Games and loop integrals

Ben Ruijl

ETH Zurich, Wolfgang-Pauli-Str. 27, 8093 Zürich, Switzerland

E-mail: `benruyl@gmail.com`

Abstract. In order to numerically integrate complicated cross sections, large expressions have to be sampled many times, which is a time consuming operation. Inspired by the use in games, artificial intelligence methods such as Monte Carlo Tree Search and Local Stochastic Search are used to simplify the expressions. As a result, large expressions can be compressed by more than a factor 20.

1. Introduction

Monte Carlo integration is the preferred way to compute complicated cross sections of scattering processes [1, 2, 3, 4]. The disadvantage of this method is that it converges slowly and thus requires a large number of samples. Sampling large multivariate polynomials is very time consuming, which makes Monte Carlo methods slow. The expressions that arise from Quantum Chromodynamics (QCD) are polynomials in many variables. The number of variables could range from a few to several hundreds. Analogously, the number of terms range from ten thousand terms to millions of terms [5]. In some extreme cases, the executable code that performs the evaluation of the expression could take up a few gigabytes. If we are able to reduce the number of operations required to evaluate these expressions, sampling becomes faster. As a result, Monte Carlo integrators can obtain precise results much faster.

We describe two methods to reduce the number of operations. The first is Horner's rule for multivariate polynomials, which is extracting variables outside brackets [6] (sec. 2.1). For multivariate expressions the order of these variables is called a *Horner scheme*. The second is called Common Subexpression Elimination (CSEE) [7], which is performed after the Horner scheme has been applied (sec. 2.2). We will investigate two methods of finding a Horner scheme that yields a near-minimal number of operations after the Horner scheme and CSEE have been applied. The first is the tree search algorithm Monte Carlo Tree Search (MCTS), which is also used for finding the best moves in games. For example, MCTS is an important component of AlphaGo [8]. The second method is a local search, which has been successful in solving the Travelling Salesman problem [9].

Our final local search algorithm is able to reduce the computation time of numerical integration from weeks to days or even hours. The methods have been implemented in Form [10] and are used by at least one other research group.

The remainder of this work is structured as follows. First, we discuss methods for expression simplification in section 2. In section 3 we discuss the experimental setup. Next, we discuss MCTS in section 4. In section 5 we examine local search methods. We discuss performance and results in section 6. Finally, we present the conclusion in section 7.

2. Horner schemes and common subexpression elimination

Expression simplification is a widely studied problem. Some examples are Horner schemes [7], common subexpression elimination (CSEE) [11], partial syntactic factorisation [12] and Breuer's growth algorithm [13]. Much research is put into simplifications using more algebraic properties, such as factorisation, especially because of its interest for cryptographic research [14, 15]. Simplification methods that depend on factorisation have the major problem of being notoriously slow. Horner schemes and CSEE do not require sophisticated mathematics: only the commutative and associative properties of the operators are used. The expressions we are considering often have more than twenty variables and more than a hundred thousand terms [5]. In this regime, computationally expensive methods are infeasible. Therefore, we consider using basic methods such as Horner schemes and CSEE.

2.1. Horner Schemes

Horner's rule reduces the number of multiplications in an expression by lifting variables outside brackets [6, 7, 16]. For multivariate expressions Horner's rule can be applied sequentially, once for each variable. The order of this sequence is called the *Horner scheme*. Take for example:

$$x^2z + x^3y + x^3yz \to x^2(z + x(y(1 + z))) \; . \tag{1}$$

Here, first the variable x is extracted (i.e., x^2 and x) and second, y. The number of multiplications is now reduced from 9 to 4. However, the order x, y is chosen arbitrarily. One could also try the order y, x:

$$x^2z + x^3y + x^3yz \to x^2z + y(x^3(1 + z)) \; , \tag{2}$$

for which the number of multiplications is 6. Evidently, this is a sub-optimal Horner scheme. There are $n!$ orders of extracting variables, where n is the number of variables, and it turns out that the problem of selecting an optimal ordering is NP-hard [16].

