B* Probability Based Search

Hans J. Berliner
Chris McConnell
22 June 1995

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was sponsored by the National Science Foundation under Grant Number IRI-9105202.

Abstract

We describe a search algorithm for two-player games that relies on selectivity rather than brute-force to achieve success. The key ideas behind the algorithm are:

1. Stopping when one alternative is clearly better than all the others, and
2. Focusing the search on the place where the most progress can likely be made toward stopping.

Critical to this process is identifying uncertainty about the ultimate value of any move. The lower bound on uncertainty is the best estimate of the real value of a move. The upper bound is its optimistic value, based on some measure of unexplored potential. This provides a **I-have-optimism-that-needs-to-be-investigated** attitude that is an excellent guiding force. Uncertainty is represented by probability distributions. The search develops those parts of the tree where moving existing bounds would be most likely to succeed and would make the most progress toward terminating the search. Termination is achieved when the established real value of the best move is so good that the likelihood of this being achieved by any other alternative is minimal.

The B* probability based search algorithm has been implemented on the chess machine Hitech. *En route* we have developed effective techniques for:

- Producing viable optimistic estimates to guide the search,
- Producing cheap probability distribution estimates to measure goodness,
- Dealing with independence of alternative moves, and
- Dealing with the Graph History Interaction problem.

This report describes the implementation, and the results of tests including games played against brute-force programs. Test data indicate that B* Hitech is better than any searcher that expands its whole tree based on selectivity. Further, analysis of the data indicates that should additional power become available, the B* technique will scale up considerably better than brute-force techniques.

KEYWORDS: Probabilistic B*, Computer Chess, Selective Search, Two-Person Games.

1 Introduction

Research in many AI domains has focussed on tasks that two and three year olds find easy. This has to do with primitive abilities that are difficult to capture algorithmically. The ability to recognize patterns, and actuate plans based upon such patterns is crucial to even low level behavior. Computer systems have trouble recognizing an "office scene" because they somehow don't "get" it is an office scene after having identified a few key objects. They have similar problems in speech understanding and in playing chess. The issue is always the same; if we only had more patterns and a plan to go with each, we would be in good shape.

When one examines verbal protocols of chess players analyzing chess positions, one is struck by the similarity of form. DeGroot [13] shows that the form of the analysis is very much alike, regardless of the strength of the player. What is different by player level is the content; the stronger players look at better moves. Just as there are no books to tell a human how to recognize an office scene, so there are no books to tell a human **how** to search. There are many chess books that tell you **what** to look for, but none that tell you how to process what you look at. Apparently such things are taken for granted.

Thus it appears that humans have a preferred form of chess analysis. This form is idea driven, and is oriented toward comparing the presumed best move with the alternatives. In this paper, we show how meat has been put on the bones of this concept. We show how ideas can be generated in ordinary positions and how they compete for attention until the best idea is found.

As we develop it, an idea is an optimistic concept that may turn out to be feasible if some other things work out. The **value** of an idea is represented by a probability distribution of the likelihoods of the values it could aspire to. This representation guides a selective search, and is in turn modified by it. This paper develops a mechanism that produces search trees that have many of the properties of human search trees as found in the protocols of DeGroot. An idea is generated and processed by a search mechanism in the presence of other ideas. An idea may be re-examined, refined, re-evaluated and compared to other ideas, and the search mechanism does it all. The importance of comparison comes up again and again in human intelligence. It is fundamental to our stopping rule which dictates termination when it is unlikely that a better alternative exists than the present one. Comparison also allows expanding the node in the tree that is most likely to produce evidence to help with the stopping decision.

Both in machines and humans there are questions about how ideas are generated. Ideas are certainly related to the semantics of the situation at hand. In our system, ideas are produced by shallow search probes. Humans almost certainly generate ideas from already learned structure (patterns) supported by a little search when needed. In practice, this may turn out to be a question of what the species-specific hardware best supports.

2 The History of Selective Adversary Searching

When one examines the search tree of a contemporary brute-force search, one is impressed with the sheer magnitude of the thing. There can be millions of nodes, more than 99% of which represent positions that no good chess player would ever dream of examining. How is one to get a search that makes more sense?

Those that use the alpha-beta search to guide their chess programs are well aware of the fact that so much of the search is devoted to irrelevant positions; however, the efficiency of alpha-beta is undeniable.

In an effort to get more out of additional computing time as it becomes available, several schemes have been proposed which build on the basic alpha-beta search. There is actually a spectrum of techniques that deal with how to decide what is important and what could be ignored. At the low-cost end of the spectrum are techniques that score moves as to their activeness and based upon such scores a move may be considered to be important to varying degrees. At the high-cost end of the spectrum, we have the B* approach. Here probe searches are done to establish likely bounds on the value of a subtree. Such schemes go collectively by the name of selective search [2]. They direct the power of the search into what is hoped to be important subtrees. Examples include:

- One popular scheme is to go to some finite depth (say 5 or 6), and then expand selectively to some further depth, only those moves that are deemed "worthy" of further attention. This scheme has some merits, but fails in situations where a leaf position is in transition (as when there has been a sacrifice that has not yet been recouped). Such positions will not get further attention. The standard notion of quiescence search of leaf nodes will be only continue the investigation of captures and escapes from check. Thus, if a sacrifice needs to be followed up by a non-capture, such moves will not be included in the quiescence search, nor will they be examined separately because the leaf node is deemed unworthy.

- Another scheme is forward pruning which discards moves which do not meet some static evaluation level. It can be applied throughout the tree or only after some basic depth has already been searched full width. The forward prune method has been shown to fail frequently in situations where good defense is required because it is difficult to specify the characteristics of useful defensive moves. Forward pruning was one of the earliest heuristic techniques. Clearly, there is always some risk in applying heuristics. However, the risk in forward pruning is far greater than in almost any other technique, since a static analysis of a move fails to capture much of what could be important.

- A null move search can be helpful at times and this is used in both tournament Hitech and B* Hitech. A null move search gives the side-on-move an extra move before doing its search. This, in effect, finds out what threats the side-on-move has. We use this in tournament Hitech to avoid doing deep searches, when a shallow null move search sees no significant threats. We use it in B* Hitech to find the Optimistic (or upper) bound.

- Other more ambitious selective alpha-beta schemes are described in [1, 7]. However, all these schemes are deficient in that they look for certain kinds of moves on which to continue the analysis, and the definitions used by the search paradigm are not broad enough to include all useful moves. For instance, it is essentially impossible to have a static analysis detect threats of mate in 3 moves. The singular extension scheme [1] which seemed so promising for such purposes, has received mixed reviews as to its efficacy.

- The most popular scheme at the moment is called "partial depths". In this scheme, moves that are "active" may not be counted as a full ply. For instance, a check may be counted at .5 ply and a capture of the queen may be counted as .7 ply. Yet the search is carried on until this partial-ply sum equals or exceeds the specified depth of the search. This tends to elongate those branches of the tree that have interesting moves in them. This technique is used by the most successful micro-processor based programs. It has now been under intensive development for 7 or more years, and the success of these micros attests to the efficacy of the procedure. Unfortunately, little has been published on this.

2.1 The Necessary Conditions for a Truly Selective Search

There are two basic precepts that a truly selective search must obey.

1. It must be able to **stop** when a clearly best alternative exists at the root. This is done by comparison and is independent of the ultimate value of the best move.

2. It must be able to **focus** the search on the place where the greatest information can be gained toward terminating the search.

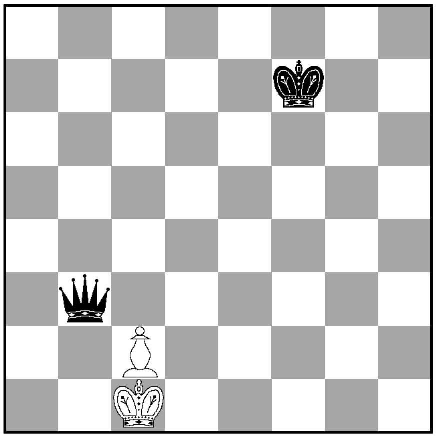

Figure 1: White to Play

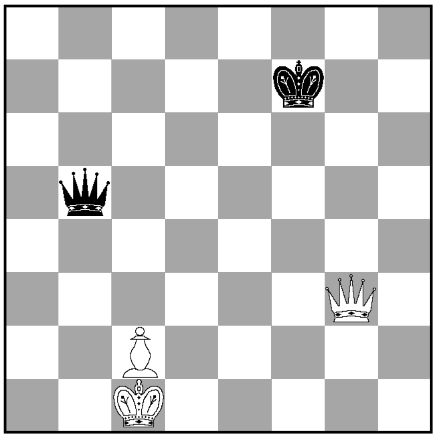

Figure 2: White to Play

We deal with the stopping issue first. Most brute-force searches have built into them the ability to move immediately when there is only one legal move. However, when there are two legal moves, and one leads to an immediate catastrophe, they are unable to discern this and continue searching as if there existed more than one realistic alternative[1]. Many past selective or best-first searches have also been unable to halt even in the face of overwhelming evidence that there is only one good alternative.

To illustrate the issue, consider Figure 1. An algorithm must convince itself that capturing the queen with the pawn is the only sensible move (all other moves lose). This is a search of 1 node in our current program. It does not matter whether the resulting position is a win for white or a draw. The only thing that

[1] In alpha-beta Hitech, we implemented a scheme in which when all alternatives except one lead to being mated, that alternative was played without further search. However, when the catastrophic alternatives are very poor but don't lead to mate, this is not powerful enough to terminate the search, because the brute-force search continues to hope for miracles.

matters is that any other move except the capture, c:b3,[2] **loses**. This is an essential idea, much like the famous alpha-beta idea [24] which showed that it was only necessary to refute an inferior move once.

The reason that the B* algorithm handles such situations properly is that it uses upper and lower bounds to bound the likely near-term value of a node. It computes an Optimistic view and a Realistic view of the move. The search is terminated when the realistic view of one alternative is at least as good as the optimistic view of the others. In Figure 1, it takes one node expansion to find that the move c:b3 has a realistic value of +1 pawn, while the optimistic values of all the other moves would be at best +1 pawn (assuming the queen did not move away). Since the realistic values of these other alternatives are poor (the queen does move away), it is easy to determine algorithmically that c:b3 is best, and do it very quickly.

For issue 2 above, the situation is much more complicated. In Figure 2 white must calculate whether the check Qb3, which forces the exchange of queens leads to a win (it does). If it does not, then white would be foolish to give away his existing chances for a win by such a move. However, once the first move is played, the follow-on, Q:b3; c:b3 is automatic. We know of no other actual or proposed search paradigm except B* that understands the dynamics of such situations.

2.2 The Difficulties with Past Approaches

Before the B* search formulation, several other selective searches were proposed. Kozdrowicki [20] developed a method for expanding only the best leaf node in the tree and its children. However, his algorithm has considerable difficulties in stopping and frequently continued examining the best alternative when it should have stopped. In Harris' method [16] the value of each node was bounded by uncertainty, and the search focussed on reducing that uncertainty. However, the uncertainty bounding a node began as a constant, and thus could not distinguish a "promising" move from others. Because the search got such a bad start, it spent much of its time on meaningless effort. His program also competed in computer chess tournaments, but ultimately succumbed to the rising alpha-beta programs. Both these formulations failed to consider the importance of competition between the likely-best alternative and the second-best as the driving force in deciding when to stop and where to put the effort directed at being able to stop.

In 1972, the first author discovered the idea that a node need not have a singular value, but could have value bounds. Up to that time, bounds had been used to precisely delimit the range of meaningful values that further searches could return; e. g. Alpha-Beta or Branch and Bound. These bounds reflected what had been learned in the previously executed part of the search. However, bounds had never been used to delimit the value of a single node independent of search[3]. By having a meaningful but **heuristic** upper and lower bound on the value of a node, it becomes possible to decide that the best node is better than its closest competitor.

---

[2]In this paper we are using the "algebraic" chess notation for describing moves. In this notation, the columns of the board are numbered from left to right from "a" to "h", and the rows are number from bottom to top from "1" to "8". A move is described by indicating the type of piece moving and its destination. When a pawn moves, the piece designation is skipped. For instance, an opening move that takes a pawn from e2 to e4 is just noted as "e4". An opening move that takes a knight from g1 to f3 is noted as Nf3. Captures are designated by ":". In the present case, c:b3 means: the pawn on the c file captures the piece on b3.

[3]The fact that in the present implementation we use searches to produce bounds has nothing to do with what has happened in other parts of the tree.

This idea resulted in the B* search formulation [5], and the ability to implement a meaningful stopping rule. We call this stopping rule **separation**. In practice, separation seldom occurs, so we define separation as **Prb (separation) > x**, for some real valued x. The original B* idea was to try to increase the lower bound of the best node (the PROVEBEST strategy) or reduce the upper bounds of competing alternatives (the DISPROVEREST strategy) until separation was obtained.

There were a number of problems with the original B* formulation. The decision rules about where to continue the search were not optimal, and this was corrected by Palay in [25]. We and others attempted to use this algorithm on a variety of games, but without great success. One problem was that the bounds were estimated using a potentially errorful static evaluation function. For instance, for chess it is important to have the notion of **threat** incorporated into the optimistic view. However, it is difficult to devise an algorithm to statically detect certain threats (such as the threat of mate in 2 moves) which are much more easily found by search. Estimation errors produce wasted analysis effort, and the possibility that important moves are never examined.

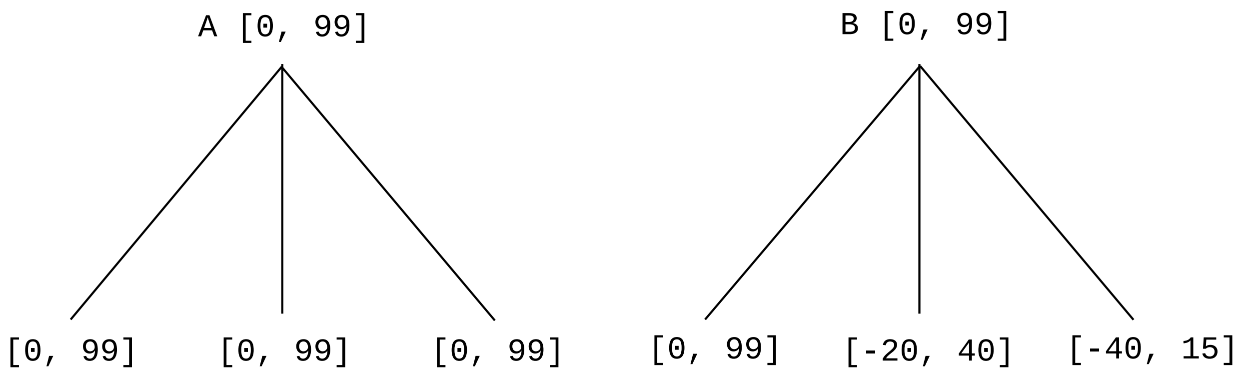

Figure 3: Bounds are not Sufficient to Portray Goodness

There was also another issue discussed by Palay. Backing up bounds to measure the potential of a subtree fails to take into account that the distribution of likely values within the backed-up range need not be uniform. This is because the bounds of the parent are the maximum bounds of the children. If many nodes in the subtree have values near one bound, this makes it more likely that the realistic value of that subtree is nearer that bound. Figure 3 shows the problem. Here both nodes A and B have the same backed up bounds. However, node A is clearly to be preferred since the maximizing player can choose among three good alternatives. Then in case some of them fail to live up to expectations, the others may succeed. The subtree of node B does not allow this; everything is staked on the success of the left most node[4]. The examination of this problem showed that it was essential to use distributions to measure the potential in a subtree. The distributions backed up for A and B will be quite different, while the ranges are the same.

[4]This notion has been variously tapped in the history of searching. Slagle [30] postulated the M & N search, and the idea was also central to singular extensions [1].

2.3 The Springboard for the Present Work

These new insights resulted in a thesis by Palay [26] which decidedly improved the performance of the B* style of searching. Unfortunately, this important contribution still remains largely unrecognized at this time.

Palay did several things:

- Use shallow searches to produce quite accurate bounding values.
- Use an extra move for the Player at the start of a search to get optimistic values.
- Introduce the notion of a Realistic value as part of the structure of a node, and partition the range of a node into two segments:
  - The range between the Optimistic value and the Realistic value is the domain of the **Player** (the one whose turn it is to move at the root), and he tries to move the Realistic value upward toward the Optimistic value.
  - The range between the Pessimistic value and the Realistic value is the domain of the **Opponent**, and he tries to move the Realistic value downward toward the Pessimistic value.
- Use a probability distribution instead of a range as the value of a node, and produce a calculus for backing up distributions. A probability distribution carries more information than a range. It expresses where the action is within a given range. This is invaluable for making good decisions.

Palay did experiments with a fundamental text of chess tactics [28] that was being used by all the programs of the day to test their prowess. He did his experiments on a VAX-780. When time was scaled according to the difference in power between this and the chess machine Belle [12], it was a close competitor to Belle (the best chess computer at the time) on the problem set. It solved simple problems considerably faster, but failed to solve some deep problems. At the time, it was not clear what the latter problem was due to. We also had the program do some problems that were more positional in nature, and it did well on these. B* with probabilities seemed destined for a bright future.

We built the chess machine Hitech [14] to serve as the B* searcher. However, Hitech was so good when it first came up doing straight alpha-beta search, that this lofty motivation faded quickly, and Palay took another job outside the Computer Science Department at CMU. In retrospect, it was foolish not to pursue both approaches in parallel.

