Admissibility in Opponent-Model Search

H.H.L.M. Donkers J.W.H.M. Uiterwijk H.J. van den Herik

Department of Computer Science,
Institute for Knowledge and Agent Technology,
Universiteit Maastricht, P.O. Box 616, 6200 MD, Maastricht, The Netherlands.
email: {donkers,uiterwijk,herik}@cs.unimaas.nl

Abstract

We start with showing experiments in the game Lines of Action (LOA) that hint at the existence of a large risk in using Opponent-Model (OM) search. Then the nature of the risk is investigated and its consequences are unveiled. It appears that, in contrast to Minimax, OM search is sometimes unable to repair large errors in the evaluation function(s). The result is that OM search, even when used in the ideal situation of perfect knowledge on the opponent, can lead to bad move decisions. Only when the pair of evaluation functions is *admissible*, OM search can be used safely and successfully. Unfortunately, in practice, admissibility is very difficult to achieve.

1 Opponent-Model Search

OM search [3] is a game-tree search algorithm designed to use knowledge on the opponent in order to exploit weak points in the opponent's search strategy. The algorithm is based on three strong assumptions on the opponent and the player. The first assumption is that the opponent (called MIN) uses a Minimax algorithm (or equivalent) with an evaluation function (V_{op}), a search depth and a move ordering that are known to the first player (called MAX). The second assumption is that MAX uses an evaluation function (V_0) that is *better* than the opponent's evaluation function. Third, it is assumed that MAX searches at least as deep as MIN. Informally, OM search is a derivative of Minimax in which MAX maximizes at max nodes, but selects at min nodes the moves that MIN would select.

2 LOA experiments

To investigate the *actual effectiveness* of OM search, we performed some experiments in the domain of LOA, a board game invented by Claude Soucie (and described in [7]). It is played on an 8×8 checkers board. The starting player uses twelve black stones, the other player uses twelve white stones. The black stones are placed in two rows along the top and bottom of the board, while the white stones are placed in two rows at the left and right edge of the board (the four corner squares remain empty). The object of the game is to move the stones until all stones of one colour are connected. The players move in turn. A player may move only one stone at the time (i.e., per turn) in a straight line in any of

the eight principal directions. The step length of the move is exactly equal to the total number of stones on the line of movement. Jumping is only allowed over the own stones. When the move lands on an opponent's stone, this stone is captured. In the implementation used in the experiments, a repetition of a board situation instantaneously ends the game (in a draw).

The game of LOA was selected for the experiments because the game is not trivial and also not too complex. Furthermore, there are very different evaluation functions available, which allows for the application of OM search. In the experiments, three evaluation functions are used. The first one (V_c) is a simple *centre-oriented function* that assigns values to stones depending on their board position: the four centre squares yield 10 points, their 12 surrounding squares yield 5 points, and the 20 squares surrounding the latter squares yield 1 point. The remaining 28 border squares yield no points. Opponent stones receive the same amount of points (expressed as minus points). The score of a board position is the total sum of all points assigned to the stones. Of course, in practice many additional aspects are used to evaluate a position. Two other evaluation functions (V_{NNa} and V_{NNb}) used in the tournament are *neural-network functions*. They are adopted from Kocsis' research on training neural networks to evaluate board positions in LOA [5].

We conducted three tournaments between artificial LOA players with the following set-up. In every tournament we tested two evaluation functions (indicated here by V_1 and V_2). In the tournaments the first player used OM search (with $V_0 = V_1$ and $V_{op} = V_2$) and the second player used α-β (with V_2). The evaluation function that the first player assumed for MIN was the same one that the second player actually used. So, the first player had perfect information of the opponent. To be able to compare the results of OM search with α-β, each tournament was repeated, but now both players used α-β, the first player using V_1, the second player V_2. The tournaments consisted of 30 different positions that had to be played to the end. Every position had to be played twice, reversing colours, i.e., the total number of games was 60. To investigate the influence of the search depth, every position was played using four different search depths (d = 2, 3, 4, and 5). No time limit was given.

The results of the LOA tournaments are presented in Tables 1, 2 and 3. For every LOA position in a match, a player obtains 2 points if the game played from that position is won by the player, 1 if the position leads to a draw, and 0 if the position is lost. The result of a match is the difference of points in that match.

