Suchet, so werdet ihr finden

Eine Reise in die Gedankenwelt eines Schachprogramms

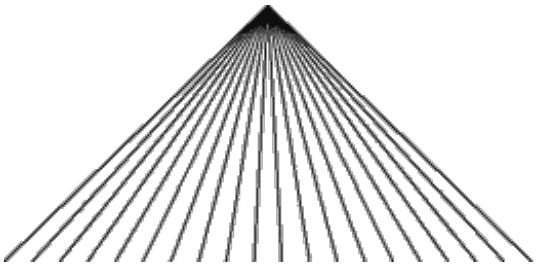
Schachprogramme durchsuchen in unglaublicher Geschwindigkeit möglichst viele Varianten auf dem Brett, um sich dann für die vielversprechendste zu entscheiden. Einige Programme schaffen dies sehr gut, andere verschwenden unnötige Zeit. Was macht diesen Unterschied aus? Wieso tappen auch Profi-Programme manchmal im Dunkeln? Im folgenden Artikel erklärt Stefan Zipproth Tricks und Fallstricke moderner Schuchverfahren.


Stefan Zipproth

Die Schachlegende Bobby Fischer hat einmal behauptet, er rechne überhaupt nicht voraus, denn er gewinne auch so. Schachprogramme sind mit einer solchen Weitsicht – zumindest bis jetzt – nicht gesegnet. Sie sind zwar durchaus fähig, eine Schachstellung auch ohne Vorausrechnung ("Suche") einzuschätzen, aber diese Bewertung übertrifft nach heutigem Stand der Technik nicht einmal das Schachverständnis eines mittleren Vereinsspielers.

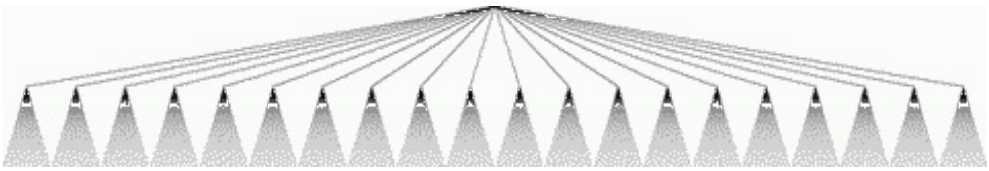
Vor allem in taktischen Stellungen sind auch modernste Programme darauf angewiesen, ihr mangelndes Schachverständnis durch ein stupides Ausprobieren möglichst vieler Varianten auszugleichen. Doch ganz so einfach ist es nicht. Selbst das wesentlich anspruchslosere Spiel "Vier Gewinnt" (vgl. CSS 2/03) konnte nur mit erheblichem Aufwand so weit durchgerechnet werden, dass es nun als gelöst betrachtet werden darf.

Das Schachspiel jedoch ist ungleich komplexer. Hier gibt es in jeder Stellung wesentlich mehr Wahlmöglichkeiten als bei "Vier Gewinnt". Nehmen wir an, es gäbe im Schach durchschnittlich 20 Züge in einer Stellung; wenn also unser Programm in einer Stellung einen guten Zug finden soll, so muss es etwa 20 Züge ausprobieren:



Zum Beispiel gibt es in der Grundstellung zufällig genau 20 Züge (16 Bauern- und 4 Springerzüge). Das Bewertungsverfahren unseres Schachprogramms würde also jede der 20 resultierenden Stellungen untersuchen und dann den Zug ausspielen, der die höchste Bewertung liefert.