2.2. Common subexpression elimination

The number of operations can be reduced further by applying common subexpression elimination (CSEE). This method is well known from the fields of compiler construction [11] and computer chess [17], where it is applied to much smaller expressions or subtrees than what we are considering here. Figure 1 shows an example of a common subexpression in a tree representation of an expression. The shaded expression $b(a+e)$ appears twice, and its removal means removing one superfluous addition and one multiplication.

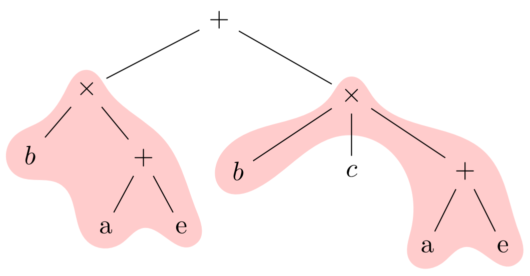

Figure 1: A common subexpression (shaded) in an associative and commutative tree representation.

CSEE is able to reduce both the number of multiplications and the number of additions, whereas Horner schemes are only able to reduce the number of multiplications.

	variables	terms	operations	eval. time (s)
res(7,4)	13	2561	29 163	0.001
res(7,5)	14	11 379	142 711	0.03
res(7,6)	15	43 165	587 880	0.13
res(9,8)	19	4 793 296	83 778 591	25.0
HEP(σ)	15	5716	47 424	0.008
HEP(F_{13})	24	105 058	1 068 153	0.4
HEP(F_{24})	31	836 009	7 722 027	3.0
HEP(b)	107	193 767	1 817 520	2.0

Table 1: The number of variables, terms, operations, and the evaluation time of applying a single Horner scheme and CSEE in seconds, for our eight (unoptimised) benchmark expressions. The time measurement is performed on a 2.4 GHz Xeon computer. All expressions fit in memory (192 GB).

3. Experimental setup

We use eight large benchmark expressions, four from mathematics and four from real-world High Energy Physics (HEP) calculations. In table 1 statistics for the expressions are displayed. We show the number of variables, terms, operations, and the evaluation time of applying a Horner scheme and CSEE.

The expressions called res(7,4), res(7,5), res(7,6), and res(9,8) are resolvents and are defined by $\text{res}(m, n) = \text{res}_x(\sum_{i=0}^{m} a_i x^i, \sum_{i=0}^{n} b_i x^i)$, as described in [12]. The number of variables is $m + n + 2$. The polynomial res(9,8) is the largest polynomial we have tested and has been included to test the boundaries of our hardware.

The High Energy Physics expressions represent scattering processes for the future International Linear Collider, a likely successor to the Large Hadron Collider [5]. A standard method of calculating the probability of certain collision events is by using perturbation theory. As a result, for each order of perturbations, additional expressions are calculated as corrections to previous orders of precision. The HEP polynomials of table 1 are second-order corrections to various processes.

HEP(σ) describes parts of the process $e^+e^- \to \mu^+\mu^-\gamma$. HEP($F_{13}$), HEP($F_{24}$), and HEP($b$) are obtained from the process $e^+e^- \to \mu^+\mu^- u\bar{u}$. The results can be used to obtain next-generation precision measurements for electron-positron scattering [5]. These four HEP polynomials represent classes of polynomials with approximately the same behaviour.

4. Tree search methods

Recently, Monte Carlo Tree Search has been shown to yield good quality Horner schemes [18]. We will describe its characteristics, so that we can see if we can improve its performance.

Monte Carlo Tree Search (MCTS) is a tree search method that has been successful in games such as Go, Hex, and other applications with a large state space [19, 20]. It works by selectively building a tree, expanding only branches it deems worthwhile to explore. MCTS consists of four steps, which are displayed in figure 2. The first step (2a) is the selection step, where a leaf or a not fully expanded node is selected according to some criterion (see below). Our choice is node z. In the expansion step (2b), a random unexplored child of the selected node is added to the tree (node y). In the simulation step (2(c)), the rest of the path to a final node is completed using random child selection. Finally a score Δ is obtained that signifies the score of the chosen path through the state space. In the backpropagation step (2d), this value is propagated back through the tree, which affects the average score (winrate) of a node (see below). The tree is built iteratively by repeating the four steps.

