The HEPGAME project

J.A.M. Vermaseren

Nikhef

in collaboration with

J. Kuipers (Nikhef/Google), J. van den Herik and A. Plaat (Tilburg University)

- Introduction
- Particle Physics
- FORM
- Recursion Relations
- Games, Minimax and MCTS
- Simplifying Formulas
- The HEPGAME Project

Introduction

Physics, and in particular particle physics, has always made intensive use of computers. Many things we see these days were either used first or invented by particle physicists. The best known example is the World Wide Web. Also computer algebra has many of its roots there because of the extreme complexity and size of the calculations in theoretical particle physics:

M. Veltman	Schoonschip
A. Hearn	Reduce
S. Wolfram	SMP and Mathematica
J.V.	FORM

Today we will discuss an attempt to use recent techniques in the field of artificial intelligence/game theory to solve very complicated systems of equations that occur in the calculation of some reactions in particle physics.

Particle Physics

Particle physics is a part of physics that investigates the fundamental building blocks of matter and the forces between them. These building blocks are:

quarks There are 6 types of quarks: u,d,s,c,b,t. Each comes in three varieties called color.

leptons Charged leptons are the electron(e), the muon(μ) and the tau(τ). The chargeless leptons are called neutrino's and they come also in three varieties (ν_e, ν_μ, ν_τ).

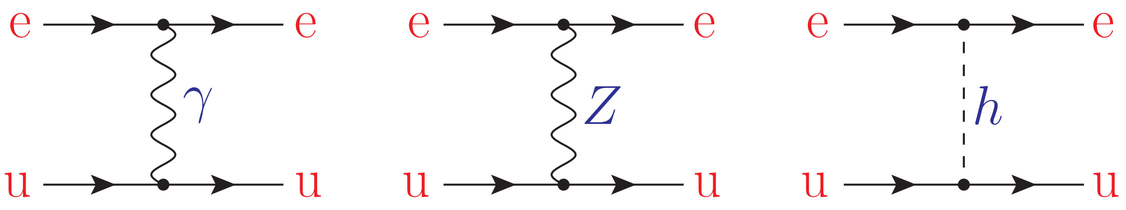
force mediators The photon (electromagnetism)(γ), the $W^\pm$ and Z (weak forces), the gluon (strong or color forces)(g) and the graviton (gravity)(G).

the Higgs particle A necessary ingredient(h) to keep the best model we have (excluding gravity) physical (ie finite).

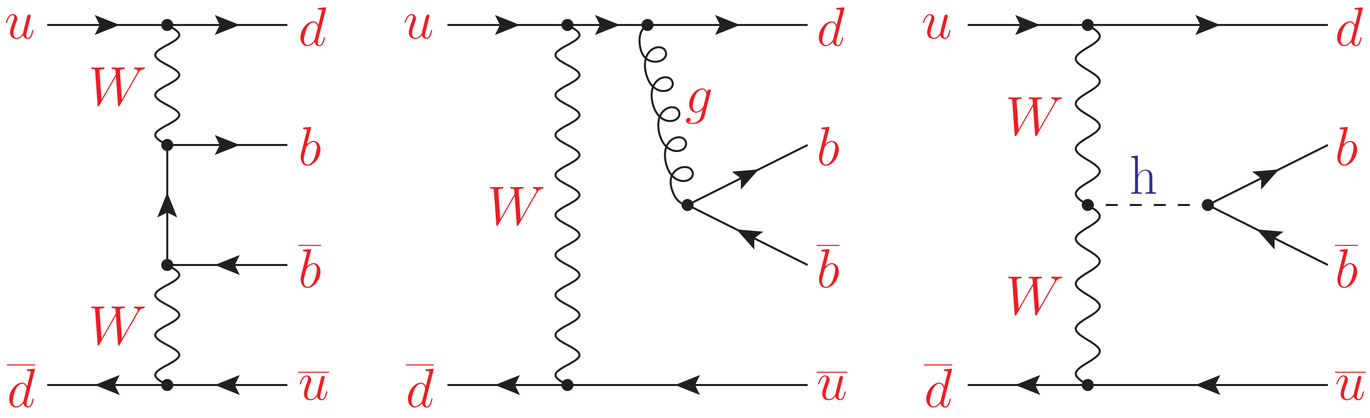
The quarks and the leptons come in two varieties: particles and anti-particles. The force mediators and the Higgs particle are their own anti-particles. (With the W we have a W^+ and a W^- which are each others anti-particles).

Currently there is no good theory that incorporates all interactions (forces), but we have a very good theory that includes everything except for gravity. This is called the "standard model". It predicted the Higgs particle although it could not predict its mass.

When particles interact with each other, we call that a reaction. These interactions can be seen as particles exchanging force mediators (or Higgs particles) as in:



Also a particle and an anti-particle can annihilate and form one or more force mediators, or force mediators can produce a particle anti-particle pair.



The above pictures are called Feynman diagrams. In a proper theory, each element in a diagram (lines and vertices) represents an element of a formula and when you want to calculate a reaction, you have to write down all diagrams that can contribute to it, write for each diagram its complete formula, square the sum of the diagrams, and work out the formulas. This does involve quite some mathematics.