Damit ist es aber nicht getan, denn auch die gegnerischen Antwortzüge sind wichtig, um sicherzustellen, dass der vermeintlich beste Zug nicht einfach eine Figur einstellt. Denn schon dies übersteigt das Schachverständnis gängiger Stellungsbewertungs-Verfahren. Für jeden der 20 Zugkandidaten, unter denen wir den besten Zug ausfindig machen wollen, müssen also wiederum (ungefähr) 20 gegnerische Züge durchprobiert werden:



–

Suchet, so werdet ihr finden

20.000 Jahre für einen Zug

2 Halbzüge Vorausrechnung reichen aber noch nicht einmal für Amateurniveau. Damit ein Computerprogramm zum Beispiel eine drohende Bauerngabel erkennen kann, muss es schon 4 Halbzüge vorausrechnen:

Wenn Weiß am Zug die Drohung d5-d4 nicht pariert, dann schlägt Schwarz im 4. Halbzug den weißen Springer oder Läufer. Für diese 4 Halbzüge sind schon etwa 20 x 20 x 20 x 20 = 160.000 Varianten möglich! Ein halbwegs guter Vereinsspieler braucht natürlich keine 160.000 Varianten durchzurechnen, um zu sehen, dass eine Bauerngabel droht. Ein Computer aber schon, weil das Wissen ("Bei einer drohenden Bauerngabel ist es wahrscheinlich gut, eine der Figuren wegzuziehen, die angegriffen sein werden, mit folgenden Ausnahmen ...") einfach viel zu kompliziert ist, um in einer Bewertungsfunktion eingebaut zu werden. Schließlich muss es immer stimmen, sonst kann unser Programm niemals Großmeister-Niveau erreichen, weil Schach voll von Regel-Ausnahmen ist.

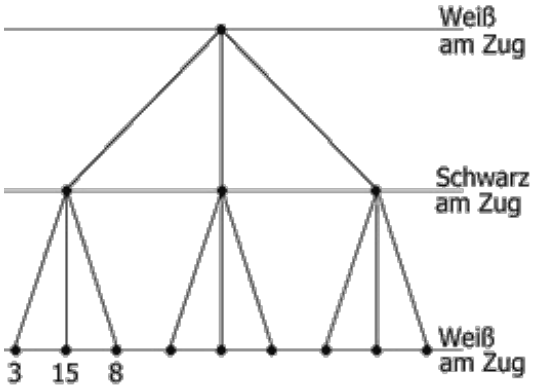
Um GM-Niveau zu erreichen, ist eine wesentlich »tiefere« Vorausrechnung notwendig – etwa 16 Halbzüge in langen Partien. Nach Adam Riese macht das etwa 600.000.000.000.000. 000.000 (600 Millionen Millionen Millionen) mögliche Varianten. Ein Computer mit 1000 (!) Gigahertz bräuchte ungefähr 20.000 Jahre, um sie alle durchzurechnen. Kasparov würde sich sicherlich weigern, auf solche Turnierbedingungen einzugehen.

Aus diesem Grund war man vor dem Jahre 1966 davon überzeugt, dass es niemals möglich sein würde, ein auch nur annähernd vernünftig spielendes Schachprogramm zu schreiben. (Nebenbei: Diesen Forschungsstand haben wir momentan im Spiel »Go«, wo pro Stellung über 300 Züge möglich sind und selbst schwache Vereinsspieler die besten Programme ohne Anstrengung schlagen.)

Was ist im Jahre 1966 so Bedeutsames passiert? Damals gab es schon seit 15 Jahren ein von Alan Turing (dem bekannten Erfinder des Intelligenztests für Computer) entwickeltes Verfahren, um zumindest 2 Halbzüge tief zu suchen. Die beiden Forscher Kotok und McCarthy entdeckten dann, dass es erstaunlicherweise möglich ist, den größten Teil der riesigen Variantenvielfalt einzusparen! Und das, ohne dabei irgendwelche Varianten zu übersehen oder Fehler zu machen.

Wohlgemerkt, wir sprechen hier nicht von der menschlichen Fähigkeit, Züge intuitiv als unsinnig zu erkennen und auf diese Weise einzusparen – davon sind wir auch heute noch weit entfernt. Es geht bei dem von Kotok und McCarthy entdeckten "Alpha-Beta-Verfahren" um eine Möglichkeit, Varianten aus ganz logischen Gründen und ohne jedes Schachwissen einsparen zu können.

