based on the values in its arrays, the History-ADS shares the advantageous quality of the Threat-ADS in that it does not require sorting. One can simply explore moves, if applicable, in the order specified by the ADS, thus allowing it to share the strengths of the Killer Moves heuristic, while simultaneously maintaining information on all those moves that have produced a cut.

Lastly, unlike the Threat-ADS, which was specific to the BRS, the History-ADS works within the context of the two-player Mini-Max algorithm. However, since the BRS views a multi-player game as a two-player game by virtue of it treating the opponents collectively as a single grouped entity, the History-ADS heuristic is also applicable to it, and it thus functions in both two-player and multi-player contexts. We will thus be investigating its performance in both these avenues in the following sections.

4 Experimental Model

Given the similarities between the two techniques (i.e., the History-ADS and the Threat-ADS heuristic), we consider it useful to employ a similar set of experiments in analyzing their performances. This will permit a fair comparison of the schemes on a level playing field. We are interested in learning the improvement gained from using the History-ADS heuristic, when compared to a search that does not employ it. We accomplish this by taking an aggregate of the Node Count (NC) over several turns. The NC measure is defined as the number of nodes that are expanded during the search, i.e. excluding those generated but then pruned before being visited. Historically, this metric has been shown to be highly correlated to runtime, while also being platform-agnostic [3]. For a vari-

ety of games, we average this value over fifty trials, with different ADS update mechanisms, and varying game states.

As in our previous work, we will employ the Virus Game, Focus, and Chinese Checkers when considering the multi-player case. Focus and Chinese Checkers are both well-known multi-player games, and the Virus Game is a territory control game described in detail in [8]. However, since the History-ADS is applicable to two-player games as well, we require an expanded set of games to handle these. We have opted to use the two-player version of Focus, Othello, and the very well-known Checkers, or Draughts. However, the requirement in Checkers that forces jumps when possible, often leads to a game with a very small branching factor. This factor can vary greatly especially in different midgame states. 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 solved, it still serves as a useful testing environment for the general applicability of a domain-independent strategy, given how well-known and documented the game is in the literature [11].

In testing the Threat-ADS heuristic, we had varied the update mechanism used, the ply depth of the search, and most recently, the starting position of the game, to test its performance in midgame states [12]. We have elected to perform a subset of these experiments, using the History-ADS heuristic, to provide as much information about its performance as possible. Specifically, we compare the Move-to-Front and Transposition update mechanisms, perform trials at both initial and midgame board states, and vary the depth of the search between games. Trials with Othello, Relaxed Checkers, and the Virus Game are done to a 6-ply depth, and trials with Chinese Checkers, and both variants of Focus to a 4-ply depth. Midgame states were generated by playing games a set number of turns using intelligent players, then taking measurements from this position. The details of how midgame states are generated is described in detail in [12].

The number of turns we aggregate the NC over is five for Relaxed Checkers, Othello, and Chinese Checkers, three for Focus, and ten for the Virus Game. To determine statistical significance, we employ the Mann-Whitney test, as we do not make the assumption of normalcy. We also provide the Effect Sizes – an easily-readable indication of the degree of savings given by the History-ADS.

Our results are presented in the next section.

5 Results

Our results are presented in the following subsections, first for two-player games, and after that, for multi-player games. In every case, the highest percentage advantage gleaned by the History-ADS heuristic is highlighted in **bold face**.

5.1 Results for Two-Player Games

Table 1 presents our results for the game of Othello. The History-ADS produced noticeable gains in terms of NC in all cases, in both initial and midgame states, and the Move-to-Front rule outperformed the Transposition rule.

Table 1. Results of applying the History-ADS heuristic for Othello.

Midgame?	Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
No	None	5,061	2,385	-	-
No	Move-to-Front	**3,827 (24%)**	1,692	6.0×10^{-3}	0.51
No	Transposition	4,057	2,096	0.015	0.42
Yes	None	20,100	9,899	-	-
Yes	Move-to-Front	**14,500 (28%)**	6,303	1.2×10^{-3}	0.59
Yes	Transposition	15,200	6,751	0.014	0.45

Consider Table 2, where we showcase our results for Relaxed Checkers. Again, we observed a very large reduction in NC, and again, the Move-to-Front achieved better performance than the Transposition rule.

Table 2. Results of applying the History-ADS heuristic for Relaxed Checkers.

