Using Aspiration Windows for Minimax Algorithms

Reza Shams *
Alcatel-ELIN
Forschungszentrum
Ruthnerg. 1-7, A-1210 Wien
Austria

Hermann Kaindl
SIEMENS AG Osterreich
GudrunstraBe 11
A-1101 Wien
Austria

Helmut Horacek
Universitat Bielefeld
Postfach 8640
D-4800 Bielefeld 1
Germany

Abstract

This paper is based on investigations of several algorithms for computing exact minimax values of game trees (utilizing backward pruning). Especially, the focus is on trees with an ordering similar to that we have actually found in game playing practice. We compare the algorithms using two different distributions of the static values, the uniform distribution and a distribution estimated from practical data. Moreover, this is the first systematic comparison of using aspiration windows for all of the usual minimax algorithms. We analyse the effects of aspiration windows of varying size and position.

The use of an *aspiration window* not only makes *alpha-beta search* competitive, but there also exists previously unpublished dependencies of its effects upon certain properties of the trees. In general, the more recently developed algorithms with exponential space complexity are not to be recommended for game-playing practice, since their advantage in having to visit fewer nodes is more than outweighed under practical conditions by their enormous space requirements. Finally, we propose a method for an analytic determination of estimates of the *optimal window size,* presupposing evidence about some characteristic properties of the domain of application. In summary, we discovered and found empirical evidence for several effects unpredicted by theoretical studies

1. Introduction and Background

For long, the only known method for computing the exact *minimax value* of a *game tree* without generating this tree in its entirety was the so-called *alpha-beta algorithm* (for a description and historical review see for instance [Knuth & Moore 75]). In the meantime, several other pruning algorithms have been found. A short review and references will be given in Section 2, and more elaborate descriptions can be found for instance in [Kaindl 90] and in the cited references.

Naturally the question arises which is the most efficient algorithm (more precisely, under which conditions). For some of these algorithms there exists exact formulae for the expected number of terminal positions on certain game trees (random trees with uniform branching degree and distinct terminal values) [Pearl 84]. These render alpha-beta (in its pure form) clearly less efficient than its competitors. However, for trees of the kind occurring in real games the situation is not this clear: [Marsland & Campbell 82) defined *strongly ordered* trees, in which 70% of the time the first branch from each node is best. [Campbell & Marsland 83] and [Marsland *et al.* 871 investigated different tree ordering types using a form of Monte Carlo simulation.

With respect to issues of relative efficiency, such criteria seem to be a promising attempt to model the actual conditions occurring in computer game playing. Therefore, we actually gathered such data (among others) with the chess program MERLIN.. The statistics compiled from 438 move decisions (most of them under tournament conditions) showed a mean value of 90.3% (with a standard deviation of 8.2%) for the relative frequency of the first branch from each node being "best" (more such statistics can be found in [Kaindl 88]). Since we were mainly interested in a comparison using parameters which reflect best the practical conditions occurring in real game playing, we choose to investigate very strongly ordered trees.

Comprehensive experiments under such conditions showed some interesting results (see [Kaindl *et al.* 89)). As a main result, alpha-beta (in its pure form) is also less efficient than its competitors on ordered trees, although it profits most from an increased ordering of the trees. In practice, however, many enhancements to alpha-beta are used (see [Schaeffer 86, 891 for a comprehensive set of experiments comparing their performance in the domain

* The contribution of this author was performed as Diplomarbeit" supervised by the second author (comparable to a Master's thesis) before joining Alcatel-ELIN Forschungszentrum. It was supported by SIEMENS AG Osterreich, especially in providing facilities.

1 MERLIN is a collaborate effort of the second and the third author together with Marcus Wagner. It played some major computer tournaments, for instance it tied for 10th out of 22 participants at the World Computer Chess Championship in 1983, and for 6th out of 24 participants at the World Computer Chess Championship in 1989.

of computer chess). While most of these enhancements just serve the purpose of achieving improved ordering, it is also common practice to use an *aspiration window* for alpha-beta, but to our best knowledge there is no published account of the effects of varying window size. Note, that alpha-beta was only compared in its pure form (in particular without such windows) in the previous theoretical studies. Our comparisons gave a strong indication of its improvements using such a window, which made it fully competitive to the newer algorithms. While this enhancement is often used for improving alpha-beta, we are not aware of a published comparison of the effects of using windows for the other algorithms. Hence, we found it interesting to compare "all" the minimax search algorithms using such a window.

