Connecting Sciences

H. Jaap van den Herik[1], Aske Plaat[1], Jan Kuipers[2], Jos A.M. Vermaseren[2]
[1]*Tilburg University, Tilburg center for Cognition and Communication, Warandelaan 2, 5037 AB Tilburg, The Netherlands*
[2]*Nikhef Theory Group, Science Park 105, 1098 XG Amsterdam, The Netherlands*
jaapvandenherik@gmail.com

Keywords: Artificial Intelligence; High Energy Physics; Horner's rule; Monte Carlo Tree Search; Go; chess

Abstract: Real progress in science is dependent on novel ideas. In Artificial Intelligence, for a long time (1950-1997) a prevailing question was: can a computer chess program defeat the human World Champion? In 1997 this turned out to be the case by using a supercomputer employing the minimax method with many enhancements, such as opponent modeling. The next question was: can we use similar techniques to outperform the top grandmasters in Go. After some ten years of intensive research the answer was that minimax could not be successfully transferred to the game of Go. In the article, we will review briefly why this transfer failed. Hence, a new method for Go was developed: MCTS (Monte Carlo Tree Search). The results for MCTS in Go are rather promising. At first glance it is quite surprising that MCTS works so well. However, deeper analysis revealed the reasons for this success.

Since the field of AI-research claims to be a fruitful test bed for techniques to be applied in other areas one might look in which areas minimax or MCTS are applicable. A possible and unexpected answer is the application area of solving categories of high energy physics equations. In that area the derivation of the formulas is often performed by the (open source) computer algebra system FORM developed by Jos Vermaseren. The derivation is usually hand guided (human decisions are needed) and becomes difficult when there are many possibilities of which only a few lead to a useful solution. The intriguing question is: can we make the computer select the next step? Our idea is that the next step can be made by using MCTS.

In the article we show the first attempts which prove that this idea is realizable. It implies that games and solving high energy physics equations are connected by MCTS. A first unexpected result is showing the existence of connecting sciences. Equally unexpected is the second result. It is obtained by using MCTS for the improvement of multivariate Horner schemes when converting large formulas to code for numerical evaluation. From the viewpoint of MCTS this is an uncomplicated application and thus it is possible, by varying parameters, to see how MCTS works. It shows that the new ideas can function as cross-fertilization for two, and maybe more, research areas. Hence, the future lies in connecting sciences.

1 Introduction

In 1965, the Soviet mathematician Aleksandr Kronrod called chess the Drosophila Melanogaster of Artificial Intelligence [26]. At that time, chess was a convenient domain that was well suited for experimentation. Moreover, dedicated research programs all over the world created quick progress. In only three decades the dream of beating the human world champion was realized. On May 11, 1997 Garry Kasparov, the highest rated human chess player ever, was defeated by the computer program DEEP BLUE, in a highly publicized six game match in New York.

So, according to some, the AI community lost their Drosophila in 1997, and started looking for a new one. The natural candidate was an even harder game: the oriental game of Go. Go is played on a 19×19 board, see Fig. 1. Its state space is much larger than the chess state space. The number of legal positions reachable from the start of the game of Go is estimated to be $O(10^{171})$ [1], whereas for chess this number is "just" $O(10^{46})$ [14]. If chess is a game of tactics, then Go is a game of strategy. The standard minimax approach that worked so well for chess (and for other games such as checkers, Awari and Othello) did not work well for Go. Therefore, Go was well equipped to become the new Drosophila. For decades, computer Go programs played at the level

Figure 1: A Go board

of weak amateur. After 1997, the research effort for computer Go intensified. Initially, progress was slow, but in 2006, a breakthrough occurred. This breakthrough and some of its consequences, are the topic of this article.

The remainder of this contribution is structured as follows. First, the techniques that worked so well in chess will be discussed briefly. Second, the new search method that caused the breakthrough in playing strength will be described. Third, an example of transfer of this method to the domain of high energy physics will be given. Then, a successful MCTS application to Horner's rule of multivariate polynomials will be shown. We complete the article by a discussion on the future of connecting sciences.

2 The Chess Approach

The heart of a chess program consists of two parts: (1) a heuristic evaluation function, and (2) the minimax search function. The purpose of the heuristic evaluation function is to provide an estimate of how good a position looks, and sometimes of its chances of winning the game [16]. In chess this includes items such as the material balance (capturing a pawn is good, capturing a queen is usually very good), mobility, and king safety. The purpose of the search function is to look ahead: if I play this move, then my opponent would do this, and then I would do that, and ..., etc. By searching deeper than the opponent the computer can find moves that the heuristic evaluation function of the opponent mis-evaluates, and thus find the better move.

Why does this approach fail in Go? Originally, the main reason given was that the search tree is so large. In chess, the opening position has 20 legal moves (the average number of moves is 38 [17, 21]. In Go, this number is 361 (and thereafter it decreases with one per move). However, soon it turned out that an even larger problem was posed by the construction of a good heuristic evaluation function. In chess, material balance, the most important term in the evaluation function, can be calculated efficiently and happens to be a good first heuristic. In Go, so far no good heuristics have been found. The influence of stones and the life and death of groups are generally considered to be important, but calculating these terms is time consuming, and the quality of the resulting evaluation is a mediocre estimator for the chances of winning a game.

