24 Interest Search

A Faster Minimax

Jeff Rollason

24.1 Introduction

Minimax has been the dominant paradigm for many board games for so long, able to deliver either good or perfect solutions to many game problems. Such is this success that in many primary areas of human game intellect the best play is based on minimax-based AI. However, minimax has its limitations, and in some domains with great combinatorial complexity, minimax has been replaced by Monte Carlo tree search (MCTS) as the preferred solution. Where unscripted MCTS is rolling out to terminal nodes (the expected case), it sadly depends on spending most of its time assessing junk variations in the simulation phase in order to finally pick a "best" move. Note that in this case, only a small fraction of MCTS moves will likely fall inside the selection/expansion tree. This is a surrender to the complexity of the problem domain, determining that it is not possible to address the domain by exploring just meaningful game variations. If we look back at minimax, the situation is better, but actually still largely quite poor. If you randomly freeze a minimax search for a typical chess program, you will find that most variations explored will be largely in the range of poor to junk.

Interest search is another way to look at minimax pruning and allows the search to concentrate on meaningful variations. This method radically impacted the fortunes of the shogi program *Shotest*, which twice ranked #3 in the Computer Shogi Association (CSA) world championships in Japan, making it the all-time highest ranked western shogi program ever. Even basic implementations of this method yield a 5× speedup for chess, with no net loss of strength. For shogi this yield was 14×. This chapter details this method and how it was used for chess and shogi.

24.2 Background

A key observation the author made from early days in working in chess is that, despite good move ordering, much of the time was spent searching variations that, to the human observer, were obviously pointless. Having just one absurd move in a branch would be followed by an expensive subtree of variations to assess that move. To an extent, alpha–beta and related methods help reduce the overhead by creating more cutoffs in these bad variations. However, this was still not enough, particularly for quiet positions, where the large bulk of the search was still dominated by exploring poor variations. This characteristic cannot have been lost on early chess developers so early work in minimax for chess originally divided primarily into two camps:

1. Fast full-width search that attempted to explore all variations, pruned by alpha–beta and related methods to optimize the tree (Shannon type A)
2. Selective search, which likely spent more time in evaluation, but restricted the number of moves searched through heuristic forward pruning (Shannon type B)

If selective search could reliably reduce the number of nodes explored, then this could be traded for deeper search. Naturally selective search came at a price, requiring heuristic assessments during search. If the cost of this was less than the saving caused by narrower search, and if this avoided missing moves that should be explored, then selective search could search deeper and therefore be stronger. Since it would also have access to slower evaluation, it would also be able to have better terminal node evaluation.

Hope and logic in those early days suggested that (2) would win out as the idea of move selection that depended on trying to explore every possible combination of move plays surely could not be the way to deliver the best possible play. However, history has shown that method (1) appeared to quickly dominate and the method of forward pruning simply was unable to compete.

24.3 Rethinking Selective Search

An issue with selective search is that as soon as you restrict the moves list to some subset, even though the confidence that any one list will have all the moves it needs might be high, that once this gets multiplied in a multimove variation, the risks geometrically increase. Even with a 99% confidence in any one moves list, after 10 plies, the chance that the key variation may have been missed is still 10%. To achieve 99% certainty within any one list would alone be very hard and to achieve it would probably mean that there was no selective search saving. You might as well search a few more moves, slightly faster, and be 100% certain that you had examined the key variation.

I think the flaw here is that trying to find the magic compromise where you sacrifice some small portion of all moves in each list to narrow the tree is perhaps doomed to failure. Trying to assess viability in terms of individual moves lists (with perhaps some peeking at previous plies in order to keep choices relevant) is probably a too limited way to look at being selective.

24.4 Selection by Variation: Interest Search

When assessing snapshots from a chess alpha–beta search, the key disappointment is not just that an absurd move was tried, but that the search then followed up such moves with comprehensive follow-ons to validate or reject that move. Intuitively wacky moves do not merit the same thorough examination as moves that are more obviously sound.

That reveals what the goal could be: Dubious moves do not warrant the same level of examination, so that as soon as a variation introduces such a move, the willingness to devote time exploring the variation should diminish.