Midgame?	Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
No	None	78,600	10,600	-	-
No	Move-to-Front	**40,800 (48%)**	5,619	$< 1.0 \times 10^{-5}$	3.58
No	Transposition	48,600	6,553	$< 1.0 \times 10^{-5}$	2.84
Yes	None	64,000	25,700	-	-
Yes	Move-to-Front	**36,100 (44%)**	12,700	$< 1.0 \times 10^{-5}$	44%
Yes	Transposition	43,600	16,600	$< 1.0 \times 10^{-5}$	0.79

Table 3 shows our results for two-player Focus, where we again see the same pattern observed in the previous cases, although with an even larger reduction in NC, for reasons that will be discussed below.

5.2 Results for Multi-Player Games

Table 4 holds our results for the Virus Game. As we have witnessed in the two-player case, the History-ADS obtained substantial gains in pruning, and the Move-to-Front rule outperformed the Transposition rule marginally.

Table 5 presents our results for Chinese Checkers. We saw the same pattern emerge as before, with a very large reduction in NC, and the Move-to-Front rule again achieved better performance than the Transposition rule.

Table 3. Results of applying the History-ADS heuristic for two-player Focus.

Midgame?	Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
No	None	5,250,000	10,600	-	-
No	Move-to-Front	**1,290,000 (75%)**	88,000	$< 1.0 \times 10^{-5}$	10.39
No	Transposition	1,800,000	158,000	$< 1.0 \times 10^{-5}$	9.07
Yes	None	10,600,000	-	-	
Yes	Move-to-Front	**2,420,000 (77%)**	637,000	$< 1.0 \times 10^{-5}$	2.37
Yes	Transposition	2,910,000	760,000	$< 1.0 \times 10^{-5}$	2.22

Table 4. Results of applying the History-ADS heuristic for the Virus Game.

Midgame?	Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
No	None	10,500,000	1,260,000	-	-
No	Move-to-Front	**4,690,000 (55%)**	1,010,000	$< 1.0 \times 10^{-5}$	4.57
No	Transposition	4,850,000	739,000	$< 1.0 \times 10^{-5}$	4.45
Yes	None	12,800,000	1,950,000	-	-
Yes	Move-to-Front	**5,940,000 (54%)**	832,000	$< 1.0 \times 10^{-5}$	3.51
Yes	Transposition	6,060,000	974,000	$< 1.0 \times 10^{-5}$	3.45

Table 5. Results of applying the History-ADS heuristic for Chinese Checkers.

Midgame?	Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
No	None	3,370,000	1,100,000	-	-
No	Move-to-Front	**1,250,000 (63%)**	316,000	$< 1.0 \times 10^{-5}$	1.92
No	Transposition	1,320,000	338,000	$< 1.0 \times 10^{-5}$	1.87
Yes	None	8,260,000	-	-	
Yes	Move-to-Front	**3,400,000 (59%)**	899,000	$< 1.0 \times 10^{-5}$	1.84
Yes	Transposition	3,650,000	920,000	$< 1.0 \times 10^{-5}$	1.74

Finally, Table 6 has our results for multi-player Focus. The same pattern, seen in all the other cases, is observed yet again.

Table 6. Results of applying the History-ADS heuristic for multi-player Focus.

Midgame?	Update Mechanism	Avg. NC	Std. Dev	P-Value	Effect Size
No	None	6,970,000	981,000	-	-
No	Move-to-Front	**2,180,000 (69%)**	184,000	$< 1.0 \times 10^{-5}$	4.88
No	Transposition	2,740,000	271,000	$< 1.0 \times 10^{-5}$	4.32
Yes	None	14,200,000	8,400,000	-	-
Yes	Move-to-Front	**3,240,000 (77%)**	1,730,000	$< 1.0 \times 10^{-5}$	1.30
Yes	Transposition	3,570,000	1,860,000	$< 1.0 \times 10^{-5}$	1.26

6 Discussion

Our results strongly reinforce the hypothesis that an ADS managing the move history, employed by the History-ADS heuristic, can achieve improvements in tree pruning through better move ordering, in both two-player and multi-player games. This confirms that such an ADS does, indeed, correctly prioritize the most effective moves, based on their previous performance elsewhere in the tree, as initially hypothesized.