Alternatives

Since a full-width look-ahead search is infeasible most early Go programs used the knowledge-based approach: (1) generate a limited number of likely candidate moves, such as corner move, attack/defend groups, connecting moves, and ladders, and (2) search for the best move in this reduced state space [28]. The Go heuristics can be generalized in move patterns, which can be learned from game databases [37, 38]. A second approach was to use neural networks, also with limited success [18].

3 Monte Carlo

In 1993, the mathematician and physicist Bernd Brügmann was intrigued by the use of simulated annealing for solving the traveling salesman problem. If randomized local search could find shortest tours, then maybe it could find good moves in Go. He wrote a 9×9 Go program based on simulated annealing [7]. Crucially, the program did not have a heuristic evaluation function. Instead it played a series of random moves until the end of the game was reached. Then the final position was scored as either win or loss. This was repeated many times, and the result was averaged. So instead of searching a tree, Brügmann's program searched paths, and instead of using the minimax function to back-up the scores, the program took the average of the final scores. The program had no heuristics, except not to fill its own territory. Could this program be expected to play anything but meaningless random moves?

It turned out that it could. Although it certainly did not play great or even good moves, the moves looked better than random. Brügmann concluded that

by just following the rules of the game the average of many thousands of plays yielded better-than-random moves.

At the time, this attempt at connecting the sciences of physics and artificial intelligence appeared to be a curiosity. Indeed, the hand-crafted knowledge-based programs performed significantly better. For the next ten years not much happened with Monte Carlo Go.

Monte Carlo Tree Search

Then, in 2003, Bouzy and Helmstetter reported on further experiments with Monte Carlo playouts, again stressing the advantage of having a program that can play Go moves without the need for a heuristic evaluation function [2, 5]. They tried adding a small 2-level minimax tree on top of the random playouts, but this did not improve the performance. In their conclusion they refer to other works that explored statistical search as an alternative to minimax [23, 33] and concluded: "Moreover, the results of our Monte Carlo programs against knowledge-based programs on 9×9 boards and the ever-increasing power of computers lead us to think that Monte Carlo approaches are worth considering for computer Go in the future."

They were correct.

Three years later a breakthrough occurred with the introduction of MCTS and UCT. Coulom [15] described Monte Carlo evaluations for tree based search, specifying rules for node selection, expansion, playout and backup. Chaslot et al. coined the term Monte Carlo Tree Search or MCTS, in a contribution that received the ICGA best publication award [11]. One year later Kocsis and Szepesvari [24] laid the theoretical foundation for a selection rule that balances exploration and exploitation and that is guaranteed to converge to the minimax value. This selection rule is termed UCT, short for upper confidence bounds for multi-armed bandits [4] applied to trees (see Eqn. (4)). Gelly et al. [20] use UCT in a Go program called MoGo, short for Monte Carlo Go, which was instantly successful. Chaslot et al. [10] also described the application of MCTS in Go, reporting that it outperformed minimax, and mentioned applications beyond Go.

Since 2006 the playing strength of programs improved rapidly to the level of strong amateur/weak master (2-3 dan). The MCTS breakthrough was confirmed when, for the first time, a professional Go player was beaten in a single game. In August 2008 at the 24th Annual Go Congress in Portland, Oregon, MoGo-Titan, running on 800 cores of the Huygens supercomputer in Amsterdam, beat 8P dan professional Kim MyungWan with a 9-stone handicap [13].

The main phases of MCTS are shown in Fig. 2. They are explained briefly below.

There has been a large research interest in MCTS. Browne et al. [8] provides an extensive survey, referencing 240 publications.

MCTS basics

MCTS consists of four main steps: selection, expansion, simulation (playout), and backpropagation (see Fig. 2). The main steps are repeated as long as there is time left. Per step the activities are as follows.

(1) In the selection step the tree is traversed from the root node until we reach a node, where a child is selected that is not part of the tree yet.

(2) Next, in the expansion step the child is added to the tree.

(3) Subsequently, during the simulation step moves are played in self-play until the end of the game is reached. The result R of this—simulated—game is $+1$ in case of a win for Black (the first player in Go), 0 in case of a draw, and -1 in case of a win for White.

(4) In the back-propagation step, R is propagated backwards, through the previously traversed nodes. Finally, the move played by the program is the child of the root with the best win/visit count, depending on UCT probability calculations (to be discussed briefly below).