$V_1 = V_{NNa}$, $V_2 = V_{NNb}$

d	OM vs. α-β	res	α-β vs. α-β	res
2	72 - 48	24	69 - 51	18
3	54 - 66	−12	62 - 58	4
4	58 - 62	−4	63 - 57	6
5	52 - 68	−16	50 - 70	−20

Table 1: Results of the first LOA tournament.

The first tournament (see Table 1) was between the two neural-net evaluators. From the fifth column it follows that the difference in quality between the two neural-net evaluators is not significant. However, the results for OM search are disappointing, especially for the depths 3 and 4. Although the opponent model is correct, the player using OM search is hardly

able to profit from the knowledge given. At search depths 3 and 4, using OM search with the correct opponent model even leads to a decrease in quality!

Because the two evaluation functions (V_{NNa} and V_{NNb}) are relatively close to each other, a new tournament was held between the evaluation functions V_{NNa} and V_c, of which we believed that V_c is much weaker than V_{NNa}. From the fifth column in Table 2 follows that V_{NNa} indeed is stronger than V_c, but the difference in quality is not overwhelming. However, the results for OM search in this tournament are disastrous. From a winning evaluation function, using the correct opponent model, OM produces a badly-losing player.

$V_1 = V_{NNa}$, $V_2 = V_c$

d	OM vs. α-β	res	α-β vs. α-β	res
2	59 - 61	-2	58 - 62	-4
3	54 - 66	-12	72 - 48	24
4	44 - 76	-32	69 - 51	18
5	44 - 76	-32	72 - 48	24

Table 2: Results of the second tournament.

Since OM search went so wrong in this tournament, a third tournament was held in which the evaluation functions switched place. This was done in order to see whether the error in OM search would diminish. Of course, the scores in the fifth column of Table 3 are just the opposites of the ones in Table 2. Now we see that the quality of the OM player is on average slightly better than in the previous tournament, but still not as good as α-β, and certainly not as good as we expected from OM search.

$V_1 = V_c$, $V_2 = V_{NNa}$

d	OM vs. α-β	res	α-β vs. α-β	res
2	63 - 57	6	62 - 58	4
3	56 - 64	-8	48 - 72	-24
4	46 - 74	-28	51 - 69	-18
5	42 - 78	-36	48 - 72	-24

Table 3: Results of the third tournament.

In conclusion, we remark that in all three tournaments, the OM player was given a serious advantage because the search depth is kept constant, independent of the time that is needed to search the tree. The intriguing question now is: what went wrong with OM search?

3 Risk in Opponent-Model Search

The assumptions that form the basis of OM search give rise to two types of risk. The first type of risk arises from imperfect knowledge on the opponent. When MIN is using a different evaluation function than assumed, or a different search depth or move ordering, MIN might select another move than the move that MAX expects. This type of risk has been described and analyzed in [4]. In the LOA experiments above, this risk is not present because MAX has perfect knowledge of the evaluation function of MIN. The second type of risk arises from the *quality* of the evaluation functions used. The assumption is that MAX uses an evaluation function that is better than MIN's evaluation function. This assumption leads to two questions: what does 'better' mean and what happens if MAX's evaluation function is not the better one? In order to define 'better', a more detailed discussion of evaluation functions in game trees is needed.

3.1 Static Evaluation Functions

A static evaluation function (or *evaluation function* for short) is used to compute a substitute for the game-theoretic value of a node in a search tree. It is called *static* because it only uses the static information from the game position at that node. The evaluation function is used when the game-theoretic value of the position is unknown.

In the ideal case, the evaluation value of a node is related to its game-theoretic value. An evaluation function is called *perfect* if at all max (min) nodes P in the game tree, the child of P with the highest (lowest) evaluation value also has the highest (lowest) game-theoretic value among its siblings. Clearly, if such a perfect evaluation function is available, search is unnecessary. In passing we note that Christensen and Korf [1] use a definition of perfect evaluation functions (Perfect Play) that is too strict. In their definition, a perfect evaluation function must yield the game-theoretic value for end positions and furthermore, making an optimal move must not change the evaluation value.

Perfect evaluation functions are not available for games that are of interest for game researchers in AI, except for the knowledge implicit in endgame databases. Hence, imperfect evaluation functions must be used. There are two ways to interpret imperfect evaluation functions. The first interpretation is that of an estimator of the *probability to win the game* from a position. This interpretation is often used by researchers that apply temporal-difference learning or other machine-learning techniques in games. The second interpretation of an evaluation function is that of an estimator of the *profitability* of a position. This second interpretation is used in most game-playing programs.

