Trappy Minimax - using Iterative Deepening to Identify and Set Traps in Two-Player Games

V. Scott Gordon
CSU, Sacramento
gordonvs@ecs.csus.edu

Ahmed Reda
CSU, Sacramento
ahmedcsus@yahoo.com

Abstract

Trappy minimax is a game-independent extension of the minimax adversarial search algorithm that attempts to take advantage of human frailty. Whereas minimax assumes best play by the opponent, trappy minimax tries to predict when an opponent might make a mistake by comparing the various scores returned through iterative-deepening. Sometimes it chooses a slightly inferior move, if there is an indication that the opponent may fall into a trap, and if the potential profit is high. The algorithm was implemented in an Othello program named Desdemona, and tested against both computer and human opponents. Desdemona achieved a higher rating against human opposition on Yahoo! Games when using the trappy algorithm than when it used standard minimax.

1. Introduction

Minimax is a well-known algorithm for performing adversarial search in two-player strategy games. Although it is known to work very well in practice, it also has its shortcomings. One of these is that it assumes best play by the opponent. Often, a human (or even a computer) may be fallible, and may be tricked into playing an inferior move. Minimax never takes this into account, and thus it never purposefully tries to set traps.

This paper introduces trappy minimax, a variation of minimax that utilizes iterative deepening to identify possible traps, and under the right conditions, set them. Traps are defined in a game-independent manner, based only on the generic search tree and the backed up values from its associated terminal evaluation function. Alpha-beta pruning still works without modification. Thus, trappy minimax can be used wherever standard minimax can be used. At times, trappy minimax chooses different moves than standard minimax. Depending on the type of opponent, the trappy version may be more effective.

Section 2 gives a brief historical background and related work; Section 3 defines the concept of a trap; Section 4 presents the trappy minimax algorithm; Section 5 gives experimental results for two board games, and Section 6 offers a summary and conclusions.

2. Background

The *minimax* algorithm was first proposed by Claude Shannon in 1950 [Shannon50], as suitable for programming a computer to play the game of chess. An improvement called *alpha-beta pruning* was described in 1962 by Kotok [Kotok62]. Alpha-beta pruning allows the algorithm to eliminate certain branches from consideration, speeding up the search without changing the result. Minimax with alpha-beta pruning has been used in most computer implementations of popular board games such as chess and checkers, and forms the basis of the search algorithms used in successful competitive programs such as Deep Blue (for chess) [HCH02] and Logistello (for Othello) [Buro97].

When minimax considers every possible move at each level, it is performing a *full-width* search. When it only considers a subset of the allowable moves, it is performing a *selective* search. A minimax algorithm is shown in Figure 1.

```
Negamax(board,depth)
If (game over) then return(result)
If (max depth) then return EVAL(board)
best = -∞
For each move
{
   make move on board
   score = -Negamax(board, depth+1)
   if score > best then best=score
   retract move from board
}
return (best)
```

Figure 1 — Minimax

Various improvements to minimax have been developed. Most attempt to alter the ordering of generated moves in order to accelerate alpha-beta pruning [Schaeffer89]. Others use game-dependent knowledge to determine whether certain branches of the tree should be ignored, allowing for deeper search.

Iterative Deepening is when a minimax search of depth N is preceded by separate searches at depths 1, 2, etc., up to depth N. That is, N separate searches are performed, and the results of the shallower searches are used to help alpha-beta pruning work more effectively.

Carmel and Markovitch [CM95] examined a variation of minimax called M*, in which an opponent model is incorporated into the search algorithm. M* allows minimax to consider whether an opponent is likely to make an inferior move, and to take better advantage of it. They also examined using M* to take advantage of opponents that search to a shallower depth, and thus linking search depth with setting traps.

In this paper, the notion of a trap specifically as a function of search depth is explored in greater detail. Whereas M* utilizes an opponent model, trappy minimax does not require it. It also does not need to know what algorithm the opponent is using or to what depth he/she/it is searching. Traps are defined as a function of search depth, and can be set at many levels. The only assumption is that certain opponents may be vulnerable to these sorts of traps - particularly humans.

3. Definition and evaluation of Traps

3.1 Definition of a Trap

A **trap** is a move that looks good in the short term but has bad consequences in the long term. Thus, in the minimax algorithm, it is a move with high evaluations at shallow depths and a low evaluation at the maximum depth. Traps have the significance that a non-optimal opponent might be tricked into thinking that they are good moves when in fact they are not.