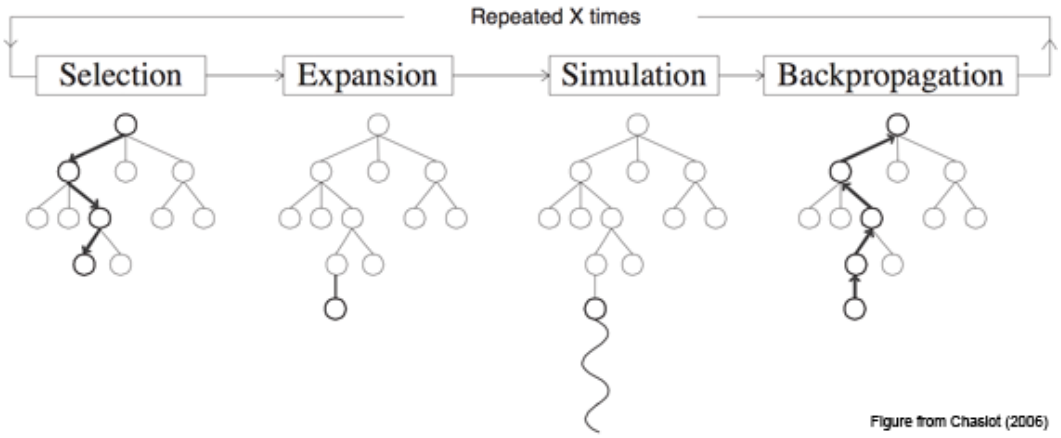
Crucially, the selection rule of MCTS allows it to balance (a) exploitation of parts of the tree that are known to be good with (b) exploration of parts of the tree that have not yet been explored.

Where MCTS originally used moves in the playout phase that are strictly random, better results are obtained by playing moves that use small amounts of domain knowledge. Many programs use pattern databases for this purpose [20]. The high levels of performance that are currenlty achieved with MCTS depend to a large extent on enhancements to the expansion strategy, simulation phase, and the parallelization techniques. (So small amounts of domain knowledge are needed after all, albeit not in the form of a heuristic evaluation function.)

Applications

The striking performance of MCTS in Go has led researchers to apply the algorithm to other domains. The ability of MCTS to find "bright spots" in the state space without relying on domain knowledge has resulted in a long list of other applications, for example, for proof-number search [34]. Moreover, MCTS

Figure 2: Monte Carlo Tree Search scheme



has been proposed as a new framework for game-AI for video games [12], for the game Settlers of Catan [36], for the game Einstein würfelt nicht [29], for the Voronoi game [6], for Havannah [32], for Amazons [30], and for single player applications [35].

4 High Energy Physics

One area where finding solutions is important, and where good heuristics are hard to find, is equation solving for high energy physics. In this field large equations are needed to be solved quickly. Standard packages such as Maple and Mathematica are often too slow, and a specialized high-efficiency package called FORM is used [31].

In particle physics and quantum field theory (QFT) calculations are needed for a detailed comparison between experimental data and theory. It is only natural to demand that the theoretical results are at least as accurate as the data, because usually the data are much more expensive to obtain. One prominent example here is the measurement and the Quantum Electro Dynamics calculation of the electric dipole moment of the electron (also called $g-2$) which gives agreement between theory and data to almost 10 digits [3]. In Quantum Chromodynamics (QCD), due to its larger coupling constant and a larger number of Feynman diagrams, currently the best accuracies are in the few percent range. With the advent of the new Large Hadron Collider (LHC) at CERN, which is expected to measure many reactions to the one percent range, it is essential that the accuracy in QFT calculations is drastically improved beyond the current state of the art.

Although there are nowadays successful nondiagrammatic methods for certain categories of reactions, the standard method for calculations in QFT is still by means of Feynman diagrams. These describe reactions pictorially, based on the terms that are present in the formulas describing the theory. A perturbative expansion in terms of the strength of the interactions corresponds to the number of closed loops in the diagrams. Hence, a more accurate calculation will involve diagrams with more loops. Each loop introduces a degree of freedom that is the four-momentum of the loop.

The number of possible diagrams increases more than exponentially with the number of loops and also the degree of difficulty to calculate them depends very strongly on the number of loops (Fig. 3).

There are various ways to calculate Feynman diagrams, all of which have a limited range of application. For the type of computations we envisage the best method is currently the integration-by-parts method (IBP). The integration here is the integration over the four-momenta related to the loops. As an example, consider the calculation of a reaction that was calculated in past years (1995-2004) [27]. This calculation was needed for a more accurate determination of the distribution of quarks and gluons inside the proton which in turn is needed by all precision calculations of reactions at the LHC in Geneva. The calculation involved $O(10000)$ Feynman diagrams with three loops, distributed over some 200 topologies. For each topology 20-30 equations were determined that related diagrams with different powers of the propagators to each other. The next step was to combine and apply these equations in such an order that the 14 parameters are reduced to either zero or one. Combining equations so that the new equations have fewer parameters, can result in rather lengthy equations (more than 1000 lines) and thus this work was done by computer, using the FORM program. The approach was as follows.

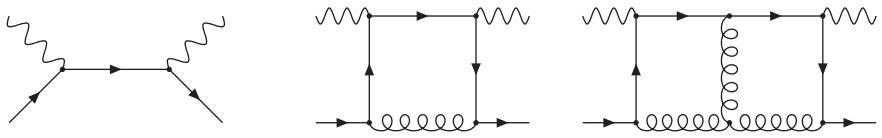


Figure 3: Examples of Feynman diagrams with zero, one and two loops. The different types of lines represent different particles.