Palay's work left things in the following state:

- Probability-based representations are better than a simple range.
- Values computed by shallow searches are considerably better than statically computed judgements.
  - Realistic values can be obtained by doing shallow searches on the node in question.
  - In order for optimism to work, it is necessary to have a rank ordering of moves in order of likelihood of success and impact if successful. Optimistic values were obtained by allowing the side-on-move an extra move to start. This implemented the time-tested notion of *threat*, as that which would happen if your opponent did not respond. However:
    - It is impossible to estimate the value of a check by giving the checking side an extra move, as he would merely capture the king.
    - A mate threat is also difficult to deal with. Clearly, mate threats are very potent,

but where do they fit in with various captures, checks, pawn promotions, etc.

- Palay's approach involved a **SELECT** phase during which the Player exercised his optimism to find the best move at the root. Then a **VERIFY** phase allowed the Opponent to exercise **his** optimism on the selected move. If the selected move remained best after the VERIFY, it became the move made. Otherwise, the SELECT phase was begun anew. This mechanism worked well; however, there were still many issues to be resolved:
    - How is separation to be achieved in the usual case where there are several closely competitive alternatives?
    - What is the best way to efficiently represent and back up probabilistic data?
    - How to use the probability distributions to decide which move to process next considering such things as:
        - How does one deal effectively with distributions that overlap?
        - How can one prevent one preferred alternative from getting all the attention?
    - How to make a complete program of all this; one that not only solves tactical[5] problems but plays a whole game, including managing its time resources, thinking ahead on its opponent's time, saving pertinent search trees from one move to the next, and other things.

In 1991, as Hitech[6] was being surpassed by other chess programs, and the various people who had worked on it had graduated or gone elsewhere, the senior author determined to make one last attempt at making the B* concept work. There had been some papers [29, 23] on new approaches to this problem, but none had been linked to a successful program. It was desired to see where the weaknesses in the B* approach were, and how they could be remedied.

3 The New B* Search

3.1 Search Depth and the Evaluation Space

Let us look at certain problems of searching. One can think of a search space as a surface with geological features on it. The X and Y coordinates define the location on the surface, and the Z coordinate is the value of the evaluation function at point X_i, Y_i (the height of the geological feature). In an ideal domain it should be possible to have a marble roll from any given point to the low point in the terrain. This would be the case if our evaluation function defined a smooth space. But in complex domains there are bound to be ridges. Ridges introduce changes in gradient that make life difficult for the marble that is trying to find the lowest point.

For a searcher to successfully navigate such a surface, it must be able to see over sufficient ridges to have some idea of where progress is to be made. Brute-force programs adopt the strategy of searching to the largest possible radius from the origin. This is good because with each additional ply of lookahead, the probability that one is looking over the last ridge is increased. It is presumed that after crossing the last ridge a marble would smoothly roll to the lowest point in the space. The probability that an additional

[5] A tactical problem is one in which the correctness of a move is decided solely based upon the material balance at the end of the principal variation.

[6] In this paper, we shall be referring to the established Hitech program variously as Hitech 5.6, alpha-beta Hitech, brute-force Hitech and tournament Hitech. These are all attributes of this program, that may be germane to the existing context.

ply will find the last ridge is not very high. Thus, this approach is based on a coarse understanding of the properties that cause a surface to have ridges.

It is possible to have a "strategic" outlook that would pay attention not only to the Z coordinate of the surface but also to some compass direction. Such a strategy would be able to implement an idea such as "head North-East for at least 100 miles regardless of the terrain you encounter". That is a long-term strategy; one that could outdo a searcher paying attention only to the height of the places on the surface at a given radius from the origin. Of course, correct strategies may not be easy to come by.

So the real question for a searching program that goes to depth D is: how much "strategic" knowledge is required for it to navigate well? Since one can never know what is over the next ridge until one has looked, this appears to be a hopeless problem for which at best heuristic approximations are possible. It is possible to have two apparently identical states; one leading to some great gain and the other to nothing. Yet both have been achieved by following some strategy that is as yet not successful.

So a selective searcher by taking small steps in what appear to be good directions hopes to be able to peer over more meaningful ridges than the brute-force searcher. This can be successful as demonstrated by the success of the various selective extension strategies used in connection with the alpha-beta search. The two extremes are:

1. A smooth space where searching further is clearly better.
2. A highly ridged space, where having any clue as to where the ridges are to be found will be a win.

It would seem that flat, featureless spaces can be avoided by having enough domain-specific knowledge so that there are almost always some interesting directions to be explored. It should be noted that the very notion of strategy implies the achieving of some goal. It is very risky to do this in the alpha-beta paradigm since the predefined halting of the search may find the strategy at various stages of development. Thus, in the alpha-beta paradigm it is very difficult to assess the success of a partially implemented strategy, since it is not certain whether it can be fully implemented.

The correct way to implement a strategic outlook is with bounding values. This is exactly what B* provides. Then even shallow searches can follow a strategic idea until it either proves successful, shows itself to be unsuccessful, or must be abandoned because of resource limitations. A brute-force alpha-beta search has no notion of potential, and must abide by whatever result it finds at its search horizon. This is why the B* approach should be preferred. We have as yet only implemented a few strategies which were discussed in Section 3.10. This whole problem is discussed at length in [9].

3.2 Some Considerations

The initial goal of the research was to see if it was possible to integrate McAllester's Conspiracy Theory [23] into B*. The idea behind conspiracy theory is that there are a certain number of leaf nodes that *conspire* to keep the value of some descendant of the root from changing by a certain amount in a certain direction. If it were desirable to make such a change, then the easiest change to effect would be the one where the *fewest* nodes conspire to keep this from happening. The senior author quickly found out that this would not work, and found out something very important about why the conspiracy approach has not worked.

While it is useful to think of conspiracies in the above way, and to plan to attack the conspiracy with the fewest conspirators, in practice this does not work. Conspiracies fall into buckets. There are buckets of conspiracies of magnitude 1, of magnitude 2, magnitude 3, etc. In chess, (and we would assume in almost all domains) there lots of potential conspiracies, and the number of conspiracies of magnitude 2 is usually already quite large. Thus, when dealing with conspiracies of magnitude 2, one must examine them in some quasi-random order, and the chances of finding *the* conspiracy that is easiest to break is quite small. It is like a breadth-first search of conspiracies, with no other clue as to what might make a given conspiracy easy to break. *This is the reason for the failure of the conspiracy approach in game playing*. There is no good method for deciding the weakest conspiracy of a given magnitude[7].

What was needed was a more real-valued approach; Palay's probability-based B*. Here moves can be ranked in order of likelihood to succeed; rather than being lumped into a few conspiracy buckets. This leads to much better discrimination. We therefore concentrated on improving probability based B* and refining its details.

3.3 Representation of a Node

The principal elements of a B* node are:

- The **RealVal**, which is the best estimate of the true value of the node[8]
- The **OptVal** which is the optimistic value of the node for the side-on-move
- The **PessVal** which is the optimistic value for the side-not-on-move, backed up from its subtree[9], and
- The **OptPrb** which is the probability that a certain target value can be achieved by future searches of the subtree rooted at this node.

The side-to-move at the root is the Player, and the other side is the Opponent. We also use the SELECT and VERIFY phases as described by Palay..

3.4 Backing up of Values

In this algorithm there is always one side that is trying to do the forcing. During the SELECT phase, it is the Player; and during the VERIFY phase it is the Opponent. The forcing player is called the **Forcer**. When backing up to a node where Forcer has a choice, the best alternative is always backed up. For the other player, the **Obstructor** it is an ANDing of alternatives that is backed up, since they must **all** be refuted.

RealVal is backed up normally from child to parent. OptVals are only computed for the Forcer leaf nodes. OptVals for one side are the PessVals for the other and are thus backed up. The most complicated backing up occurs with OptPrbs. When the backed up value is an AND, the OptPrbs of the

[7]In passing it is worth noting that this is very similar to the problem that keeps the A* search from being efficient. What happens is that long lists of similar valued nodes are accumulated waiting to be expanded, and with the original A* formulation it is impossible to discriminate which has the greatest promise.

[8]In a game-theoretic sense a node can only have three values: Win, Lose, or Draw. However, Win/Lose/Draw values are unlikely to be found by the search. Therefore, it is necessary to have values that permit ordering leaves by goodness. Palay in [26] discusses the Oracular value of a node. This is the relatively stable value that a node would have on the existing valuation scale based upon a deep search of (say) 10 ply.

[9]It should be noted here that all the 'Vals are the result of alpha-beta searches, so they represent what can be achieved against best resistance. Thus, it is important to recognize that the OptVal is only that which can be achieved against best resistance.

child nodes are multiplied; otherwise, the best is backed up.

3.5 Overview of the Probability Based B* Search

Before launching into a full description of the algorithm, we present a stylized version in which we do not attempt to define constructs but merely give a feel for how it works. This is followed by an actual search example. We urge the reader to read the algorithm lightly together with the example.

```
integer TargetVal;
SELECT: while (RealVal(BestMoveAtRoot) < OptVal(AnyOtherMoveAtRoot))
        {
         TargetVal = (OptVal(2ndBest)+RealVal(Best))/2;
         Select Root Node with greatest OptPrb;
         Trace down the child subtree selecting
                For Player-to-Move nodes, child with largest OptPrb
                For Opponent-to-Move nodes, child with best RealVal
         Until arriving at a leaf node;
         Get RealVal for each Child Node of this leaf;
         If it is a Player-to-Move node get OptVals for each Child;
         Back up Values;
         if (EffortLimitsExceeded) Break;
        }
TargetVal = RealVal(2ndBstMoveAtRoot) - 1;
VERIFY: while (RealVal(BestMoveAtRoot) >= TargetVal)
        {
         Select Reply Node with greatest OptPrb;
         Trace down the child subtree selecting
                For Opponent-to-Move nodes, child with largest OptPrb
                For Player-to-Move nodes, child with best RealVal
         Until arriving at a leaf node;
         Get RealVal for each Child Node of this leaf;
         If it is an Opponent-to-Move node get OptVals for each Child;
         Back up Values;
         if (EffortLimitsExceeded) GoTo MakeMove;  !! It passed inspection.
        }
GoTo SELECT;                              !! Selected move was refuted.
MakeMove:
```

3.6 A Search Example

In this example we show how a simple search could progress through the search phases, showing how node selection is done and how the backing up works. The example is shown in both table and graphical form. The table headings should be self explanatory except for Prt which is the parent of a node. Figure 6 has the legend for trees. Solid lines indicate the path to the node to be expanded. Within each node, the top number (either RealVal or OptPrb) is used for selecting the node to expand.

The initial expansion of the root produces 3 nodes that are in competition for best. As usual, the maximizing player is to play at the root, and he is named PLAYER. He is pitted against the minimizing player who is named OPPONENT. The OptVals and RealVals are produced by probe searches, and since the PessVal is backed up from descendant nodes, it is undefined at this time.

TargetVal, which is defined in Section 3.9.3 as (OptVal(2ndBest) + RealVal(Best))/2 evaluates to 30. OptPrbs (probability of moving RealVal up to TargetVal) are computed by linear approximation in the range from OptVal to RealVal. Figure 4 shows this initialization.

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	100	20	Undef.	30	.875
B	0	40	18	Undef.	30	.455
C	0	25	10	Undef.	30	.000

Figure 4: Initialization

Since we are in the SELECT phase, the node with the highest OptPrb is selected for expansion. Intuitively, this is the node with the best chance of moving RealVal up to TargetVal. Figure 5 and Figure 6 show the tree after expanding node A.

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	100	20	Undef.	30	.875
B	0	40	18	Undef.	30	.455
C	0	25	10	Undef.	30	.000
D	1/A	Undef.	20	100	30	.875
E	1/A	Undef.	60	100	30	1.00
F	1/A	Undef.	80	100	30	1.00

Figure 5: SELECT After Expanding A

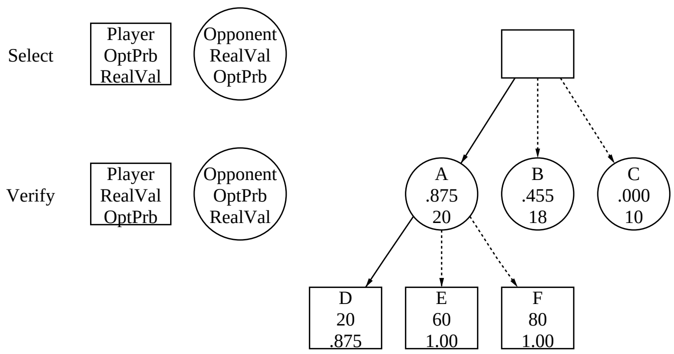

Figure 6: LEGEND; SELECT After Expanding A

The result of expanding node A has produced three successors, D, E, & F. The PessVals and OptVals of each of these are inherited from the parent; the RealVals are computed by doing probe searches. Note that the alternatives are listed in order of goodness for the Opponent, who is trying to get the most negative value. Thus, D is the most promising node. The OptPrbs are the probability that the Player can achieve TargetVal in this subtree. These OptPrbs represent an AND situation since any can be selected by the Opponent. Therefore, they are multiplied and the value is backed up to node A. If nodes E or F had RealVals that were better than the present TargetVal, these values would lower the OptPrb and thus indicate that this node is less promising. Figure 6 shows the expansion of node D.

The SELECT phase now continues by tracing down the best OptPrb at even depths (node A), followed by the best RealVal at odd depths (node D). So node D is expanded. Backing these values up to node A, produces the following table in which the OptVal and RealVal of node A have changed, and the RealVal of node D as well as the PessVals of nodes D, E, & F. Since RealVal(Best) has changed, TargetVal is recomputed, and with it the OptPrbs.

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	76	34	Undef.	37	.929
B	0	40	18	Undef.	37	.136
C	0	25	10	Undef.	37	.000
D	1/A	Undef.	34	76	37	.929
E	1/A	Undef.	60	100	37	1.00
F	1/A	Undef.	80	100	37	1.00
G	2/D	76	34	Undef.	37	.929
H	2/D	36	12	Undef.	37	.000
I	2/D	32	16	Undef.	37	.000

Figure 7: SELECT After Expanding D

At this point node A is the overwhelming favorite to be the best choice at the root. However, there is still some overlap between RealVal(A) and OptVal(B). It is conceivable that the search could spend quite a bit of time trying to get complete separation. For this reason, we have a parameter called MinAct, which is originally set to 0.15. No node whose OptPrb < MinAct is deemed to still be in contention. As the search continues, and the time available to complete the search gets shorter, MinAct is gradually increased, thereby reducing the likelihood that marginal possibilities will be examined. As the search develops, the TargetVal may take on many values and this will change the OptPrbs usually in the direction of lowering them as RealVal(Best) increases. We have found that MinAct is an excellent labor saving device.

Since no other node at the root except A has an OptPrb > MinAct, the initial SELECT phase is now over. The search now moves into the VERIFY phase where the OPPONENT attempts to use his optimism to find moves that may upset the analysis as it now stands. The node selection algorithm now reverses the selection procedure in that at even depths the node with the best RealVal is selected, and at odd depths the node with the best OptPrb. This is because now the OPPONENT is exercising his optimism.

The first step here is to compute OptVals for all the nodes at which it was OPPONENT's turn to play. In this example, these are nodes D, E, & F. Nodes G, H, & I inherit PessVal from D. After computing these, the table looks like this. Remember that for OPPONENT, negative values look best. Thus, nodes D, E, & F have been juxtaposed to reflect their goodness. Note further, that the objective in the VERIFY phase is to show that RealVal(A) is not as good as RealVal(2ndBest). The second best RealVal is that of B, so reducing RealVal(A) to 17 would be sufficient to show it is not best. Therefore, it is most effective to set TargetVal to RealVal(2ndBest)-1. Then the OptPrbs can be computed. Now OptPrb is the probability that the OPPONENT can lower RealVal to TargetVal. Figure 7 shows the tree's values for VERIFY. In Figure 9, the tree at the end of the SELECT can be seen on the left, and the same tree at the start of the VERIFY on the right.

The descendant of node A that has the highest OptPrb is F. Expanding F produces Figure 9. Again, note that the OptPrb being backed up to F is the product of the descendant nodes. Also note that the PessVals of even depth (PLAYER) nodes are the OptVals for the OPPONENT.