Wie ist so etwas möglich? Betrachten wir zunächst an einem einfachen Beispiel, wie ein Computer vorgehen muss, um in einer Stellung den besten Zug zu finden. Der Einfachheit halber sei angenommen, dass es pro Stellung nur 3 Züge gäbe und wir nur 2 Halbzüge tief suchen. Das Programm soll also immer 2 beliebige Halbzüge in seinem Speicher ausführen und dann die Stellung bewerten. Danach wird der letzte Zug zurückgenommen und es werden andere Zugmöglichkeiten probiert, solange bis sämtliche Varianten erfasst sind. Dieser Vorgang lässt sich durch einen so genannten "Suchbaum" folgendermaßen darstellen:



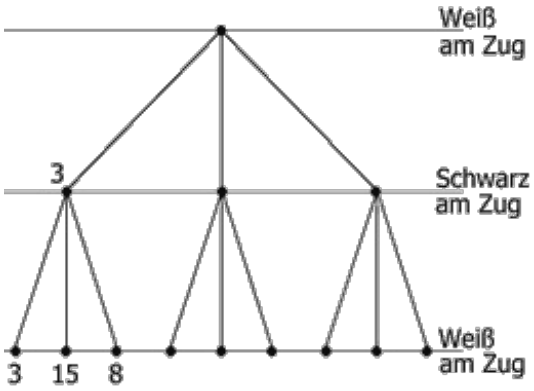
Im obigen Suchbaum hat Weiß einen Zug gespielt und alle drei möglichen Antwortzüge von Schwarz wurden ausprobiert. Jede der drei resultierenden Stellungen wurde mit einer Bewertungsfunktion untersucht (die Materialbilanz und einfache strategische Merkmale bewertet).

Da die Bewertungen 3, 15 und 8 aus weißer Sicht sind, gehen wir davon aus, dass unser Gegner "Schwarz" denjenigen Zug spielen wird, der zur kleinsten Bewertung ("3") führt – denn dies ist aus schwarzer Sicht die beste Variante.

Minimax & Co

Schwarz versucht also immer, die Bewertung zu minimieren, und Weiß versucht, sie zu maximieren; deswegen nennt man dieses Verfahren auch Minimax. Natürlich könnte unser Gegner auch andere Varianten wählen, als wir erwarten, da er vermutlich eine andere Bewertungsfunktion hat oder gar ein Mensch ist. Das ist das grundlegende Risiko bei Suchverfahren, denn wir haben keine andere Möglichkeit, als davon auszugehen, dass unsere Bewertungsfunktion auch die Züge des Gegners mit hinreichender Genauigkeit abschätzen kann. Wählt der Gegner einen anderen Zug, als wir erwarten, so können wir nur hoffen, dass wir im Recht sind und der gegnerische Zug tatsächlich Chancen vergibt.

Im Beispiel gehen wir also davon aus, dass Schwarz denjenigen Zug spielen wird, der zur Bewertung "3" führen wird. Daraus ergibt sich, dass wir die Stellung (auch "Knoten" genannt), in der Schwarz am Zug ist, mit "3" bewerten können:



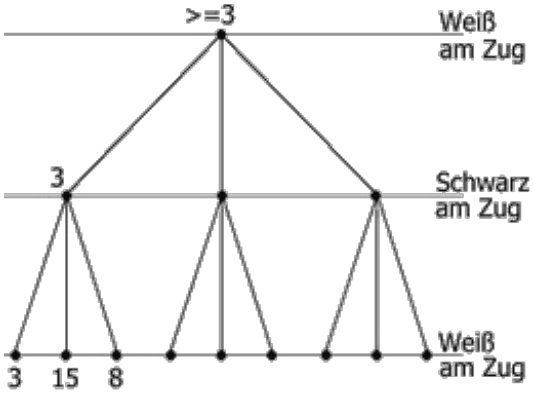
Stellen wir uns vor, dieses Schema so lange fortzusetzen, bis alle Stellungen bewertet sind. Bis jetzt haben wir nur für eine der drei Stellungen mit Schwarz am Zug eine Bewertung, nämlich "3". Wenn wir zum Beispiel annehmen, die beiden anderen Stellungen würden mit "2" und "14" bewertet, so würden wir (Weiß) denjenigen Zug ausspielen, der zur höchsten Bewertung ("14") führt.