This is the underlying principle the drives interest search. (It should be noted that this principle also drives the selection and expansion phase of MCTS, but not the simulation phase, where most moves are usually selected.)

24.5 Quantifying the Interest Search Idea

Interest search passes down a value through the branch indicating how interesting that branch is. For aesthetic reasons, a high value might obviously have been chosen to indicate "high interest," but actually, this is inverted and the value actually represents "interest cost," so that a high value means that the variation is already expensive, perhaps because it is already deep into the tree or that it includes mediocre moves.

This value is passed down the tree and is used to determine when a branch ends, completely replacing the usual "iterate by depth" by "iterate by interest," where the interest threshold increases with each iteration. Search paths full of highly interesting moves get explored at depth, whereas poor variations are terminated at shallow depths. Once a variation contains one very poor move, it uses up a high proportion of its interest value, so that it only receives limited follow-on search. The same impact might be achieved by a variation containing two mediocre moves, which net may generate the same high value of one very poor move. This is intuitively a reasonable idea.

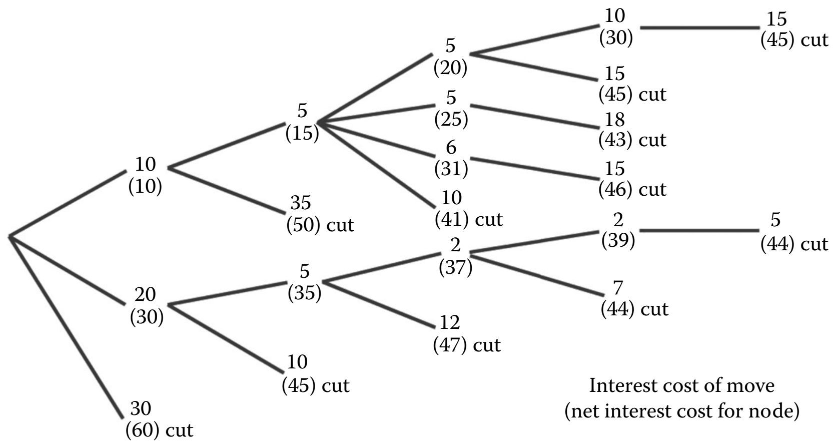

Figure 24.1

An interest search tree with an interest threshold of 40.

Figure 24.1 shows how this principle might impact the structure of a tree search, terminated on interest cost rather than depth. The upper number shows the interest cost of that move and the lower number the net interest cost of that branch. As you move left and down, the lower number increases in value. When its value exceeds the interest threshold, a cutoff occurs.

24.6 Classifying the Moves in Terms of Interest

This idea earlier could be managed in wide variety of ways, unlike alpha–beta, which has just one exact definition. The description given here is the method used on AI Factory's Treebeard chess. A more comprehensive implementation follows later, for Japanese chess (shogi).

At each ply, a full plausibility analysis is performed, to achieve an expected ordering of moves. This delivers a score for "how good a move is likely to be" but in addition, a second value is calculated for how "interesting" a move is. If a move is "good," then it is also interesting, but it might also be very interesting because it forces a reply.

Our Treebeard chess has a relatively simple set of criteria, each of which adds a value to the interest. A simplified overview of this is as follows:
For each move the plausibility analysis evaluates (separate from static terminal evaluation) the following:

1. Interest is initialized to 1000/number of legal moves.
2. If in check, value increases += 1000 (function of number of replies).
3. Move is in the transposition table += 25.
4. Ply1 existing previous iteration best move += 100.
5. Killer move += 100.
6. Move is voted global "best reply" to previous move in branch += 100.
7. Best follow-on from previous move 2 plies ago += 100.

8. Pawn attacks piece += f(value piece).
9. Value of capture += f(value of capture).
10. We give check. += 100, more if our previous ply was also check.
11. Attack pieces, extra if fork += f(value of piece).
12. Moving a piece under attack += f(value of piece).
13. Capture checking piece += 100.
14. Promotion += 100.
15. Diminish interest if beyond iteration depth.
16. Move piece that is target of pin/tie += f(value of pin/tie).
17. Move is best noncapture += 100.