Applying the algorithm for selecting the next node to expand, we get A (best RealVal), D (best OptPrb), G (best RealVal). Expanding G, computing the OptPrbs, and backing up values, we get the table below. (remember that during the VERIFY phase, we backup the best OptPrb from odd depths, and take the product at even depths). The tree at this point can be seen in Figure 11 and Figure 11. For the next

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	76	34	-80	17	Undef.
B	0	40	18	Undef.	17	Undef.
C	0	25	10	Undef.	17	Undef.
F	1/A	-80	80	100	17	.606
D	1/A	0	34	76	17	.500
E	1/A	40	60	100	17	.000
G	2/D	76	34	0	17	.500
H	2/D	36	12	0	17	1.00
I	2/D	32	16	0	17	1.00

Figure 8: VERIFY Initialization

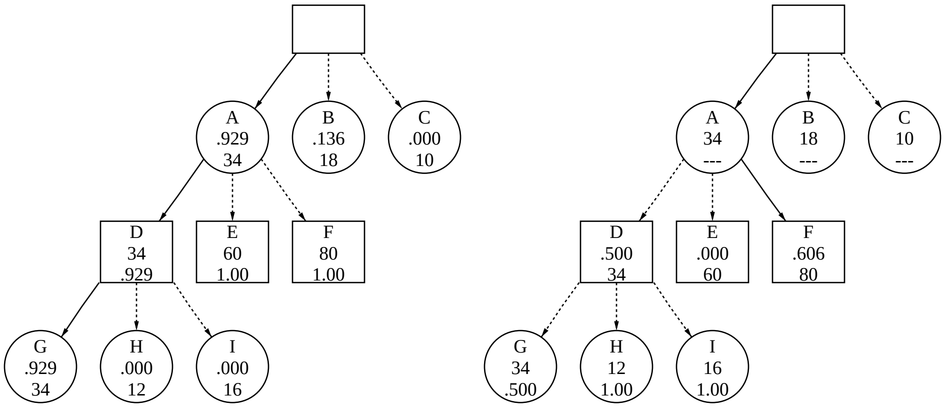

Figure 9: SELECT to VERIFY Transition

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	76	34	-80	17	Undef.
B	0	40	18	Undef.	17	Undef.
C	0	25	10	Undef.	17	Undef.
D	1/A	0	34	76	17	.500
F	1/A	-80	80	100	17	.313
E	1/A	40	60	100	17	.000
G	2/D	76	34	0	17	.500
H	2/D	36	12	0	17	1.00
I	2/D	32	16	0	17	1.00
J	2/F	100	80	-80	17	.606
K	2/F	100	60	-80	17	.693
L	2/F	100	50	-80	17	.746

Figure 10: VERIFY After Expanding F

node expansion we get: A,D,G,M. Expanding M and backing up the values, we get Figure 12. Now the best path is A, F, J. Expanding J and backing up values yields Table(TableJ).

Since all of the alternative refutations of node A {D, E, F} now have OptPrbs < MinAct, the VERIFY phase comes to a close with the conclusion that node A almost certainly is the best node. The final tree can be seen in Figure 15. One should note that RealVal(T) turned out to be outside the original bounds. This occasionally happens, since the bounds are, after all, only heuristic estimates. This example is a relatively short one, but still quite complex and attempts to display all the features that go into controlling the search.

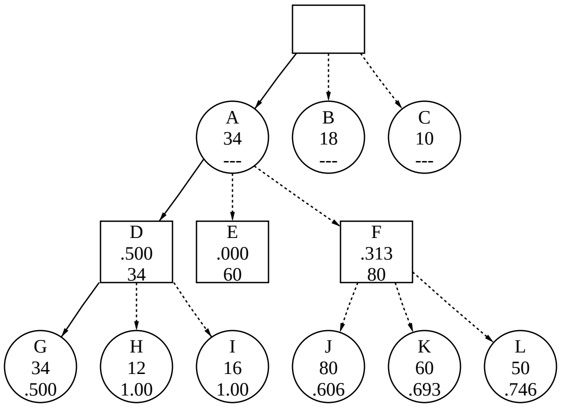

Figure 11: First Verify Expansion

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	76	34	-80	17	Undef.
B	0	40	18	Undef.	17	Undef.
C	0	25	10	Undef.	17	Undef.
D	1/A	10	34	76	17	.700
F	1/A	-80	80	100	17	.313
E	1/A	40	60	100	17	.000
G	2/D	76	34	10	17	.700
H	2/D	36	12	0	17	1.00
I	2/D	32	16	0	17	1.00
J	2/F	100	80	-80	17	.606
K	2/F	100	60	-80	17	.693
L	2/F	100	50	-80	17	.746
M	3/G	10	20	76	17	.700
N	3/G	15	30	76	17	.133
P	3/G	20	34	76	17	.000

Figure 12: VERIFY After Expanding G

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	76	34	-80	17	Undef.
B	0	40	18	Undef.	17	Undef.
C	0	25	10	Undef.	17	Undef.
D	1/A	10	34	76	17	.133
F	1/A	-80	80	100	17	.313
E	1/A	40	60	100	17	.000
G	2/D	76	34	10	17	.133
H	2/D	36	12	0	17	1.00
I	2/D	32	16	0	17	1.00
J	2/F	100	80	-80	17	.606
K	2/F	100	60	-80	17	.693
L	2/F	100	50	-80	17	.746
M	3/G	10	55	76	17	.034
N	3/G	15	30	76	17	.133
P	3/G	20	34	76	17	.000
Q	4/M	76	55	10	17	.156
R	4/M	76	32	10	17	.318
S	4/M	76	20	10	17	.700

Figure 13: VERIFY After Expanding M

Node	Depth/Prt	OptVal	RealVal	PessVal	TargetVal	OptPrb
A	0	76	34	-80	17	Undef.
B	0	40	18	Undef.	17	Undef.
C	0	25	10	Undef.	17	Undef.
D	1/A	10	34	76	17	.133
F	1/A	4	80	100	17	.086
E	1/A	40	60	100	17	.000
G	2/D	76	34	10	17	.133
H	2/D	36	12	0	17	1.00
I	2/D	32	16	0	17	1.00
J	2/F	100	40	4	17	.167
K	2/F	100	60	-80	17	.693
L	2/F	100	50	-80	17	.746
M	3/G	10	55	76	17	.034
N	3/G	15	30	76	17	.133
P	3/G	20	34	76	17	.000
Q	4/M	76	55	10	17	.156
R	4/M	76	32	10	17	.318
S	4/M	76	20	10	17	.700
T	3/J	4	82	82	17	.167
U	3/J	11	70	80	17	.102
V	3/J	16	40	80	17	.042

Figure 14: VERIFY After Expanding J

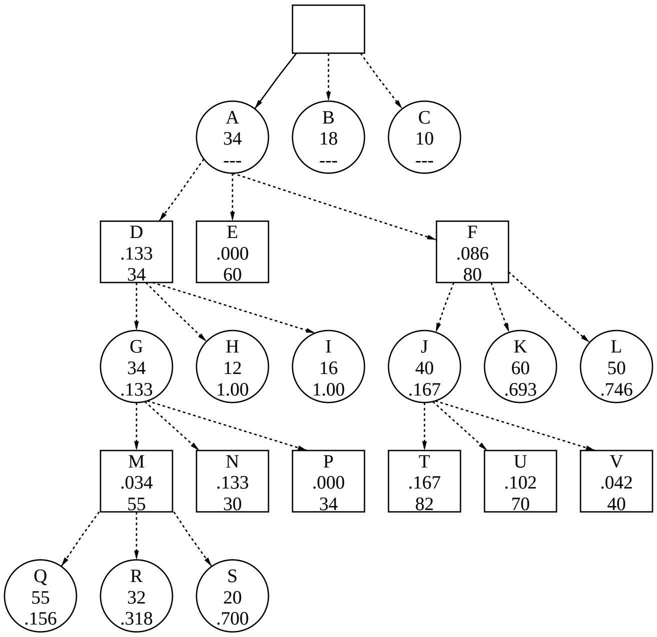

Figure 15: Final Tree

3.6.1 OptPrb and Probability Distributions

Probability distributions collect the information needed to make good decisions about what to explore and when to stop. When a node is expanded, its RealVal and OptVal are computed. The conditions for computing OptVal are quite complicated and are explained in Section 3.8. It is assumed that the actual value of a leaf is somewhere between these two values, with it most likely being nearer the RealVal.

Figure 16 shows a leaf node distribution. The RealVal=20, and the OptVal=250. We assume a

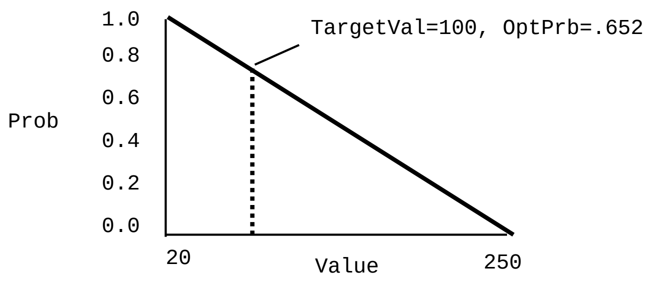

Figure 16: Probability Distribution and Meaning

uniform distribution between RealVal and OptVal, and this appears to work well in practice. The OptPrb at 250 is 0, while the OptPrb at 20 is 1.0 since that value has already been achieved. Intermediate values can be read off along the X and Y axes.

In our method of computing OptPrbs we differ somewhat from Palay's approach. In Palay's method, after a node was expanded, the distribution of likely values to be achieved in every subtree was computed. This was quite time consuming, but allowed the selection of the path through the tree that was likely to yield the greatest improvement in the value at the root, according to the leaf node estimates. Thus, the backing up of values consisted of backing up the best distribution *in toto* whenever it was a OR node, and doing a point-by-point multiplication of distributions at AND nodes. It was this latter operation that was expensive computationally. This method of backing up distributions is explained in [26, 27].

Instead, we use a method of projecting a **TargetVal**, which is a value that seems to be a reasonable target to which to try to move the best RealVal. The computation of TargetVal is explained in Section 3.9.3. In Figure 16 the TargetVal=100, and OptPrb(TargetVal) = .652.

After a node has been expanded, we back up all new values including OptPrbs, however far back they percolate in the tree. Then we check whether the parameters that determine TargetVal have changed. If so, then a new computation of OptPrbs throughout the tree is done. This computation is very inexpensive, requiring only a single multiply per node in the AND case. The OptPrbs in any subtree now represents the likelihood of achieving TargetVal in that subtree.

We have found that the performance of the B* algorithm is hardly affected by the exact value of TargetVal; it only needs to be an approximate distance from the best RealVal and reflect the potential of the best alternative OptVal.

3.7 Independence of Alternatives in the Search

When a player has many alternatives it is more likely that he has a good one than if he only has few. This is reflected in the way distributions are backed up. When the Player chooses, we assume that he will choose his best alternative. However, when the Opponent chooses he may choose from many, and the quality and quantity of his alternatives are reflected by the product method of backing up distributions[10].

---

[10] It should be noted that this captures the idea behind **conspiracy** without being subject to the categorization effect that comes from putting things into conspiracy buckets.

It is well known that all moves and branches of a search tree are not independent. Thus, a certain move may appear non-forcing because it allows (say) five replies. However, if each of these replies is refuted in the same way, then one may question whether this represents as much freedom as when each of the five replies is met in different ways. Actually, the problem of independence of alternatives is complicated by the fact that no one knows how to define "in the same way", despite the fact that it is fairly clear to humans what is being talked about. In some earlier work [4], the first author backed up descriptions of failures, and used these to try to discover those moves that could not be refuted in the same way. It turned out that the descriptions were not complete enough to be effective all of the time.

We have addressed the independence problem in the modern style of approximating behavior by starting with the simplest concepts. Thus, as a first approximation, it is good to categorize moves into **Reply Groups**. All moves in a reply group have the same best response. These responses are discovered during the RealVal searches. The value backed up for a reply group is the best value found in searches of all members of the reply group. This avoids the risk of having the OptPrb of a parent node reflect that there are many responses that may or may not be independent. This method has produced better behavior, but we do not want to justify it by other than empirical means.

3.8 The Search Mechanism

The participants in the adversary search are the Player who is the side to move at the root, and the Opponent, who is the other side. In the original formulation of B*, the search was guided by selecting either the ProveBest or DisproveRest strategy [5] in an attempt to gain separation. In Palay's formulation these strategies were replaced by two phases. In the SELECT phase, the search proceeds until a candidate best descendant of the root is determined. Here, the Player's optimism is extremely important in trying to find the best move. In the VERIFY phase this move is subjected to intense scrutiny to determine if it can hold its value. Here the opponent has the chance to exercise his optimism. For instance, in the SELECT phase a certain piece may be attacked by the Player, and prudence dictates that the Opponent should move or defend it. However, in the VERIFY phase it may be found that a counter-attack best solves this problem. If both sides exercise their optimism simultaneously, total chaos would result. If the Player were to put a rook where it could be taken it behooves the Opponent, as a first attempt, to take it. If he were allowed to exercise optimism at this point, he might prefer to attack the queen, which if successful could be better. However, it would widen the scope of the investigation unnecessarily to the point where convergence would be problematical. This problem was first discussed in [4].

3.9 Node Expansion

Expanding a node consists of doing probe searches to get bounding values for its children. The algorithm **always** gets the RealVal bound for any node being expanded by doing a D-ply search of the move in question. Here **D** is referred to as the probe depth, and is adjustable depending upon the power of the machine and the amount of material on the board. If it is the SELECT phase, then the children of all Player-to-move nodes also get an OptVal. OptVals are obtained by making the move leading to the child node, and reversing the side-to-move at the start of the D-ply search. This gives the Player an extra move, and gives him a chance to show his threats. The reasons for this method are discussed in Section 3.10. Since the SELECT phase only deals with Player optimism, it is not necessary to get OptVals for the children of Opponent-to-move nodes. On the other hand, in the VERIFY phase, OptVals are obtained only for the children of Opponent-to-move nodes, because here we are dealing with Opponent optimism.[11]

3.9.1 The SELECT Phase

In this phase, we only deal with the Player's optimism. OptPrbs define the Player's potential at each node. All phases are governed by effort limits, which are discussed in Section 3.11.

The search begins with a limit on the number of nodes that are to be expanded in trying to find the best move for the Player. When this limit is reached, we begin the VERIFY phase. If during the SELECT phase, the best move is considered "clearly best", the SELECT phase is terminated early. By clearly best, we mean that competitive OptPrbs are valued in such a way that it is unlikely that any other move can achieve as good a RealVal. If the search later returns to the SELECT phase because the selected move was refuted during the VERIFY phase, then a new effort limit is set, and things proceed as described above.

3.9.2 The VERIFY Phase

Despite Palay's assertion that the VERIFY phase is the reverse of the SELECT phase, this is not correct. The purpose of the VERIFY phase is not to find the best reply for the Opponent, but merely to find one that is good enough to reduce the RealVal of the selected move to the point where it is no longer best. These two goals are not the same. Finding the best move in the SELECT phase is an open-ended activity terminated only by exceeding the effort limit or finding a clearly best move. Finding any move that is an **adequate** refutation of the selected move is a much more circumscribed task.

In the VERIFY phase, the Opponent is allowed to exercise his optimism, which has been suppressed during the SELECT phase. For instance, assume the successful Player move in the SELECT phase was to sacrifice a rook which the Opponent captured. In the VERIFY phase the Opponent will be allowed to try *his* optimism. He could threaten mate, and this may refute the rook sacrifice. However, it would have been folly to allow this threat during the SELECT phase. Only after a concrete selection has been made does it make sense to allow the opponent to try to refute by non-standard means.

The length of the VERIFY phase is also governed by an effort limit. If the selected move is refuted before that limit is reached, control returns to the SELECT phase to begin anew finding the best Player move. If the effort limit is reached, the selected move has stood up to testing, and is played. It is also possible for the VERIFY phase to give up trying to refute the selected move, if none of the moves remaining to be investigated appear to have the potential for a refutation.

3.9.3 Selecting the Next Node to Expand

At the root, the node with the largest OptVal is not always the best one to look at. An OptVal of 500 may represent a subtree where Prb(500) = .01, and Prb(100) = .02. An OptVal of 250 represent a subtree where Prb(100) = .8, even though Prb(300) = 0.0. So a more likely lower goal may be preferred to a less likely higher goal. This is why distributions are so desirable as estimators of goodness, and

[11] It should be noted here, that when we discuss **node expansion**, we are speaking of B* nodes. Each B* node expansion involves doing a certain number of probe searches depending upon how many descendants the node has, and what phase the search is in. This in turn will involve many thousands of nodes in the probe (hardware) searches. We do present the latter data at various places in this report, but for the purposes of understanding the B* algorithm, it does not matter how many descendants a node has, nor how many nodes were expanded in the search to find the bounding values.

OptPrb for a suitable goal gives an excellent idea of which branches may lead there.

In each phase, one player is goal-oriented and forcing and the other is an obstructor. The goal-oriented player **(Forcer)** tries the move that has the greatest chance of achieving some optimistic level of success, and the obstructor tries to limit the effectiveness of that move. Limiting the effectiveness of a move is a much more prudent strategy for the obstructor than looking for wild refutations. The Forcer and the Obstructor form a pair, and they succeed in limiting the scope of the investigation to the benefit of the problem solving process. In the two phases, the two opponents exchange roles. Thus, in the SELECT phase the Player is the Forcer, and in the VERIFY phase the Opponent is the Forcer. This exchanging of roles continues until the optimism of the Forcer does not change the existing view of what is going on.

The level of success that the Forcer attempts to achieve is called TargetVal. At levels in the search where the Forcer is on move, the branch chosen is the one whose OptPrb indicates that it is most likely to reach TargetVal. At levels where the obstructor is to choose, the branch chosen is the one that has the best RealVal from the obstructor's point of view. This produces the firmest resistance.

The expression that computes TargetVal is (OptVal(2ndBest) + RealVal(Best))/2. We have no theory for this expression, but it seems to do an excellent job of keeping the goal at the right distance from what has been achieved already. As these values change, TargetVal will also change. The major function of TargetVal is to avoid computing the cumulative probability distributions that caused Palay's program to spend so much time. Each time TargetVal changes, the program goes through the whole tree (which is never more than 20,000 nodes, and seldom more than 8000) and computes the probability of any node's subtree having the potential to reach the TargetVal. These values are backed up by the product rule methods explained in Section 3.6.1, and this yields the OptPrbs for each node. This computation takes about 0.2 seconds on a SUN-4 workstation. As new information comes in during the search, the OptPrbs are updated along with the OptVal and RealVal in the affected subtrees.