1950 entdeckte Claude Shannon das oben beschriebene Minimax-Verfahren für die Schachprogrammierung. Es dauerte 16 Jahre, bis jemand auf die Idee kam, dass es gar nicht notwendig ist, im obigen Beispiel alle 9 Stellungen zu bewerten. Obwohl es, wenn man weiß, wie es geht, gar nicht so schwierig ist. Kommen Sie auf die richtige Idee?

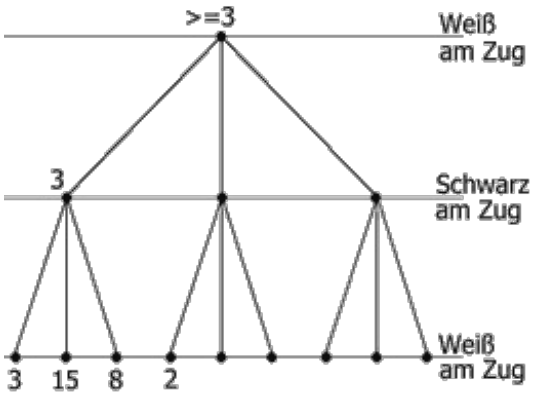
Wie können wir im Voraus wissen, welcher Zug für Weiß der günstigste ist, ohne überhaupt alle 9 Stellungen (auch "Endknoten" genannt) zu bewerten? Und ohne dabei Gefahr zu laufen, einen wichtigen schwarzen Antwortzug zu übersehen?

Überlegen wir einmal, was wir alles anhand des obigen Suchbaums wissen. Es steht fest, dass Weiß einen Zug spielen kann, der zu einer Stellung führt, die wir mit »3« bewerten. Das bedeutet, egal was bei den anderen beiden weißen Alternativzügen geschieht, wir wissen auf jeden Fall, dass wir mindestens die Bewertung "3" erreichen können – weil wir sowieso »maximieren«, das heißt, selbst wenn die beiden anderen Züge eine schlechtere Bewertung haben, werden wir auf jeden Fall den mit der besten Bewertung auswählen.

Wir können also unseren Suchbaum um eine entscheidende Information erweitern:



Nun probieren wir den zweiten weißen Zug aus, der zum mittleren "Teilbaum" führt, und natürlich einen darauf folgenden schwarzen Antwortzug. Wie gehabt, bewerten wir dann die sich daraus ergebende Stellung mit unserem Bewertungsverfahren:

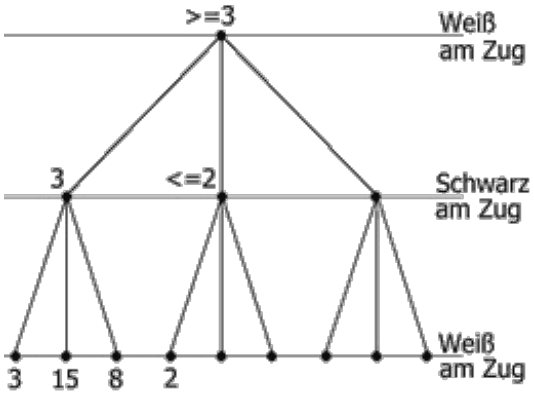


Wir haben also (rein zufällig) einen schwarzen Antwortzug ausgespielt, der auf unseren Zug hin eine Stellung mit Bewertung "2" erreicht. Schwarz kann also auf unseren (dem mittleren Teilbaum entsprechenden) Zug hin eine Stellung erreichen, die mindestens die Bewertung "2" zur Folge hat. Eine schlechtere Bewertung ist gar nicht möglich, da Schwarz sicherlich keine schlechtere Variante wählen wird, wenn er die Möglichkeit hat, Bewertung "2" zu erreichen.