Moreover, we had serious doubt that in practice all the possible values are equally likely resulting from the static evaluator, as usually assumed in theoretical studies. Therefore, we became interested in the effects of using a distribution of the values estimated from practical data, compared to having them equally distributed. The question is, how the simplifying assumption of uniform distribution affects the results.

Since using an aspiration window always bears the danger of having to repeat the search *(re-search),* in case the minimax value is outside the window, the effort for computing it may even become greater than that when using no window at all. While the techniques of estimating the positioning of such a win dow are quite developed, its *size* is only determined by experienced guessing. Hence, we present the first method for an analytic determination of estimates of the optimal window size.

2. Algorithms Compared

Like alpha-beta (AB), the newer algorithms SCOUT [Pearl 80] and *palphabeta* [Fishburn & Finkel 80] have linear space requirements. These have similar average performance [Musczycka & Shinghal 85]. In our experiments, we actually used a slightly improved variant called *negascout* (NS) [Reinefeld 83]. SSS* [Stockman 79] and DUAL* [Marsland *et al.* 87] are *best-first* search algorithms and consequently have exponential space requirements.

Enhancements to NS have led to INS *{informed* NS), which saves information for the case that subtrees must be re-searched [Marsland *et al.* 87]. Hence, INS has storage requirements comparable to SSS* or DUAL* (see also [Schaeffer 86]). These enhancements have similar effects as the use of the usual hash tables in game playing programs.

AB is normally used in practice in its refined version *aspiration alpha-beta* (AAB). Usually, game playing programs can estimate the minimax value quite accurately to lie within a certain range, even before the search is done. (This ability is strongly related to the use of *iterative deepening* in most programs: searching the tree successively deeper and deeper, see e.g. [Korf 85].) This range is usually called the *aspiration window,* and its bounds are used as an (artificial) initialisation of the parameters a and B at the root. Quite similarly to NS, a re-search with modified parameters is necessary, when the actual minimax value is not inside this window.

More precisely, a slightly modified version of the alpha-beta algorithm is used in this context today which was proposed by [Fishburn & Finkel 80]. In case the minimax value is outside the original window, it can provide tighter bounds for the re-search. Hence, they called it "Fail-soft Alpha-Beta" (abbreviated here by FAB). More details on this can be found in [Marsland & Campbell 82]. However, we could not find any quantitative data on the savings compared to the original version of aspiration alpha-beta which is *not* utilizing the advantages of FAB. (This version we call OLDAAB here.)

In the following we generalize the method of using a window for all the given algorithms. The use of an aspiration window for NS and INS is analogous to that for AB. We call these algorithms ANS and AINS. However, for the corresponding versions of SSS* and DUAL*, ASSS* and ADUAL*, the window is useful only from *one* side, by initialising their internal bound with the upper or the lower bound of the window, respectively. When the minimax value is inside the window, this is obvious from their way of handling their internal bound. Of course, when the value *of* the second bound of the window is reached or exceeded, stopping the search is possible according to [Campbell & Marsland 83] and [Reinefeld 89]. However, when the exact minimax value has to be determined this is not to be recommended. A re-search with the new bound would have to be done, in effect wasting resources by searching again parts of the tree whose results would be available when continuing the stopped search. Therefore, ASSS* and ADUAL* cannot utilize their second bound, and thev should not be hindered by it. A pseudo-code formulation of the key part of ASSS* looks like the following.

```
Value := SSS*(Position, b),
if Value = 6 then
    Value := SSS*(Position, +<*>);
end if;
```

Analogously, the key part of ADUAL* can be formu lated as follows:

```
Valuer DUAL*(Position,a);
if Value = a then
    Value := DUAL*(Position, -°°);
end if;
```

3. Experiment Design

As usual for mathematical analysis as well as simulation studies, the standard model of a *uniform* game tree of width *w* and depth *d* has been used. In our experiments, we kept *w* fixed *(w* = 5) while varying *d* from 3 to 9.

Since the approach of generating such trees proposed and used by [Schaeffer 86] requires only *O(wd)* storage, we have also adopted it for our experiments. It works on the principle of deriving the value of a subtree from information available at its parent node. A minimax value is chosen and used with the specified ordering criteria to build a tree from the root down. The ordering parameters can be specified here in the form of *w* weights reflecting the chance that each of the *w* moves from anv node will lead to a subtree having the minimax value of this node. A more detailed description can also be found in [Marsland *et al.* 87].