This method should be compared to Palay's process which produced a rough estimate of the whole probability distribution associated with each node. That allows reading off the probability of achieving any given level of success in that subtree. However, the processing cost is large. Instead, we estimate a target distance, and then compute the ratio of area under the curve (OptVal-TargetVal)/(OptVal-RealVal). This is the probability of achieving a value greater or equal to TargetVal in the given range. The computation is considerably faster, since it does not require producing a distribution capable of being interrogated on a point-by-point basis. If it should turn out that the selected TargetVal is too difficult to achieve, it is moved closer and the process repeated, at still a fraction of the cost of Palay's scheme.

In Palay's method it is possible to choose the node that has the greatest likelihood of achieving the greatest success. The first author had many opportunities to observe this algorithm in action, and does not believe that it in any way outperformed the present scheme. It turns out that within limits it only makes a small difference which of several promising nodes are chosen. They will all be examined eventually, and it is sufficient to examine them in some approximately correct order. If one of them turns out to be a winner, it will eventually be found. If there are several winners, the first one found will be selected. The quality of the data, while truly excellent for the given purposes, does not deserve the kind of high quality mathematical manipulation that Palay's scheme involved. It is, in fact, necessary to introduce some variation into the selection process to guarantee a reasonably broad, if shallow, coverage of all candidates. This is treated under the subject of dithering (see Section 6). Any somewhat second-rate decision will actually support this process.

During the SELECT phase, TargetVal is always greater than the best RealVal, and as gains in RealVal are made during the search, TargetVal is adjusted accordingly. However, during the VERIFY phase TargetVal always remains at the value that the RealVal of the selected move must be reduced to for it no longer to be best. In both phases, we have found that it is not the best policy to always investigate the node with the highest OptPrb. It could be that such a subtree has already had a large amount of effort expended on it. In such a case, it is wise to select for investigation a node that is not quite as good, but has had less effort expended on it.

3.9.4 The Two Phases Working Together

It is the function of the SELECT phase to identify the best move for the PLAYER, and the function of the VERIFY phase to show that this move is not best. This sets the tone for everything else. At all times the move with the greatest RealVal is considered to be the best move. The Forcer will at each choice point select for search the node with the best OptPrb with the proviso that the dithering factor may cause occasional effort to be diverted to a node with a less good OptPrb. The latter is useful to assure that the most promising node does not get **all** the search effort. In the SELECT phase it is important to examine every possible move that has potential, and to pursue those with the greatest potential until they are:

- shown to be decisively better than the competition,
- found to be lacking, or
- time expires.

In the VERIFY phase the same logic is applied in attempting to find that the selected move is not best. In the SELECT phase all effort is put into being sure that every possible promising move is tried. In the early stages of a search when optimism abounds, this means that the TargetVal is a good distance from the best RealVal. As the optimism abates, the TargetVal comes closer to the best RealVal. However, in the VERIFY phase, TargetVal is a firm value. If the value of the selected move can be brought down below this value, then the selected move is considered refuted, and control returns to the SELECT phase.

3.10 Knowledge Issues in Optimistic Evaluation

There are certain problems associated with assigning OptVals to moves that are checks, that threaten mate, or that start with certain kinds of captures. These are domain specific issues for chess, that Palay did not succeed in solving.

All values are the result of probe searches, which produce more consistent values than static estimates [26]. Optimism is best portrayed by allowing the Forcer to make an extra move before beginning the probe search that produces the OptVal for this move. In chess, this amounts to discovering if the Forcer has a "threat" that he could execute on the next move. However, there are many cases where the extra move gives a false impression of what is going on.

The most obvious one of these is a checking move. Clearly, one cannot make an extra move here, as this would result in the capture of the king that is in check, which would not tell us anything useful. Instead, we have determined that the strength of a check is a function of the following positive properties:

1. Is the checking piece safe from capture?
2. How few replies to the check are there?

3. Is the king forced to move?

Thus, we produce a value based on these considerations. We discount the value of a check, so that checks near the root of the tree tend to be preferred.

Another problem is the threat of mates. A normal 3-ply Hitech alpha-beta search with extensions is capable of always finding mates in 2 moves, and frequently mates in 3 or even 4 moves (because of extensions). Such threats are very potent, and are something that is essentially impossible to discover without probe searches. We consider mate threats to be better than all checks that have 3 or more replies. The number of moves to mate is not important. We also discount such threats as a function of distance from the root.

A third type of problem, we call hit-and-run captures. Here a piece will capture a man that is well guarded, and then use the extra move to escape with the capturing man, thus giving the illusion that there is a threat to capture whatever was grabbed in the hit-and-run. We have solved this problem by careful case analysis of the principal variations brought back by the optimistic and realistic searches. If:

1. In the optimistic line of play the piece making the extra move is the same as the one that does the capturing, and
2. In the realistic line of play, the capturing man is itself captured,

then only a small fraction of credit is allowed. Similar considerations deal with moves in which a piece moves twice, but would have been captured in the line of play that produced the RealVal. Such "threats" are not to be taken too seriously.

We have been able to use the pattern recognizing ability of Hitech [6] to create patterns that produce optimism in special situations. If a king has wandered away from his home base, it is worth endowing such positions with a few points of optimism for the opponent to encourage further exploration. Another situation occurs when there is a pin (which standard Hitech does not detect). Here again, we encourage the side that is making the pin to consider the position optimistic for itself.

We have only developed a few optimistic patterns, but they are producing excellent results. It should be pointed out that brute-force programs have had problems with the issue of optimism since time immemorial. Consider, for instance, the value of a pin. It could be that the pinned piece will be lost completely, or that it will escape, or that part of its value will be lost. The quiescence search will do a fairly good job of arbitrating this. However, there are times when (say) a knight is pinned by a bishop, yet the knight is defended and cannot be captured immediately with gain. However, in the long run the knight cannot escape and some severe detriment will result from this. Of course, not every knight pinned by a bishop will suffer such a fate. It is impossible to decide statically how much of a debit to associate with such a situation; todays programs tend to use some heuristic average. This will encourage pinning and avoiding being pinned, but fails to address the main question. A pin is a source of optimism for the side doing the pinning. Only by pursuing this issue for some distance in the search, can the truth about the value of the pin be discovered. This is an example of a fundamental class of issues. There are very few things in the real world for which an exact value can be determined. Some sort of fuzziness represented by optimistic and realistic evaluation bounds is a good way of bounding the problem until more investigative effort can be brought to bear.

3.11 Effort Limits

In a performance program one needs effort limits, which may very well be quite *ad hoc*. It is rarely the case that one move is clearly superior to the rest, and in such situations the algorithm speedily determines this. However, in most cases there must be some assurance that the algorithm is taking appropriate action under conditions of uncertainty. For instance, how much effort should be spent in the first SELECT phase to decide the best move. Assuming that the search does not terminate due to an overwhelming preference for a move, there are three types of effort limits:

- The amount of time available for the investigation,
- How close are the competing alternatives, and
- That no effort be expended on any branch that has less than a likelihood, X, of achieving the TargetVal. As search time is used up, X is increased accordingly.

Similar limits control the effort to be allocated to refuting the selected move in the VERIFY phase. It could be that the very next move to be investigated would refute the selected move. However, in a pragmatic world there must be a balance between the likelihood of success and its cost. Each phase terminates when further investigation costs more than it is likely to help.

We set the effort limit at the start of the whole move selection process based upon the size of the saved tree (discussed in the next section), and the average amount of time that is available per move until the next time control. This is expressed as the "maximum" number of nodes that the algorithm expects to expand. In this implementation, that number is 225. 40% of the nodes are allotted to the first SELECT phase, and 50 nodes to the first VERIFY. After that, future selects and verifies, use approximately 50% of the remaining nodes with the proviso that there never be less than 15 nodes allotted to a VERIFY. This can on occasion result in exceeding the maximum node target. However, those occasions are rare, and are more than made up for by the large number of times the algorithm terminates without using up its node allotment.

3.12 Thinking Ahead on the Opponent's Time

Brute-force programs think ahead on the opponent's time by guessing the move they expect to be made, and then figuring out what they would do if that move were played. If the program guesses successfully, it may be able to respond instantaneously to the expected move. If the time taken by the opponent is inordinately long, it may even be able to penetrate more deeply in the search, and make a much better move than it would under normal time controls. Tournament Hitech improved upon this technique by putting a maximum time on thinking ahead for any move the opponent may make [7]. When that time was reached, it would think ahead on the next most likely opponent move, etc. There were occasions where it was prepared for any of 6 potential replies, and would respond instantly if any of these were played. In brute-force programs, saving parts of trees from the previous move has been thought to be impractical. The maximum savings would only represent about 5%[12] of the total work that is going to be expended on the next move.

---

[12] It takes a factor of 4.5 to do an additional ply of search [14]. To replace the part of the tree lost by the move made and the opponent's reply is $1 - (1/4.5)^2 = 0.95$. For chess the branching factor is about 35. [32]. The reason that 4.5 is smaller than the "theoretical" maximum savings due to alpha-beta, i.e. sqrt(branching factor) which is approximately 6.0, is that the transposition table introduces many savings [31].

The situation with B* think-ahead is quite different. At the time a move is actually played, the search tree represents a fabric of moves that justifies the selection of the played move. Although the program considers the move to be best, it does not know the best reply. All it knows is that there is no reply which can make a difference with respect to the choice that it made. After having made its move, it behooves the program to identify the best reply and the best answer to it.

Given this understanding, the correct method for think-ahead becomes easy to understand. The germane parts of the tree are saved when the B* search makes its move. Then the Opponent's best move is found using the procedure used for Player move selection. If the Opponent moves before this procedure terminates, then the analysis tree will have been enriched with new pertinent data. If the best Opponent move is found before he plays, the search will turn its attention to finding the Player's best reply. This can continue as time allows. We have observed occasions where B* Hitech has calculated certain parts of a tree 6 moves in advance, and is prepared to reply immediately as long as the game continues along the line of perceived best moves. If at any time, the analysis indicates that a move that was previously thought best is no longer so, the analysis will return to that stage and resume from that point.

In selective searches, where only small numbers of nodes are searched, it is feasible to save and enrich trees. In the brute-force paradigm everything is hit-or-miss. If the right move is guessed some effort is saved; if not, the think-ahead effort is all wasted. It is worth noting that even when B* Hitech is not quite sure which of three or so plausible moves is best, it is enriching the tree with its findings. A fairly high percentage of these are useful after an actual move has been made.

3.13 The Graph History Interaction Problem

The analysis of a two-player game can best be represented in the form of a tree. However, in games like chess where the same position can be reached along many different paths, it is more efficient to use a graph. Using a graph allows each position to be evaluated to a given depth only once [31]. The benefits go beyond the obvious one of preventing duplication of effort since different length paths can lead to the same position. For example, if we are normally searching to a depth of 7, and at depth 5 encounter a node that has a value brought back from a 5 ply search, it is as if we were searching to a depth of 10. This is a very important gain that really makes itself felt in the end game, frequently allowing the analysis to go to more than twice the expected maximum depth.

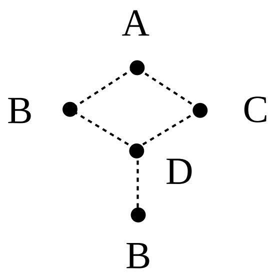

Figure 17: History can Affect the Value of a Node

Unfortunately graphs have one fundamental flaw in games like chess where a path that loops back on itself is scored as a draw. Figure 17 shows the problem. Each node in the figure represents a position and each arc some sequence of moves. The path ABDB is a draw by repetition. However, the path

ACDB does not repeat. How should position B be scored? As a draw or as the result of searching B? It clearly depends on the path that is taken, yet ignoring the path to a position is the source of the efficiency of graphs! This is the Graph History Interaction (GHI) problem; so christened by Andy Palay [26]. It is an insidious effect of using a graph to represent the analysis.

In a brute-force program, an analysis involves millions of positions. For this reason, graphs are typically represented as a hash table without overflow. Each entry in the hash table stores the essential information about a search done on a particular position. If two different positions hash to the same entry, the entry is changed to keep the more valuable information [7]. If a position that is flushed from the hash table is encountered again, it will have to be searched again. The benefits of compactly and efficiently storing a large part of the graph more than make up for the cost of collisions since most searches are not very valuable and the cost of searching an individual position again if needed is low.[13] Given this representation, how can the GHI problem be dealt with? There is no optimal solution, but how collisions are handled can help [11]. It has been found empirically that the likelihood of some deleterious effect due to GHI in the brute-force paradigm is far outweighed by the benefits described above. This is because most of the entries in a brute-force search are nonsensical, and the likelihood of some meaningful collision is small.

The situation is different for the B* selective search, thousands of nodes are generated instead of millions and information is continuously being passed around in the tree. The search tree must be preserved in its entirety, or the whole process would not work. For this reason we represent search trees as an explicit graph with each node in the graph pointing to its parents and children. The GHI problem is even more severe for a selective search because most entries in the search tree are well reasoned out, and there may be several paths to meaningful goals. One might question whether a tree representation would be preferable to a graph representation. There are two good reasons for still preferring a graph:

- Efficiency -- In the usual case, the path to a position does not matter and the work of exploring the subtree below any given position only needs to be done once. In a tree representation every time the same position is encountered the whole subtree below the position needs to be explored again. Given the relatively few nodes that a selective search can afford to expand, this duplication of effort would be ruinous.
- Logical Necessity -- If there were a sequence starting at the root that went A, B, C, and this leads to the same position as the sequence C, B, A, it would be important to know about it. Otherwise, the search would be under the impression that there are two equally good alternatives, A & C, and would have an impossible time deciding between them. This problem is solved in brute-force alpha-beta searches by just choosing the first of the set of alternatives with equal value. However, this is not possible when the search termination criterion is **deciding that one move at the root is better than all others**.

Hence, graphs and the problems they bring are necessary. This leaves the issue of how to deal with GHI. In selective searches an error due to GHI could be ruinous. We deal with the problem by using conditional values (**CV**). A CV for a node has a value[14] only if a particular combination of nodes are on or not on the path that leads to that node. CVs are only found in nodes that are on a strongly connected cycle. Most nodes are not on a cycle and do not need CVs.

---

[13]The cost of searching is low because the results of searching the children of the flushed position are usually still in the hash table.

[14]Actually a set of values including value bounds and the OptPrb.

The process for computing CVs starts by finding all nodes that are on a particular cycle. The possible paths through the cycle are then explored by starting from each possible entry point into the cycle and recursively exploring all cycle children. When a path loops, a repetition CV dependent on the looping node is created. CVs are backed up to each parent node by combining the CVs for all cycle children with the values from all non-cycle children. This process continues until all possible paths are explored. The details involved in making this an efficient process will be described in a forthcoming paper.

4 Examples of the B* Search in Action

For the examples in this section, the effort limit is 225 nodes, of which 90 are dedicated to the first SELECT phase. We wish to again remind the reader that probe searches are done for each node expansion, and that these searches are done on the Hitech hardware, yielding a move and a value. Such searches typically involve an effort of about 2000 hardware nodes for a typical 3-ply probe search. In our method of accounting, we count the expansion of one software node as one node as this gives the best view of what is going on. To read the logs presented, the following information is needed:

There are 10 node expansions listed on each line of the log. The number at the left of each line followed by a ":" is the number of the first node expansion on that line. The node that is expanded is in the subtree of the top level move named in algebraic notation. For instance, the first node expanded in Figure 18 was the root node, and this resulted in the move "b7c6" having the best RealVal, as indicated by the "*" in front of this move. A letter following the 4 character encoding of the move indicates the depth at which the probe searches were dispatched. For the third node, the "a" following the "d7c6" indicates the probe searches were sent out at depth 1 (there is a mapping of the letters of the alphabet to the first 26 integers). When a search is dispatched at depth N, it will, of course, reach to depth N+D, where D is the probe depth, plus any quiescence moves that may be explored below that. When there is no depth letter following a move, this means that this move became the best as the result of expanding a node in another move’s subtree. When a number appears by itself, this means that this is the new best RealVal for the whole search, and the "^" or "v" indicates whether this is an upward or downward movement in the value. Values are measured on a scale where a pawn is worth 128, and positional values of much lesser magnitude are included. Finally, a "$" indicates that the move following it is the best reply during the VERIFY phase. In the VERIFY phase, the depth indicator letters are in capitals rather than lower case.

Figure 18 is an example of a move that every program would make correctly. The only issue is how much time would it take to make this move. While there are 25 legal moves, there are really only two that do not lose material: d:c6 and b:c6. Even though this is the case, and even though b:c6 is vastly superior to d:c6, it is safe to say that any brute-force program would stop only when its effort limit is reached, which would usually be 3 minutes. B* Hitech, on the other hand, at regular tournament settings takes only 5 nodes to discern that b:c6 is clearly best, and then does a minimal VERIFY to see that nothing unexpected can happen. The result is that the move is made in 22 seconds instead of 180. Less extreme examples of this behavior abound.

The search penetrated to a maximum depth of "G" (7) + probe depth (3) = 10. At the end of the search is given the time taken, the number of software nodes, the number of hardware nodes, an indication of the lower bound[15] on the value of the move, and the projected line of best play. The "#" indicates that the

[15]Recall that the B* search only develops the tree to the point where it is sure it has the best move, at which point it only has a lower bound on the value of that move.

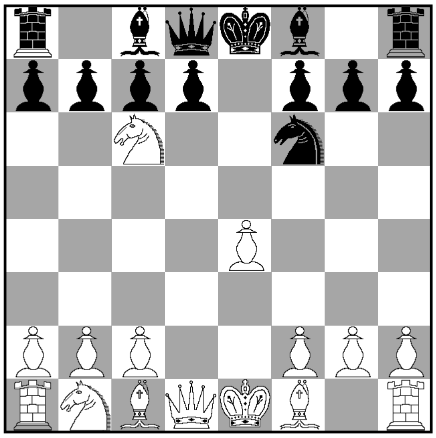

Figure 18: Black to Play

```
  1: *b7c6     30v  d7c6a  f8b4a  d7c6b $e4e5B  c1g5B $f1d3  $e4e5D    32^
 11:  e4e5F  e4e5D  e4e5F  e4e5D  e4e5D  e4e5G
   22.34 16 2154338 <= 32  b:c6 | e5 Qe7 Qe2 Nd5 Nd2 # c5
5. ....   b:c6        ***[b7c6]***
```

moves to the right were found by the hardware probe search. Since Black is the player, negative values are preferred.