In the game of Go, each node represents a player move and in the expansion phase the game is played out, in basic implementations, by random moves. In the best performing implementations heuristics and pattern knowledge are used to complement a random playout [19]. The final score is 1 if the game is won, and 0 if the game is lost. The entire tree is built, ultimately, to select the best first move.

For our purposes, we need to build a complete Horner scheme, variable by variable. As such, each node will represent a variable and the depth of a node in the tree represents the position in the Horner scheme. Thus, in figure 2(c) the partial Horner scheme is x,z,y and the rest of the scheme is filled in randomly with unused variables. The score of a path in our case, is the improvement of the path on the number of operations: the original number of operations divided by the number of operations after the Horner scheme and CSEE have been applied. We note that for our purposes the *entire* Horner scheme is important and not just the first variable.

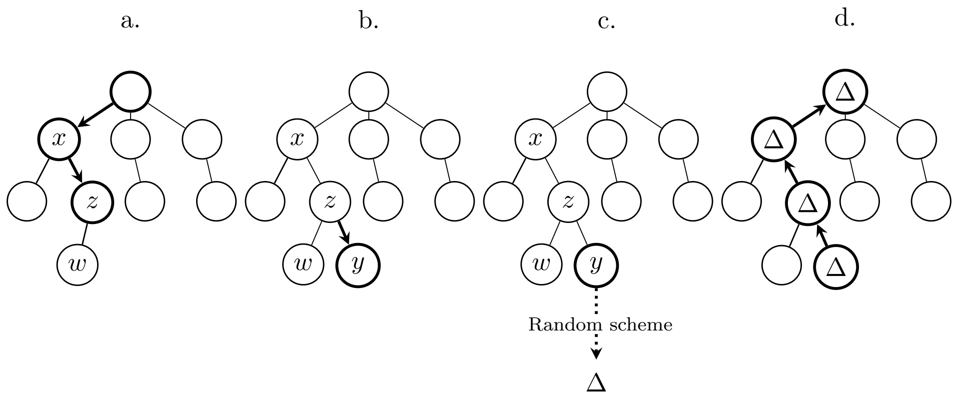

Figure 2: An overview of the four phases of MCTS: selection (a), expansion (b), simulation (c), and backpropagation (d). The selection of a not fully expanded node is done using the best child criterion (in our case Upper Confidence bounds applied to Trees (UCT)). Δ is the number of operations left in the final expression, after the Horner scheme and CSEE have been applied. See also [21].

In many MCTS implementations UCT (formula 3) is chosen as the selection criterion [21, 22]:

$$\underset{\text{children } c \text{ of } s}{\operatorname{argmax}} \; \bar{x}(c) + 2C_p\sqrt{\frac{2\ln n(s)}{n(c)}} \,, \tag{3}$$

where c is a child node of node s, $\bar{x}(c)$ the average score of node c, $n(c)$ the number of times the node c has been visited, C_p the exploration-exploitation constant, and argmax the function that selects the child with the maximum value. This formula balances exploitation, i.e., picking terms with a high average score, and exploration, i.e., selecting nodes where the child has not been visited often compared to the parent. The C_p constant determines how strong the exploration term is: for high C_p the focus will be on exploration, and for low C_p the focus will be on exploitation.

We consider the role of the exploration-exploitation constant C_p in the UCT formula. We notice that in the first iterations of the simulation there is as much exploration as there is in the final iterations, since C_p remains constant throughout the search. For example, the final 100 iterations of a 1000 iterations MCTS run are used to explore new branches even though we know in advance that there is likely not enough time to reach the final nodes. Thus we would like to modify the C_p to change during the simulation to emphasise exploration early in the search and emphasise exploitation towards the end.

We introduce a new, dynamic exploration-exploitation parameter T that decreases linearly with the iteration number [23]:

$$T(i) = C_p \frac{N - i}{N} \ , \tag{4}$$

where i is the current iteration number, N the preset maximum number of iterations, and C_p the initial exploration-exploitation constant at $i = 0$.

We modify the UCT formula to become:

$$\underset{\text{children } c \text{ of } s}{\text{argmax}} \ \bar{x}(c) + 2T(i)\sqrt{\frac{2 \ln n(s)}{n(c)}} \ , \tag{5}$$

where c is a child of node s, $\bar{x}(c)$ is the average score of child c, $n(c)$ the number of visits at node c, and $T(i)$ the dynamic exploration-exploitation parameter of formula (4).