These formulas can become rather big in two ways:

- It can happen that one starts with one (complicated) diagram, both the input and the output fit on a few lines, but at intermediate stages one could have many Gbytes of formula.
- One may end up with a formula that takes millions of terms and this formula needs to be integrated over by numerical means.

In both cases it should be clear that this is way beyond manual processing.

Computers!

FORM

For manipulating the formulas of the previous section (and many other types of formulas) we use symbolic manipulation programs, also called computer algebra systems (CAS).

The most widely known systems are Mathematica and Maple. They are designed to act a bit like the way people work by hand and serve as a "mathematicians work bench". Their generality however makes them pay dearly in efficiency. For most applications this is no objection.

If, however, one needs to do truely large calculations, it is better to use a different system which is called FORM. This is designed for being fast and handling very big formulas. The trade-off is that it uses a completely different programming model and a number of complicated things are harder to program. A big advantage of FORM is that it is free and open source (http://www.nikhef.nl/∼form).

A few examples of (simple) FORM programs are:

```
    Symbols a,b;
    Local F = (a+b)^5;
    Print;
    .end

Time =       0.00 sec    Generated terms =          6
               F         Terms in output =          6
                         Bytes used      =        204

   F =
      b^5 + 5*a*b^4 + 10*a^2*b^3 + 10*a^3*b^2 + 5*a^4*b + a^5;
    Symbols a,b,c,d,e,f,g;
    Local F = (a+b+c+d+e+f+g)^30;
    .end

Time =       6.51 sec    Generated terms =    1947792
               F         Terms in output =    1947792
                         Bytes used      =   99425612
    Symbols n;
    CFunction f;
    Local Fibonacci = f(21);
    Repeat;
      id f(n?{>2}) = f(n-1)+f(n-2);
    EndRepeat;
    id f(1) = 1;
    id f(2) = 1;
    Print;
    .end

Time =       0.05 sec    Generated terms =      10946
       Fibonacci         Terms in output =          1
                         Bytes used      =         20

   Fibonacci =
      10946;
```

The next example is more cryptic. It computes all possible drawings for the eighth finals of the champions league in December 2012. It vetoes teams of the same nationality playing against each other. There were two groups of eight teams and teams from the first group had to play against teams of the second group. Each group had two Spanish, one French and one Italian team. Other teams could not cause conflicts.

```
    Tensor f;
    Index  i1,...,i8;
    Local  F = f(i1,...,i8)*e_(i1,...,i8)*e_(1,...,8);
    Contract;                        * Generates the 8! permutations
    id f(i1?{1,2},?a) = 0;           * First Spanish team in group 2
    id f(i1?,i2?{1,2},?a) = 0;       * Other Spanish team in group 2
    id f(i1?,i2?,3,?a) = 0;          * Italian teams
    id f(i1?,i2?,i3?,4,?a) = 0;      * French teams
    .end
```

```
Time =        0.05 sec    Generated terms =      17088
                F         Terms in output =      17088
                          Bytes used      =     527676
```

The program first generates all $8! = 40320$ possibilities and then eliminates the 'forbidden' combinations. There are 17088 possibilities left. Hence the chance that the trial run and the real drawing would give the same result is 1/17088. Note how short such a program can be if you know what you are doing.

The next example is one that shows the use of a (physics) library for calculating a category of extremely difficult Feynman diagrams. The output does not show all the steps, just a few crucial ones. The program:

```
#define LONGINT "1"
#include mincer3.h
Vectors Q,p1,...,p8;
Symbols n1,...,n9;
CFunction I;
Local F = I(5,5,5,5,5,5,5,5,5);
id I(n1?,...,n9?) = Q.p2^n9/<p1.p1^n1>/.../<p8.p8^n8>
 *Q.Q^(-6+n1+...+n8-n9)*ep^3;
#call integral(no)
print;
.end
```

and the output (with selected intermediate statistics):

```
                     .
                     .
Time =      60.48 sec    Generated terms =     204053
               F         Terms in output =     202857
       noplane recursion Bytes used      =   44371728
                     .
                     .
Time =    1160.61 sec    Generated terms =   14082803
               F         Terms in output =     250615
        Second recursion Bytes used      =   69574612
                     .
                     .
Time =    1644.60 sec    Generated terms =         19
               F         Terms in output =         19
     Expression G scheme Bytes used      =       1928

Time =    1644.60 sec    Generated terms =          6
               F         Terms in output =          6
                         Bytes used      =        308
```

```
   F =
    - 3240951453409071499704259699924964673271060630291648832174 75
         /48443834647717410851389 44
    + 458253263713157238125/96*ep^-3
    + 114859844526235874013420 4625/33606144*ep^-2
    - 460871712641734469235346182768 61775/12241101400399872*ep^-1
    + 27194951372965943042395499580000000 0*z5
    - 171778183525143046757527348768500668 75/96*z3;
```

We see here an example of a ‘simple’ answer with complicated intermediate results.

In the HEPGAME project we will be using FORM as the main tool for formula manipulation. We will also use the fact that we have complete control over it to extend it with new generic concepts that will facilitate our research.