The interest value, which is higher for more interesting moves, is finally added to the plausibility score, so that interesting moves have an increased chance of selection. Therefore, move ordering is a combination of how good a move is and how interesting it is.

Within the search, once a move has been selected for assessment, its net interest cost is calculated as K/(move interest), so this converges to "K" for the least interesting move (where "K" is an arbitrary constant, picked solely to fit the word size of the target hardware; if using floating point this might simply be 1.0) and reduces to near zero for extremely interesting moves.

At the point that a move may be selected for search, the following calculation is made:

```
net_Interest = total_Path_Interest + total_Interest_This_Ply
               + interest_This_Move
```

where

`total_Path_Interest`	The net interest cost of the branch leading to this ply
`total_Interest_This_Ply`	Net interest cost of all previous moves explored at this ply
`interest_This_Move`	The interest cost of the move to be considered

This is then compared to the "iteration interest limit" and if it is greater, then a cutoff occurs and no further moves are explored at this node. This iteration interest limit simply increases the limit for each iteration as each iteration is executed.

If a cutoff does not occur, and the move is searched, then "`net_Interest`" becomes the new "`total_Path_Interest`" passed on for the next ply.

24.7 Does This Work?

If you follow through the idea given earlier in a thought experiment, it is not hard to observe that this appears to have a very serious flaw. If you test a move that appears to result in a very strong outcome a few plies later, then the opponent could avoid this outcome by exploring a move with a high interest cost, so that it cuts off before the outcome is reached. Indeed, this destroys the search, which quickly descends into poor play. The key here is to separate the interest tally for different search parities—that is, for each player in the game.

To make this work, you need to track separate interest cost for each of the two search parities. Now, the search cannot avoid an opponent variation outcome by attempting to insert a dull move to force an early cutoff. However, this does not stop each side avoiding horizon events. The search could find that a "good" move actually leads to disaster, so avoids the good move by searching a "bad" move that causes an earlier cutoff. This is actually not so different to the existing defect that minimax already has. The difference is that minimax would allow time to be wasted exploring the "bad" move, whereas interest search will cut it off earlier.

Note that if side A has run out of interest but side B has not, then A might cause a termination, but this is after side B having just moved. So A forcing a cutoff will leave B with the advantage of last move.

24.8 Performance of Interest Search for Treebeard Chess

From testing, the addition of interest search resulted in a 20× speedup in search time (500 games) but with only a 41% win rate as there was some inevitable loss of critical moves. If the number of iterations was increased to achieve strength parity, then the net speedup was 5×, as shown in Table 24.1.

Note that this method makes no absolute limit on any one moves list. If the previous branch contained high interest moves, then at the current depth, there will be the option to potentially explore all moves. Note also that in follow-on iterations, the best moves will have already become hash and best reply choices, so they will be promoted in interest value. With the increased interest threshold of the next iteration, then new moves will be considered throughout the tree for examination, not just the usual terminal node extension you normally get with iterated minimax.

Our implementation of interest search for chess could undoubtedly be optimized. With more attention, we would expect that a 10× speedup or more is likely. We have not needed to explore this as our chess already easily fulfils what it needs to, as the mobile version, from its Elo rating, appears to be already stronger than 99.5% of online players on chess.com.

However, this method was actually primarily developed at length with our shogi program, where the optimization was vital as shogi grows combinatorially faster than chess.

24.9 Applying Interest Search to Japanese Chess (Shogi)

Shogi is a much greater challenge than chess as is combinatorially explosive. To start, the average number of moves is nearer 80 rather than 35 for chess but can easily peak at 200 moves near the end game. The reason for this is that pieces captured can be replayed on the board (9 × 9) on any square as a move (a drop), so that if many pieces have been captured,

Table 24.1 Comparing the Performance of Search with and without Interest Search for Chess

		Result for A Compared to B		
Player A	Player B	Time Used	Win%	Comment
Interest ON	Interest OFF	20× faster	41	
Interest ON	Interest OFF	5× faster	50	Rebalanced win rate
Interest ON	Interest OFF	Equal time	72	Rebalanced time

the number of legal moves goes up. This makes the game very challenging as pieces are never eliminated from play, so that swapoff exchange simplifications are not possible.