Although our definition of a trap does not require opponent modeling, it is useful to consider the sort of opponent that would fall for a trap. An opponent that performs a full-width minimax search to a depth greater than the trappy minimax player, would not fall for a trap. Humans, however, are always susceptible to traps, since they in general do not perform full-width search, and do not consider every variation to a uniform depth.

Setting a trap is only of practical benefit if the result of an opponent *not* falling for the trap is not much worse than the evaluation of the move recommended by standard minimax. In other words, if the opponent does not fall for the trap, setting the trap which didn't materialize should not yield a significant negative cost.

3.2 Trap Evaluation

A trap is evaluated using two factors: (1) how likely the opponent will fall into the trap, (2) how profitable the trap is if the opponent falls into it. A trap could be very deceiving to a human opponent yet not profitable, and similarly, a trap could be highly profitable but easy to avoid. A good trap is hard to spot, and profitable.

Since most minimax implementations utilize iterative deepening, it is relatively easy to accumulate evaluations at various search depths. Thus, accumulating the data needed for evaluating traps is very inexpensive, and alpha-beta pruning can still be employed at each search depth.

To assess how deceiving a trap is, evaluations are made using depths other than just at the maximum depth. When the opponent is a human, looking at all the depths other than the maximum depth is useful, while for an opponent using a shallower minimax search, looking at just that depth would be sufficient. We examined three different methods for assessing evaluations at shallow tree depths:

1. The _median_ method: calculates the median of the evaluations at all but the maximum depth,
2. The _best value_ method: uses the best evaluation for the opponent among the shallower evaluations,
3. The _last level_ method: uses only the evaluation at MaxDepth-1 (MaxDepth is the deepest level of search examined by the program).

The first two methods are directed towards human opponents, while the third method is targetted at opponents who use the minimax algorithm either at a shallower depth (up to MaxDepth-2), or selective search.

4. Trappy Minimax

Depending on which assessment method is being used, the resulting aggregate shallow evaluation is compared to the evaluation at the maximum depth. Two factors are then determined: *trappiness*, and *profitability*.

Trappiness is based on the distance between a high positive score and an actual negative score, and is calculated from the point of view of the opponent. That is, it attempts to measure the likelihood that the opponent could miss the trap.

Profitability is the gain to the program if the opponent falls for the trap. It is calculated from the point of view of the algorithm.

Trappiness and profitability are both factored into the evaluation of each possible computer move, along with the standard minimax evaluation. The resulting trappy minimax algorithm is shown in Figure 2.

```
TrappyMinimax(board,maxdepth)
best, rawEval, bestTrapQuality = -∞
{  For each move
   {  make move on board
      for each opponent response
      {  scores[maxdepth] :=
            -Negamax(board,maxdepth)
         if (scores[maxdepth] > rawEval)
            rawEval := scores[maxdepth]
      }
      for each opponent response
         for d := 2 to maxdepth-1
            scores[d] := -Negamax(board,d)
      Tfactor := Trappiness(scores[])
      profit := scores[maxdepth]-rawEval
      trapQuality := profit * Tfactor
      if (trapQuality > bestTrapQuality)
         bestTrapQuality := trapQuality
      adjEval :=
         rawEval + scale(bestTrapQuality)
      if (adjEval > best) best:=adjEval
      retract move from board
   }
   return(best)
}
```

Figure 2 — Trappy Minimax

The trappiness factor has a range from [0,1], and is calculated according the formulas shown in Figure 3, which apply regardless of which trap assessment (median, best, or last) was chosen. The calculations are slightly different depending on whether the backed-up score is for a minimizing or a maximizing level (i.e., odd or even depth).

Assessments	Maximizing Trappiness
U <= M	0
M < U < M+aM	.75(U-M)/aM
M+aM <= U < M+4aM	.75+.25(U-M-aM)/(3aM)
U >= M+4aM	1
Assessments	Minimizing Trappiness
U >= M	0
M > U > M-aM	.75(M-U)/aM
M-aM >= U > M-4aM	.75+.25(M-U-aM)/(3aM)
U >= M-4aM	1

Figure 3 - Trappiness Formulas

U = Upper levels assessment (median, best or max-1)
M = Maxdepth assessment
aM = abs(M)

After the trappiness factor is calculated, profitability is determined by calculating the difference between the evaluation at the maximum depth for the trap and that for the best guaranteed evaluation. The higher the evaluation the trap has over the best guaranteed evaluation, the better the profitability the trap has.