Suchet, so werdet ihr finden

Eine folgenschwere Idee

Wir können also unseren Suchbaum schon jetzt um eine weitere Information ergänzen. Schwarz kann mindestens Bewertung "2" erreichen – da wir die Bewertungen aber aus weißer Sicht betrachten, bedeutet dieses "mindestens", aus weißer Sicht ein höchstens (<=2).

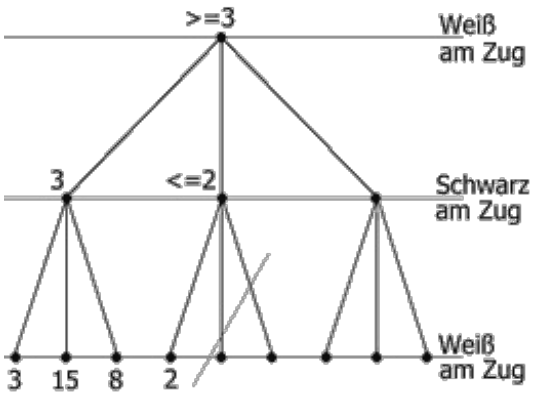


Nun der entscheidende Gedankenschritt. Denken Sie sich nichts, wenn Sie ihn in einer Stunde nicht mehr nachvollziehen können – immerhin hat es 16 Jahre gedauert, bis überhaupt jemand auf diese Idee kam.

Aufgrund des linken Teilbaums (unserem ersten ausprobierten Zug) wissen wir, dass Weiß mindestens die Bewertung "3" erreichen kann. Nun haben wir im mittleren Teilbaum (unserem zweiten ausprobierten Zug) einen gegnerischen Zug gefunden, der für Schwarz mindestens Bewertung "2" herausholt. Wozu sollen wir also die beiden anderen schwarzen Züge ausspielen? Schließlich wissen wir, dass der mittlere Teilbaum unattraktiv ist, denn dort können wir höchstens die Bewertung "2" erreichen – aber durch den linken Teilbaum haben wir schon mindestens "3".

Ein Beispiel: Nehmen wir an, wir würden trotzdem den nächsten schwarzen Zug ausspielen, und dieser würde mit "5" bewertet. Dies würde nichts ändern, weil Schwarz minimiert, und somit in jedem Fall den anderen Zug mit Bewertung "2" wählen würde. Wenn hingegen der nächste schwarze Zug mit "1" bewertet werden würde, so würde dies gleichfalls nichts ändern, denn ob nun "1" oder "2", Weiß wird in jedem Fall den linken Teilbaum wählen, wo er die Bewertung "3" erreichen kann.

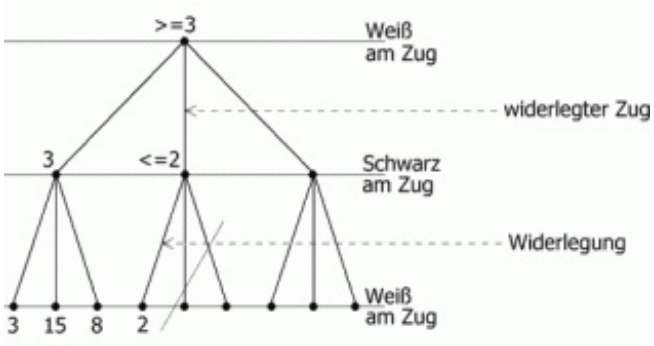
Folglich brauchen wir den mittleren Teilbaum gar nicht weiter zu untersuchen. Wir können einen so genannten "Cut" durchführen und uns damit den weiteren Aufwand für die Bewertung der beiden anderen schwarzen Züge ersparen, ohne dabei etwas übersehen zu können. Welche Bewertung sie auch immer haben würden – sogar ein Matt für Weiß oder Matt für Schwarz würde nichts ändern.