Figure 19 is a much more difficult problem. Here B* Hitech examines the correct move, R:h2+, as early as its second node expansion, but does not recognize its value until expansion 68, at which point it discovers the correct follow-up 1.--R:h2+; 2. K:h2,g5! This is despite having gone to depth 9 (6+3) on expansion 23 without hitting on the right follow-up. In this search the issues are much less clear, and B* Hitech takes the full 90 expansions allowed for the SELECT phase and the full 50 allowed for the VERIFY. It should be noted that even though there is only one legal response for white during the VERIFY phase, this is still explored at times to a depth of 10, since it is possible that some later turn of events will show the idea to be unsound for black.

Again, the principal line of play delivered is correct, but obscures what is most important to human eyes. Namely, if after 1.--R:h2; 2. K:h2,g5; white were to make a safer-appearing move such as 3. Qd2, then would follow 3.-- Qh8+; 4. Kg2,Qh3+; 5. Kf3,Bg4 mate. Below the move output, we show the tree profile for the search, which shows the number of nodes expanded at each depth. It shows that the maximum depth reached was 11+3(probe search depth) = 14, which is about 5-ply deeper than a normal alpha-beta search would reach in this amount of time. The profile has the characteristic bell-shaped outline first noted in [4]. The average depth of the tree is 4.8+(3) = 7.8 which compares well with the 8-ply search that brute-force Hitech executes in the same amount of time. We assume that usually this effort is more effectively distributed into the interesting parts of the tree. In the present case the idea is not very deep (but complex to humans) and brute-force Hitech finds the winning line during the 6-ply search.

There is, of course, much more to the decision process than we have presented above. At each point where node expansion decisions must be made, the RealVals, OptVals and OptPrbs of the competitor nodes must be considered. It is impossible to portray all this information in such a short report. To give some idea of how things change, we present in Figure 20 detail the first 10 nodes from this search.

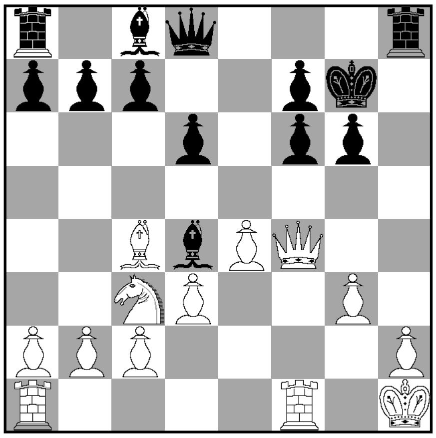

Figure 19: Black to Play

```
  1: *c8h3   h8h2a  h8h2b *d8d7a  h8h2c  h8h2c  -188v  c8g4a  g6g5a  h8h2c
 11:  g6g5b  g6g5c  g6g5d  d8d7c  d4e5a  d8d7d  g6g5e  d4c3a  d8d7e  g6g5c
 21:  h8h2d  h8h2e  h8h2f  d8d7c  d8d7f  d4e5b  d4e5c  d4e5d  d4e5e  d4e5e
 31:  d8d7c  d4c3b  d4c3c  d4c3c  d4c3d  d4c3e  d4c3e  d8d7e  d8d7f  d8d7g
 41:  d8d7f  d8d7g  d8d7h  d8d7i  d8d7i  d8d7g  d8d7g  d4c3d  h8h2d  h8h2e
 51:  g6g5c  d8d7g  g6g5d  d4e5c  g6g5f  g6g5g  h8h2e  g6g5h  d4e5c  h8h2f
 61:  d4e3a  d4e5f  d4e5g  d4e5g  d8d7d  d8d7e  d8d7g *h8h2c  d4c3f  c8g4b
 71:  c8g4c  h8h2e  d4e5g  d8d7c  d4e5h  g6g5i  d8d7d  d8d7j  d8d7k  d4e5h
 81:  d8d7g  d8d7h  d8d7i  d4e5d  d4e5e  d4e5f  d4c3g  d4c3f  d4e5g  h8h2d
 91: $h1h2D  h1h2D  -247^  h1h2E  h1h2E  h1h2D  h1h2E  h1h2F  h1h2E  -241^
101:  -245v  h1h2D  h1h2G  -243^  h1h2F  h1h2F  h1h2D  h1h2D  h1h2D  h1h2D
111:  h1h2D  -241^  h1h2D  h1h2D  h1h2D  h1h2D  h1h2E  h1h2E  h1h2E  h1h2E
121:  h1h2F  h1h2E  h1h2D  h1h2D  h1h2E  h1h2F  h1h2F  h1h2G  h1h2D  h1h2G
131:  h1h2D  h1h2D  h1h2D  h1h2D  h1h2D  h1h2D  h1h2D  h1h2G  h1h2D  h1h2D
  253.99 140 29397772 <= -241  R:h2+ | K:h2 g5 Qf5 B:f5 # e:f5 Qh8+ Kg2
        R:h2+        ***[h8h2]***
```

Depth	Nodes
0	1
1	7
2	6
3	21
4	34
5	23
6	18
7	17
8	7
9	4
10	1
11	1

5 What is the Rationale for a Working B* Search?

It is one thing to propound an algorithm, and show examples of it in action. However, it is still more satisfying to achieve a notion of why such an algorithm should work. We have shown that B*'s analysis is strongly dependent upon notions of:

1. Comparisons among alternatives with a view toward terminating when a clearly best

```
!! comment: After the root node is expanded (node 1), this is the state of
!! things.  Since Black is the player, negative values are preferred.

TargetVal=-1417 BestRealVal=-70 2ndBest=-66 MinAct=0.15
!! comment: MinAct is the probability level below which a move is not deemed
!! as worth expanding. TargetVal is based upon optimism available.
!!       OptVal  RealV PessV   OptPrb
c8h3  [  -296    -70  32766]  0.0000    !! This is the best move thus far
d8d7  [ -2764    -66  32766]  0.4993
h8h2  [ -4621    456  32766]  0.6311    !! Selected for Xpand; best OptPrb
 and 37 inferior moves

Node 2: R:h2+
h8h2  [ -4621    453  32766]  0.6309

Node 3: R:h2+ K:h2
d8d7  [ -2764    -66  32766]  0.4981
h8h2  [ -2764    207  32766]  0.4151
 and 37 inferior moves
!! comment:  All control parameters are still the same, but now d8d7 has the
!! best OptPrb since h8h2’s OptVal has worsened (though RealVal improved).
!! So d8d7 (which threatens mate in two by Rh2+; K:h2,Qh3++) is chosen next.

Node 4: Qd7
d8d7  [ -2764    -85  32766]  0.0000
New Best Move, New Best RealVal, New PV  <= -85  Qd7 Qd2
New TargetVal=-1446

Node 5: R:h2+ K:h2 Qd7
New TargetVal=-765                !! Comment: The failure of this threat
                                  !! reduces expectations by quite a bit.
Node 6: R:h2+ K:h2 Qh8+

Node 7: Qd7 Qd2
d8d7  [ -1807   -188  32766]  0.2572
New Best RealVal, New PV Qd7 Qd2 Qh3
New TargetVal=-894        !! Comment: Reducing TargetVal has made d8d7
                          !! into an acceptable action.  Now with some success
                          !! it goes up.

Node 8: Bg4

Node 9: g5

Node 10: R:h2+ K:h2 Bg4
New TargetVal=-541        !! Comment: Has not found the right idea yet, but is
                          !! narrowing down the possibilities.
```

Figure 20: Detail of First 10 Nodes

alternative emerges.

2. Use of optimism to guide the search.
3. Finding lines of play based upon simple resistance by the Opponent before allowing him to venture to find a refutation in the same style as the Player used.
4. Using probability distributions to capture the notion of goodness.

It is legitimate to speculate whether there exist any comparable notions in the real world that are similar to these.

1. The notion of comparison is fundamental to all intelligent activities, and the word "better" is found abundantly in the protocols of chess players collected by DeGroot [13].
2. The notion of optimism is also to be found everywhere. Human ambition is based upon optimism. Thus, any attempt to find something that is not immediately apparent must in some way be supported by an optimistic view of what is likely to happen.
3. The notion of optimism being met by firm resistance may be new. There are certain similarities to finding solutions in AND/OR trees. The correct strategy at an OR node is to examine the branch that is most likely to succeed, while at AND nodes it is correct to examine the branch most likely to fail. Thus, at Forcer nodes B* acts as if it were an OR node, trying to find the best possible alternative. Obstructor nodes are AND nodes. Examining the node with the best RealVal first is the most likely way to terminate the search. This node has established its RealVal as best, and is most likely to make the whole branch fail. This paradigm seems to capture what is going on in thinking about moves in a two person game.
4. Probability functions have long been known to capture the essence of subtrees, not only in the work of Palay but also in [27]. Probability distributions and the notion of fuzziness go well together and produces robustness. Point-value representations of nodes usually aim for too much precision and can end up being catastrophically wrong.

Thus, the B* search relies upon fundamental mechanisms that appear to be in ready supply in human intellectual activity. It is interesting to note that the idea of finding a crisp method of terminating the search was motivated by observation of human chess players. The notion of optimism is also supported by such observations. However, the first author has long held that human protocols just capture the tip of the intelligence iceberg. Thus, the need for distributions, and the need for the Forcer/Obstructor relation are not pointed to by anything we know of in the literature. They are the result of experimentation to find a workable algorithm.

6 B* Hitech and its Experiences

The present paradigm was not developed in a vacuum. The algorithm was implemented on the Hitech special purpose hardware [14, 6]. Most of the problems and solutions discussed herein were first discovered during the implementation which began in the fall of 1990.

6.1 Development and Testing of Early Versions

Initially, we attempted to combine the Conspiracy idea with the B* paradigm. As related in Section 3.2 this was not possible. The early development of B* Hitech dealt with how to evaluate the optimism of checks and mate threats, and how to deal with the notion of threat in general. Our findings are related in Section 3.10. Palay [26] had already investigated these problems and found certain remedies, but these were not quite good enough. The key to progress was being able to compare the lines of best play in the probe searches that produced both the RealVal and the OptVal. This allowed the discrimination of moves that were truly threatening from those where the threat was unlikely to be real because the threatening piece could be removed without any detriment for the opponent. This type of winnowing was absolutely necessary to reduce the number of moves worth looking at to some reasonable number that could be processed within the effort limits.

Much trial and error was involved here, since it was not at all clear, *a priori* just how much optimism good moves needed to have in order to be preferred, and in what order different types should be investigated. We believe that this is rather dependent on the evaluation function. Our experiences are described in Section 3.10, and we believe any programmer used to bringing up systems such as this could readily do the tuning appropriate for his system.

Once these knowledge problems were ameliorated, it was apparent that B* Hitech was doing very well on tactical problems. It was using the probability based approach and getting better results on the same set that Palay used in his thesis. In fact, these results were comparable to the results that had been achieved on this set by the best programs[16]. One reason was that we were using 3-ply probe searches, whereas Palay, due to computing time limitations, was only able to use 2-ply searches. All these searches are with quiescence, as a search without quiescence does not produce reliable values.

At this point, we began to run practice games against tournament (Brute-Force Alpha-Beta) Hitech that had an accredited USCF rating of 2400+. We had not yet dealt with the GHI problems, but were content to find out how well the new program could compete against the old.

In tactical situations the new program at least held its own with the old, but in positional situations the old program usually found steady answers while the new was at a loss for what to do. It was in these circumstances that we found that having a **dithering factor** (allowing moves to be tried that had less potential but had been explored less) was very useful. Dithering was a function of the number of node expansions in a subtree and the OptPrb of that subtree. Thus, a certain amount of OptPrb would be discounted if the subtree had already had a great deal of attention. This provided a moderate breadth-first component to the search. Hitech is smart enough to recognize a good position when it sees one. However, the goodness may not be apparent until a depth far removed from the root of the tree is reached. Thus, the existing optimistic functions could have considerable difficulty in plotting a course to such a position. Therefore, in positions where there are no clear tactical issues, the program is encouraged to explore those moves that have some potential for improving the position generally. This is an important facet of this work, as it is essential to be able to play this common type of position.

In the fall of 1992, after about 1.5 man-years of effort, the new program was still running at about 3 times real-time; that is about 10 minutes and 350 nodes/move. However, improvements in various decision procedures, and the realization that being able to think ahead on the opponent's time would produce certain gains, eventually got B* Hitech to the point where it was reasonable to believe it could play at the average rate of 3 minutes/move. We now had to put in things like time controls, tight effort limits, thinking ahead on opponent's time, and other little things that are needed in a tournament program. Because B* Hitech was not completely a real-time program, we determined that when the time remaining got too short, it would ask to be switched over to tournament Hitech which could manage time better, and be sure to forge a reply no matter how little time remained.

In test games versus tournament Hitech, B* Hitech got 41.7% of the total points. Most games were draws. It frequently got very strong positions but was unable to produce the consistent play to win. When it got into trouble it seldom escaped. It did, however, win some very nice games, based upon pure depth of calculation, and in these wins it beat tournament Hitech in a way that only very select humans had been able to do (see Appendix). But this only happened about once out of every 20 games. However,

---

[16]This early version of B* Hitech got 286 right out of 299 of the positions in the book of standard tactical problems [28].

these results were very encouraging, and certainly not taken for granted when the first author began this work.

This version of Hitech participated in three tournaments, being subject to some improvement, albeit with new bugs, each new time out. Since B* Hitech was able to play at only approximately 1.5 real-time we had to arrange for a transition to Hitech 5.6 when time became too short. Also, in the endings, we deferred to Hitech 5.6. The results are shown in Table 1.

Tourney	Result	Place
7th World Computer Champ. 1992	3 - 2	7th out of 22.
ACM Internat. Comp. Champ. 1993	3 - 2	3rd out of 12.
AEGON Human-Computer Tourn. 1993	3 - 3*	14th out of 32.

(*) Some of these games were played by Hitech 5.6 as bugs were found and corrected in B* Hitech.

Table 1: B* Hitech Tournament Results

The result of the AEGON tournament where humans play only computers and vice versa, is somewhat marred by the fact that B* Hitech had a serious bug, and we only played it in two games, in which it scored 1-1. However, the results in general seemed to indicate we were in the ballpark of a reasonably competitive program.

In other testing, we found that the original set of problems in [28] were no longer much of a challenge, and switched our major testing to a volume [17] that we had recently been using to test tournament Hitech. This volume contains positions that are much more difficult as they intermix tactical with positional and strategic themes. B* Hitech outperformed tournament Hitech on quite a few problems in this set, and discovered several errors in the book.

6.2 Testing of the Most Recent Version

The most recent version is really the first that we feel is relatively bug-free. This version appears to have no difficulties with GHI problems, and this has made it possible to solve some very difficult problems.

6.2.1 How Problems are Scored

The book "The Best Move" [17] that we used for testing has been widely regarded as the best instruction book for tactics and position play combined. We were quite surprised that out of the 230 examples in the book over 50 contained errors, some quite serious[17]. Each problem can earn the solver

[17]The reader may wonder how we can determine that a highly regarded book contains errors. The method used is to investigate positions where a program disagrees with the book. If in following the recommended line, the program finds some obvious error or improvement, then one must conclude that the book is errorful. Thus, this is a clean scientific approach to the validity of test material. Of course, it is always possible that both the book and the program make the same error; however, we will not concern ourselves with that case. It should be noted that the first author, a former chess by correspondence World Champion was unable to find any of these errors, and was instead enthralled by the imaginative presentation. The competence of the two programs is what brought the errors to light.

There is an amusing experience that can be related about the use of the book. There are two problems [#166 & #187] from the same game. In the first the book indicates that the situation is not yet ripe for the win which must be prepared. In the second, it gives a long and involved win. Both programs find a win in the first position, which is very delicate and clearly escaped the attention of the participants in the game. Neither program finds the win in the second position under tournament settings.

a certain number of points, depending upon whether it

- Judges the value of the position correctly,
- Finds the best first move, and
- Finds the best follow-up line.

We took the approach that the book's method of scoring was correct (we do indeed believe it is excellent), and when the given solution was wrong, we gave the intended number of points for the correct or better solution. When a program made the correct first move, but the principal variation did not represent best play thereafter, we continued the game until the program either found the best play or deviated from it.

Recall that both the B* and alpha-beta algorithms are not charged with anything other than finding the best first move. Thus, a given first move could lead to the win of a rook, but if the algorithm believes it can win a pawn by the first move, and no other first move can do as well, then it is justified in making that first move. B* Hitech will do the minimal amount of work to convince itself in such a situation. Brute-force Hitech may find the correct first move, but the principal variation it tenders in support of its decision may not have penetrated deeply enough due to resource limitations to find the absolutely best possibility. Therefore, both are given a chance to pursue their idea of the best line of play as the game continues. If this is the correct line they get full credit, otherwise whatever partial credit the book deems appropriate.