Recursion Relations

Some categories of Feynman diagrams can be computed by the use of recursion relations. One of the best known recursion relations in science is the Fibonacci series:

```
1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,
   6765,10946,...
```

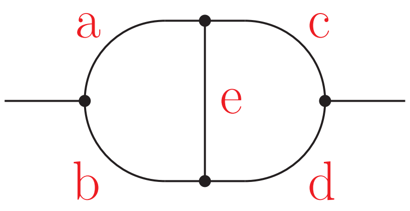
It is defined by

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \\ F(n) &= F(n-1)+F(n-2) \end{aligned}$$

We also call these difference equations. This would be a second order difference equation, because it goes down to $F(n-2)$. To solve an m-th order difference equation, you need m startup values, and like with differential equations, some mathematatics. Sometimes such equations can be solved as in

$$F(n) = \frac{(1+\sqrt{5})^n-(1-\sqrt{5})^n}{2^n\sqrt{5}}$$

Things can become much more complicated when there are more variables. Take for instance the following diagram which represents a complicated integral:

$I(a, b, c, d, e) =$ 

For this integral one can write the recursion relation (never mind how):

$$\begin{aligned}(4-2\epsilon-a-b-2e)\ I(a,b,c,d,e) \;=\; &+aI(a+1,b,c,d,e-1) - aI(a+1,b,c-1,d,e)\\ &+bI(a,b+1,c,d,e-1) - bI(a,b+1,c,d-1,e)\end{aligned} \tag{1}$$

The variable ϵ is a parameter in the theory.
We see that either c, d or e is lowered by one while either a or b is raised by one.
If the variables c, d, e are positive integers, repeated application of this recursion relation will make one of them zero eventually. As it turns out, if one of the parameters a, b, c, d, e becomes zero or a negative integer, we have a closed form solution for this integral. Hence, if either c, d, e or a, b, c are positive integers, we can solve the integral by repeated use of the recursion relation.

Such recursions are easy to program, but things can run out of hand at times:

```
    Symbols a,b,c,d,e,ep,x1,x2;
    CFunctions I,rat;
    PolyRatfun rat;
    Local expr = I(4,4,4,4,4);
    repeat id I(a?pos_,b?pos_,c?pos_,d?pos_,e?pos_) =
              rat(1,4-2*ep-a-b-2*e)*(
               +a*I(a+1,b,c,d,e-1)-a*I(a+1,b,c-1,d,e)
               +b*I(a,b+1,c,d,e-1)-b*I(a,b+1,c,d-1,e)
              );
    .end
```

```
Time =         6.57 sec    Generated terms =     102070
            expr           Terms in output =        160
                           Bytes used      =      51104
```

The next level of complication is that for the calculations we like to do one of the parameters might not be an integer. If it is a or b, no problem. If it is c or d, also no problem, because we flip the diagram. When e is not an integer, we do have a problem.

We can write a second recursion relation. It is

$$\begin{aligned}(4-2\epsilon-2a-b-e)\, I(a,b,c,d,e) &= bI(a-1,b+1,c,d,e) - bI(a,b+1,c,d,e) \\ &+ eI(a-1,b+1,c,d,e) - eI(a,b,c,d,e+1)\end{aligned} \quad (2)$$

We can rearrage this equation into

$$\begin{aligned}I(a,b,c,d,e) &= I(a-1,b,c,d,e) + \frac{e}{b-1} I(a-1,b,c,d,e) \\ &- \frac{e}{b-1} I(a,b-1,c,d,e+1) - \frac{4-2\epsilon-2a-b-e}{b-1} I(a,b-1,c,d,e)\end{aligned}$$

This relation can be used to bring either the second argument (b) down to one, or make the first argument zero. When b is one, nothing can be done.

Using symmetry, we can do the same for the first, third and fourth argument, leaving us with integrals that either have a zero (and hence can be done) or the integral $I(1,1,1,1,e)$.

Some manual inspection reveals that if we combine the first equation (1) with the second (2) and a flipped version of the second that can be used to lower the first parameter (2a), we set the parameters a, b, c, d equal to one, and we replace e by $e-1$ in the last two equations, we obtain a new equation that is precisely what we want to have:

$$\begin{aligned} I(1,1,1,1,e) &= -\frac{4-2e-4\epsilon}{2-2e-2\epsilon} I(1,1,1,1,e-1) \\ &\quad + \frac{2}{2-2e-2\epsilon}\left(I(0,2,1,1,e-1) - I(2,1,0,1,e)\right) \end{aligned}$$

which is a relation that can be used to bring e in the range between 1 and 2. Usually this will be $1+\epsilon$ as e starts as an integer plus ϵ.

At this point we have reduced the integrals to the single integral $I(1,1,1,1,1+\epsilon)$. This integral has to be done by different means and hence is called a master integral. In the project we will run into very few master integrals (only two) and fortunately those integrals are known.

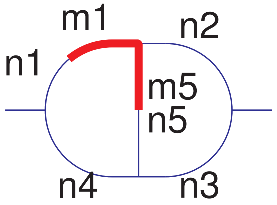
The above shows basically what we will encounter also in the more complicated cases: we obtain a number of relations (dozens) and then we have to make various combinations and transformations of these relations to reduce all integrals either to integrals of a simpler type, or to a master integral. It may happen that the simpler integrals still need similar techniques to be reduced to even simpler integrals, until we run into integrals that we know how to do.