1. The equations are generated by a FORM program.
2. The equations are inspected.
3. One of the equations to use is selected and a few lines to the FORM program are added to substitute it in the other equations.
4. The FORM program is run. This may take anywhere from a few seconds to up to many minutes when we are working on the very last equations.
5. The output is inspected. If satisfactory, we continue with step 2, unless we are done already. If the result is not satisfactory then the code of the last phase has to be undone and another equation must be tried. Maybe we have to backtrack by much more than one phase. With bad luck we may have to start over again.

Future Research

In many cases this approach led to a satisfactory solution, even though the work for a single topology might take anywhere from hours to weeks. Extending the same calculation to one additional loop would increase the number of topologies to more than 1000, the number of parameters would increase by 6 and the number of equations would go to 30-40.

Without extra levels of automatization such calculations are currently impossible. We believe that MCTS can provide some of the needed levels of automization. Connecting the science of artificial intelligence with physics is future research which we believe has a promising future ahead.

5 Horner's rule for multivariate polynomials

The research on MCTS in FORM was started with work on improving the speed of the evaluation of multivariate polynomials. Applying MCTS to this problem resulted in an unexpected improvement, reported in [25]. It is discussed here as an illustration of the usefulness of Go as the Drosophila of AI, as in intriguing way of connecting the sciences of artificial intelligence and mathematics. As part of this work a sensitivy analysis of MCTS was performed, which is reported here.

Polynomial evaluation is a frequently occurring part of equation solving. Optimizing its cost is important. Finding more efficient algorithms for polynomial evaluation is a classic problem in computer science. For single variable polynomials, Horner's method provides a scheme for producing a computationally efficient form. It is due to Horner (1819) [19], although references to the method go back to the 3rd century Chinese mathematician Liu Hui. For multivariate polynomials Horner's method is easily generalized but the order of the variables is unspecified. Traditionally greedy approaches such as using most-occurring variable first are used. This simple approach has given remarkably efficient results and finding better approaches has proven difficult [9]. Horner's rule is a convenient application to perform experiments that provide insight into how MCTS works.

For polynomials in one variable, Horner's method provides a computationally efficient evaluation form:

$$a(x) = \sum_{i=0}^{n} a_i x^i = a_0 + x(a_1 + x(a_2 + x(\cdots + x \cdot a_n))). \tag{1}$$

With this representation a dense polynomial of degree n can be evaluated with n multiplications and n additions, giving an evaluation cost of $2n$.

For multivariate polynomials Horner's method must be generalized. To do so one chooses a variable and applies Eqn. (1), treating the other variables as constants. Next, another variable is chosen and the same process is applied to the terms within the parentheses. This is repeated until all variables are processed. As an example, for the polynomial

$a = y - 6x + 8xz + 2x^2yz - 6x^2y^2z + 8x^2y^2z^2$ and the order $x < y < z$ this results in the following expression

$$a = y + x(-6 + 8z + x(y(2z + y(z(-6 + 8z))))). \quad (2)$$

The original expression uses 5 additions and 18 multiplications, while the Horner form uses 5 additions but only 8 multiplications. In general, applying a Horner scheme keeps the number of additions constant, but reduces the number of multiplications.

After transforming a polynomial with Horner's method, the code can be further improved by performing a common subexpression elimination (CSE). In Eqn. (2), the subexpression $-6 + 8z$ appears twice. Eliminating the common subexpression results in the code

$$\begin{aligned} T &= -6 + 8z \\ a &= y + x(T + x(y(2z + y(zT)))), \end{aligned} \quad (3)$$

which uses only 4 additions and 7 multiplications.

Horner's rule reduces the number of multiplications, CSE also reduces the number of additions.

Finding the optimal order of variables for the Horner scheme is an open problem for all but the smallest polynomials. Different orders impact the cost evaluating the resulting code. Simple variants of local search have been proposed in the literature, such as most-occurring variable first, which results in the highest decrease of the cost at that particular step.

MCTS is used to determine an order of the variables that gives efficient Horner schemes in the following way. The root of the search tree represents that no variables are chosen yet. This root node has n children. Each representing a choice for variables in the trailing part of the order. n equals the depth of the node in the search tree. A node at depth d has $n - d$ children: the remaining unchosen variables.

In the simulation step the incomplete order is completed with the remaining variables added randomly. This complete order is then used for Horner's method followed by CSE. The number of operators in this optimized expression is counted. The selection step uses the UCT criterion with as score the number of operators in the original expression divided by the number of operators in the optimized one. This number increases with better orders.

In MCTS the search tree is built in an incremental and asymmetric way, see Fig. 4. During the search the traversed part of the search tree is kept in memory. For each node MCTS keeps track of the number of times it has been visited and the estimated result of that node. At each step one node is added to the search tree according to a criterion that tells where most likely better results can be found. From that node an outcome is sampled and the results of the node and its parents are updated. This process is illustrated in Fig. 2.

Selection During the selection step the node which most urgently needs expansion is selected. Several criteria are proposed, but the easiest and most-used is the UCT criterion [24]:

$$UCT_i = \langle x_i \rangle + 2C_p \sqrt{\frac{2 \log n}{n_i}}. \quad (4)$$

Here $\langle x_i \rangle$ is the average score of child i, n_i is the number of times child i has been visited and n is the number of times the node itself has been visited. C_p is a problem-dependent constant that should be determined empirically. Starting at the root of the search tree, the most-promising child according to this criterion is selected and this selection process is repeated recursively until a node is reached with unvisited children. The first term of Eqn. (4) biases nodes with previous high rewards (exploitation), while the second term selects nodes that have not been visited much (exploration). Balancing exploitation versus exploration is essential for the good performance of MCTS.