The role of T is similar to the role of the temperature in Simulated Annealing [24]: in the beginning of the simulation there is much emphasis on exploration, the analogue of allowing transitions to energetically unfavourable states. During the simulation the focus gradually shifts to exploitation, analogous to annealing. Hence, we call our new UCT formula "Simulated Annealing UCT (Simulated Annealing - UCT (SA-UCT))".

In the past related changes have been proposed. For example, Discounted UCB [25] and Accelerated UCT [26] both modify the average score of a node to discount old wins over new ones. The difference between our method and past work is that the previous modifications alter the importance of exploring based on the history and do not guarantee that the focus shifts from exploration to exploitation. In contrast, this work focuses on the exploration-exploitation constant C_p and on the role of exploration during the simulation.

We implemented four improvements over UCT. (1) The final iterations are used effectively. (2) There is more exploration in the middle and at the bottom of the tree. This is due to more nodes being expanded at lower levels, because the T is lowered. As a consequence, we see that (3) more branches reach the end states. As a result, (4) there is exploration near the bottom, where there was none for the random playouts.

In order to analyse the effect of SA-UCT on the fine-tuning of C_p (the initial temperature), we perform a sensitivity analysis on C_p and N [23]. We find that (1) SA-UCT increases the region of C_p that gives good results by and order of magnitude, and (2) produces Horner schemes of the same quality. Thus, we may conclude that SA-UCT reduces the fine-tuning problem without overhead.

5. Stochastic Local Search

There are some intrinsic shortcomings to using a tree representation, especially if the depth of the search tree becomes (too) large. We noticed that many branches do not reach the bottom when there are more than 20 variables (we remind the reader that the problem depth is equivalent to the number of variables) as is the case with many of our expressions. MCTS determines the scores of a branch by performing a random play-out. If the branch is not constructed all the way to the bottom, the final nodes are therefore random (no optimisation). For Horner schemes, the *entire* scheme is important, so sub-optimal selection of variables at the end of the scheme can have a significant impact.

The issue of poor optimisation at the bottom of the tree motivated us to look for a method that is symmetric in its optimisation: both the beginning and the end should be optimised equally well. In this section we (re)consider which class of algorithms is best suited for the Horner scheme problem. The Horner scheme problem belongs to the class of permutation problems. Many algorithms for optimising permutation problems have been suggested in the literature, such as Stochastic Hill Climbing (SHC) [27], Simulated Annealing (SA) [24], Tabu Search [28],

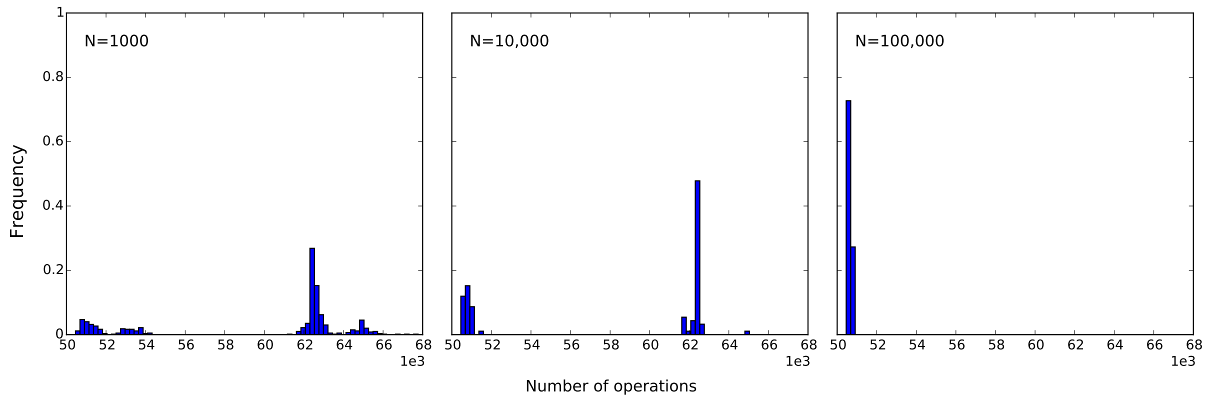

Figure 3: The distribution of the number of operations for the HEP(F_{13}) expression with 1 swaps for 1000 iterations (left), 10 000 iterations (middle) and 100 000 iterations (right) at $T = 0$. There appears to be a local minimum around 62 000. However, as the number of iterations is increased, the local minimum becomes smaller relative to the global minimum region (middle) and completely disappears (right). We may conclude that the apparent local minimum is not a local minimum, but a saddle point, since SHC is able to escape from the 'minimum'.