The big problem is to select each time which equations to combine.

Also, after combining equations the new equations may become rather lengthy (like many thousands of terms). This is why we use computer algebra.

Finding the proper order and combination that will lead to an acceptable solution is by far the most people-time consuming part in calculations that use this method.

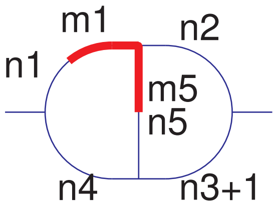
Even though the equations may be very complicated, one can be artistic about it. See the next definition.



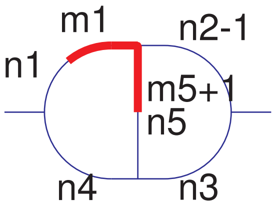
$$= \int d^Dp_1\; d^Dp_2 \frac{1}{(p_1^2)^{n_1}(p_2^2)^{n_2}(p_3^2)^{n_3}(p_4^2)^{n_4}(p_5^2)^{n_5}((p_1+P)^2)^{m_1}((p_5+P)^2)^{m_5}}$$

This is messy if we write the equations in terms of integrals, but in pictures we obtain:

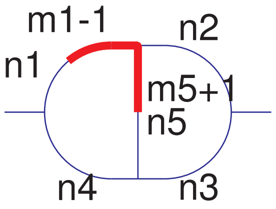
n_3Q^2



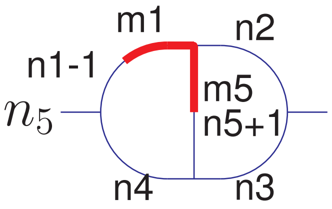
$= m_5$



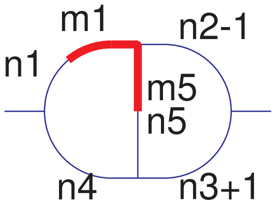
$- m_5$



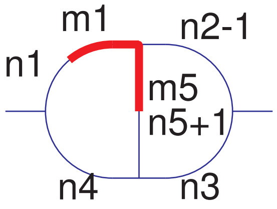
$- n_5$



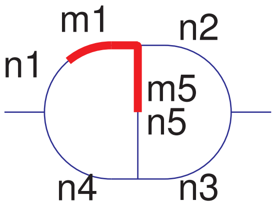
$+\ n_3$



$+ n_5$



$+\ (n_5 + n_3 + 2n_2 + m_5 - 4 + 2\epsilon)$



Again, this is a very short and ‘simple’ equation.

Games, Minimax and MCTS

In the early days of computers the ‘intelligence’ of the computer was measured by how well the computer could play chess.
Eventually however a brute force approach, in combination with a number of algorithms made it possible to beat the world champion.
This was not what we liked to call (artificial) intelligence.
Already attention was directed towards the game of Go, because there brute force was not going to do the job and the strongest programs were still at a level that might be called “intermediate amateur”.

How would one construct a computer program to play a game? There are various approaches.

1. Try by means of pattern recognition or other heuristic rules to find a move that according to these rules must be good. If possible, look to avoid immediate disasters (like mate in one).
2. Try all possible moves, try all possible answers by the opponent and then, for each of them, evaluate the position. Select for each of our moves the worst that the opponent can do to us, and out of those worst cases select what is best for us.
3. in a variation on the last one, try a number of the best ones and repeat this procedure for each of them. Repeat this a few times, so that the "better candidates" will be investigated a bit deeper.
4. Combining all of this we may not try all moves, but reject obvious bad patterns immediately. This can make the number of possibilities much smaller.

The second method is the principle of Minimax. Whereas the opponent is trying to minimize our score in the evaluation, of those minimal scores we select the maximum.

Going through the gametree this way can be very time consuming and hence we have to limit our searches. But if we consistently select what the computer thinks to be the best move, we may miss many effects. One is the famous horizon effect. In chess one can imagine thinking no more than n moves deep and having then a bishop more, while in move n+1 ones queen is lost. Another is the opposite. A branch looks awful and hence it not visited again, but actually there is a magnificent move waiting there that reverses the whole position. For a human player these cases may even not be very difficult to find.

The problem with patterns is that computers are notoriously weak at pattern recognition. In the game of Go it is already very difficult to give a good definition of a group, and even then, what at one stage was a good solid group may at a later stage be cut in pieces.

For a minimax method to work well, one needs a good evaluation function. If the evaluation function is perfect, method two will be perfect. There will not be any horizon effects. Unfortunately.....

Time to come up with a new method.

Assume that we compensate for the lack of an evaluation function by playing the game out randomly and then counting the score.

This sounds completely crazy!

But we add something to it: We will try this many times (this is the essence of a Monte Carlo method) and we will let the part of the tree that we will try depend on the results of previous attempts.

The selection of the next attempt will be according to the UCT (Upper Confidence level for Trees), introduced by Kocsis and Szepesvári in 2006:

$$UCT_i = \langle x_i \rangle + 2C_p \sqrt{\frac{2 \log n}{n_i}}$$

At any point in the tree the child with the highest UCT value is selected. Here

$\langle x_i \rangle$ is the average score of child i over the previous traversals

n_i is the number of times child i has been visited before

n is the number of times the node itself has been visited

C_p is a problem-dependent constant. Should be determined empirically.

$$UCT_i = \langle x_i \rangle + 2C_p\sqrt{\frac{2\log n}{n_i}}$$

The first term in the equation favours trying previously successful branches in the tree. This is called exploitation. The second term favours branches that have not been visited much before (if never, the term is even infinite). This is called exploration. The value of C_p determines the balance between the two.