While we have made some few experiments with *random* trees *{independent and identically distributed,* see [Pearl 84]), our emphasis was on *very* strongly ordered trees. For the chance of the first branch being best we chose the values 70%, 80%, and 90%. Since the effect of the ordering of the remaining moves appears to be largely negligible [Kaindl *et al.* 89], we assumed them to have equal chance of being best.

Let the distribution function $Fx(x) = P(X \leq x)$ characterise the random variable X ranging over the same interval as the static evaluation function *fin).* The probability $p — P(X — x)$ for node n is the probability of the event that *fin)* returns x. $FX(x)$ has been assumed to be uniform (as usual) and alternatively estimated from practical data of a chess program. The compiled statistics revealed that the static values around 0 occured very frequently. For this reason, a normal distribution could not fit these data well (according to statistical tests). Hence, we defined a distribution function $Fx_1(x)$ approximating our data as an estimate of Fx (x) (see [Shams 90]). It seems that the exact shape of the distribution function is not really important for the results of the simulation runs, only the values around 0 must have a significantly higher probability.

Several different *window sizes* were chosen in our experiments: 1, 79, 159, 239, 319, 399, 479, and 639 values *inside* the window, out of the range -999...999. (This corresponds to 0.05%, 4%, 8%, 12%, 16%, 20%, 24% and 32% of the total value range.) The window containing 1 value has been called *narrow window* by [Marsland *et al.* 87] and is of special interest, because it is the smallest window possible for a successful search. The *window position* has been chosen uniformly over the whole range. In case the minimax value is outside the window, we have performed additional experiments, in which it was more likely that the window position is "closer" to the minimax value (a more realistic assumption).

In these experiments, 20 simulations were done for the case, that the minimax value *MM* is inside the window (a, 6), and 20 in case it is outside. This means 10 simulations with $MM \leq a$, and 10 with $MM \geq b$. In both of these cases AAB, ANS and AINS perform a re-search in the sense of "failing low" and "failing high", respectively. However, ASSS* only re-searches if $MM \geq b$ and ADU AL* does so if $MM \leq a$. The overall performance figure is gained by weighting the data according to the frequency observed in practice (see below).

Generally, every stochastic event has been simulated by a call to a *pseudo-random number generator*, parameterized independently of the relative frequencies achieved earlier in the tree generation process. While the *seed* for the random number generator was different for each of the generated trees, each of the algorithms searched the same trees, of course. As a measure of performance we selected the *number of bottom positions* (NBP), as usual. It is known, that some of the algorithms (SSS* and DUAL*) have more overhead than others (AB, NS and INS) [Marsland *et al.* 871. However, whether this is significant (in the sense of running time) depends strongly on the cost of going from one node to another and evaluating terminal nodes in the domain of application, as well as the encodings. In particular, the running times of SSS* and DUAL* in the simulations are significantly larger than those of AB, NS and INS. The situation is atypical, however, since in the simulations just calls to a random number generator are performed instead of real operations, which are usually more expensive.

The empirical data have been prepared in illustrative figures. Of course, only a very limited selection of these can be presented here. A comprehensive set is available, however, in [Shams 90].

4. Results of Using Aspiration Windows

Specific results regarding AB and AAB should be noted. Many of the published comparisons of minimax algorithms as well as our experiments show that AB (in its pure form) is less effective than its newer competitors. Some investigations of its improvement through an *aspiration window* indicate savings on the order of 20% (according to our experience with MERLIN and [Gillogly 78, Baudet 78, Marsland 83]), which make the alpha-beta algorithm competitive. This is also supported by our results with AAB. They showed even a slight superiority of AAB in case of using a window of realistic size.

The deeper the search, the more can be saved in absolute terms of NBP using AAB instead of AB. Compared to the algorithms which do not use a window, AAB shows the least degradation in performance (relative to the minimal tree) when the search depth is increased. From our empirical data it seems that the use of an aspiration window leads to a certain reduction in branching factor, depending on the window size.

So the question arises, whether the other algorithms can utilize a window just as well. Based on experiments with chess programs, we could conjecture that NS and its variants only marginally improve by use of an aspiration window. In fact, these algorithms use a similar idea internally. As described above, ASSS* and ADUAL* can use windows only from one side. Actually, the results of our experiments provide empirical evidence that AB profits most from using an aspiration window.