Expansion The selection step finishes with a node with unvisited children. In the expansion step one of these children is added to the tree.

Simulation In the simulation step a single possible outcome is simulated starting from the node that has just been added to the tree. This simulation can consist of generating a complete random outcome starting from this node or can be some known heuristic for the search problem. The latter typically works better if specific knowledge of the problem is available.

Backpropagation In the backpropagation step the results of the simulation are added to the tree, specifically to the path of nodes from the newly-added node to the root. Their average results and visit count are updated.

This MCTS cycle is repeated a fixed number of times or until the computational resources are exhausted. After that the best result found is returned.

Sensitivity to C_p and N

The performance of MCTS-Horner followed by CSE has been tested by implementing in FORM [31]. MCTS-Horner was tested on a variety of different multivariate polynomials, against the currently best algorithms. For each test, polynomial MCTS found better variable orders, typically with half the number of operators than the expressions generated by previous algorithms. The results are reported in detail in [25].

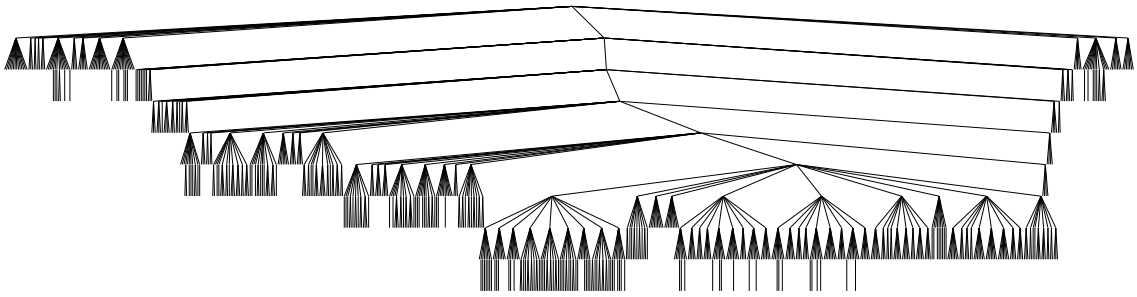


Figure 4: The asymmetric search through a MCTS search tree during the search for an efficient Horner scheme.

The experiments showed that the effectiveness of MCTS depends heavily on the choice for the exploitation/exploration constant C_p of Eqn. (4) and on the number of tree expansions (N).

When C_p is small, MCTS favors parts of the tree that have been visited before because the average score was good ("exploitation"). When C_p is large, MCTS favors parts of the tree that have not been visited before ("exploration").

Horner is an application domain that allows relatively quick experimentation. To gain insight into the sensitivity of the performance in relation to C_p and to the number of expansions a series of scatter plots have been created, which give some explanatory insight into the nature of this relatively new search algorithm.

The results of MCTS followed by CSE, with different numbers for tree expansions N as a function of C_p are given in Fig. 5 for a large polynomial from HEP, called HEPpoly. For equal values of C_p different results are produced because of different seeds of the Monte Carlo random number generator. On the y-axis of each graph the number of operations of the resulting expression is plotted. The lower this value, the better the algorithm performs. The lowest value found for this polynomial by MCTS+CSE is an expression with slightly more than 4000 operations. This minimum is achieved in the case with 3000 tree expansions for a value of C_p of between 0.7 and 1.2. Dots above this minimum represent a sub-optimal search result.

For small values of the numbers of tree expansions MCTS cannot find a good answer. With 100 expansions the graph looks almost random. Then, as we move to 300 tree expansions per data point, some clearer structure starts to emerge, with a minimum emerging at $C_p \approx 0.6$. With more tree expansions the picture becomes clearer, and the value for C_p for which the best answers are found becomes higher, the picture appears to shift to the right. For really low numbers of tree expansions there is no discernible advantage of setting the exploitation/exploration parameter at a certain value. For slightly larger numbers of tree expansion, but still low, MCTS needs to exploit each good result that it obtains. As the number of tree expansions grows larger, MCTS achieves better results when its selection policy is more explorative. It can afford to look beyond the narrow tunnel of exploitation, to try a few exploration beyond the path that is known to be good, to try to get out of local optima. For the graphs with tree expansions of 3000, 10000 and 30000 the range of good results for C_p becomes wider, indicating that the choice between exploitation/exploration becomes less critical.

For small values of C_p, such that MCTS behaves exploitatively, the method gets trapped in one of the local minima as can be seen from scattered dots that form "lines" in the left-hand side of the figure. For large values of C_p, such that MCTS behaves exploratively, many of the searches do not lead to the global minimum found as can be seen from the cloud of points on the right-hand side. For intermediate values of $C_p \approx 1$ MCTS balances well between exploitation and exploration and finds almost always a Horner scheme that is very close to the best one known to us.