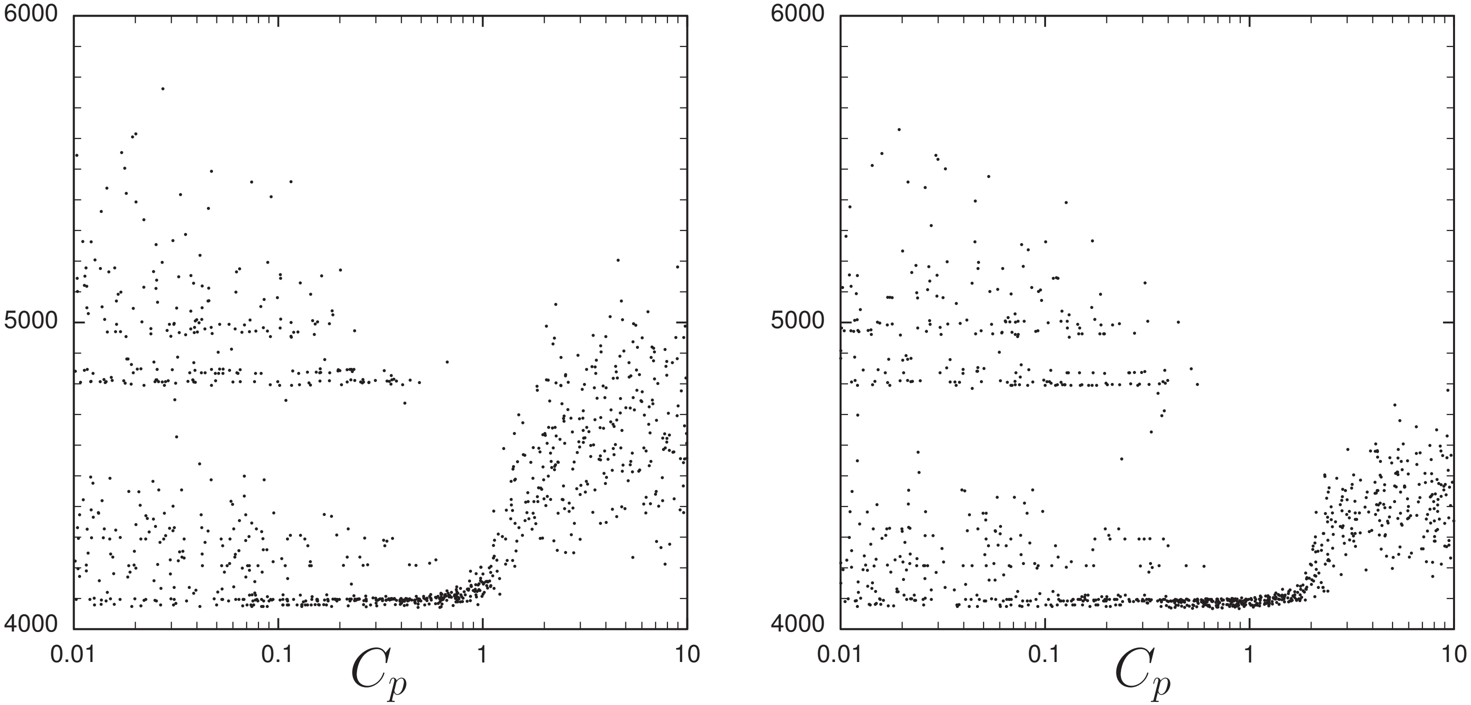
This approach can be successful if positive outcomes are clustered in the tree.

In games this often works because a good move will usually leave many more favourable endpositions than a bad move.

When the value of C_p is too small, we will only sample one seemingly good branch in the tree and eventually end up in a local maximum.

When the value of C_p is too big, we will basically be sampling randomly and forget to pursue branches that seem promising.

Let us have a look at this in an example based on what we will discuss in the next section. The aim is to have as small an outcome as possible.



The graphs are based on 1000 runs, each with a random value for C_p. The left one has 1000 tries in the MCTS, the right one has 3000 tries in the MCTS.