Ant Colony Optimisation [29], and Evolutionary Algorithms [30]. Since measurements take weeks per algorithm, we limit ourselves to two. In [31] we provide qualitative motivations for focusing on SHC and SA. In summary, the absence of heuristics and the high cost of sampling to tune parameters make the other options less interesting.

A Stochastic Hill Climbing procedure, also known as iterative improvement local search, has two parameters: (1) the number of iterations N, and (2) the neighbourhood structure, which defines the transition function [9]. Fewer tunable parameters is an advantage in our use case, since the evaluation time is often too long for extensive tuning. We find that running half of the simulations with a neighbourhood structure that makes minor changes to the state (i.e., a single shift of a variable), and running the other half with a neighbourhood structure that involves larger changes (i.e., the mirroring of a random sublist) is a good strategy for all of our benchmark expressions [31]. Consequently, only the computation time remains as an actual parameter. A Stochastic Hill Climbing procedure only moves to a neighbour if the evaluation score (number of operations) is improved. As a consequence, SHC could get stuck in local minima.

To obtain an idea on the number of local minima, we measure how often the simulation gets stuck: if there are many local minima, we expect the simulation to get stuck often. In figure 3, we show the distribution of HEP(F_{13}) for 1000, 10 000, and 100 000 SHC runs respectively. For 1000 and 10 000 runs we see two peaks: one at the global minimum near 51 000 and one at an apparent local minimum near 62 000. As the number of iterations is increased, the weight shifts from the apparent local minimum to the global minimum: at 1000 iterations, there is a probability of 27.5% of arriving in the region of the global minimum, whereas this is 36.3% at 10 000 iterations. Apparently the local minimum is 'leaking': given sufficient time, the search is able to escape. The figure on the right with 100 000 iterations confirms the escaping possibility: the apparent local minimum has completely disappeared. Thus, the local minimum is in reality a saddle point, since for a true local minimum there is no path with a lower score leading away from the minimum. Since SHC requires many iterations to escape from the saddle point, only a few transitions reduce the number of operations.

We observe that apparent local minima disappear for our other benchmark expressions as well. SHC runs with 100 000 iterations approach the global minimum for all of our benchmark expressions. For example, for HEP(F_{13}) mentioned above, the result is 50636 ± 57 and for

	vars	original	MCTS 1k	MCTS 10k	SHC 1k	SHC 10k	$E_{min,4}$ 1k
res(7,4)	13	29 163	$(3.86 \pm 0.1) \cdot 10^3$	$(3.84 \pm 0.01) \cdot 10^3$	$(3.92 \pm 0.28) \cdot 10^3$	3834 ± 26	3819 ± 9
res(7,5)	14	142 711	$(1.39 \pm 0.01) \cdot 10^4$	13768 ± 28	13841 ± 441	13767 ± 21	13770 ± 5
res(7,6)	15	587 880	$(4.58 \pm 0.05) \cdot 10^4$	$(4.54 \pm 0.01) \cdot 10^4$	46642 ± 3852	$(4.61 \pm 0.25) \cdot 10^4$	$(4.55 \pm 0.16) \cdot 10^4$
res(9,8)	19	83 778 591	$(5.27 \pm 0.25) \cdot 10^6$	$(4.33 \pm 0.31) \cdot 10^6$	$(4.13 \pm 0.34) \cdot 10^6$	$(4.03 \pm 0.17) \cdot 10^6$	$(3.97 \pm 0.18) \cdot 10^6$
HEP(σ)	15	47 424	4114 ± 14	4087 ± 5	4226 ± 257	4082 ± 58	4075 ± 25
HEP(F_{13})	24	1 068 153	$(6.6 \pm 0.2) \cdot 10^4$	$(6.47 \pm 0.08) \cdot 10^4$	$(5.99 \pm 0.51) \cdot 10^4$	$(5.80 \pm 0.55) \cdot 10^4$	$(5.37 \pm 0.40) \cdot 10^4$
HEP(F_{24})	31	7 722 027	$(3.80 \pm 0.06) \cdot 10^5$	$(3.19 \pm 0.04) \cdot 10^5$	$(3.16 \pm 0.23) \cdot 10^5$	$(3.06 \pm 0.23) \cdot 10^5$	$(2.98 \pm 0.09) \cdot 10^5$
HEP(b)	107	1 817 520	$(1.81 \pm 0.04) \cdot 10^5$	$(1.65 \pm 0.08) \cdot 10^5$	$(1.50 \pm 0.08) \cdot 10^5$	$(1.40 \pm 0.06) \cdot 10^5$	$(1.44 \pm 0.04) \cdot 10^5$