The results of the test with HEPpoly for different numbers of tree expansions are shown in Fig. 6, reproduced from [25]. For small numbers of tree expansions low values for the constant C_p should be chosen (smaller than 0.5). The search is then mainly in exploitation mode. MCTS quickly searches deep in the tree, most probably around a local minimum. This local minimum is explored quite well, but the global minimum is likely to be missed. With higher numbers of tree expansions a value for C_p in the range $[0.5, 2]$ seems suitable. This range gives a good balance between exploring the whole search tree and exploiting the promising nodes. Very high values of C_p appear to be a bad choice in general, proven nodes are not exploited anymore so frequently. Here we note that these values hold for HEPpoly, and that different polynomials give different optimal values for C_p and

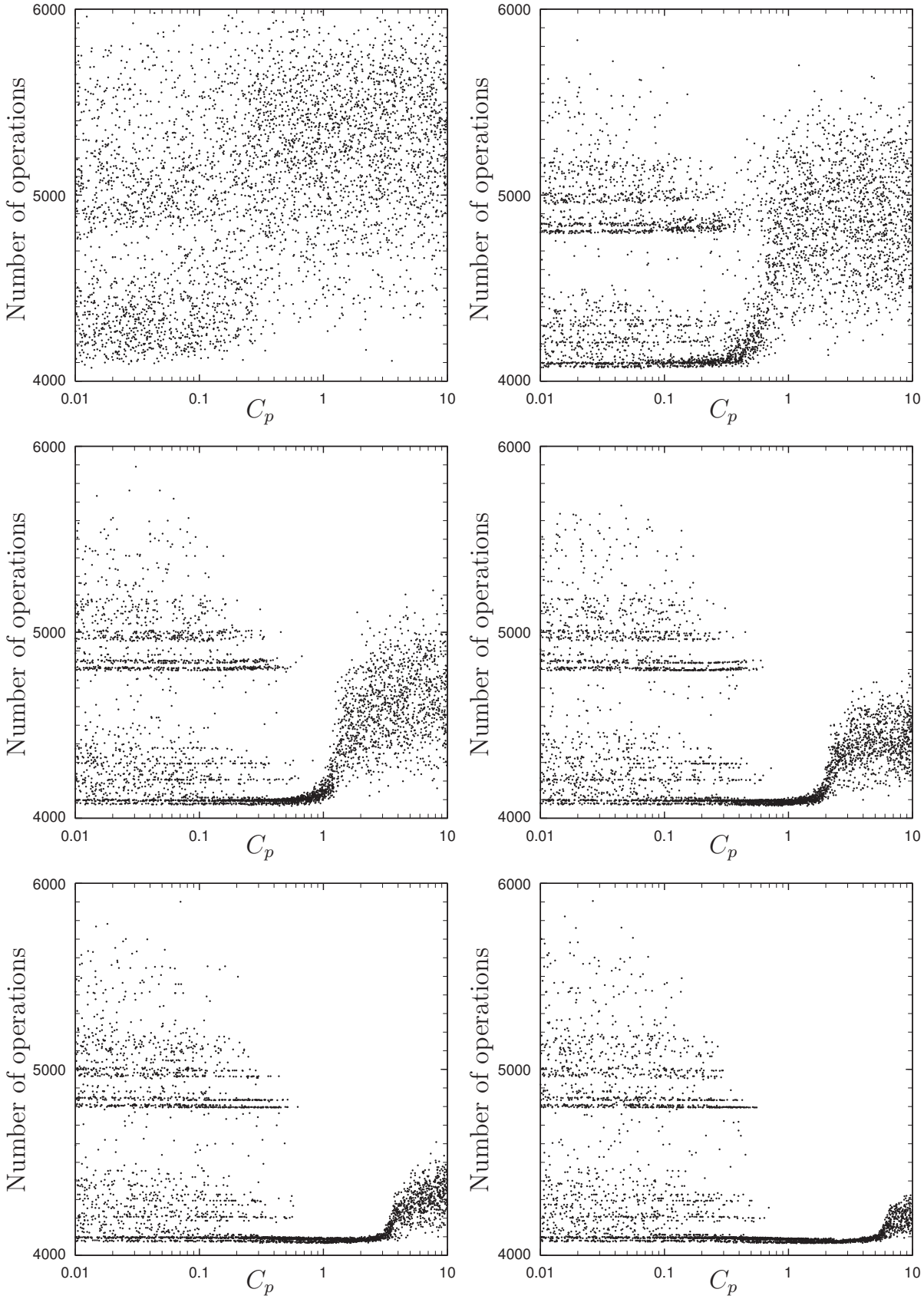


Figure 5: Six scatterplots for different numbers of expansions: n=100, 300, 1000, 3000, 10000, 30000, left to right, top to bottom

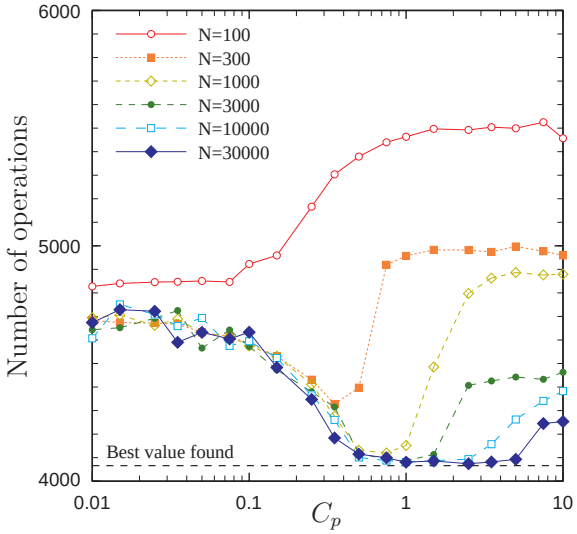


Figure 6: The performance of MCTS Horner as function of the exploitation/exploration constant C_p and the number of tree expansions N. For $N = 3000$ (green dots) the optimum is $C_p \approx 1$.