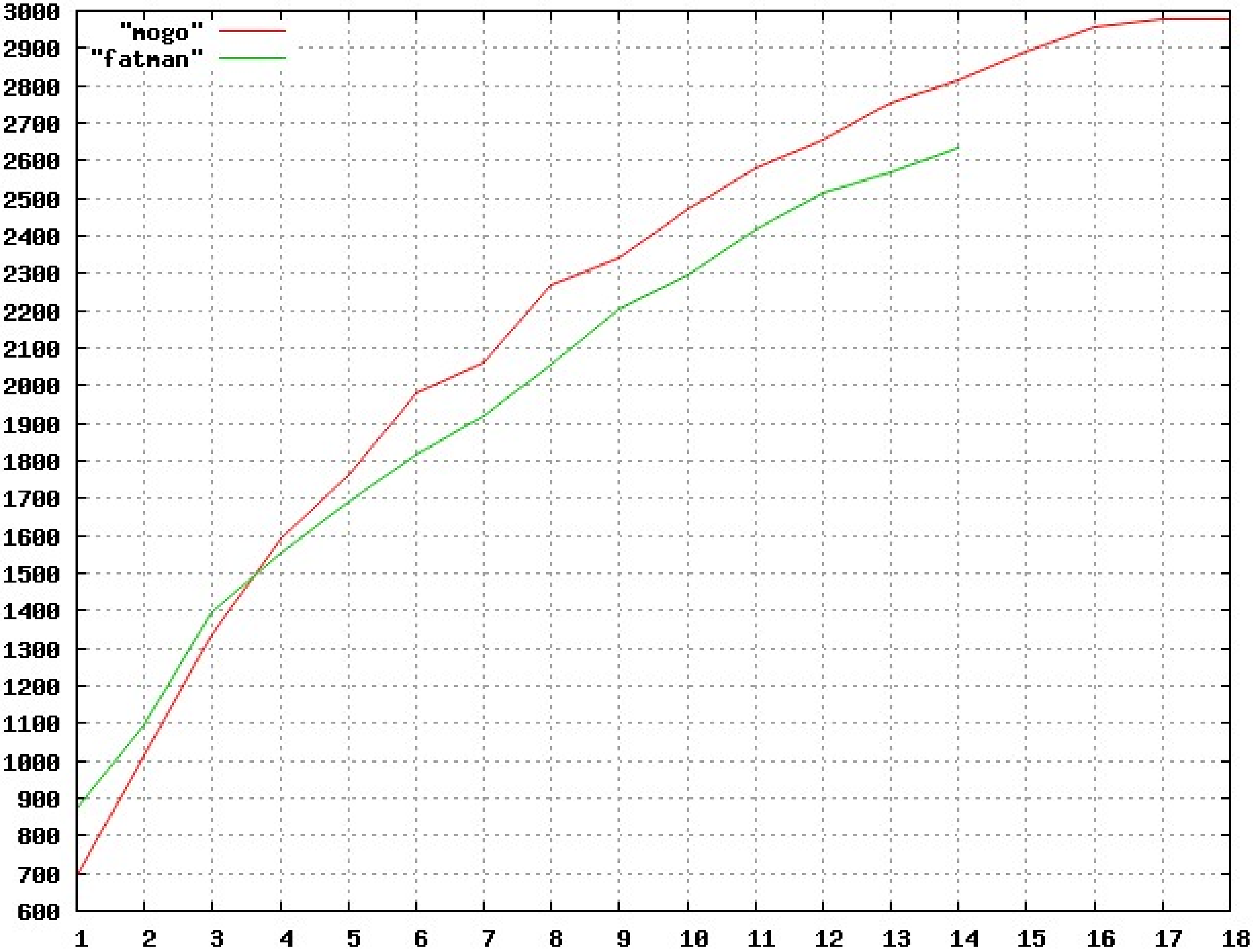
We see that in the right graph the right side is lower than in the left graph. This is due to the larger statistics. In the left side of the graphs we can see the local minima. The valley in the middle has values of C_p that guarantee an optimal result in (nearly) each run.

One of the nice things about MCTS is that is can make use of many processors simultaneously. The major problem is the updating of the tree information. There are of course also other problems to be considered, but that would take us too far.

The scalability of such Monte Carlo algorithms was investigated in 2008 by Don Dailey. He tested two programs on a 9x9 board (and had them play many games against GnuGo of which the strength was known). The number of tree evaluations was:

Mogo	$64 \times 2^{(N-1)}$ simulations
FatMan	$1024 \times 2^{(N-1)}$ simulations

Both programs used MCTS. Mogo was at the time the strongest program which can be seen from the fact that it needed fewer tree evaluations for comparable strength. The result is in the figure.


"mogo"
"fatman"
3000
2900
2800
2700
2600
2500
2400
2300
2200
2100
2000
1900
1800
1700
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

The application of MCTS has caused a big revolution in game theory. Specifically in Go.

For the first time a computer managed to beat a top ranked professional Go player with a 9 stone handicap. (using 800 cores on the Huygens supercomputer in a match organized by Prof. van den Herik).

Of course some simple heuristics are put in to avoid obviously bad choices.

The current status is that the program Zen (Zen19) has managed for the first time to beat a very strong professional Go player (Takemiya, a former champion) at 4 stones handicap. This was totally unexpected.

In some things the computer programs are relatively weak though, but to go into that would be rather technical.

Simplifying Formulas

Often the result of symbolic computations is a lengthy formula that needs further numerical processing. In the case of a Monte Carlo integral this may mean many thousands or even many millions of function evaluations. It is therefore important that the output expression is written as economically as possible. This is called simplification.

The basic example is an expression in a single variable as in

$$\begin{aligned} F &= a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \\ &= ((((a_5x + a_4)x + a_3)x + a_2)x + a_1)x + a_0 \end{aligned}$$

In the first form there are 5 additions and at least 9 multiplications, while in the second form there are 5 additions and 5 multiplications. Such a rewrite is called a Horner scheme.