Table 2: SHC compared to MCTS. The MCTS results for all expressions except res(9,8) and HEP(b) are from [18]. All the values are statistical averages over at least 100 runs. SHC results have a larger standard deviation, and thus the expected value of the minimum is often lower than these values (see last column).

HEP(σ) the result is 4078 ± 9. The small standard deviations indicate that no runs get stuck in local minima (at least not in local minima significantly higher than the standard deviation).

From these results we may conclude that true local minima, from which a Stochastic Hill Climbing cannot escape, are rare for Horner schemes.

6. Performance of SHC vs. MCTS

Below, we compare the results of Stochastic Hill Climbing to the previous best results from MCTS, for our eight benchmark expressions res(7,4), res(7,5), res(7,6), res(9,8), HEP(σ), HEP(F_{13}), HEP(F_{24}) and HEP(b). The results of all the MCTS runs except for res(9,8), and HEP(b) are taken from [18].[1] The results are displayed in table 2.

The results for MCTS with 1000 and 10 000 iterations are obtained after considerable tuning of C_p and after selecting whether the scheme should be constructed forward or in reverse (i.e., the scheme is applied backwards [23]).

For smaller problems, we observe that the averages of SHC are on a par with MCTS. However, we see that the standard deviations of SHC are higher than MCTS. This is because for MCTS the first nodes are fixed rather fast, which limits the variety. Consequently, we expect SHC to outperform MCTS if several runs are performed in parallel. Indeed, this is what we see in the last column of table 2. The standard deviations of MCTS are often an order of magnitude smaller than those of SHC, so the benefits of running MCTS in parallel are smaller. We may conclude that SHC has a better minimal behaviour if run in parallel.

For our largest expressions, HEP(F_{13}), HEP(F_{24}) and HEP(b), we observe that SHC with 1000 iterations yields better results than MCTS with 10 000 iterations. For HEP(F_{24}), the average of SHC with 1000 iterations is about 20% better than the average for MCTS with 1000 iterations. In fact, the results are slightly better than MCTS with 10 000 iterations. If we take the $E_{min,4}$ into account, the expected value for HEP(F_{24}) is an additional 7% less.

The fact that SHC outperforms MCTS when the number of variables is larger than 23, may be due to the fact that there are not sufficient iterations for the branches to reach the bottom, making the choice of the last variables essentially random (see section 5 and [23]). This may also be the reason why for MCTS it is important whether the scheme is constructed forward or in reverse: if most of the performance can be gained by carefully selecting the last variables, building the scheme in reverse will yield better performance.

SHC is 10 times faster (in clock time) than MCTS, since most of the time is spent in the evaluation function. It is able to make reductions up to a factor of 26 for our largest expression.

[1] We only consider optimisations by Horner schemes and CSEE. Additional optimisations that are mentioned in [18], such as 'greedy' optimisations, can just as well be applied to the results of SHC.