The trappiness factor of a move along with the profitability can then be combined to give an estimate of how good a trap is, and then whether the potential profit should be factored into the final evaluation of a move.

In our implementation, the quality of a trap is the product of trappiness and profitability. Consider the sample game tree shown in Figure 4, for a 10-ply search. The top two plies are shown, plus backed-up values for searches of depths 3 through 10. The lists of values at the leaves are for search results of various depths, with the bottommost values representing standard minimax values at maxdepth. Evaluation proceeds thusly:

In Figure 4, the computer has three moves from which to choose. Suppose the second move (marked with an asterisk) is chosen. In that case, a score of at least 6 is guaranteed, presuming the opponent plays the best response (marked with a "+"). However, the alternative for the opponent (marked with a "$") looks very appealing when evaluated at depths 3 through 9. However, when evaluated at depth 10, the node has an evaluation that is considerably worse than even the correct move, despite all the good evaluations using the upper levels.

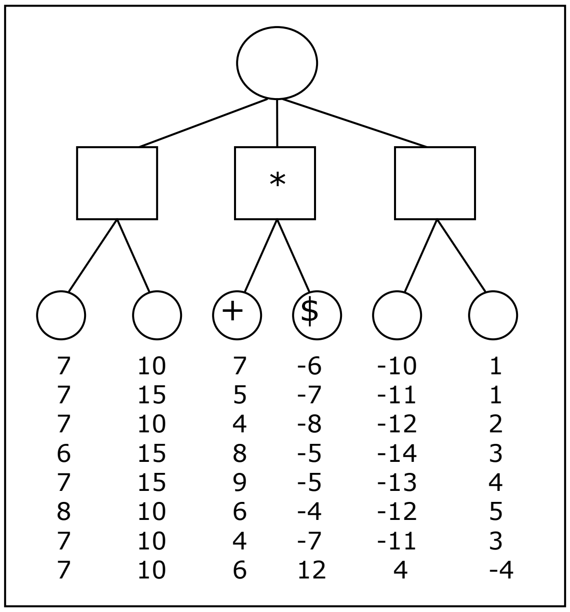

Figure 4 - example Search Tree

Assuming we are using the median method, the assessment for the evaluations of node “$” is equal to 6. And, using the appropriate formula from Figure 3, the trappiness will be 0.792. The profitability of the trap will be `12-6=6`, which indicates that if the opponent falls for the trap, there will be an extra 6 evaluation points scored for the computer beyond the guaranteed 6. The trap quality is therefore the product of these values, or 4.75.

After evaluating all possible traps for the opponent, the algorithm accounts for the traps in the move evaluation so that moves with higher potential of tricking the opponent are favored over moves that have lower or no potential. This is done by first picking the trap that is most likely to trick the opponent, that is the one with the highest trap evaluation. In the current implementation, the trap evaluation is first scaled so that it never exceeds 25% of the best guaranteed evaluation, then each of the scaled trap evaluations are added to their associated final move scores.

Scaling is done according to the formulas shown in Figure 5. Note that any trap values better than twice the move evaluation are adjusted to 25% of the move evaluation.

Trap evaluation	Final adjustment
`T < 0`	`0`
`0 >= T > .25M`	`.2M/.25`
`.25M >= T > 2M`	`.2+.05(T-M)/1.75`
`T >= 2M`	`.25M`

Figure 5 - Final move evaluation adjustment

T = trap evaluation
M = abs(best move evaluation)

5. Experimental Results

We implemented and tested trappy minimax using the game of Othello. We tested our implementation against both standard minimax and against humans.

5.1 Othello - *Desdemona*

Our Othello implementation, named *Desdemona*, utilizes the full trappy Algorithm described previously in Section 4. Desdemona can be set to use either standard minimax or trappy minimax (both with alpha-beta pruning), for various search depths. This enabled us to test the two algorithms against each other, and to evaluate both the efficacy of setting traps against weaker opposition, and the possible danger in trying to set traps against equal or stronger opposition.