The probability interpretation of evaluation functions does not make sense in relation to OM search, because the probability to win is related to the strength of the opponent. Evaluation functions in OM search must therefore be interpreted in terms of profitability. Unfortunately, the concept of profitability is not exact and not measurable. In spite of this, assume that an evaluation function W exists that measures the profitability exactly. This function, by definition, is a perfect evaluation function. Furthermore, W should bare the property for all positions P and Q that if $W(P) \geq W(Q)$, the game-theoretic value of Q should not be lower than that of P. Imperfect evaluation functions are estimations of this function W, if the profitability interpretation is used. Now a 'better' evaluation function can be interpreted as a function that estimates W better. Of course, there are many ways to define estimation and, hence, to define 'better'. Fortunately, OM search itself provides a solution that will be exposed in the next subsection.

Before doing so, one additional definition has to be given. Two evaluation functions V_1 and V_2 are called *equivalent* if they impose the same order on positions $(V_1(P) \geq V_1(Q) \Leftrightarrow V_2(P) \geq V_2(Q))$.

3.2 Estimation errors in evaluation functions

How does the quality of the evaluation functions V_0 and V_{op} influence the performance of OM search? If V_0 is a perfect evaluation function and V_{op} is not, then clearly OM search will be successful. But V_0 being perfect discharges us from

using any search at all. Also if V_0 and V_{op} are equivalent, then OM search will be successful. But in this case, OM search will behave equal to Minimax and the extra efforts of OM search are a waste of time. But if V_0 and V_{op} are not equivalent and V_0 is not perfect, under which conditions will OM search then be guaranteed to be successful?

To give a clear answer to this question, the following alternative view on OM search is presented. Let T be the game tree that is under consideration by OM search. At every min node P, MIN determines which branch is selected. Let T_{min} be the subtree of T in which all branches at min nodes are removed, except for the branches selected by MIN and let $\mathcal{P}_{min}$ be the set of positions corresponding with the leafs of T_{min}. The remaining task of MAX is to find the position P^* in $\mathcal{P}_{min}$ that has the largest value of V_0.

Obviously, evaluation function V_{op} determines set $\mathcal{P}_{min}$, and V_0 determines P^*. Assume that V_0 and V_{op} start as perfect evaluation functions ($V_0 = V_{op} = W$). Assume further that we introduce *estimation errors* in V_0 and/or V_{op}. These are modifications in the evaluation functions that change the order that the functions impose on the positions. Estimation errors in V_0 and V_{op} only have effect on the outcome of OM search if they influence $\mathcal{P}_{min}$ or P^*. There are four types of estimation errors that can lead to such an effect.

Type-I error: V_0 overestimates a position in $\mathcal{P}_{min}$ This is the most serious error of OM search. V_0 can overestimate any of the positions in $\mathcal{P}_{min}$ and select any of them as the maximum. A type-I error is "repaired" if V_{op} also sufficiently overestimates the position that was overestimated by V_0. The overestimation by V_{op} of this position causes it to be removed from $\mathcal{P}_{min}$.

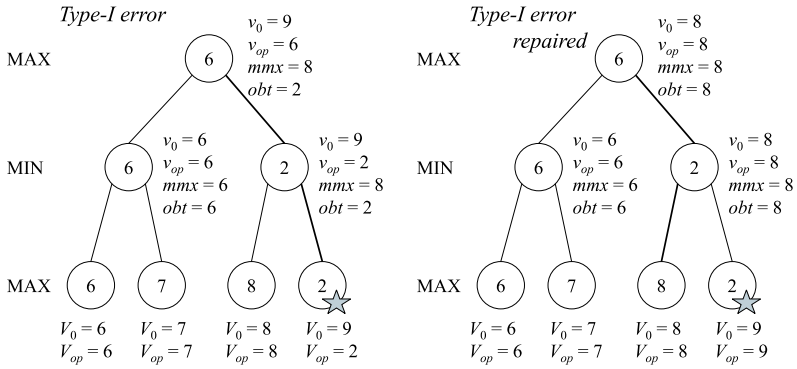

Figure 1: Example of a type-I error and a repair of this error.