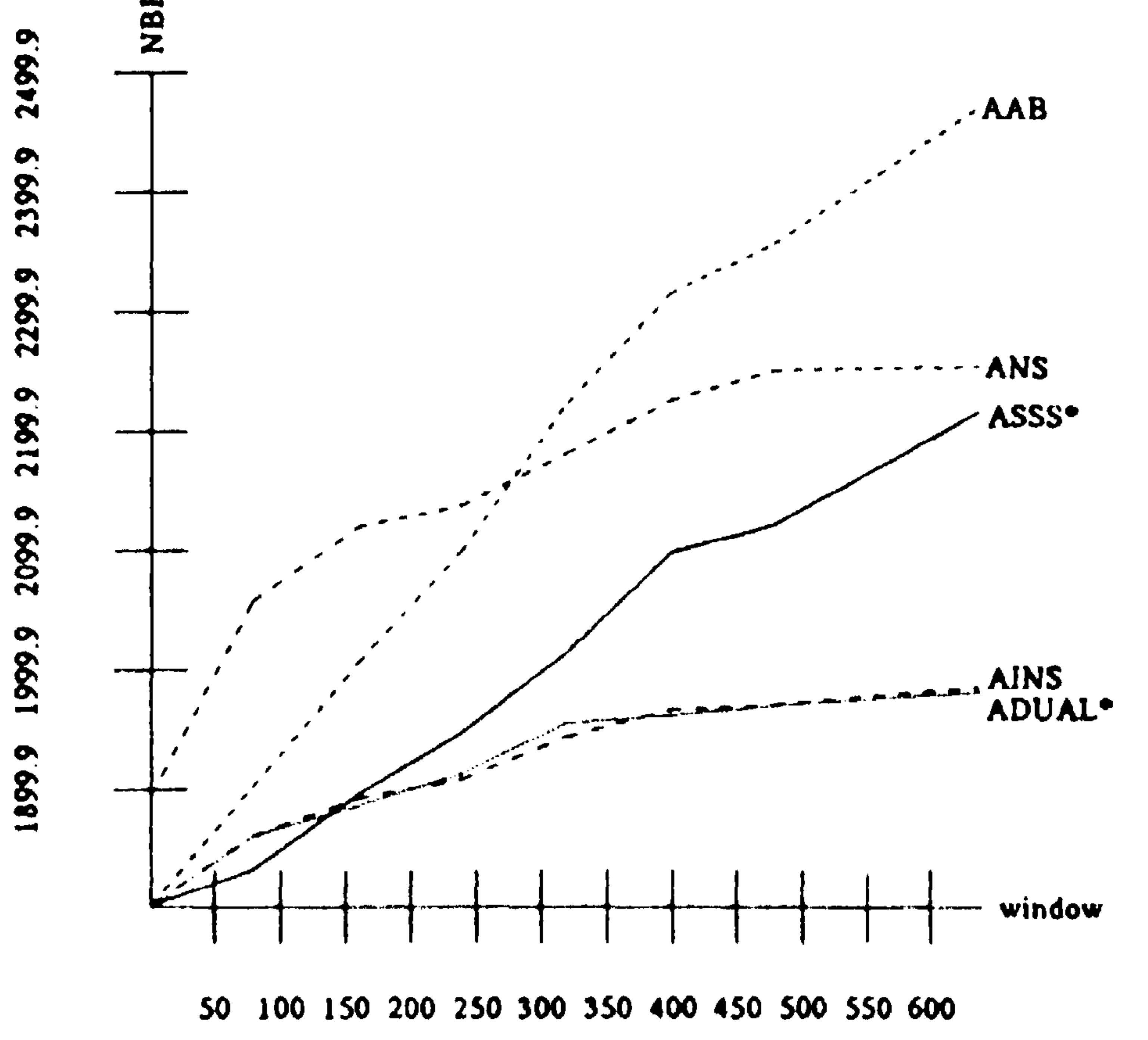

Fig. 1. Comparison of algorithms with varying window size to minimal tree - NBP, averaged over depths 3 to 9, distribution of terminal nodes' values according to FX_1 (x), width = 5.