N. The different values for C_p are so far determined by trial and error. Automatic tuning of C_p is part of ongoing research.

6 Discussion

From the beginning of AI in 1950, chess has been called the Drosophila of AI. It was the testbed of choice. Many of the findings from decades of computer chess research have found their way to other fields, such as protein sequencing, natural language processing, machine learning, and high performance search [22]. After DEEP BLUE had defeated Garry Kasparov, research attention shifted to Go.

For Go, no good heuristic evaluation function seems to exist. Therefore, a different search paradigm was invented: MCTS. The two most prevailing characteristics are: no more minimax, no need for a heuristic evaluation function. Instead, MCTS uses (1) the average of random playouts to guide the search, and (2) by balancing between exploration and exploitation, it appears to be able to detect by itself which areas of the search tree contain the green leaves, and which branches are dead wood. Having a "self-guided" (best-first) search, without the need for a domain dependent heuristic, can be highly useful. For many other application domains the construction of a heuristic evaluation function is an obstacle, too. Therefore we expect that there are many other domains that could benefit from this technology, and, indeed, many other applications have already been found their way (see, for example, [6, 12, 29, 30, 32, 34–36]). In this paper two links have been discussed, viz. (1) with High Energy Physics, and (2) with Horner's rule. Finding better variable orders for the classic multivariate Horner's scheme algorithm is an exciting first result [25].

In summary, scientific research has branched out into many fields and subfields. As we see communication between disparate subfields can result in fertile cross pollination. Thus our conclusion unavoidably is that a new Drosophila was born out of a connection between physics and artificial intelligence. It is a Drosophila that now connects the sciences of High Energy Physics, Mathematics, and Artificial Intelligence.

REFERENCES

[1] Victor Allis (1994). Searching for Solutions in Games and Artificial Intelligence (Ph.D. thesis). University of Limburg, Maastricht, The Netherlands.

[2] Ingo Althöfer. "The origin of dynamic komi," ICGA Journal, volume 35, number 1, March 2012, pp. 31-34

[3] T. Aoyama, M. Hayakawa, T. Kinoshita and M. Nio, e-Print: arXiv:1110.2826 [hep-ph]

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer: "Finite-time Analysis of the Multiarmed Bandit Problem," Mach. Learn., vol. 47, no. 2, pp. 235-256, 2002

[5] Bruno Bouzy and Bernard Helmstetter. "Monte-Carlo Go developments." H.J. van den Herik, H. Iida, E.A. Heinz (eds.), 10th Advances in Computer Games conference (AGC-10). pp. 159-174, 2003

[6] Bruno Bouzy, Marc Métivier, Damien Pellier: "MCTS experiments on the Voronoi Game," Advances in Computer Games 2011, Tilburg, The Netherlands, pp. 96-107, 2012

[7] Bernd Brügmann. "Monte-Carlo Go." AAAI Fall symposium on Games: Playing, Planning, and Learning http://www.cgl.ucsf.edu/go/Programs/Gobble.html 1993

[8] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton: "A survey of Monte Carlo Tree Search Methods," IEEE Transactions on Computational Intelligence and

AI in Games, March 2012, Volume 4, issue 1, pages 1-43, 2012

[9] M. Ceberio, V. Kreinovich, ACM SIGSAM Bull. **38** pp. 8–15, 2004

[10] G. M. J.-B. Chaslot, S. de Jong, J.-T. Saito, and J. W. H. M. Uiterwijk, "Monte-Carlo Tree Search in Production Management Problems," in Proc. BeNeLux Conf. Artif. Intell., Namur, Belgium, pp. 91-98, 2006

[11] G.M.J-B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, and B. Bouzy. "Progressive Strategies for Monte-Carlo Tree Search." In P. Wang et al., editors, Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007), pages 655-661. World Scientific Publishing Co. Pte. Ltd., 2007; also in: New Mathematics and Natural Computation, 4(3):343-357.

[12] G.M.J-B. Chaslot, S. Bakkes, I. Szita, and P. Spronck: "Monte-Carlo Tree Search: A new framework for Game AI." In. M. Mateas and C. Darken, eds, Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference. AAAI Press, Menlo Park, CA, 2008

[13] Chaslot, G. M. J.-B., Hoock, J.-B., Rimmel, A., Teytaud, O., Lee, C.-S., Wang, M.-H., Tsai, S.-R., and Hsu, S.-C. (2008a). "Human-Computer Go Revolution 2008." ICGA Journal, Vol. 31, No. 3, pp. 179-185.