The left tree in figure 1 gives an example of a type-I error. In the figure, the numbers inside the nodes give the true values (W) of the profitability of the nodes. Next to the nodes the values of V_0 and V_{op} (or v_0 and v_{op} for the internal nodes) are given. Two additional values are presented at an internal node: mmx is the minimax value that MAX would obtain if V_0 was used with minimax, and obt gives the truly obtained profitability for MAX, e.g., the true value of the chosen variation.

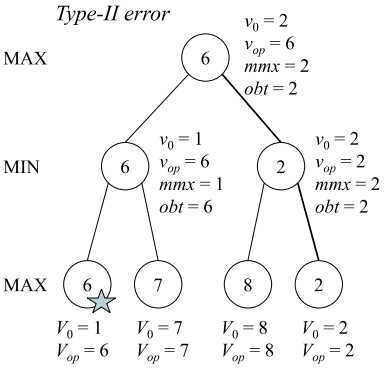

Figure 2: Example of a type-II error.

In the example a type-I error is generated by MAX's overestimation of the right-most leaf. This causes that MAX *beliefs* to obtain a value of 9 but in reality, MAX only obtains 2, which is 4 less than the true profitability of the root. The tree at the right gives a possible repair for this error. If MIN sufficiently overestimates the same position that MAX overestimates, this position will not be selected by MIN anymore. The difference between believed and obtained value disappears (both values are now 8).

Type-II error: V_0 underestimates a position in $\mathcal{P}_{min}$ This error is less serious for MAX because it only results in another move if the best position P^* is underestimated (figure 2). Furthermore, if P^* is heavily underestimated, the second-best position will take its place and the overall harm is limited. Type-II errors are also repaired when V_{op} overestimates the same positions, causing them to disappear from $\mathcal{P}_{min}$.

Type-III error: V_{op} underestimates a position that enters $\mathcal{P}_{min}$ This is the major error for MIN. By underestimation, any position could enter $\mathcal{P}_{min}$, even positions that are very unprofitable for MIN (see figure 3, left). MAX can profit the most from this error. The error is repaired if V_0 also underestimates these positions.

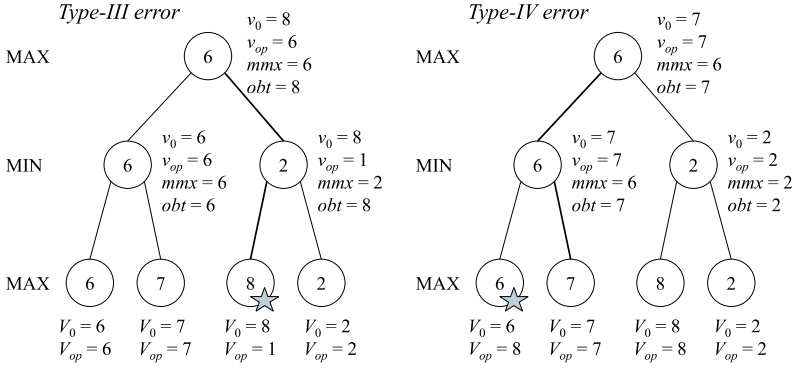

Figure 3: Example of a type-III error and a type-IV error.

Type-IV error: V_{op} overestimates a position in $\mathcal{P}_{min}$ The last type of error is less serious for MIN. It causes a position to disappear from $\mathcal{P}_{min}$ (see figure 3, right). MIN is not able anymore to profit from opportunities in this position, but a second-best position will take its place. This type of error cannot be repaired.

Errors due to overestimation and underestimation are not unique for OM search, but the effect of these errors in Minimax is bounded by the value of siblings,

just like the errors of type II and IV above. A single underestimation or overestimation cannot do much harm in Minimax, whatever the size of the error. The example in Figure 1 illustrates this. The overestimation of the right-most leaf with any amount larger than 8 causes Minimax to select the sibling of this node. Once this sibling is selected, the extent of the overestimation does not matter anymore. In OM search, the extent matters seriously. The value v_0 of the root depends on this overestimated value. If the example tree is a subtree of a larger search tree, this root value will be propagated up this larger search tree. The larger the overestimation, the larger the damage will be that it causes in this search tree. A single large type-I error can thus cause major damage to the position of MAX. Fortunately, there is also another side of this coin: a single large type-III error can lead MAX to a certain win.