The core mechanism to drive this is essentially the same as for chess earlier, but backed by a much more comprehensive and sensitive plausibility analysis, with 48 separate components. Among the more significant additions are as follows:

1. Move to a square that is now safer (less attacks) than it was at root ply.
2. Move is now legal but not legal at root ply (i.e., a new move).
3. Index into various positional arrays to credit better squares or desirable target squares.
4. Various context-sensitive credits, including moves to defend against threatened mates.
5. Negative plausibility: Penalize moves that in previous branches were searched before the best alpha was actually found.

These complex terms mean that each move may get multiple credits as each move performs multiple roles. Such moves are more likely to be ordered high in the search list. The complexity of this also makes it more likely that few moves will have the same score as they will likely each have at least some differentiating value.

However, such is the nature of the game shogi that this is not enough, as most of the time multiple drops will be possible. For example, the move list might contain 40 possible drops by a rook. This could easily flood the top of the moves list with multiple alternative rook drops, all of which look equally good, but clearly, it would not make sense to explore all of these before trying some other kind of move.

Therefore to address this, the plausibility needs to dynamically modify the score and interest value to change the move ordering, to diminish selection of moves similar to those already played.

24.10 Dynamic Calculation of Score and Interest

As each move is searched, the list of existing unsearched moves is updated depending on how similar they are to the move just played.

These similarity checks penalize the following matching criteria, to different degrees:

1. Dull moves that have no tactical component.
2. Vacate moves that avoid some tactical threat such as a capture.
3. Vacates per square. This is like (2) but counts separately for each square.
4. Unsafe moves that are expected to result in some loss.
5. Moves by piece type.
6. Mate threats.
7. Checks.
8. Move from square and move to square.
9. Drop move.
10. Blocking move.
11. Attacking move that creates a new threat.
12. Discovered attack.

This then changes the move ordering by score so that the next move to be considered may have changed from what otherwise might have been selected. That new move will also have a modified interest value.

24.11 Impact of Interest Search on Shogi

Note first that the aforementioned reordering still applies, even if interest search is not in use. Also, the search control is not "naked" if the interest search is switched off as the program will already have the usual move ordering mechanisms such as killer, hash, history heuristic, best reply, and best follow-on tables. It also has a number of other pruning methods to contain the tree search, as follows:

1. Razoring (prune by comparing to beta)
2. Gamma pruning (prune by comparing to alpha)
3. Alpha fail pruning (stop selection of the next move if too many moves fail to replace alpha)

The aforementioned are necessary to limit the search that otherwise would be very slow.

Comparing this to chess, it is clear that with shogi, interest search is eliminating more critical lines than chess as simply switching off interest search decimated the performance (threshold initialized for similar depth), dropping to a 8.3% win rate, but 119× faster (see Table 24.2). With rebalancing by adding more iterations to restore a 50% win rate, the speedup was still 14× the speed of search with no interest search. If we rebalanced time taken, then interest search manages a healthy 90.5% win rate.

24.12 Analysis

The behavior of this type of search was intuitively much more natural to my own perception of what AI search should be doing. I was always uneasy that minimax on its own felt more like a theorem prover than AI. Determining a move primarily from variations that humans could never consider did not feel like AI.

However, justifying the underlying basis of why AI methods are actually valid can be hard. Large tree searches cannot be easily visualized or intuitively understood. Minimax search, although in its core foundation is easy to understand, is harder to comprehend once you are considering vast numbers of nodes.