Things become messier when more than one variable is involved. In that case we can use a Horner scheme in one of the variables and consider that the coefficients in it are expressions in the remaining variables. Then these expressions are treated in a similar way. The problem is that the efficiency of such a multivariate Horner scheme may depend on the order in which the variables are treated. The differences can be quite considerable.

As an example we take the following polynomial in x,y,z:

$$a(x,y,z) \;=\; y-3x+5xz+2x^2yz-3x^2y^2z+5x^2y^2z^2$$

Evaluation of this polynomial takes 18 multiplications and 5 additions (18, 5).

Horner Order	Formula	Operations
x,y,z	$y+x(-3+5z+x(y(2z+y(z(-3+5z)))))$	8,5
x,z,y	$y+x(-3+5z+x(z(y(2-3y)+z(5y^2))))$	9,5
y,x,z	$x(-3+5z)+y(1+x^2(2z)+y(x^2(z(-3+5z))))$	11,5
y,z,x	$-3x+z(5x)+y(1+z(2x^2)+y(z(-3x^2+z(5x^2))))$	14,5
z,x,y	$y-3x+z(x(5+x(y(2-3y)))+z(x^2(5y^2)))$	11,5
z,y,x	$-3x+y+z(5x+y(2x^2+y(-3x^2))+z(y^2(5x^2)))$	14,5

While doing this, one may run into common subexpressions like $-3+5z$, x^2 in some of the orderings. This can bring the number of operations down, but still the differences between the orderings remain.

The easy solution would be to try all orders, but once the number of variables becomes large, this is not practical ($20! = 2432902008176640000$).

The various orderings of 20 variables define a tree. Selecting the first variable can be done in 20 ways, the second in 19 ways, etc. The challenge is to find a path through the tree that gives an optimal or near optimal solution.

Searching through trees to find a good/best solution is what MCTS is all about.

We have made an implementation of various simplification algorithms in FORM so that it can produce output code based on the results of a search with MCTS in combination with other improvement algorithms like a search for common subexpressions or something called greedy optimization.
The results are better than anything we could find in the literature.
We will show some examples of the various optimization levels first. Without MCTS the Horner ordering is by 'occurrence' which means that the variables are ordered by the number of occurrences in the formula. Actually the program tries two orderings: occurrence and anti-occurrence.

O1 Occurrence+anti-occurrence followed by common subexpression elimination.

O2 Occurrence+anti-occurrence followed by common subexpression elimination. After that a 'greedy' optimization.

O3 MCTS with common subexpression elimination and greedy optimizations.

```
    Symbols x,y,z;
    Local F = 6*y*z^2+3*y^3-3*x*z^2+6*x*y*z-3*x^2*z+6*x^2*y;
    Format O1,stats=on;
    Print;
    .end
      Z1_=y*z;
      Z2_= - z + 2*y;
      Z2_=x*Z2_;
      Z3_=z^2;
      Z1_=Z2_ - Z3_ + 2*Z1_;
      Z1_=x*Z1_;
      Z2_=y^2;
      Z2_=2*Z3_ + Z2_;
      Z2_=y*Z2_;
      Z1_=Z2_ + Z1_;
      F=3*Z1_;
*** STATS: original  1P 16M 5A : 23
*** STATS: optimized 0P 10M 5A : 15
```

```
      Z1_=z^2;
      Z2_=2*y;
      Z3_=z*Z2_;
      Z2_= - z + Z2_;
      Z2_=x*Z2_;
      Z2_=Z2_ - Z1_ + Z3_;
      Z2_=x*Z2_;
      Z3_=y^2;
      Z1_=2*Z1_ + Z3_;
      Z1_=y*Z1_;
      Z1_=Z1_ + Z2_;
      F=3*Z1_;
*** STATS: original  1P 16M 5A : 23
*** STATS: optimized 0P 9M 5A : 14
```

```
      Z1_=x + z;
      Z2_=2*y;
      Z3_=Z2_ - x;
      Z1_=z*Z3_*Z1_;
      Z3_=y^3;
      Z2_=x^2*Z2_;
      Z1_=Z1_ + Z3_ + Z2_;
      F=3*Z1_;
*** STATS: original  1P 16M 5A : 23
*** STATS: optimized 1P 6M 4A : 12
```

With simple expressions that have only a few variables the MCTS can use brute force and try all possibilities. When there are more variables this is of course not possible and it tries a default 1000 times, unless the user specifies a different number.

The next example is a 12x12 determinant with 14 variables. The expression is rather large. After some tests we decided to put the MCTS constant to 0.1. The output has

	Operations	Time
Original	142711	0.00 sec
O1	20210	0.41 sec
O2	16398	5.37 sec
O3	11171	183.61 sec

In the literature we found here 19148 operations obtained in 22 sec on a probably slightly faster computer.

Of course one could argue that the extra time due to the MCTS may cost more than one will gain in the numerical program. This is entirely a function of the number of function evaluations needed. In addition some time is gained back by a much faster compilation. And as extra benefit: fewer operations means also a shorter executable. We have had already a case for which it allowed us to get the executable program 'smaller' than 2 Gbytes this way (bigger than that would not run).
One should notice the statistical nature of the MCTS procedure. This may mean that different runs will give different results. The above result is neither our best result, nor our worst.