4 Admissibility

For OM search to be at least as successful as Minimax, type-I errors are undesirable. The relation between V_0 and V_{op} must be such that type-I errors do not (or rather unlikely) happen. Furthermore, type-III and type-IV errors should occur as often as possible in order to let OM search profit from them. This leads to the following condition on the evaluation functions V_0 and V_{op}:

Admissible pair A pair of evaluation function (V_0, V_{op}) is admissible for OM search if V_0 is a better profitability estimator than V_{op}, and when V_0 never overestimates a position that V_{op} does not overestimate likewise.

Notice that this condition seems too strong, because only type-I errors should be avoided. But since it is impossible to foresee all search trees for a given game, we have to stay on the safe side. The condition is also not sufficient to prevent type-I errors completely. Especially when the quality of V_0 is poor, the repair of type-I errors might fail because in such case it could happen that all children of one min node are seriously overestimated, and one of them enters $\mathcal{P}_{min}$. But in such case, also Minimax would yield bad results.

Managing the risk of estimation errors in OM search is analogous to risk management in the heuristic optimization algorithm A^*. In order for A^* to result in an optimal solution, the heuristic function $h()$ must be *admissible*, which means that $h()$ should never *overestimate* the real distance ($h^*()$) to the goal. This restriction on $h()$ is quite similar to the restrictions stated above. Pearl [6] provides a suggestion on how to discover an admissible function $h()$: create a simplified model of the problem domain by either relaxing or overconstraining the original model of the problem domain. Then use the distance function of the simplified model as an estimate of the true distance. The construction of the simplified model must be such that it can be proven that $h()$ is admissible.

Although this model-based approach to heuristic functions is also used in manually-built static evaluation functions for games, the need for some kind of admissibility only appears in OM search. It is possible to apply Pearl's method to produce suitable pairs of evaluation functions. Instead of one simplified model, now two models are needed, one for MAX and one for MIN. The model for MIN

should be almost the same as MAX's, but it must clearly overlook some aspects of the game. By building an evaluation function based on these simplified models, it is likely that these evaluation functions are suited for OM search. (An example of this method would be to take a regular chess evaluation function for MAX and to give MIN the same function but leaving out the threats by the opponent's knights.)

Analogous to the situation in A^*, complete admissibility is not always necessary. If suboptimal solutions are acceptable in A^*, non-admissibility is tolerable to a certain extent. For OM search, minor non-admissibility is acceptable if the extent of the estimation errors is limited, e.g., when the quality of the evaluation function V_0 is high.

5 Conclusions

Unless (nearly) admissible pairs of evaluation functions are used, it is unwise to apply OM search. The experimental evidence and the theoretical argumentation make this clear. This does not mean that OM search must be abandoned. In some cases, it will be possible to construct admissible pairs of evaluation functions, in other cases, the quality of the evaluation function V_0 is known to be sufficiently high to allow non-admissibility. A third approach is to adapt OM search in such way that the disadvantage of the possible errors is alleviated. One such adaptation is called *probabilistic* OM search, which is described in [2].

References

[1] Christensen, J. and Korf, R.E. (1986). A Unified Theory of Heuristic Evaluation Functions and its Application to Learning. *Proceedings of the AAAI'86*, pp. 148–152.

[2] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2001). Probabilistic Opponent-Model Search. *Information Sciences*, 135(3-4), pp. 123–149. ISSN 0020-0255.

[3] Iida, H., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Herschberg, I.S. (1993, 1994). Potential Applications of Opponent-Model Search. Part 1: The Domain of Applicability. *ICCA Journal*, 16(4), pp. 201–208. Part 2: Risks and Strategies. *ICCA Journal*, 17(1), pp. 10–14. ISSN 0920-234X.

[4] Iida, H., Kotani, I., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1997). Gains and Risks of OM Search. In *Advances in Computer Chess 8* (eds. H.J. van den Herik and J.W.H.M. Uiterwijk), pp. 153–165. Universiteit Maastricht, Maastricht, The Netherlands. ISBN 90-6216-2347.

[5] Kocsis, L., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2000). Learning Time Allocation using Neural Networks. In *Working Notes of the Second International Conference on Computers and Games, CG'2000*, pp. 297–314. Hamamatsu, Japan.

[6] Pearl, J. (1984). *Heuristics, Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, Reading, Massachusetts. ISBN 0-201-05594-5.

[7] Sackson, S. (1992). *A Gamut of Games.* Dover, New York, NY. ISBN 0-486-27347-4.