Our results confirm that the History-ADS heuristic is able to achieve a statistically significant reduction in NC in the three two-player games, Othello, Relaxed Checkers, and the two-player variant of Focus, as well as three multi-player games, the Virus Game, Chinese Checkers, and the multi-player variant of Focus. We observed a drastic variation in the reduction between cases, i.e., from 24% to a value as high as 77%, depending on the game. Despite this wide range, these savings are very high, substantially exceeding those obtained by the Threat-ADS heuristic [8]. Savings remain high in both initial and midgame board positions, demonstrating that the History-ADS heuristic can perform well over the course of a game.

Our second observation is that the Move-to-Front rule outperformed the Transposition rule in all cases. This is a reasonable outcome, as the adaptive list may contain dozens to hundreds of elements, and it would take quite a bit of time for the Transposition rule to migrate a particularly strong move to the head of the list. As opposed to this, the Move-to-Front would migrate the move to the front quickly, and would likely keep it there. This phenomenon also confirms that the order of elements within the list matters to the move ordering. In other

words, merely maintaining an unsorted collection of moves that have produced a cut will not perform as well as employing an adaptive list, further supporting the use of the History-ADS heuristic.

When considering the wide range of savings between the various games, we observe that in general, the larger the overall game tree, the greater are the proportional savings. There are a number of possible reasons why this is the case. Firstly, we note that the History-ADS retains information on all moves that produce a cut, and in a larger tree, it is likely that there will be more moves of this nature. Thus, the History-ADS has, available to it, more information to work with. Secondly, in games with a very large branching factor, such as Focus or Chinese Checkers, many of the available moves tend to not be very good, and the prioritization of a small number of strong moves can thus prune huge "chunks" of the search space. Lastly, in the case of deeper trees, the History-ADS has more opportunities to apply its knowledge of strong moves gleaned from other levels of the tree, similar to the reasoning behind the History heuristic.

As expected, the History-ADS heuristic achieved noticeably greater savings compared to the Threat-ADS heuristic, which obtained between a 5% and 20% reduction in tree size, with most advantages being around 10% [8, 10]. This is not particularly surprising, however, as there are many more possible moves than opponents, and so the History-ADS achieves that improved saving with a larger adaptive list. That the Threat-ADS can, in some cases achieve half of the History-ADS' savings with a list of size 3-4 is, in and of itself, impressive. Thus, it remains a viable option when the more complex History-ADS is not necessary, perhaps due to other move ordering strategies being employed.

7 Conclusions and Future Work

Our work presented in this paper introduces the History-ADS heuristic, a powerful move ordering technique, which employs an ADS to rank moves based on their historical performance within the search. The development of the History-ADS builds upon our previous work applying ADSs to move ordering, specifically the Threat-ADS heuristic. Compared to it, the History-ADS is able to achieve substantially greater savings in terms of tree pruning in a wide variety of cases, and serves to extend the applicability of ADS-based techniques to the scope of two-player games. Given the domain-independent nature of the History-ADS heuristic and its excellent performance in those games tested in this work, we hypothesize that it will perform well in a very wide variety of two-player and multi-player games beyond those explored here.

The History-ADS heuristic represents our initial attempt to extend ADS-based techniques to two-player games, and employ the powerful move history metric. As this is a first attempt, the intuitive idea of storing all moves that produce a cut within a single adaptive list was employed. However, it may be that only a subset of these stored moves are particularly important, and limiting the length of the ADS may preserve the majority of its performance, while using much less memory and time to traverse the list. Furthermore, applying

the knowledge of the cuts obtained from all levels of the tree, equally suffers from the issue of disproportionately favouring those moves that are strong near the leaf nodes. Thus another avenue of research is to explore more complex ADSs that mitigate this issue. Alternatively, it is possible that maintaining separate ADSs for different levels of the tree can mitigate this as well. Both of these concepts are the subjects of current and ongoing research.

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. D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, pp. 293–326, 1975.
3. 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.
4. A. Reinefeld and T. A. Marsland, "Enhanced iterative-deepening search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 701–710, 1994.
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*, pp. 169–171, 1979.
6. J. H. Hester and D. S. Hirschberg, "Self-organizing linear search," *ACM Computing Surveys*, vol. 17, pp. 285–311, 1985.
7. S. Albers and J. Westbrook, "Self-organizing data structures," in *Online Algorithms*, pp. 13–51, 1998.
8. 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*, pp. 125–138, 2013.
9. 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.
10. 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.
11. 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.
12. 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.