Genau das ist das schon erwähnte Alpha-Beta-Verfahren. Alpha-Beta deswegen, weil man Mindestwerte (wie im obigen Beispiel >=3) "Alpha-Schranken" oder "untere Schranken" nennt, und Maximalwerte (wie oben <=2) "Beta-Schranken" oder "obere Schranken". Einen Cut, der aufgrund einer Alpha-Schranke möglich ist (wie oben wegen Alpha=3), nennt man deswegen auch Alpha-Cut und einen, der sich aufgrund einer Beta-Schranke ergibt, Beta-Cut.

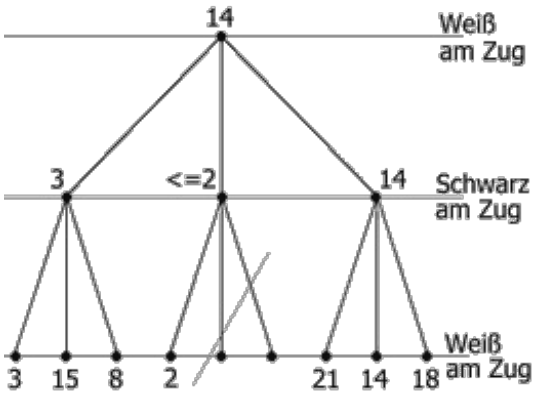
Der dazwischen liegende Bewertungs-Bereich wird Suchfenster genannt – also alle Bewertungen, die zwischen Alpha und Beta liegen. Sobald das Programm auf eine Bewertung stößt, die besser ist und somit außerhalb dieses Suchfensters liegt, kann ein Cut durchgeführt werden.

Alpha-Cuts und Beta-Cuts sind im Grunde dasselbe, nur einmal aus Sicht unseres Programms und im anderen Fall aus Sicht des Gegners. Ein Schachprogrammierer bezeichnet normalerweise alle Cuts als Beta-Cuts, weil er die Bewertungen nicht immer aus weißer Sicht betrachtet, sondern immer aus Sicht der gerade am Zuge befindlichen Partei.



Schachlich gesprochen ist der schwarze Zug, der mit "2" bewertet wurde, nichts anderes als eine Widerlegung des vorausgehenden weißen Zuges. Weil dieser Zug bereits den weißen Zug widerlegt, ist es unnötig, noch bessere Widerlegungen zu finden (denn der erste ausprobierte weiße Zug im linken Teilbaum war sowieso besser). Wenn wir beispielsweise herausgefunden haben, dass unser Zug widerlegt wurde, weil er einen Läufer einstellt, dann ist es nicht mehr notwendig zu überprüfen, ob vielleicht sogar die Dame verloren geht.

Nachdem also im mittleren Teilbaum die Bewertung »2« gefunden wurde, kann sofort mit dem rechten Teilbaum fortgesetzt werden. Auch hier gilt: Sobald eine Bewertung gefunden wird, die kleiner als 3 ist, brauchen die restlichen Züge nicht mehr betrachtet zu werden. Wenn natürlich selbst der beste schwarze Zug nur zu einer Stellung führt, die beispielsweise mit "14" bewertet wird, so wäre der rechte Teilbaum die neue beste Variante für Weiß, womit unser Programm letzten Endes den rechten Zug ausspielen würde:



Das Schöne am Alpha-Beta-Verfahren ist, dass man unglaublichen Aufwand betreiben kann, um möglichst viel aus ihm herauszuholen. Je bessere Voraussetzungen man schafft, desto mehr Cuts entstehen. Letzten Endes lässt sich mithilfe dieses Verfahrens die Anzahl der Züge, die man pro Stellung ansehen muss, auf etwa ein Viertel reduzieren.

Fachleute sprechen hier vom Verzweigungsfaktor. Im Minimax-Verfahren muss man in jeder Stellung sämtliche Züge ausprobieren, also wäre hier der Verzweigungsfaktor des Suchbaumes gleich 20. Das Alpha-Beta-Verfahren senkt den Verzweigungsfaktor auf etwa 5. Das klingt zwar zunächst recht unspektakulär, hat aber in Wahrheit drastische Auswirkungen. Denn wenn wir weit oben im Suchbaum weniger Züge untersuchen müssen, so sparen wir uns ja automatisch alle darunter liegenden Teilbäume.