[14] S. Chinchalkar, "An Upper Bound for the Number of Reachable Positions." ICCA Journal, Vol. 19, No. 3, pp. 181-182, 1996

[15] Remi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." In H.J. Van den Herik, P. Ciancarini and H.H.L.M. Donkers, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy, pp. 72-83, 2006

[16] H.H.L.M. Donkers, H.J. van den Herik, J.W.H.M. Uiterwijk, "Selecting Evaluation Functions in Opponent Model Search." Theoretical Computer Science (TCS), Vol 349, No. 2, pp. 245-267, 2005

[17] A.D. de Groot. "Het denken van den schaker," Ph. D. thesis in dutch 1946; translated in 1965 as "Thought and Choice in chess." Mouton Publishers, The Hague (second edition 1978). Freely available as e-book from Google.

[18] M. Enzenberger. "Evaluation in Go by a Neural Network Using Soft Segmentation" In Proceedings of the 10th Advances in Computer Games Conference, Graz, Austria, 2003.

[19] Horner, William George (July 1819). "A new method of solving numerical equations of all orders, by continuous approximation." Philosophical Transactions (Royal Society of London): pp. 308-335. Reprinted with appraisal in D.E.Smith: A Source Book in Mathematics, McGraw-Hill, 1929; Dover reprint, 2 vols 1959

[20] S. Gelly, Y. Wang, R. Munos, and O. Teytaud: "Modification of UCT with Patterns in Monte-Carlo Go," Inst. Nat. Rech. Inform. Auto. (INRIA), Paris, Tech. Rep., 2006

[21] D. Hartmann, "How to Extract Relevant Knowledge from Grandmaster Games. Part 1: Grandmaster have insights—the problem is what to incorporate into Practical Problems." ICCA Journal, Vol. 10, No. 1, pp 14-36, 1987

[22] H.J. van den Herik, "Informatica en het Menselijk Blikveld." Inaugural address Rijksuniversiteit Limburg, Maastricht, The Netherlands, 1988

[23] Junghanns, A: "Are there Practical Alternatives to Alpha-Beta?" ICCA Journal, Vol. 21, No. 1, pp. 1432, 1998

[24] L. Kocsis and C. Szepesvàri: "Bandit based Monte-Carlo Planning," in Euro. Conf. Mach. Learn. Berlin, Germany: Springer, pp. 282293, 2006

[25] Jan Kuipers, Jos A.M. Vermaseren, Aske Plaat, H. Jaap van den Herik, "Improving multivariate Horner schemes with Monte Carlo tree search," arXiv 1207.7079, July 2012

[26] Evgenii Mikhailovich Landis and I.M. Yaglom, "About Aleksandr Semenovich KronRod" Russian Math. Surveys 56:993-1007, 2001

[27] S.A. Moch, J.A.M. Vermaseren, A. Vogt: Nucl.Phys. B688 (2004) 101-134, B691 (2004) 129-181, B724 pp. 3-182, 2005

[28] Müller, Martin. "Computer Go," Artificial Intelligence 134: p151, 2002

[29] Richard Lorentz: "An MCTS Program to Play Einstein Würfelt nicht!" Advances in Computer Games 2011, Tilburg, The Netherlands, pp. 52-59, 2012

[30] Julien Kloetzer: "Monte Carlo Opening books for Amazons." Computers and Games 2010, Kanazawa, Japan, pp. 124-135, 2011

[31] J. Kuipers, T. Ueda, J.A.M. Vermaseren, J. Vollinga, "FORM version 4.0," *preprint* arXiv:1203.6543 (2012)

[32] Richard Lorentz: "Experiments with Monte Carlo Tre Search in the Game of Havannah." ICGA Journal, Vol 34, no 3, 2011

[33] Rivest, R: "Game-tree searching by min-max approximation," Artificial Intelligence, 1988 Vol. 34, No. 1, pp. 77-96, 1988

[34] J-T. Saito, G.M.J-B. Chaslot, Jos. W.H.M. Uiterwijk, and H.J. van den Herik: "Monte-Carlo Proof-Number Search." In Computers and Games, 2007

[35] Maarten Schadd, Mark H.M. Winands, H.Jaap van den Herik, Guillaume Chaslot, Jos W.H.M. Uiterwijk: "Single Player Monte Carlo Tree Search." In: Computers and Games 2008: pp. 1-12, 2008

[36] I. Szita, G.M.J-B. Chaslot, and P. Spronck: "Monte-Carlo Tree Search in Settlers of Catan." In Proceedings of the 12th International Advances in COMputer Games Conference (ACG'09), Pamplona, Spain, May 11-13, 2009

[37] Erik C.D. van der Werf, H. Jaap van den Herik and Jos W.H.M. Uiterwijk. "Learning to score final positions in the game of Go." Theoretical Computer Science. Vol. 349. No. 2. pp. 168-183, 2005

[38] Erik C.D. van der Werf, Mark H.M. Winands and H. Jaap van den Herik, Jos W.H.M. Uiterwijk. "Learning to predict Life and Death form Go game records." Information Sciences. Vol.175, No. 4. pp. 258-272, 2005