7. Conclusions

Monte Carlo Tree Search (MCTS) with UCT has proven to be a good candidate to simplify large expression [18]. A downside of this method is that the constant C_p that governs exploration versus exploitation has to be tuned. The quality of the final scheme largely depends on this constant. We have modified the UCT algorithm so that C_p decreases linearly with the iteration number [23]. As a result, the final iterations are spent on *optimising* the end of the tree, instead of *exploring*. We show that using this modified UCT, the sensitivity to C_p is decreased by at least a factor 10 [23, 32]. Thus, the tuning is simplified.

Tree search methods, even with SA-UCT, have the problem that the beginning of the tree is optimised more than the end. For Horner schemes this does not lead to optimal solutions. Therefore we considered other algorithms that optimise uniformly. Since sampling is slow for our use case, tuning many parameters is infeasible. For this reason, we preferred straightforward algorithms over sophisticated ones. In the case of Horner schemes, we have found that one of the most basic algorithms, Stochastic Hill Climbing, yields the best results.

Stochastic Hill Climbing provides a search method with two parameters: (1) the number of iterations (computation time) and (2) the neighbourhood structure, which is a tunable parameter. We found that running half of the simulations with a neighbourhood structure that makes minor changes to the state (i.e., a single shift of a variable), and running the other half with a neighbourhood structure that involves larger changes (i.e., the mirroring of a random sublist) is a good strategy for all of our benchmark expressions. Consequently, only the computation time remains as an actual parameter. From our experimental results we arrive at three subconclusions: (1) SHC obtains similar results to MCTS for expressions with around 15 variables, (2) SHC outperforms MCTS for expressions with 24 or more variables, and (3) SHC requires ten times fewer samples than MCTS to obtain similar results. Therefore we may conclude that SHC is more than 10 times faster [31].

The result that a basic algorithm such as SHC performs well is surprising, since Horner schemes have at least two properties that make the search hard: (1) there are no known local heuristics, and (2) evaluations could take several seconds. In the previous sections we have shown that the performance of SHC is so good, because the state space of Horner schemes is flat and has few local minima.

The number of operations is linearly related to the time it takes to perform numerical evaluations. The difference between the number of operations for the unoptimised and the optimised expression is more than a factor 24 compared. As a consequence, we are able to perform numerical integration (via repeated numerical evaluations) at least 24 times faster.

For High Energy Physics, the contribution is immediate: numerical integration of processes that are currently experimentally verified at CERN can be done significantly faster. Our algorithms are implemented in the open source symbolic manipulation system Form 4.2 [10] and are used by multiple research groups.

Acknowledgements

This work is supported by the ERC Advanced Grant no. 320651, “HEPGAME”.

References

[1] Berends F A, Kuijf H, Tausk B and Giele W T 1991 *Nucl. Phys.* **B357** 32–64

[2] Stelzer T and Long W F 1994 *Comput. Phys. Commun.* **81** 357–371 (*Preprint* `hep-ph/9401258`)

[3] Campbell J M and Ellis R K 1999 *Phys. Rev.* **D60** 113006 (*Preprint* `hep-ph/9905386`)

[4] Boughezal R, Campbell J M, Ellis R K, Focke C, Giele W, Liu X, Petriello F and Williams C 2017 *Eur. Phys. J.* **C77** 7 (*Preprint* `1605.08011`)

[5] Fujimoto J, Ishikawa T, Kato K, Kaneko T, Nakazawa N, Shimizu Y, Vermaseren J and Yasui Y 2006 *Nucl. Phys. Proc. Suppl.* **160** 150–154 [,150(2006)]

[6] Horner W 1819 *A New Method of Solving Numerical Equations of All Orders by Continuous Approximation* (W. Bulmer & Co) dover reprint, 2 vols 1959 URL `http://books.google.nl/books?id=2fLpGwAACAAJ`

[7] Knuth D E 1997 *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms* (Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.) ISBN 0-201-89684-2

[8] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, Driessche G v d, Graepel T and Hassabis D 2017 *Nature* **550** 354–359 ISSN 0028-0836 URL `http:https://doi.org/10.1038/nature24270`

[9] Aarts E and Lenstra J K (eds) 1997 *Local Search in Combinatorial Optimization* 1st ed (New York, NY, USA: John Wiley & Sons, Inc.) ISBN 0471948225

[10] Ruijl B, Ueda T and Vermaseren J 2017 (*Preprint* `1707.06453`)