6.2.2 Test on a Book of Problems

	Poss Pts	Hitech A-B	4xTime	B* Hitech	2xNodes	Probe+1
Opening	**28**	**18**	**22**	**17**	**17**	**17**
MidGame	**837**	**559**	**587**	**547**	**604**	**613**
Ending	**235**	**166**	**179**	**145**	**149**	**151**
All	**1100**	**770**	**815**	**704**	**765**	**781**

Table 2: Results of Tests on "The Best Move"

Table 2 shows the performance of the two programs on the set of problems in the book "The Best Move" [17]. The first column shows the maximum number of points that it is possible to get in that category. The next five columns show the performance of five different programs by type of position. The five programs are respectively tournament Hitech, that Hitech given 4 times as much time (a factor of 4.5 allows a one-ply deeper search), B* Hitech at tournament settings, B* Hitech allowed twice as many nodes (which amounts to twice as much time), and B* Hitech with the probe searches going 1-ply deeper than normal (which amounts to 4.5 times as much time).

The results show that Hitech 5.6 is slightly better than B* Hitech in all phases of the game under tournament conditions. It is interesting to note the performance of the other programs that represent various ways of investing additional time, if such were available due to better hardware in the future.

In the opening, Hitech 5.6 benefits from going 1-ply deeper than normal by improving its performance by 22%. On the other hand B* Hitech does not benefit at all from having more time to explore or by having deeper probe searches. This clearly indicates that B* Hitech's understanding of the opening is limited in some way that the deeper searching Hitech 5.6 is not.

In the middle game, we see the potential of the B* search. Under tournament conditions Hitech 5.6 still outperforms B* Hitech. But when given 4 times as much time Hitech 5.6 improves by 5%, while B* Hitech improves 9% with a factor of only 2 more in time. Some additional improvement on this is seen if the probe searches are made one ply deeper, which allows a still better understanding of threats.

In the ending, it is again a case of the brute-force search being clearly better, although the improvements in B* Hitech with additional time are slightly better than those of Hitech 5.6. We should note that 3 or 4-ply probe searches are just not enough for B* to understand endgame play at the Master level, without additional knowledge. As indicated, B* Hitech runs in tournament mode only until the amount of material is reduced to a certain level. It then defers to Hitech to finish the game. We deal with some of the issues of how to remedy this in Section 9.

6.2.3 Games versus Tournament Hitech

Opening Type	Closed	Semi-Open	Open	Total	%
Points for B*	2.5	4.0	3.5	10.0	41.7
Points for A-B	5.5	4.0	4.5	14.0	58.3
Avg. B* Stopped	35.5	29.4	30.0	32.6	
Avg. Game Ended	54.1	46.4	41.4	47.3	

Table 3: Test Games versus Hitech 5.6[18]

Table 3 shows the results of 24 games between B* Hitech and Hitech 5.6. These games were the last tests made and came after a number of serious bugs were remedied, including bugs in the hardware and some very old software bugs. The latter two affected both programs. The games were played from 12 opening positions which represented a variety of situations. For each starting position, two games were played. In one B* Hitech had white, and in the other it had black. Both programs had identical time limits for making moves. Thus, the contest was completely balanced. Games were terminated when one side had won, when a draw was declared by repetition or insufficient material, or when B* Hitech announced that it was no longer qualified to continue to play the current ending because of reduced material. In the latter cases, the following procedure was used to evaluate the outcome:

- If the programs agreed that neither side had more than .125 of a pawn advantage, the game was declared a draw. This was the outcome of 3 games.
- If the programs agreed that one side had an advantage of two pawns or more, the game was declared a win for that side. This terminated 6 games.
- In all other situations, the game was continued by tournament Hitech playing both sides, with appropriately inherited time information, until a definite result was obtained. There was a proviso that if both programs agreed one side had the advantage when the game was continued, it was not allowed that the other program win the game. This proviso never needed to be invoked.

This procedure simulates the play of B* Hitech under tournament conditions. The conditions for terminating play were decided on before any games were run, and were invoked to remove randomness

[18]Open, Semi-Open, and Closed refer to the character of the position as determined by a particular starting position. Closed means no pawns have been captured or that two opposing center pawns abut; Semi-Open means one of the two center pawns of one side has been captured, and Open means that at least one center pawn of each side has been captured.

that could occur in nearly even endings due to some horizon effect [3], or other aberration. They act equally on both programs. It can be seen that the above is a purely mechanical procedure, requiring no chess knowledge.

B* Hitech got 41.7% of the total points. This is a very good result. It certainly is decisive to lose 14-10. However, it is also amazing that a selective search program can get this many points from an established brute-force program. It should be remembered just how much work the computer chess community has put into the development of the alpha-beta search. It is now a very robust vehicle, capable of performing in all phases of the game. B*, on the other hand, requiring many more control decisions, has not had nearly that amount of work put into it (despite some intensive effort on this project). We have avoided the problem of B* Hitech's poor endgame play by not allowing it to play the ending. 13 games were played out in this manner. We have included in Table 3 the average move of switch over from B* to regular Hitech for these games, and the average move when the game was over. This will allow the reader to gauge the degree of involvement of endgame play by tournament Hitech in finishing off games of B* Hitech. It is not surprising that games in closed positions were continued further, as swaps of material are rarer in such positions. In its current state, B* Hitech with Hitech 5.6 playing the ending is rated only 65 points worse than tournament Hitech on the human scale. This puts it near a US Chess Federation rating of 2350, based upon tournament Hitech's rating of 2413, and makes it the best purely selective search chess program known[19].

Looking in detail at Table 3 one can discern that almost all of B*'s deficit can be attributed to its play of closed positions. In such situations strategy is of paramount importance. In Hitech 5.6, the battle-hardened veteran, strategy has been built in by means of pattern recognizers that find useful patterns far from the root of the search [6]. However, B* Hitech can only find patterns 3-ply removed from where the probe search originates. This frequently does not allow the patterns to be fully developed as yet. Hitech 5.6 occasionally finds some "deep" idea as a result of brute-force search, but independent of the pattern recognizers. However, more frequently B* Hitech found such ideas. This resulted in turns of events, that Hitech 5.6 did not have an inkling of until the game had turned decisively in the direction dictated by B* Hitech. However, to become a real competitor B* Hitech needs, at a minimum, to be able to play closed positions better.

7 Discussion of Test Results

7.1 B* Search Examples

One interesting and *extreme* example of the differences between the understanding of a brute-force and B* search is shown in Figure 21. Here black can make a draw by correct play. Tournament Hitech sees that after 1.- f5!; 2. Ke5,f4; 3. Ke4,f3; 4. Ke3,h5; 5. Kf2,h4; 6. Bd6,Kh3 the position is a draw because 7. K:f3 is stalemate. This is the kind of thing that humans can easily miss, but tournament Hitech finds it in 8 seconds. B* Hitech on the other hand, looks at 10 nodes to decide that 1.- f5 is the only reasonable move. It continues that way until it is faced with 4.- h5 where it chooses h6 instead. But this makes no difference. It goes down the only path available to it and at the end, finds to its surprise

[19]There are micro chess programs that may be better, but these programs use one of the hybrid search strategies discussed in Section 2. We wish to make the point that the B* paradigm has been pushed very far.

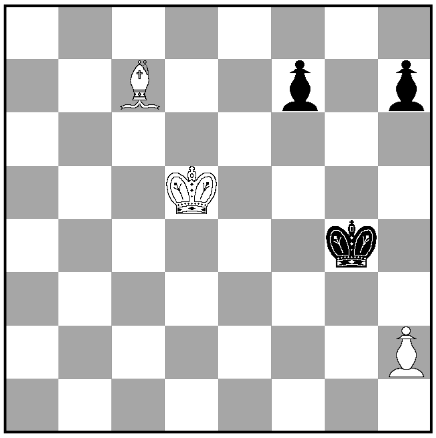

Figure 21: Black to Play

that the position is a draw. Actually, neither program understands the position is a draw without a great deal of additional work. They both will avoid the move K:f3 as long as possible, always believing that white still has a chance as long as he avoids this move. This is what every computer program that plays in tournaments would also do. However, it is interesting to see how B* Hitech only looks at enough of the solution to be sure what the best next move is, while tournament Hitech looks as deeply as it can.

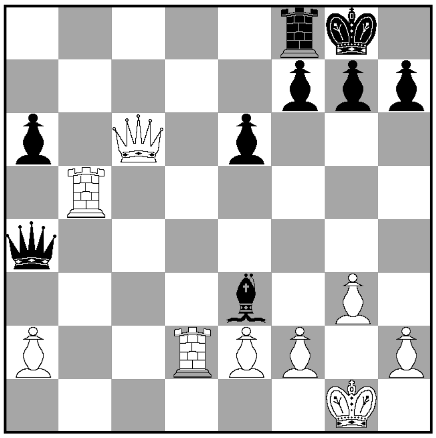

Figure 22: White to Play

A more typical example of how things work, where B* Hitech looks much deeper into a situation can be seen in Figure 22. This is from an infamous article by former World Chess Champion Botvinnik [10], in which he demonstrated that his "program" did not at all understand the position[20]. However, the resulting commotion about his failure inspired us to see that B* Hitech would solve this problem correctly, since this is exactly the kind of problem it should excel on. This resulted in some refinements in our treatment of the GHI problem. The way B* Hitech now solves this problem is exemplary of its best play.

Within 120 nodes B* Hitech discovers the key move 1. Rd8!!, and has found that if R:d8; 2. Rd5!! wins. The principal variation gives 1. Rd8,Q:b5; 2. Qd6!,B:f2+!; 3. K:f2 because B* Hitech considers this to be

[20]From a game Kasparov-Ribli, 1991, in which the World Champion missed a win.

the line that gives White the best chances. This is correct. After Black plays Q:b5, it quickly decides on Qd6!. Then after B:f2+ it realizes that there are really two competing alternatives. It explores K:f2 to a depth of 29 ply, and finds the complete win. The principal variation it delivers is 3. K:f2, Qf5+; 4. Kg1!!(not Kg2 whereupon Qd5+ gives good drawing chances),Qb1+; 3. Kg2,Qe4+; 4. Kh3,Qf5+; 5. g4,Qf1+; 6. Kg3,Qe1(or g1)+; 7. Kf3,Qf1+; 8. Ke3,Qh3+!; 9. Kd4 leading to a winning position. Since it does not consider that longest resistance is the main criterion of successful defense it does not give the line e5+!; 10. Kd5,Qg7+; 11. Kc5,Qg1+; 12. Kc6,Qh1+; 13. Kb6,Qb1+; 14. Ka7,Qg1+; 15. K:a6 and there are no more useful checks. However, this line is documented in its log as part of its supporting analysis, as are less interesting sidelines such as 9.--Q:g4+; 10. Kc3,Qh3+; 11. Kb2.

To understand the chess part of the analysis, it is useful to note that if instead of 3.--Qf5+ Black plays 3.--Re8, then 4. a4! forces Q:a4; after which 5. Qe7! wins. But not immediately 4. Qe7?, as Botvinnik's program purportedly played, as then Qb6+ wins. B* Hitech discovers all the analysis in less than 10 minutes. The major point of this example is to show how deep the analysis can go when it makes a **top level decision difference** in the choice of moves.

When this position was presented to tournament Hitech it played 1. Rd8, expecting: Q:b5; 2. Qd6,B:f2+; 3. K:f2,Re8; 4. R:e8+,Q:e8; 5. Q:a6 with a small advantage. Then after 1.--Q:b5; 2. Qd6,B:f2+ it found that Black had better play, and after 3. K:f2,Qf5+; 4. Kg1,Qb1+; 5. Kf2,Qf5+ it was content with a draw, since it could see no advantage to venturing out into the open with 5. Kg2,Qe4+; 6. Kh3,Qf5+; 7. g4,Qf1+; 8. Kg3. The critical difference here is that B* Hitech has its optimism (the mate threat at f8) to keep it going, while tournament Hitech assesses the current situation with no understanding of such issues. We have confirmed that many top chess programs behave exactly like tournament Hitech, being afraid to venture into the open with the king, without being able to see the end of the situation.

There are some general observations on the interaction of a selective search and a brute-force search that are worth making:

- When the selective search program has the better position, it will attempt to win by finding ideas that improve its position further. This is mostly done in an excellent manner. However, sometimes it goes astray when not fully exploring all defenses and thus overlooking some possibility that can foil an idea. On the other hand, the brute-force program will defend meticulously in such situations. This is with one exception: When the brute-force program sees that everything it can try has approximately the same result, it then chooses to defend in a manner that looks random to a human. Since it has no notion of difficulty, this results in not requiring much understanding for the opponent to make progress. This phenomenon reminds strongly of a similar point we made with respect to games in which a program that searches to depth N has the better position against a program that searches to depth N+k [8]. Here the deeper program will frequently see too much for its own good, and retreat without any notion of whether it is offering resistance in the human sense.
- When the brute-force program has the better position, it essentially plows ahead always considering the best defenses as it sees them (which are usually the correct ones). However, a selective search has little idea of what a "defensive" idea is. It is essentially impossible to define statically, and thus would have to be the result of some probe search which tells what the opponent is threatening. The only implementation of something such as this in B* Hitech is what comes up during the VERIFY part of the search. However, this is only in response to moves that have been proposed by the SELECT phase, and these are essentially random when one considers that a defensive idea is called for. We intend to try to identify defensive situations, and deal with these in a somewhat different way. It is clear that, while the extra-move paradigm is an excellent way of gauging optimism, there are several situations in which other methods are required in order to have some way of "lighting

a path" for the search.

In general, the depth that a B* search reaches has little to do with the number of nodes expanded. If the position has a forcing character, B* Hitech will plummet to whatever depth is required to find a solution. Explorations to depth D have been found empirically to seldom take more than 1.2^D nodes, which for D=20 is only 38 nodes. It is in situations that lack a forcing character that additional effort is useful. Unlike the forcing positions where the deepest node expansion could be at depth 35 or so, placid positions seldom need be explored to a depth greater that 14. We believe the effective branching factor for such positions is in the range 1.5 - 1.6. Thus, the expected number of nodes needed for exploration is between 1.5^{14} = 292, and 1.6^{14} = 720. It is in this area that the most additional computing power could be used. This is because the OptVals are not always able to capture the ideas that are pertinent, and large searches are required to find out if anything interesting can be done. Of course, it is in positions such as this that the brute-force search performs so admirably. However, to do a 14-ply brute-force search would require the power of 4.5^5 = 1845 Hitech's in a single machine. With **that much** power, it would be possible to relax the criteria for search termination, and investigate much more of the tree than is currently done.

It should be clear from the above data that B* has its greatest advantage over alpha-beta when the branching factor of the tree is high. This augurs well for its use in games such as Go.

7.2 The Horizon Effect and the B* Search

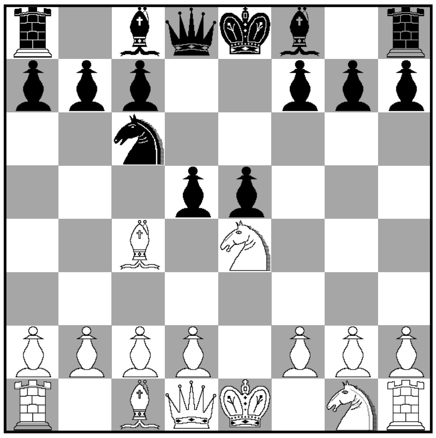

Figure 23: White to Play; last move was d7d5

Let us assume the position in Figure 23 occurs as a leaf node in a brute-force search. Since white has no meaningful captures and is not in check, the quiescence search will allow him to pass, and the position will be scored as good for white since he is ahead in material. It is possible that a brute-force program with extension heuristics may notice that white has two pieces *en prise*, and that there is a good possibility that it may lose one of them. Such a fact could cause the search to continue and come to the conclusion that white is, in fact, slightly worse off. However, detecting such a situation is computationally expensive, and not always correct. For instance, if the black king were at f7, then the pawn at d5 would be pinned and the N at e4 would not be *en prise*. This is but one kind of problem that brute-force searches have in deciding whether the score being attributed to a leaf position is correct. There are many other kinds of problems such as when a piece is pinned but can't be captured immediately [3]. It has been the view of the computer chess community that such problems can be largely done away with by ever deeper searches. That appears to be true; however, if such a problem occurs in a critical branch, it will cause bad effects in the search.

Now let us examine how the B* search deals with such problems. Let us say that the B* program is playing white in Figure 23, and this is a leaf position in the principal line of play at the end of the SELECT phase. Now when the opponent begins to VERIFY, he exerts his optimism in connection with his last move d7d5, and notices that with an extra move he will be a pawn ahead after the capture d:c4 or d:e4. This optimism will cause the search to want to explore the consequences of d7d5, and white will be forced to respond because black wants to know what is going to happen. In this way the full consequences of the position will be investigated until it is clear what will happen, or that it is not of interest.

This same kind of **I-have-optimism-that-needs-to-be-investigated** attitude will be applied to all leaf nodes in which good effects are waiting to be discovered. It is only necessary that the good effect be reasonably probable and can be found within the probe search. Thus horizon effects, which push undesirable results over the search horizon, do not seem to occur in B* searches. If they do, they must be the result of the probe search (which uses brute-force) being subject to a horizon effect. During this work, we have seen some bad bounds being brought back for this reason. However, we have noted no horizon effects of any other type. This is clearly an important consideration, all other things being equal.