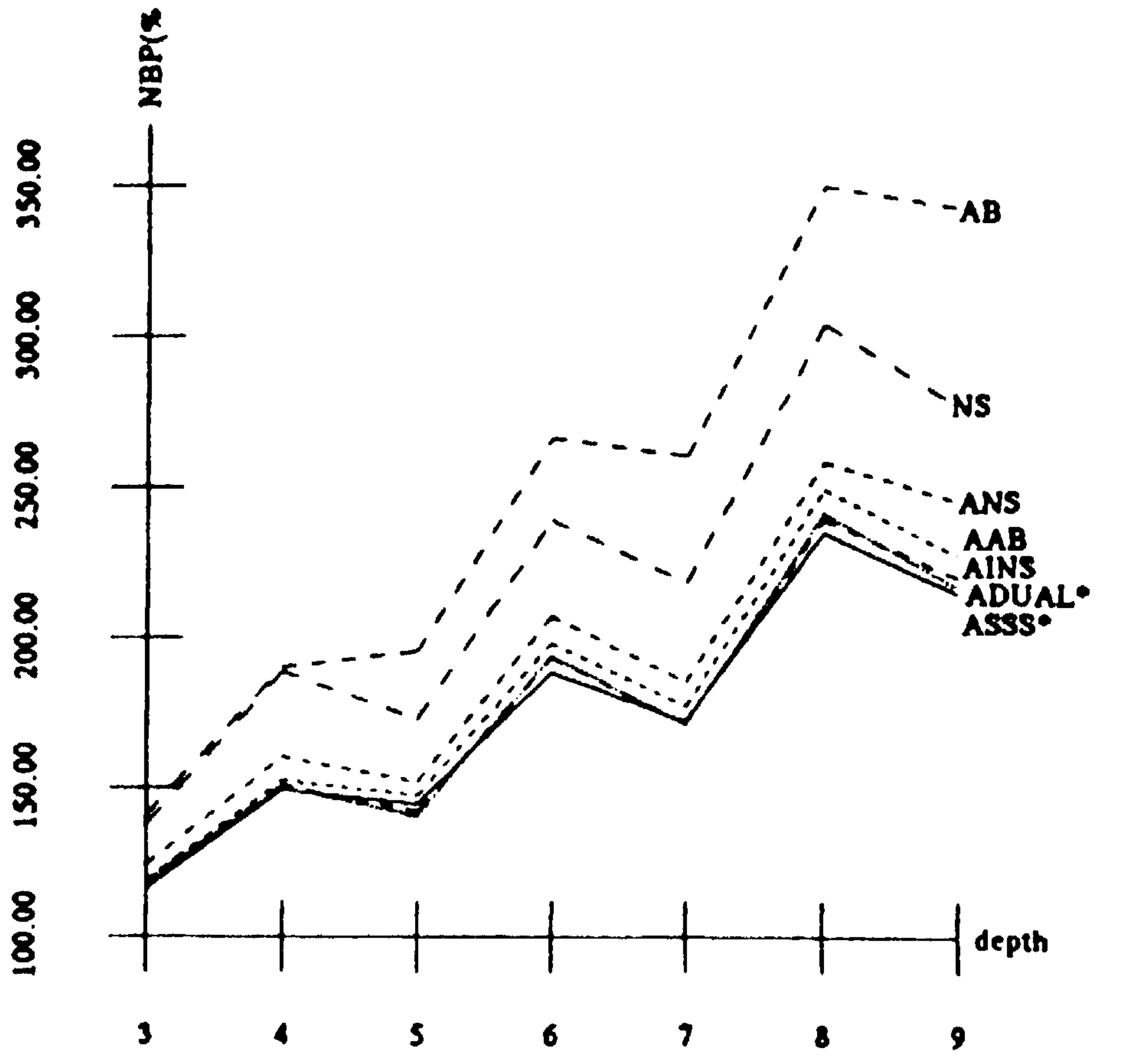

Fig. 2. Comparison of algorithms to minimal tree -NBP(%), one re-search (in case of aspiration algorithms), window containing 79 values, 80% first-move-best, distribution of terminal nodes' values according to FX, (x), width = 5.

Now let us have a more detailed look at these results. First we consider the case that the minimax value is *inside* the window. When using small windows like those in practice, ANS is the worst of the algorithms compared here, although NS is clearly better than AB. In the average, the gain in efficiency by decreasing window size is approximately "linear", though with a different slope for the various algorithms (see Fig. 1). Moreover, we observed some exceptional cases, when a larger window results in a more efficient search than a smaller one. This is particularly remarkable for AAB, but it is only possible here when both bounds of one window differ from the bounds of the other. An example and a proof sketch for this phenomenon can be found in [Shams 90].