[11] Aho A V, Sethi R and Ullman J D 1988 *Compilers: Principles, Techniques and Tools* (Addison-Wesley)

[12] Leiserson C E, Li L, Maza M M and Xie Y 2010 Efficient Evaluation of Large Polynomials *In Proc. International Congress of Mathematical Software - ICMS 2010* (Springer)

[13] Breuer M A 1969 *Commun. ACM* **12** 333–340 ISSN 0001-0782 URL `http://doi.acm.org/10.1145/363011.363153`

[14] Wang P S 1978 *Mathematics of Computation* **32** 1215–1231 ISSN 00255718, 10886842 URL `http://www.jstor.org/stable/2006346`

[15] Wu W, Chen J and Feng Y 2014 *Science China Mathematics* **57** 2123–2142 ISSN 1869-1862 URL `http://dx.doi.org/10.1007/s11425-014-4850-y`

[16] Ceberio M and Kreinovich V 2004 *SIGSAM Bull.* **38** 8–15 ISSN 0163-5824 URL `http://doi.acm.org/10.1145/980175.980179`

[17] Adelson-Velsky G M, Arlazarov V L and Donskoy M V 1988 *Algorithms for Games* (New York, NY, USA: Springer-Verlag New York, Inc.) ISBN 0-387-96629-3

[18] Kuipers J, Plaat A, Vermaseren J and van den Herik J 2013 *Computer Physics Communications* URL `http://www.sciencedirect.com/science/article/pii/S0010465513001689`

[19] Lee C S, Wang M H, Chaslot G, Hoock J B, Rimmel A, Teytaud O, Tsai S R, Hsu S C and Hong T P 2009 *IEEE Trans. Comput. Intellig. and AI in Games* **1** 73–89

[20] Hayward R 2009 *International Computer Games Association (ICGA)* **32** 114–116

[21] Browne C, Powley E, Whitehouse D, Lucas S, Cowling P, Rohlfshagen P, Tavener S, Perez D, Samothrakis S and Colton S 2012 *Computational Intelligence and AI in Games, IEEE Transactions on* **4** 1–43 ISSN 1943-068X

[22] Kocsis L and Szepesvri C 2006 Bandit based Monte-Carlo Planning *In: ECML-06. LNCS 4212* (Springer) pp 282–293

[23] Ruijl B, Vermaseren J, Plaat A and van den Herik H J 2014 *Proceedings of ICAART Conference 2014* **1** 724–731 (*Preprint* `1312.0841`) URL `http://arxiv.org/abs/1312.0841`

[24] Kirkpatrick S, Gelatt C D and Vecchi M P 1983 *Science* **220** 671–680

[25] Kocsis L and Szepesvri C 2006 Discounted UCB. Video Lecture in the lectures of PASCAL Second Challenges Workshop 2006, Slides: `http://www.lri.fr/sebag/Slides/Venice/Kocsis.pdf`.

[26] Hashimoto J, Kishimoto A, Yoshizoe K and Ikeda K 2012 *Lecture Notes in Computer Science* **7168** 1 – 12 URL `http://id.nii.ac.jp/0023/00008786`

[27] Hoos H and Stützle T 2004 *Stochastic Local Search: Foundations & Applications* (San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.) ISBN 1558608729

[28] Glover F 1986 *Computers & Operations Research* **13** 533 – 549 ISSN 0305-0548 applications of Integer Programming URL `http://www.sciencedirect.com/science/article/pii/0305054886900481`

[29] Dorigo M and Gambardella L M 1997 *Trans. Evol. Comp* **1** 53–66 ISSN 1089-778X URL `http://dx.doi.org/10.1109/4235.585892`

[30] Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* 1st ed (Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.) ISBN 0201157675

[31] Ruijl B, Vermaseren J, Plaat A and van den Herik H J 2014 Why Local Search Excels in Expression Simplification (*Preprint* `1409.5223`) URL `http://arxiv.org/abs/1409.5223`

[32] Ruijl B, Vermaseren J, Plaat A and van den Herik H J 2014 *Proceedings of the 11th International Workshop on Boolean Problems* URL `http://arxiv.org/abs/1405.6369`