Ein kleines Rechenbeispiel: Wenn wir vorher 20.000 Jahre gebraucht haben, um Tiefe 16 zu erreichen, so sind es nun nur noch 20.000 Jahre mal 1/4 x 1/4 x ... x 1/4, also 16-mal der Faktor 1/4. Aus 20.000 Jahren werden somit knapp 3 Minuten!

Zugsortierung

Selbst sehr geringe Verbesserungen des Verzweigungsfaktors haben also große Auswirkungen auf die notwendige Anzahl von Bewertungen. Die wichtigste Optimierungs-Methode überhaupt ist es, die vielversprechendsten Züge zuerst zu untersuchen. Nehmen wir an, im obigen Diagramm wäre der Zug im mittleren Teilbaum, der den Cut verursacht hat, nicht als Erster, sondern als Dritter ausgespielt worden. Dann hätten wir gar nichts gewonnen.

Nur wie erreicht man, dass die besten Züge von vornherein zuerst ausgespielt werden? Schließlich kann unser Programm nicht hellsehen – wenn von vornherein klar wäre, welches die besten Züge sind, dann bräuchten wir erst gar nicht zu suchen.

Der Trick lautet "Internal Iterative Deepening". Wenn wir in einer bestimmten Stellung herausfinden wollen, welches wohl der vielversprechendste Zug ist, dann führen wir eine Vor-Suche durch, und zwar zur Zeitersparnis mit einer um 2 Schritte verkürzten Suchtiefe. Eine solchermaßen verkürzte Suche hat den Vorteil, dass sie im Vergleich zur eigentlichen Suche sehr wenig Zeit erfordert. Auf diese Weise gelangen wir zu einem Zug, von dem zwar nicht sicher ist, dass er der beste ist, aber zumindest wahrscheinlich.

Die Vor-Suche dient nur dazu, in der Haupt-Suche einen guten Zug als Ersten ausspielen zu können, also die Reihenfolge der Züge zu beeinflussen. Obwohl natürlich für die Vorsuche zusätzliche zu bewertende Stellungen anfallen, kommen wir unter dem Strich besser weg, weil der Aufwand dafür eher gering ausfällt. Es ist viel wichtiger, in der Hauptsuche einen guten Zug als Ersten auszuspielen, um das Alpha-Beta-Verfahren auszureizen.

Es ist interessant sich vorzustellen, welche Auswirkungen diese Vorsuchen auf den Suchbaum haben – er wird ziemlich undurchschaubar. Denn innerhalb der Vorsuchen können ja ohne weiteres wieder Vorsuchen stattfinden. Jeder der Knoten im obigen Beispiel wird zunächst mit einer Vorsuche behandelt, um anschließend in der eigentlichen Suche einen günstigen Zug als Ersten ausspielen zu können.

Auch in den Startstellungen selbst – die jeweils am Anfang jeder Suche vorliegen – wird dies getan, obwohl es programmiertechnisch etwas anders gelöst wird. Denn die Vorsuchen in der Startstellung sind der Grund dafür, dass bei praktisch allen Schachprogrammen die Tiefe schrittweise nach oben zählt – denn es wäre ohne weiteres möglich, sofort auf der angestrebten Tiefe anzufangen zu suchen. Nur vergibt man damit unter dem Strich viel Zeit, weil man nicht weiß, welcher Zug als Erster günstig ist.

Richtig effektiv werden die Vorsuchen aber erst durch gleichzeitige Verwendung von Hashtabellen (s. CSS 3/02, S.38ff; 4/01, S.34) Durch die Vorsuchen werden viele Stellungen mehrmals behandelt (was auch durch Zugumstellungen geschehen kann) und wenn wir uns in einer Art Gedächtnis merken, welche Züge in früheren Suchen zu einem Cut geführt haben (das heißt Widerlegungen waren, siehe Diagramm nächste Seite), so können wir uns viele der Vorsuchen sparen, weil in diesem Gedächtnis bereits ein günstiger Zug vermerkt ist, den wir dann als Ersten ausprobieren.