In *very* strongly ordered trees, the differences caused by various window sizes are small. There may be the hint for practice to avoid too small windows, since the risk of having to re-search may be greater than the savings. The method for estimating the window size presented below takes this into account implicitly.

If the first search fails, a re-search has to be done with an opened window. In these cases, ASSS* is the most efficient algorithm. With increasing ordering of the trees, AINS becomes nearly as efficient as ASSS*. ANS is here better than AAB, just as NS is better than AB, while without 'Tailing" and when using smaller windows AAB is more efficient than ANS.

Again, there is an approximately linear behavior according to the window size, though with different slopes and more pronounced differences among the algorithms. Moreover, there is a different reason here for exceptions: An initially larger window may well cause a more costly first search, but this may result in a better bound for the second one, in effect reducing the overall effort. These exceptional cases are more frequent than those occurring when the minimax value is *inside* the window.

There is also an interesting phenomenon when the minimax value is identical to one of the bounds. Each of the two searches may visit fewer bottom nodes than the minimal tree contains, while the sum must of course be at least as large.

In general, our results showed that "failing low" is more expensive than "failing high" This observation coincides with the results of experiments with the minimal window as reported by [Marsland *et al.* 87).

As described above, ASSS* and ADUAL* can use only one of the window bounds. Hence in our experiments, re-searches are performed only in 10 instead of 20 cases, in which the minimax value is outside the window. Taking the results of all 20 simulation runs into consideration, these algorithms are the most efficient ones, ASSS* being the best. However, concentrating on the re-search cases only, ADUAL* is surprisingly inefficient, while ASSS* is still the best. Plausible reasons for this phenomenon are that ADUAL* visits comparably many nodes in the initial failing search, and it "fails low", which we have observed to be more costly than 'Tailing high" (as SSS*does).

Focussing on practical conditions, a comparison should be made on combined data. Hence, we have chosen the frequency of *re -searches* (1 out of 20 cases) according to our experiences with MERLIN, which have been confirmed by statistics compiled from the five games played at the World Computer Chess Championship in 1989. There, out of 815 individual searches using AAB (usually, *iterative deepening* uses more than one search for one move decision) a mean value of 5.4% (with a standard deviation of 22.6%) resulted for the relative frequency of re-searches. Therefore, we weighted the re-search data with a factor of 1/20 and the remaining ones with 19/20. This means that the essential results of the case in which the minimax value is inside the window still hold. Fig. 2 shows the relative efficiency of the investigated aspiration algorithms, and compares them also to AB and NS.

The results of our experiments with *random* trees suggest that the relative gain in efficiency by using an aspiration window is nearly the same for all investigated algorithms, except that AB and NS (the ones with linear storage requirements) profit more. However, their aspiration versions are still less efficient than those of the others.

Our comparison of AAB with OLDAAB revealed a previously unknown property of FAB. With increasing search depth their efficiency becomes more and more the same. Empirical data showing this phenomenon are illustrated in [Shams 90, Figs. 90-101]. As we found out, the main reason is that with increasing depth it is more likely that FAB returns the same value as the original AB, namely the alpha or beta bound. Hence, it appears that the advantage of FAB over the original alpha-beta algorithm is only marginal. However, since it cannot be worse and its overhead is negligible, FAB is still to be preferred.

5. Effects of Different Distributions

All the algorithms are more efficient on trees with uniform distribution of terminal values than on those having terminal values distributed according to Fx_1 • A plausible reason for this effect is as follows. As usual in practice, the minimax value lies more likely near the mean of the distribution of all the values. In contrast, when using the uniform distribution, more often large values (positive and negative) outside the window occur, which result more frequently in cutoffs.

Especially AB shows more gain in efficiency from a better ordering on trees with terminal values distributed according to Fv , presumably because there is simply more to gain. In the average, DUAL* is most efficient on trees with both distributions, but using Fx_t the performance of INS becomes essentially the same (see [Shams 90, Figs. 9-11]). On trees with uniform distribution of the terminal values, SSS* is even slightly more efficient than DUAL* when searching to even depths, while it is clearly worse for odd depths. Fx_1 makes SSS*'s performance slightly worse. In general, the results of dependencies on the distribution of terminal values also hold for the aspiration variants of these algorithms.