Trappy minimax sometimes risks making non-optimal moves in the hope of gaining more advantage if the opponent makes the wrong response. An opponent who uses standard minimax to search the same depth would not be expected to fall for any traps set by the trappy algorithm. An opponent searching to a shallower depth would, however, be susceptible to the traps.

Thus we tested Desdemona by playing it against itself with a variety of combinations of configurations. Some of the games involved the two algorithms using the same search depth, to determine if the trappy moves tend to be sufficiently less accurate to lead to inferior play against strong opposition. Some of the games involved setting the trappy version to a deeper search level, and comparing its margin of victory against how well standard minimax would have done in the same circumstances. In each case, games were played with each algorithm having the chance to be white and black (i.e., move first or second -- black always moves first).

Match	Black	Depth	White	Depth	Result
1	original	8	original	8	43:21
2	trappy	8	original	8	13:51
3	original	8	trappy	8	49:15
4	original	8	original	6	40:24
5	trappy	8	original	6	41:23
6	original	6	original	8	8:56
7	original	6	trappy	8	10:54
8	original	9	original	9	26:38
9	trappy	9	original	9	36:28
10	original	9	trappy	9	39:25
11	original	9	original	7	55:9
12	trappy	9	original	7	56:8
13	original	7	original	9	34:30
14	original	7	trappy	9	39:25
15	original	10	original	10	3:61
16	trappy	10	original	10	49:15
17	original	10	trappy	10	30:34
18	original	10	original	8	53:11
19	trappy	10	original	8	58:6
20	original	8	original	10	27:37
21	original	8	trappy	10	21:43

Figure 6 - Trappy and Standard Minimax match results

When pitting trappy minimax against standard minimax at the same search depth, four games (2, 3, 10, and 17) were won by standard minimax and two games (9 and 16) were won by trappy minimax. That is, standard minimax performed slightly better at the same search depth, as expected.

For matches with different search depths, in four cases (5, 12, 19, and 21) trappy minimax outperformed standard minimax against the weaker opponent. In two cases (7 and 14), standard minimax outperformed trappy minimax against the weaker opponent. That is, trappy minimax performed slightly better overall than standard minimax when faced with weaker opposition, as was also expected.

Examining particular moves played is instructive. Consider the log files of games 6 and 7. As can be seen in Figure 7, the trappy and standard versions both made similar decisions for the first 6 moves:

Move	Black Standard 6-ply	White Standard 8-ply	White Trappy 8-ply
1	D3	C3	C3
2	C4	E3	E3
3	F2	C5	C5
4	B3	F2	F2
5	C1	D1	D1
6	E1	F3	F3
7	F6	F5	E6

Figure 7 - First seven moves of games 6 and 7

Closer examination of each program's alternatives for the 7th move show that playing at square F5 will guarantee an evaluation of 400. Since that was the highest possible evaluation, that move was the choice of the standard minimax algorithm. The trappy version found that playing at E6 also has a guaranteed evaluation of 400 in addition to a potential trap - if the opponent misses the correct move B4 and instead chooses to play to square F7, which has a better upper evaluation (that is, appears better at lower plies) than the correct B4. The weight of the move was adjusted accordingly and E6 was selected over F5 because of the trappiness and potential profit. Indeed, the weaker (6-ply) opponent chose F7 giving an additional 100-point bonus to the white player. In this case, interestingly, this advantage did not ultimately lead to a better result at the end of the game (although in more cases, such traps were beneficial).

Note, in Figure 8, that a trap was also identified for the move C2, but the move was considerably worse in both cases and was therefore not selected.

Move	F1	G1	A2	C2	A3	F5	E6
Standard	150	50	-150	-50	0	400	400
Trappy	150	50	-150	-37.5	0	400	448

Figure 8 - Options & corresponding scores for move 7

Although the overall results were as expected, they did not hold in 100% of the cases. This is natural, since the terminal evaluation function is itself non-optimal, and therefore some noise in the results is to be expected.

5.2 Results against Human Opponents

Desdemona also played a series of matches in the Yahoo! Games Online environment against a variety human opponents. Desdemona played 50 games using standard minimax, and 50 games using trappy minimax.