A consideration in assessing the validity of interest search is that its concentration on plausible variations reduces the chance of suspect evaluation results arising from obscure random variations. For example, when exploring a complete space with full-width

Table 24.2 Comparing the Performance of Search with and without Interest Search for Shogi

		Result for A Compared to B		
Player A	Player B	Time Used	Win%	Comment
Interest ON	Interest OFF	119× faster	8.3	
Interest ON	Interest OFF	14× faster	50	Rebalanced win rate
Interest ON	Interest OFF	Equal time	90.5	Rebalanced time

minimax, any possible odd-ball position might be reached, and among these, some may manage to deliver a high evaluation value that causes a beta cutoff, simply because the evaluation is unable to reliably assess all random positions thrown at it. This will expose the terminal evaluation weaknesses by exposing it to positions that it was probably never designed for.

Interest search avoids the chance of this by ensuring that it only explores highly plausible search lines, which may contain a very limited number of odd-ball moves. Therefore, the evaluation is mostly only exposed to positions that a human player might consider "reasonable." This is a healthy premise as it is otherwise an unfortunate impediment that without selective search, the evaluation needs to not only be plausible in reasonable positions but also provide meaningful comparative assessments with the larger bulk of absurd positions as well. Without this handicap, the evaluation can be freed to only needing to comparatively assess a narrow set of related reasonable positions, rather than the wider spectrum of absurdity.

This may not be obvious, but actually, the map of the tree explored by this method will likely resemble the map of an MCTS tree, with exploration concentrated to explore "better" lines of play. The driving mechanism to achieve this is somewhat different, but with a likely similar end product.

It is hard to reliably project the full practical consequences of this method, but Treebeard chess has a great following, being by far the most played chess on Android, and is also used in Microsoft's MSN chess, so therefore probably the most played chess program on the planet. It is rated strongly for its agreeable humanlike style (see online reviews of "Chess Free AI Factory"), which may well be the end consequence of its use of interest search. The programs and related methods used are discussed in an online periodical [AIF 14] and previous paper [Rollason 00].

24.13 Other Games

MCTS is now becoming a preferred paradigm to minimax in combinatorially complex games, but interest search may offer a widening of the net that minimax can cope with. Its advantage is that it offers the possibility of providing near-proven best plays, whereas MCTS only offers a probability of best play. It also offers an analysis that confines positions being analyzed to being more reasonable than minimax on its own and much more reasonable than MCTS.

It should be noted that there is no reason why interest search could not also be applied to the MCTS simulation phase. This might offer a much faster MCTS if less time is spent considering very low quality lines of play.

24.14 Conclusion

Without interest search, *Shotest* would not have been able to twice rank #3 in the World Computer Shogi Championships in Japan. Although my program had other important evaluation features that contributed well to its performance, it is very clear that without interest search that I would have been very lucky to even make the top 20.

The methodology of this seems to be applicable to any minimax program that has more than a trivial plausibility analysis. It risks missing variations that regular minimax would pick up but the vast volume of what it rejects will certainly be very poor. The loss of key variations will have been traded by much more available time that can be used to explore high probability branches. Empirically, this has worked very well in both chess and shogi and there is no obvious reason why this would not work in other games.

Note also that this method avoids the forced geometric discrete stepping associated with iteration in minimax. In the latter, the chance to progress for more analysis is severely restricted by the large jump in time needed for each new iteration. With interest search, you can set the size of each progression, so that each step might be quite small. This can even be dynamically changed between moves, which might be much more effective if small steps where the search is struggling to pin down the right line of play. The key here is that the option is available, whereas iteration by depth is hardwired and inflexible.

Finally, note that this single currency of "interest" to control search offers all kinds of flexibility to modify how interest is assigned per move. This gives the developer the chance to dynamically select between shallow bushy or deep sparse trees, as they see fit. There is a great deal to explore here. This method has served the author well for some 18 years and, as yet, they have not found any other paradigm to replace it for supporting minimax.

References

[AIF 14] AI Factory Newsletter. 2003–2014. Currently 46 articles on game AI. www.aifactorynewsletter.co.uk (accessed September 10, 2014).

[Rollason 00] Rollason, J. 2000. SUPER-SOMA—Solving tactical exchanges in Shogi without tree searching. In *Lecture Notes in Computer Science*, ed. G. Goos, J. Hartmanis, and J. van Leeuwen. LNCS 2063. Springer, Berlin, Germany, ISSN 0302-9743.