The simplification is very nice to study the MCTS procedure by itself.

The HEPGAME Project

The HEPGAME project: try to combine all of the above to create new techniques or methods to do calculations of Feynman diagrams.

Let us start with some history.

In 2004 a big calculation, involving some 20 fte and more than a year of CPU time (exclusively Form) was finished by S.Moch, A.Vogt and J.V. This calculation drew much attention in the particle physics community and it is currently at the basis of all calculations of processes at the LHC (the collider at CERN that recently discovered the Higgs particle) to the 1% level. Such calculations are needed to make optimal use of the machine, because this is the accuracy that the experiments can obtain eventually. Previously calculations could only be done to an accuracy of about 10%.

It was noticed that the way used for solving the systems of recursion relations has very much in common with the way one approches a game of Go. Hence it was proposed that if one would have a good strategy unit for playing Go by computer, one might automate the derivation of the programs needed for these calculations to a high degree.

At the time not much was done with this idea. It is something that requires good knowledge in several fields, most of which are familiar to me, but of game theory and how this is implemented in computers I do not know very much.

More recently my attention was drawn to the ERC advanced grants. These grants are awarded on the basis of personal reputation and innovative, potentially risk bearing ideas. Multi-disciplinarity is appreciated. They are very prestigious and involve much money (up to 2.5 million Euros). Of course one needs to show sufficient expertise to make a chance, because in a multi-disciplinary jury there will be experts in several fields giving their opinion. Hence it was only natural to look for an expert in game theory with whom to enter into a collaboration. Because I used to be a decent Go player (by Dutch standards) and had contacts in the past with people making Go programs, I had heard the name van den Herik and contact was made. The idea was received enthousiastically. Because of his expertise Prof. Plaat also joined the collaboration and he spent much time explaining a number of concepts. Together we prepared a proposal and it was submitted in Februari 2012. Finally in December it was approved, but some cuts were applied in the budget we requested. Such cuts are not uncommon.

The basic idea of the project is to combine elements of game theory with techniques used in theoretical particle physics to create new methods to compute Feynman diagrams and use these methods to do a number of physics calculations that were impractical/impossible till now.

The major objective is to change the 20 fte + 1 year of CPU time into something like 1 fte + 20 years of CPU time. The last is much easier to come by.

Because the calculations – or the parts we are interested in – are purely analytical, we need a powerful computer algebra system that has to be extended/programmed in such a way that all needed AI can be implemented and all needed physics can be run. The best system for this is FORM because it is (a) by far the most powerful for such manipulations, (b) it is open source and (c) we have all the needed expertise for using and extending it. Creating the intended extensions will introduce completely new concepts in the field of computer algebra (one of them is in the planning stage already).

As we see, the center of the project is FORM with a physics wing and a game theory wing.

How does this work in practise?

1. Set up the basics of one or more physics calculations with varying degree of complexity.
2. For each calculation we will have a large number (20-500?) of sets of equations. Each set may have 20-100 equations. Bring these into a standard form for further processing (we have to determine what is a good standard form).
3. Set up a good MCTS program to guide FORM through solving the system. Of course other techniques than MCTS are also welcome. It should be clear that this step will require much experimentation.
4. A solved system defines a set of substitution statements to calculate the diagrams of a given type by use of FORM. When all systems have been solved, we can do the calculation.

The expertise required for the third step is completely different from that needed for the first and fourth steps. Therefore, it seems best to split the project into two groups, one concentrating more on the physics and the other concentrating more on the game theoretical aspects. The games group is best off under the wings of the game theorists (Profs van den Herik and Plaat, hence in Tilburg) while the physics group is best located at Nikhef. This leaves of course a logistics problem, because much innovation is the result of frequent discussions. Hence in the project we allocated travel money to allow for weekly gettogethers. Also many discussions can be handled either by e-mail or a Skype connection. Nikhef has also good facilities for tele-conferencing.

The original proposal asked for 18 fte in the form of postdocs and 8 fte in the form of 2 PhD students. The jury decided to trim this down to 10 fte for postdocs and 8 fte for the PhD students. This leads to the two subgroups being:

Nikhef J. Vermaseren, 10 fte (postdocs), visitor money for calling in physics expertise to get the maximum out of the results of the calculation. (e.g. A.Vogt and S.Moch).

Tilburg J. van den Herik, A. Plaat, 8 fte (2 Phd students), visitor money to have occasional visits by experts in the field of AI/game theory.

Both groups will have travel money to send its members to conferences and/or schools.

Of course a proposal like HEPGAME concentrates on a fixed aspect of the science involved. At any time it is possible that one discovers new interesting spinoffs. This has been the case already. After the proposal was submitted we (Jan Kuipers and JV) decided to have a look at formula simplification in Form. As it turned out, using what was learned while writing the proposal (plus a few sessions with Prof. Plaat), we could make good use of MCTS as explained already. It is expected that there are more fields that can use it. And if we discover completely new methods in game theory, there may be even more fields benefitting.

The formula simplification also provides a rather clean laboratory to study MCTS itself. This may lead to techniques that allow us to select its parameters in an automated way. And maybe it will lead to other methods to search through the tree.

The end is not in sight!