The environment not only provides a mechanism for playing games, but also awards ratings for performance over time. Ratings are calculated using the ELO system [Elo78], and range roughly from 0 to 3000, with 1500 being an average competitor. The ELO system determines ratings not only by win-loss record, but also factors in the strength of the opponents. Players on Yahoo! Games vary in skill level, so Desdemona faced a variety of opposition, and it is likely that the two versions (standard and trappy) faced a somewhat different set of opponents. The ELO rating system takes such variations into account.

After 50 games, Desdemona with standard minimax achieved a rating of **1680** (39 wins, 9 losses, 2 draws). With trappy minimax, Desdemona achieved a rating of **1702** (40 wins, 9 losses, 1 draw).

The log file for the games played by the trappy version of Desdemona was examined, and statistics are shown in Figure 9. While it is understandable that humans would frequently not play the move predicted by minimax, it is interesting to note that in 30% of those cases, humans fell into traps set by the algorithm.

Number of games	50
Traps set	198
Humans played optimally	34
Humans fell for traps	61
Humans made non-optimal move but did not fall into trap	103

Figure 9 - Trappy Minimax versus Humans

Overall, it appears that the imprecise nature of human play makes humans good candidates for succombing to the types of traps set by trappy minimax in Othello. It is particularly interesting that Desdemona achieved a higher ELO rating when using the trappy algorithm (rather than standard minimax), despite the fact that it was knowingly playing moves that were deemed inferior by its own evaluation. This supports the efficacy of setting traps in the manner proposed.

6. Conclusions

Trappy minimax is a game-independent extension of the minimax algorithm that further takes advantage of human frailty. Whereas minimax assumes best play by the opponent, trappy minimax tries to predict when an opponent might err. It works by identifying moves that qualify as potential traps, by comparing the various backed-up scores returned through iterative-deepening. Three methods were described for aggregating those scores: *median*, *best*, and *last level*. Traps were evaluated for both trappiness and profitability, and the results factored into the algorithm's move selection.

Our Othello implementation, named *Desdemona*, was used to test the algorithm both against computer and human opposition.

The trappy algorithm was better able to capitalize on weaker computer opposition than standard minimax. In 67% of the cases, using the trappy algorithm enabled Desdemona to achieve a better final score than when standard minimax was used. This was because a weak opponent was likely to fall into some of the traps.

Desdemona achieved a higher rating against human opposition when using the trappy algorithm than when it used standard minimax. Humans frequently fell for the traps set by the algorithm. Yahoo! Games was used to facilitate the games, and the rating calculation was done by a neutral (online) system.

Desdemona's occasional deliberate choice of a slightly inferior move, in the interests of setting a trap, caused it to perform slightly worse against strong computer opposition. This did not come as a surprise.

Trappy minimax requires no more time and space complexity than standard minimax. In fact, when iterative deepening is used (and it usually is), the mechanism for including trappy analysis is already present. Alpha-beta pruning can also still be used without modification. The trappy method is therefore an inexpensive and effective form of minimax, particularly against human opposition.

7. Future Work

Given the encouraging results shown by Desdemona, we intend to test the trappy algorithm on other games such as Chess and Checkers. Checkers is well-known to be a game that lends itself to short-term tactical analysis, so we optimistically are hoping to realize benefits that match or exceed those observed in Desdemona.

8. References

[Buro97] M. Buro, *The Othello Match of the Year: Takeshi Murakami vs. Logistello*, ICCA Journal 20(3), 1997 pp 189-193

[CM95] D. Carmel and S. Markovitch, "Opponent Modeling in a Multi-Agent System", Workshop on Adaptation and Learning in Multiagent Systems - IJCAI, Montreal 1995.

[Elo78] *The Rating of Chessplayers, Past and Present.* Batsford, 1978.

[HCH02] A. Hoane., M. Campbell, and F. Hsu, *Deep Blue*, Artificial Intelligence 134, 2002, pp 57-83.

[Kotok62] A. Kotok, *A Chess Playing Program for the IBM 7090 Computer*. B.S. Thesis, MIT, June 1962.

[Schaeffer89] J. Schaeffer, *The History Heuristic and Alpha-Beta Search Enhancements in Practice*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 1989, pp 1203-1212.

[Shannon50] C. Shannon, *Programming a Computer for Playing Chess*, Philosophical Magazine 41, 1950, pp 256-275.