6. Estimating an Optimal Window Size

One goal of our experiments is to produce a basis for determining suitable search parameters to minimize the overall cost of the search (measured by NBP). Hence, certain properties of the domain have to be estimated or empirical evidence has to be collected by means of statistical observations. In order to determine a window size which can be expected to produce the lowest possible cost, we need data about two properties of the domain:

1. the distribution of the static values *Fx (x)* as described above, and
2. the distribution of the minimax values *Fy (x) — P (Y <* x) which characterises the random variable *Y* also ranging over the same interval as *X, p = P* (Y =x) for node *n* is the probability of the event that the minimax value *MM (n)* is identical to *x.*

The function *Fy ix)* is used to estimate the probability of avoiding a re-search, depending on the window size. Of course, *Fx(x)* and *Fy(x)* are related to the static evaluation function used.

The cost of a search depends on whether the minimax value is inside (C_{in}) or outside $iC_{out})$ the pregiven window, and consequently in the average case on the frequency of each of these occurrences. C_{in} and C_{out} depend on *Fx(x)* and on the window size s. As slight simplifications which seem to be justified in most practical cases of interest we assume s to be odd, and a window centered around 0 (the data collected have been shifted appropriately). The total cost can be computed according to the formula

$$Ctotal = C\hat{}\ iFx <x),\ S) \blacksquare {}_{(>}\ {}_{M}\ f\text{-}™ F_Y' (x)\ dx\ +$$
$$Cout\ (Fx <x).\ s)\quad (l\text{-}\hat{}_{lv2}\ r^{l>n} F_Y'\ (x)\ dx)$$

If the functions can be formulated analytically, the derivation $oiC_{total}i$ with respect to *s* can be computed to get the extreme values, the minimum of which is the optimal window size. For our purpose, *Fy (x)* is based on statistics compiled from the tournament games of the chess program MERLIN at the World Championship in 1989. Unfortunately, the usual continuous distributions for estimating *Fy* (x) could not fit these data well (according to statistical tests). Therefore, we had to use the discrete values of our

TABLE 1 s_{opt} is the calculated value for the estimate of the optimal window size based on our data from practice. The value in parantheses is scaled in "pawn units", the usual measure in computer chess practice. $C_{total}i$ is the total cost in NBP for this minimum window size based on the simulation results.

	AAB	ANS	AINS	ASSS*	ADUAL*
s_{opt}	81(1.23)	53(0.83)	135(2.11)	81(1.27)	123(1.92)
Ctotal	1997.59	2102.85	1914.82	1881.82	1898.33

statistics. As for the functions C_m and C_{out} we have applied interpolation between the data obtained from the simulation runs. The results are summarized in Table 1.

Although the optimal window size is significantly different for each of the algorithms, their absolute differences of cost are within a margin of about 5 percent for "reasonable" choices of the window size (about 50 to 150). Nevertheless, a correlation between total cost and optimal window size can be observed. The difference between AINS and ADUAL* manifests in larger cost of re-searches for AINS. Hence, a larger window for AINS is plausible despite its larger cost. The estimate of an optimal window size for SSS* is due to its comparably low value for C_{in} at a specific window size (79). In other regions, its cost is always slightly higher than that of DUAL*.

7. Conclusion

The main results can be summarized as follows:

- Pure alpha-beta search is the least effective backward-pruning algorithm also on very strongly ordered trees. However, using an *aspiration window* can make it competitive since it profits most from using such a window. Under realistic conditions AAB is fully competitive (see Fig. 2), which is also supported by experience with computer chess. Moreover, we have observed previously unpublished dependencies of using such a window upon certain properties of the trees.
- While ASSS* and ADUAL* visit slightly fewer nodes than their counterparts, we would suggest that they should not be used especially in domains where good ordering can be achieved, because this minor advantage is more than outweighed by their exponential space complexity. Achieving such an ordering should be feasible in most structured domains, using methods like the *history heuristic* of [Schaeffer 86, 89]. Moreover, even though these algorithms visit fewer nodes, they are likely to be slower in practice due to their overhead in managing the list of "open" nodes.
- Regarding the algorithms with linear space complexity, NS can only slightly be improved by using an aspiration window (ANS). Using smaller windows, AAB has been found to be more efficient than ANS, and only the variant of NS with exponential space complexity, AINS, is comparably better (see Fig. 2).
- Our comparison between the variants of aspiration AB using FAB in contrast to the original version of alpha-beta search revealed that the deeper the search the more likely FAB returns one of the bounds as its result.
- The assumption of uniform distribution of terminal values must be handled with care, since it affects the results.
- Presupposing evidence about some characteristic properties of the domain of application, we found a method for an analytic determination of estimates of the optimal window size for each of the algorithms compared.