Null-Züge

Im Rechenbeispiel erreichen wir also nun Tiefe 16 in knapp 3 Minuten. Bei diesen 3 Minuten sind wir aber von einem, zurzeit völlig illusorischen, 1000-GHz-Rechner ausgegangen. Irgendetwas muss also noch fehlen, um unser Programm auf Großmeister-Stärke zu bringen. Natürlich sind die hier aufgeführten Zahlen nur als grobe Faustregeln zu verstehen, denn in einem tatsächlichen Schachprogramm greifen so viele optimierende Verfahren ineinander, dass es unmöglich ist, die Auswirkungen einzelner Module unabhängig von anderen zu betrachten.

Einerseits haben wir den Effekt der Hashtabellen sowie der verfeinerten Zugsortierung unbeachtet gelassen, andererseits gibt es noch ein weiteres, entscheidendes Puzzle-Stück: das »Nullmove-Pruning«. Als Pruning bezeichnet man eine Methode, den Suchbaum in gewissen Positionen zu beschneiden, um sich auf diese Weise die Bewertung des darunter liegenden Suchbaums zu ersparen. Im Unterschied zu den Beta-Cuts ist hier allerdings nicht 100%ig gewährleistet, dass die Abschneidung wirklich korrekt ist - daher können in seltenen Fällen wichtige Teilbäume dem Pruning-Verfahren zum Opfer fallen.

Das Nullmove-Pruning hat sich als äußerst erfolgreich erwiesen und wird in fast jedem Schachprogramm verwendet, natürlich manchmal intensiver und manchmal weniger intensiv. Interessanterweise bewirkt es eine Erhöhung des Verzweigungsfaktors, aber durch die Tatsache, dass komplette Teilbäume einfach weggelassen werden, wird dieser Nachteil mehr als aufgewogen.
Dabei macht man sich eine besondere Eigenschaft des Schachspiels zunutze: Es lässt sich grundsätzlich sagen, dass es besser ist, zu ziehen, als nicht zu ziehen. Die einzige Ausnahme von dieser Regel sind Zugzwang-Stellungen, die sich aber leicht abfangen lassen, da sie normalerweise nur im späten Endspiel auftreten.

Das Nullmove-Pruning ist ein Schritt in die Richtung, unsinnige Züge von vornherein zu erkennen und den dadurch entstehenden Suchbaum erst gar nicht durchzurechnen. Betrachten wir folgende Stellung:

Unser Programm zieht gerade den Zug g2-g4 in Erwägung, an den ein Mensch wohl nie denken würde. Normalerweise müsste das Programm nun sämtliche Varianten durchrechnen, um dann am Schluss festzustellen, dass g2-g4 einen Bauern einstellt und außerdem die Königsstellung stark schwächt.

Dieser Suchbaum lässt sich aber durch das Nullmove-Pruning komplett einsparen. Wir spielen den Zug g2-g4, lassen dann den Gegner einen »Nullzug« ausspielen – wodurch sich nur das Zugrecht ändert und Weiß ein zweites Mal ziehen darf – und führen dann eine normale Suche für Weiß aus (allerdings mit verkürzter Suchtiefe). Wenn es nun Weiß nicht gelingt, den Vorteil des zweimaligen Ziehens, der ja nicht den Schachregeln entspricht, auszunutzen (indem eine Bewertung größer Alpha gefunden wird), dann kann der erste Zug (g2-g4) nicht gut gewesen sein und wir können uns die eigentliche Suche ersparen.

Dabei unterstellen wir, dass Schwarz nach g2-g4 auf jeden Fall einen Zug hat, der besser ist, als überhaupt nicht zu ziehen. Normalerweise müssten wir alle schwarzen Antwortzüge auf g2-g4 in Erwägung ziehen, so aber können wir sehr schnell feststellen, dass Schwarz sogar g2-g4 widerlegen kann, indem er einfach überhaupt nicht zieht. (*sz*)

Informationen zum Autor:

Stefan Zipproth