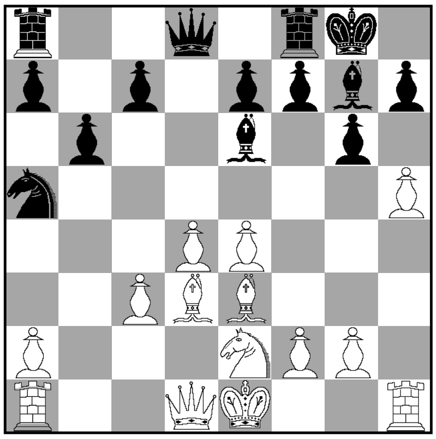

Figure 24: Black to Play

In Figure 24 Black (Hitech 5.6) plays Qd7 based upon a 10-ply search, unaware that there is any problem. Of course, any human would be leery of such a move, since the response d5 creates a situation where it is problematical whether the bishop at e6 can survive. However, Hitech 5.6 horizons the situation with the variation 1. d5, Rad8 after which the bishop can delay moving for a move or two, and even after it moves it can not be captured immediately. After the faulty Qd7, B* Hitech does a 118 node search in which all except 17 of the nodes expanded are in the subtrees of the responses d5 and h:g which are the only two moves that good humans would look at. It then decides on h:g, but quickly finds that this allows f:g, after which d5 could be met by Bf7. It then immediately switches to d5, and after the VERIFY comes up with a principal variation of 1. d5,Bg4; 2. f3,B:h5; 3. g4,B:g4; 4. f:g4,Q:g4; 5. Qd2,c6 which is about what a human Grandmaster would calculate here. To do this B* Hitech penetrated to a depth of 14 in certain parts of the tree, but the whole analysis revolved about the issue of whether the unfortunate bishop at e6 could escape, whereas Hitech 5.6 had no inkling of the problem. This is an extreme but very to the point example of the difference in outlook between B* and its brute-force competitors.

8 Speed and Potential Parallelization

Our method of getting bounds by the use of shallow searches, requires large quantities of computing power. Yet, having tried to produce bounding estimates by means of static evaluation functions, it appears clear that there is essentially no choice but to do shallow searches at the current state of the art. This limits the use of the B* search algorithm to machines that can do the required shallow searches in real time. It is certainly conceivable that in some domain other than chess it would be possible to get bounding estimates more cheaply. However, it is the first author's experience that when it is easy to get good bounds statically, there is no significant problem to be solved.

It is always interesting to examine an algorithm to determine how it would scale up with additional computing power. Section 6.2 shows that the deeper the probe searches the better the problem solution. Also, the larger the set of nodes investigated, the better the solution. While it may be possible to improve the performance of the system by running it on a faster chess-specific piece of hardware, this is unlikely as it would require the construction of a machine especially for that purpose. It is much more sensible to think of how one would use a general-purpose parallel machine to get increased power.

Our argument regarding the advantages of parallelizing B* takes the following form:

1. B* performance has already been shown to increase with additional nodes investigated, and depth of probe searches over a modest range
2. B* decomposes easily for parallelism
3. Potential losses in efficiency come from having to predict which nodes need to be expanded, and there may be some loss if that node is expanded but would never have been if a single machine were at work. We estimate the magnitude of such losses.
4. We review the literature on parallel decomposition of alpha-beta search and what the efficiency losses are.
5. Since the losses due to alpha-beta are are on the order of 90% and the losses due to B* are on the order of 15%, we conclude that as computing power increases, B* searches must overtake alpha-beta searches.

The B* algorithm decomposes nicely for limited parallelism. Since there are on average 35 legal moves in a given position [32], one can use on the order of 70 processors to do the probe searches required by a single node expansion. There will at times be as many as 70 legal moves, and at times very few. Thus, it appears that 128 processors could be kept busy doing in parallel what B* Hitech does serially now without any particular effort to schedule processors. At present B* Hitech takes about 2 seconds to do all the processing of a node. With 128 processors equal to Hitech, this reduces to .02 seconds. The fact that certain processors may not finish on time has been dealt with in the literature on the tree-splitting approach to parallel alpha-beta searching; cf. [19, 22]. At present Hitech searches about 120 nodes in the three minutes it is allotted on average for a move. These 240 seconds contain about 80 seconds worth of work done on the opponent's time.

To do an N+1 ply search takes a factor of 4.5 more than it took to do the N ply search [14]. If a machine were capable of doing a 13-ply search, it could instead do 37,367 6-ply searches. This would be enough to investigate about 747 nodes, which should be sufficient in 99.9% of all cases. In fact, since the search would use less than half this number of nodes most of the time, it will have saved up enough time to investigate any difficult position that may come up.

The whole issue in scaling up is how many processors must be kept busy at the same time. If it is 128 or less, there should be no loss of efficiency as they will all be working on the current node that is being expanded. If there are many more processors, then there is the issue of what tasks to assign them. The normal thing to do would be to attempt to find the node that is most likely to be expanded after the present node and start work on that. This is not difficult, as we have explained in 3.9.3. There are many candidate nodes to expand, and we even do dithering to be sure to not be too myopic about things. So it seems fair to say that 512 processors (4 nodes worth) can be kept busy with essentially no loss. After that there are bound to be some losses due to the fact that node expansions will bring on enough new views of the world, so that those nodes that looked good under the old view will no longer look that good. We have done some examination of search trees generated by B* Hitech to support the above views. Based on our examinations, it seems highly unlikely that more than 1 out of 5 nodes selected in advance for expansion would not have to be expanded later anyway. To be safe, let us assume the loss is actually 30% for up to 1K processors (14 nodes in advance). When we have more than 1K processors, it would be wisest to invest the additional power in deeper probe searches.

The critical number is the amount of time that is available to process a single node. Under chess tournament conditions the time limit allows approximately 3 min./move. If we plan to expand (say) 300 nodes in the process of selecting a move, that each node must be done in about 0.5 sec. If a probe search cannot be done in this time on a single processor, then several processors must cooperate to do the job. If there is a surplus of processors, then getting 4 of them working on the same search will produce a gain of about 2.5 (or .5-ply) [19, 22]. Large amounts of computer power will have to be used by doing deeper probe searches using parallel alpha-beta. This will involve losses due to parallel alpha-beta. However, despite several claims for linear losses in parallel alpha-beta, the data seem to indicate that a small number of processors can cooperate with much greater efficiency than a large number. So the bottom line is a large number of processors would most effectively be used by having 1K N-unit processors doing parallel alpha-beta, and thus have the ability to process about 14 B* nodes in parallel. We now turn to the issue of how powerful such an N-unit alpha-beta search might be.

We estimate that a 3-ply search is approximately equivalent to the instantaneous visualization of a Grandmaster without doing any calculation. We conjecture that this is what drives the human search, and it is how B* Hitech operates presently. If 1K units were to be available, each doing a 6-ply search, this would be a very formidable machine. And yet, such machines are presently doing pure alpha-beta searches. We have just completed additional experiments that show that one gets approximately a 13% gain in performance on the problem solving set by going from probe-depth = 3 to probe-depth = 4 and doubling the TargetNodes. This is approximately a factor of ten of additional power. A 6-ply search completely dominates a 3-ply search. In [8] are recorded some experiments with Hitech where a 7-ply search wins 15.5 - 0.5 over a 4-ply search. This reference contain much similar data including some results of Thompson which showed 6-ply beating 4-ply 19.5 to 0.5. So the perceptual power of a 6-ply search should be considerably more powerful than that of a 3-ply search.

In a recent speed chess tournament in Munich a program named Fritz3 running on a Pentium Plus processor finished tied for first with World Champion Kasparov at the head of a field of 18 of the World's best Grandmasters. If such a program were fast enough to do probe searches of depth N in less .5 sec, then in three minutes, 128 such processors could generate a B* search tree of 360 nodes using depth-N probe searches.

We now turn to the efficiency of alpha-beta. There are a number of papers in the field [18, 15, 21] that make claims of a linear degradation for additional processors doing alpha-beta in parallel. However, field results do not seem to bear this out. There appear to be ever greater percentage losses as more processors are added. This is hardly surprising, since alpha-beta is a recursive algorithm which depends on its efficiency on processing the right hand part of a tree knowing exactly what has happened in the left-hand part. This does not happen in parallel alpha-beta and this is the major problem. On the other hand, using each parallel processor to do part of a B* search as described above does not encounter such problems.

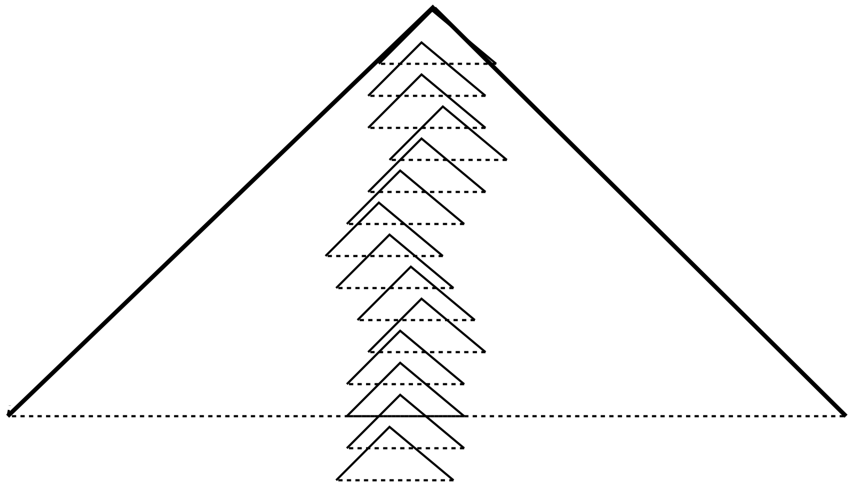

Figure 25: Division of Labor Pays Off

In Figure 25 the work tradeoff between the B* method of doing searches and the alpha-beta becomes clear. Each triangle represents a node expansion doing probe searches. If that discovers something interesting, then the first move is a step in some direction that is further explored. Thus, the triangles can reach to almost any depth. If it were possible to do 6-ply probe searches, then the first would see almost half as deep as a 13-ply search. If that looks interesting, it only takes 7 more node expansions in the same subtree to reach the limit that the 13-ply search reached. If the guidance was good in that particular line of play, it will have discovered all that the 13-ply search saw. If the guidance was less than perfect then there is consolation in the fact that investigating this single line of play is only .00615 of the total effort that the brute-force would have expended to reach this depth. Thus, even a moderately informed program should have enough resources to reach the important parts of the tree, and go well beyond the depth of 13 when required.

The bottom line is that when enough processing power becomes available to search one ply deeper, this is best invested by increasing the depth of probe searches of a B* search rather than going from depth N to N+1 in an alpha-beta search. Although the chess strength gain is still subject to further testing, it is clear that efficiency issues very much favor the B* approach.

9 Summary and Conclusions

What can one say about the play of Hitech using the B* algorithm? It is very illuminating to watch. When observing a brute-force search in a complicated position one never knows what to expect. If there is some turn of events that can be made explicit by a depth 8 search, then there is no inkling of this until it suddenly appears at depth 8. It is as if a grade D average student suddenly comes home and announces he made the honor roll. When there are two pots of gold at the end of the 8 ply search, brute force will the select the larger, and if there is no pot of gold, it will bring home the most pennies it can find. Any ideas that exist can only to pointed to in retrospect.

With the B* search things are very different. One gets the feeling of ideas. This is because the notion of threat carries the analysis in some direction. It is trying to do this or that. Much of this will, perforce, not turn out; but that is the nature of exploration. However, as long as an idea is promising it will be pursued to whatever depth it takes to come to a conclusion. Like humans, this search will bring back the first pot of gold it finds, quite satisfied, and not go looking for more.

One cannot over-emphasize the importance of **a timely termination of the search**. It is illuminating to find B* Hitech making a move without knowing what the opponent's best reply is. All that it knows is that:

- It has found the best move, and
- The opponent's best reply is not a timely topic before it makes its move.

All it needs to be sure of is **that it is, in fact, the best move**. This is the cardinal idea behind the B* approach: to find the move that is considered no worse than all other siblings at the root. This distinguishes the decisions made at the root level from the decisions made at lower levels. It distinguishes the decisions that are made during the SELECT phase from those that are made during the VERIFY phase. It is quite amazing to see B* Hitech make a simple non-capturing, non-checking move in a few nodes of investigation, because it realizes it is the best. It could be something simple such as occupying an open file with a rook, or moving a king toward the center in the endgame. These are things that humans, too, do quickly. The move selected is obviously the best of a set of siblings, and does not allow any important counteraction. Therefore, it is selected without much ado.

However, there are some things that the B* search still does not do very well despite considerable effort to overcome them. Chief among these is defense, which is an area where the brute force search excels.

One may wonder why it is so difficult to find good defensive moves. It seems to go with the territory. It was not unusual that the best defensive human chess player was also the World Champion. This was the case with Steinitz, Lasker, Capablanca, Petrosian, and Fischer. Not a bad collection. However, the main point is that defensive play is very hard in the human style because one must anticipate **all** of the opponent's worthwhile ideas, and figure out how to meet them. This is the kind of thing that a brute force search does routinely. However, when it comes to following some important idea to great depth, then the selective search does much better.

Harking back to our original premise about "natural" searches, we believe we have accumulated some evidence that there is a "natural" way for humans to search two-player situations. We feel this is as ingrained as the notion that (A>B) & (B>C) -> (A>C). It is part of some fundamental human armament. We have not tried to make a strong case for this, but made many oblique references to phenomena that support this view. The fundamental properties of this search are the same as the precepts of the B* search. Some of these properties can also be found in the protocols of chess players in [13].

- Search only until there is a clearly preferred alternative. This implies the need for strong comparison of alternatives at all times.
- Use optimism to guide your search.
- Assume that your opponent will make steadfast replies while you are formulating your plan, and then reverse the view to see if he can upset your plan by implementing his own ideas.

Our experiences show that the knowledge required to get good optimism is non-trivial. There are a variety of situations, and each seems to require its own special optimism in order to produce worthwhile moves. We have made no attempt to deal with how humans get their optimism functions, but merely noted the difficulties that keep the notion of "extra move implies threat" from being a general notion. As in many other domains, the acquisition of limited amounts of very important information is not sufficient to completely conquer the domain.

Finally, it should be remembered how well the B* algorithm scales up with increases in power. The present work was done on a piece of hardware that can in 3 minutes do a 9-ply brute-force search, or a B* search with 3-ply probe searches. This confrontation is still minimally in favor of brute-force. However, if one were to scale backward to 2-ply probe searches versus an 8-ply brute-force search, the latter would win easily. So it appears from both analysis and experimentation that doing deeper probe searches is more beneficial than doing deeper brute-force searches. A machine that can do depth-11 alpha-beta searches, could do depth-5 probe searches. Such a B* searcher would almost certainly outperform the time-equivalent brute-force program. Also, in games with a high branching factor B* has great promise. Thus, the future of the B* approach appears bright.

10 Acknowledgements

The efforts of Andy Gruss who supported the second author in massive investigations of flaws in the Hitech hardware is very much appreciated. We wish to thank Gordon Goetsch and Sergey Iskotz who worked on early versions of B*. Jonathan Schaeffer and David Kosbie made valuable comments on the final manuscript. Jay Kadane helped us in understanding some of the decision theoretic aspects of our algorithm. We also would like to thank the (unknown) referees of this paper, as their comments very much improved the understandability of the text. This paper is dedicated to the memory of Allen Newell and Araxie Berliner.

11 Appendix

We here present some games that B* Hitech has played that are considered noteworthy.

```
   April 24, 1993.
White          Black
B* Hitech    Hitech 5.6

  1   e4      c5
  2   Nf3     Nc6
  3   Bb5     g6
  4   O-O     Bg7
  5   c3      Nf6
  6   Re1     a6
  7   Ba4     O-O
  8   d4      c:d4
```

9	c:d4	b5
10	e5!?	b:a4
11	e:f6	B:f6
12	d5	Na7
13	Q:a4	Qc7?
14	Nc3!	B:c3?
15	b:c3	Q:c3
16	Bd2	Qg7
17	Qh4	Re8
18	Bh6	Qh8
19	d6!	Bb7
20	d:e7	B:f3
21	Bg5!	Qb2
22	Bf6	Qb6
23	g:f3	Nb5
24	Re4	h5
25	Re5!	Nd4
26	R:h5	Resigns

May 1, 1993.

White	Black
Hitech 5.6	B* Hitech

1	d4	d5
2	c4	d:c
3	e4	e5
4	Nf3	e:d4
5	B:c4	Nc6
6	O-O	Be6
7	B:e6	f:e6
8	Qb3	Qd7
9	Q:b7	Rb8
10	Qa6	Nf6
11	Nbd2	Bb4
12	Nc4	O-O
13	a3	Be7

All book to here, now black gets into trouble.

14	Re1	Rbe8
15	Bd2	Bc5
16	b4	Bb6
17	Bg5	h6!
18	B:f6	R:f6
19	Rac1	d3!
20	Rcd1	Rd8
21	N:b6	a:b6
22	Re3	R:f3!
23	g:f3	d2
24	Rc3	Ne5!
25	Qe2	b5

Hitech 5.6 almost played 25. Qa7,Rf8!; Q:c7 when N:f6+ and black is much better. Also, 25. f4, Qd4 is strong. Now black seems to have enough play to draw easily.

26	Rc2	Nc4
27	a4	c6
28	f4	Qd6
29	a:b5	c:b5
30	Qg4	Qd3
31	Ra2	Q:e4
32	Ra7	g5
33	Qh5	Rf8
34	Q:h6	Qe1+
35	Kg2	Ne3+
36	f:e3	Qe2+
37	Kg3	Q:e3+
38	Kg2	Draw

38.-Qe2; Kg3,g:f+; Kh3 leads nowhere.

March 25, 1994.