Purely theoretical studies did not provide us with these results, since realistic modelling of the conditions in practice makes the analysis at least very difficult. More and more simplifications often lead to the loss of important conditions. For instance, alphabeta search under realistic conditions is by far not as inefficient as suggested by theoretical analysis. Purely empirical studies in specific domains, on the other hand, always leave doubt on the generality of their results. Our experience with computer chess is consistent with the results of the simulation studies (as far as they are comparable). We hope that our approach of less rigid modelling and the use of simulation studies can help to bridge the gap between theory and practice.

Acknowledgments

We would like to thank Jonathan Schaeffer for *giving* us a very good basis for our experiments, in providing us especially with the C code of his tree generation algorithm. Wolfgang Ginzel's effort in writing software For plotting the figures is greatly acknowledged.

References

Campbell, M.S., and Marsland, T.A., A Comparison of Minimax Tree Search Algorithms, *Artificial Intelligence* 20 (4), 1983, 347 367.

Fishburn, J., and Finkel, R.A., *Parallel Alpha-Beta Search on Arachne.* Tech. Report #394, Department of Computer Science, University of Wisconsin, Madison, Wis., July, 1980.

Kaindl, II., Useful Statistics from Tournament Programs, *ICCA Journal 11(4)* 1988, 156-159.

Kaindl, H., Tree Searching Algorithms, in *Computers, Chess, and Cognition.* T. A. Marsland and J. Schaeffer, Eds., New York: Springer-Verlag, 1990, 133-158.

Kaindl, H., Wagner, M., and Horacek, H., Comparing Various Pruning Algorithms on Very Strongly Ordered Game Trees, *Proc. New Directions in Game-Tree Search Workshop,* Edmonton, Canada, May, 1989, 111 120. A comprehensive version is avail able as Tech. Report #50, Department of Statistics and Computer Science, University of Vienna, Austria, January, 1988.

Knuth, D.E., and Moore, R.W., An Analvsis of Alpha Beta Pruning, *Artificial Intelligence 6(4),* 1975,293-326.

Korf, R.E., Depth-First Iterative-Deepening: An Optimal Ad missible Tree Search, *Artificial Intelligence 21* (1), 1985, 97-109

Marsland, T.A., and Campbell, M.S., Parallel search of strongly ordered game trees, *ACM Comput.Surv.* 14(4), 1982,533-552.

Marsland, T.A., Reinefeld, A., and Schaeffer, J., Low Overhead Alternatives to SSS*, *Artificial Intelligence* 31 (2), 1987, 185-199.

Musczycka, A., and Shinghal, R., An empirical comparison of pruning strategies in game trees, *IEEE Trans. Svst. Man Cybern.* 15(3), 1985,389-399.

Pearl, J., Asymptotic Properties of Minimax Trees and Game Searching Procedures, *Artificial Intelligence* 14(2), 1980, 113-138.

Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, Mass: Addison-Wesley Publ. Co., 1984.

Reinefeld, A., An Improvement of the Scout Tree Search Algorithm, *Journal of the International Computer Chess Association* 6 (4), 1983,4-14.

Reinefeld, A., *Spielbaum-Suchverfahren.* Berlin: Springer-Verlag, 1989.

Schaeffer, J., *Experiments in Search and Knowledge.* Ph.D. Thesis, University of Waterloo, Ontario, May, 1986.

Schaeffer, J., The History Heuristic and Alpha Beta Search Enhancements in Practice,*IEEE Trans. Pattern Anal. Mach. Intell.* PAMI-11 (11), 1989, 1203-1212.

Shams, R., Ein experimenteller Vergleich ausgewahlter Suchverfahren. Diplomarbeit, Institut fur Praktische Informatik, Technische Universitat Wien, 1990.

Stockman, G.C., A Minimax Algorithm Better than Alpha-Beta?, *Artificial Intelligence* 12(2), 1979, 179-196.