White	Black
Hitech 5.6	B* Hitech

1	d4	Nf6
2	c4	g6
3	Nc3	Bg7
4	e4	d6
5	f3	c6
6	Be3	a6
7	Bd3	Nbd7
8	Nge2	b5
9	a3	Bb7
10	O-O	b:c4
11	B:c4	O-O
12	Qc2	e5
13	Rac1	e:d4
14	N:d4	Qe7
15	Qd2	d5
16	e:d5	N:d5
17	N:d5	c:d5
18	Ba2	Rfc8
19	Bh6	Bh8
20	Rfe1	R:c1
21	R:c1	Re8
22	b4	Qh4
23	Ne2?	R:e2

White has the edge but his 23rd is a terrible blunder based upon not seeing deeply enough.

24	Q:e2	Q:h6
25	Rc7	Be5!
26	R:b7	Qc1+
27	Kf2	Bd4+
28	Kg3	Qg5+
29	Kh3	Qh6+!!

A brilliant move based upon letting the Q have access to d6. The normal Qh5+ does not work. Now if White plays 31. h3 to defend the mate Qh4, then Qd6+ mates.

30	Kg3	g5!!
31	Rb8+	N:b8
32	Qe8+	Kg7
33	Qe5+	B:e5+
34	Kf2	g4
	Resigns	

April 16, 1994
White Black
B* Hitech Hitech 5.6

1	d4	d5
2	c4	d:c4
3	e4	e5
4	Nf3	e:d4
5	B:c4	Nc6
6	O-O	Be6
7	B:e6	f:e6
8	Qb3	Qd7
9	Q:b7	Rb8
10	Qa6	Nf6
11	Nbd2	Bb4
12	Nc4	O-O
13	a3	Be7
14	Re1	Rb3
15	Nfe5	N:e5
16	N:e5	Qd6
17	Qc4	Rb6
18	Nd3	Qd7
19	Bf4	c5
20	b4	c:b4
21	a:b4	Rc8
22	Nc5	B:c5
23	b:c5	Rbc6
24	Ra5	Ra8
25	Rd1	Rd8
26	Qd3	Qe8
27	Bd6	e5
28	R:a7	Rd7
29	Qb3+	Kh8
30	R:d7	N:d7
31	Qd5	Rc8
32	f3	h5
33	c6	Nf6
34	Q:e5	Q:c6
35	Q:d4	Rd8
36	Kh1	Ne8
37	e5	Kg8
38	Qd3	Rd7
39	Qg6	N:d6
40	e:d6	Qb5
41	Re1	Rf7
42	Qe6	Qb7
43	Qe8+	Kh7
44	Rd1	Qb3
45	Qe4+	g6
46	Qd3	Q:d3
47	R:d3	

Here the game was discontinued because B* Hitech does not play endings with such reduced material. The position is an easy win for white.

The next two examples are both quite difficult to understand for a non-master chess player, but show what B* Hitech is capable of at its best. Figure 26 shows the position from a game after 30 moves have been played. It is instructive because it highlights the differences of "view" between the selective search and the brute-force search.

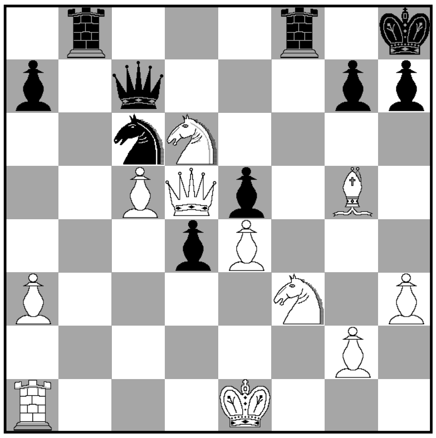

Figure 26: White (can no longer castle) to Play

In this position, white is comfortably ahead with an advantage of two pieces for a rook. All he need do is move his king to g1, where it is safe, and then he can win the game at his leisure. But, being a brute-force program, it does not believe in anything that exists beyond its search horizon. Thus, seeing no danger, it decides it can augment its gains now.

```
   May 5, 1994
White          Black
Hitech 5.6     B* Hitech

 31   N:e5     N:e5
 32   Q:e5     Rb2!
```

A good human player might become suspicious of white's position after this black move. It can be seen that white's king is now trapped in the center, being unable to cross the f-file, and being trapped on the back rank. Yet, the seriousness of this is a matter of deep calculation. Both programs are aware that white's king position is undesirable, but white is winning two pawns to compensate for this. At this point white noticed that 33. Bd2, which prevents the coming check Qa5+, is refuted by 33.-- R:d2!; 34. K:d2,Qa5+; 35. Ke2,Qa6+; and the black pieces will penetrate to mate the white king. This itself is a deep calculation (since there is a tremendous difference between some checks and a mate), and it is only because of the many extension mechanisms that Hitech 5.6 is able to see it at this point. Having realized this, it now acquiesces in loss of some material, rather than play Bd2 which loses immediately. It should be noted that B* Hitech understood it had a very good position after Rb2, but had not yet investigated how to meet 33. Bd2. After the game, it found this refutation in four nodes. However, the most impressive thing about this example is that when the colors are reversed, and B* Hitech is given the task of finding the 31st move for white, it also chooses 31. N:e5 as the selected move, BUT after 36 nodes of verification finds the refutation, and plays 31. Bd2 instead.

```
 33   Q:d4     Qa5+
 34   Bd2      R:d2
 35   Q:d2     Rf1+
 36   K:f1     Q:d2
Although the material is
near even, black has a win.
```

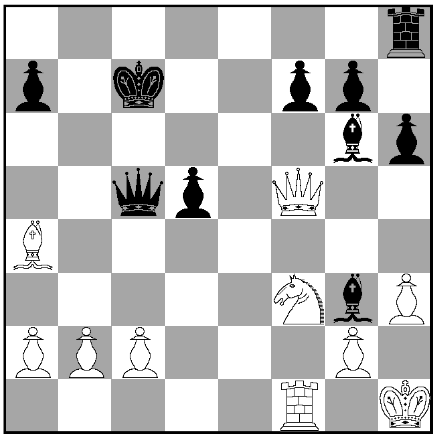

Figure 27: White to Play

In the position of Figure 27, B* Hitech is playing white and it has a certain advantage because there is insufficient shelter for the Black King. Hitech 5.6 as black is expecting 28. Qd7+ and has just found that the intended reply Kb6, is not so good because of 29. Nd4 when black cannot play Q:d4 because of 30. Qc6+,Ka5; 31. c3 and the threats to the queen and b4+ win. So it intended to meet 28. Qd7+ by Kb8 with a marked inferiority, since Bc6 wins a pawn. However, B* Hitech has seen much further. It plays 28.

Qg4!! relying on the fact that a move of the attacked bishop on g3 to (say) d6 would allow 29. Nh4! with the subsequent removal of the bishop at g6 and penetration of the white rook to f7 with fatal effect. If this is met by 29.-- Rf8; then 30. Rf3 allows the rook to penetrate on the Q-Side with devastating effect. If the bishop goes to f2, then Ne5 has the same effect. In turn, Hitech 5.6 sees these things too after white's 28th move has been made. It then plays 28.-- Qc4 which both programs judge best.

May 15, 1994

	White	Black
	B* Hitech	Hitech 5.6
28	Qg4!!	Qc4
29	Q:g3+	Kb7
30	Re1!	Q:a4
31	Re7+	Ka8!
32	Qc7	Qa6

If 32.--Rb8, 33. Ne5 and the threat of Nc6 is fatal.

33	Nd4!	Rc8
34	Nb5!	R:c7

If black plays a waiting move such as B:c2, then white plays 35. Qd7! forcing Kb8 (Be4,Re8! wins); 36. Q:d5,f6; 37. Kh2! and black is helpless against the many threats to his king.

35	N:c7+	Kb7
36	N:a6+	K:a6
37	c3	

Now black must lose either the d-pawn or one of the K-side pawns after which he is hopelessly lost. A masterful exploitation of a weak king position.

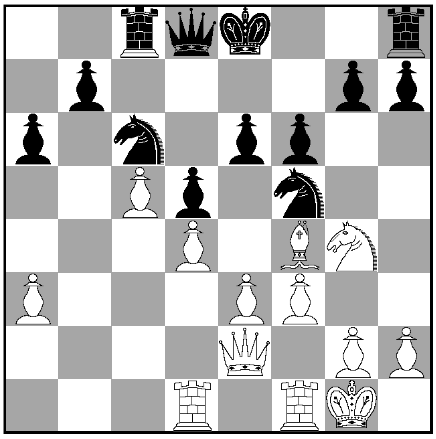

Figure 28: White to Play

Finally, we have an example of truly great play that would do justice to any Grandmaster. The position of Figure 28 looks relatively even. White may have some chances with e4, but it is hard to see how to make them work out. Further, black threatens to play g5 and/or h5 which would embarrass the white pieces on the king's side. We are using this example to show how much further ahead B* Hitech can think than Brute-Force Hitech. We show the variations that each was relying on when it made the move it played. The values at the right are in the usual style where the value of a pawn = 128, and plus is good for white. The game continued:

June 2, 1994

	White	Black		
	B* Hitech	Hitech 5.6	Principal Variation	Value
23	Qb2!		Qb2 h5 e4 Nf:d4 e:d5 e:d5 Rfe1+ Kf8 Ne3	-3
		Qd7	Qd7 Rfe1 h5 Nf2 Ra8 e4 d:e4 R:e4 O-O	32
24	e4!!		e4 Nfe7 Bd6 Na5 Rb1 Nc4	61

			if e4 d:e4 d5 e:d5 f:e4 Nfe7 e:d5 Q:g4 d:c6 R:c6 White thinks d:e4 is not a good move.	121
		d:e4	d:e4 f:e4 Nf:d4 Bd6 e5 h3 h5 Ne3 Ne7 Nd5 N:d5 e:d5 h4 But black thinks it gives him the advantage.	-38
25	d5!!		d5 e:f3 d:e6 Qe7 Bd6 f2+ Q:f2 N:d6	76
		e:d5	e:d5 f:e4 Nfe7 Ne3 Na5 N:d5 N:d5 R:d5 Qg4 Now black agrees it is good for white.	75
26	f:e4		f:e4 Nfe7 e:d5 Q:g4 d:c6 R:c6 Bd6 Qg5 White likes the position even better.	121
		Nfe7	Nfe7 Ne3 Nd8 N:d5 N:d5 R:d5 Qa4 e5 O-O Qa2 Kh8 Be3 They disagree on the best follow-up but both think white has made further gains.	94
27	e:d5		e:d5 N:d5 Rfe1+ Kf7 Bd6 Q:g4 Q:b7+ Nde7 B:e7 N:d5 appears better than Q:g4 (liked earlier).	90
		N:d5	N:d5 Rfe1+ Kf7 Bd6 b5 Nf2 Nc7 Qb3+ Ne6 Bf8 Qc7 B:g7 K:g7 Q:e6 Black produces a line with many extensions and realizes he is in trouble.	125
28	Ne3		Ne3 Nce7 N:d5 N:d5 Rfe1+ Kf7 Bd6 Nc7 Re7+ Q:e7 B:e7 K:e7 Qe2+ Kf8 Rd7 Re8 Qd2 Ne6 Qd6+ Kg8 R:b7 The price has gone up again.	271
		Nce7	Nce7 N:d5 N:d5 Bd6 b5 Qb3 Ne7 Rfe1 Rd8 B:e7 Q:d1. Too late black realizes he is lost.	476
29	N:d5		N:d5 N:d5 Qe2+ Kf7 Qh5+	403
		N:d5		
30	Qe2+		Qe2+ Kf7 Qh5+ Kf8 R:d5 Qe6	411
		Kf8	Kf8 Bd6+ Kg8 Qc4 Qf7 Q:d5 Ra8	421
31	Qc4		Qc4 Ne3 B:e3 Qc6 Bf4	422
		Rd8	Rd8 Rd4 b5 c:b6ep Qb7 Bc7 Rd7 Qd3 In a hopeless position Hitech has been programmed to make the last move that "looked reasonable". It realizes this is not best, and the value is based upon a 6-ply search when a 9-ply search showed Re8 best, but it hopes for a faulty reply.	153
32	Bd6+	Kg8		
33	Q:d5+	Qf7		
34	Rfe1	Q:d5		

35 R:d5 **White has an easy win. It is interesting to see how the brute-force program does not understand its problems until it is well into the situation and can't get out.**

References

[1] Anantharaman, T., Campbell, M., and Hsu, F.
Singular Extensions: Adding Selectivity to Brute Force Searching.
In *AAAI Spring Symposium on Computer Game Playing*. March, 1988.

[2] Anantharaman, Thomas S.
A Statistical Study of Selective Min-Max Search in Computer Chess.
PhD thesis, Carnegie-Mellon University, 1990.

[3] Berliner, H. J.
Some Necessary Conditions for a Master Chess Program.
In *Third International Joint Conference on Artificial Intelligence*, pages 77-85. IJCAI, 1973.

[4] Berliner, H. J.
Chess as Problem Solving: The Development of a Tactics Analyzer.
PhD thesis, Carnegie-Mellon University, 1974.

[5] Berliner, H.
The B* Tree Search Algorithm: A Best-First Proof Procedure.
Artificial Intelligence 12(1), 1979.

[6] Berliner, H. and Ebeling, C.
The SUPREM Architecture: A New Intelligent Paradigm.
Artificial Intelligence 28(1), February, 1986.

[7] Berliner, H. J.
Some Innovations Introduced by Hitech.
Journal of the International Computer Chess Association 10(3), September, 1987.

[8] Berliner, H. J., Goetsch, G., Campbell, M., and Ebeling, C.,
Measuring the Performance Potential of Chess Programs.
In *Advances in Computer Chess 5*. Elsevier Science Publishing Co., 1989.

[9] Berliner, H. J.
Producing Behavior in a Searching Program.
In R. F. Rashid (editor), *CMU Computer Science: A 25th Anniversary Commemorative*, pages 311-344. Addison-Wesley Publishing Co., 1991.

[10] Botvinnik, M. M.
Three Positions.
Journal of the International Computer Chess Association 16(2), June, 1993.

[11] Campbell,Murray.
The Graph-History Interaction: On Ignoring Position History.
In *Proceedings of the ACM National Conference*, pages 278-280. ACM, 1985.

[12] Condon, J. H., and Thompson, K.
Belle Chess Hardware.
In M. R. B. Clarke (editor), *Advances in Computer Chess 3*. Pergamon Press, 1982.

[13] De Groot, A. D.
Thought and Choice in Chess.
Mouton & Co., 1965.

[14] Ebeling, C.
All the Right Moves -- A VLSI Architecture for Chess.
MIT Press, 1986.

[15] Feldmann, R., Mysliwietz, P., and Monien, B.
Game-Tree Search on a Massively Parallel System.
In v. d. Herik, Herschberg, and Uiterwijk (editor), *Advances in Computer Chess 7*, pages 203-219. University of Limburg, Maastricht, Netherlands, 1993.

[16] Harris, L.
The Bandwidth Heuristic Search.
In *Proceedings of the 3rd International Conference on Artificial Intelligence*, pages 23-29. 1973.

[17] Hort, V. and Jansa, V.
The Best Move.
RHM Press, New York, NY, 1980.

[18] Hsu, F-H.
Large Scale Parallelization of Alpha-Beta Search: An Algorithmic and Architectural Study with Computer Chess.
PhD thesis, Carnegie-Mellon University, 1990.

[19] Hyatt, R. M., Gower, A. E., and Nelson, H. L.
Cray Blitz.
In D. F. Beal (editor), *Advances in Computer Chess 4*. Pergamon Press, 1986.

[20] Kozdrowicki, E. W.
An Adaptive Tree Pruning System: A Language for Programming Realistic Tree Searches.
In *Proceedings of the ACM National Conference*. 1968.

[21] Kuszmaul, B. C.
The StarTech Massively-Parallel Chess Program.
The Journal of the International Computer Chess Association 18(1), March, 1995.

[22] Marsland, T. A., Olafsson, M., and Schaeffer, J.
Multiprocessor Tree-Search Experiments.
In D. F. Beal (editor), *Advances in Computer Chess 4*. Pergamon Press, 1986.

[23] McAllester, D. A.
Conspiracy Numbers for Min-Max Search.
Artificial Intelligence 35(3):287-310, 1988.

[24] Newell, A., Simon, H., and Shaw, C.
Chess Playing Programs and the Problem of Complexity.
In E.A. Feigenbaum and J. Feldman (editor), *Computers and Thought*. McGraw-Hill, 1963.

[25] Palay, A. J.
The B* Tree Search Algorithm -- New Results.
Artificial Intelligence 19(2), 1982.

[26] Palay, A. J.
Searching with Probabilities.
Pitman Research Notes in Artificial Intelligence, 1984.

[27] Pearl Judea.
Heuristics: Intelligent Search Strategies for Computer Problem Solving.
Addison-Wesley Publishing Co., 1984.

[28] Reinfeld, F.
Win At Chess.
Dover Books, 1958.

[29] Russell, Stuart, and Wefald, Eric.
Multi-Level Decision-Theoretic Search.
In *AAAI Spring Symposium on Computer Game Playing*, pages 3-7. IJCAI, 1988.

[30] Slagle, J. R. and Dixon, J. K.
Experiments with some Programs that Search Game Trees.
Journal of the ACM 16(2), 1969.

[31] Slate, D. J., & Atkin, L. R.
CHESS 4.5 -- The Northwestern University Chess Program.
In P. Frey (editor), *Chess Skill in Man and Machine*. Springer Verlag, 1977.

[32] Slater, E.
Statistics for the Chess Computer and the Factor of Mobility.
In *Symposium on Information Theory*, pages 150-152. Ministry of Supply, London, 1950.

Table of Contents

List of Figures

List of Tables