Genetic Programming and Its Application to HEP

Computing in High Energy and Nuclear Physics 2004

30 September 2004

Eric W. Vaandering

`ewv@fnal.gov`

Vanderbilt University

and

FOCUS Collaboration

CHEP'04
INTERLAKEN

Overview

- Introduction to Genetic Programming
  - Populations and Generations
  - Mutation and Crossover
  - Fitness and Natural Selection
- Genetic Programming applied to the doubly Cabibbo suppressed decay $D^+ \to K^+\pi^+\pi^-$

What is Genetic Programming

Genetic programming is a machine learning algorithm based on two assumptions:

To find the best solution to a problem, maybe we should take a clue from biology and the evolutionary process. ($\longrightarrow$ Genetic Algorithms)

Since we will use computer programs to implement our solutions, the *form* of our solution should *be* a computer program. ($\longrightarrow$ Genetic Programming)

Genetic Programming applies a biological model which includes reproduction, mutation, and survival of the fittest to automatically discover computer programs.

CHEP'04
INTERLAKEN

Genetic Programming: Defined

Genetic Programming is a probabilistic search algorithm that iteratively transforms a set (population) of programs, each with an associated fitness value, into a new population of offspring programs using the Darwinian principle of natural selection and operations that mimic naturally occurring genetic operations, such as sexual recombination (crossover) and mutation.

- Pioneered by John Koza in 1989
- Reference: *Genetic Programming: On the Programming of Computers by Natural Selection* (1992)
- Since 1992, more than 3,000 papers applied to a wide range of problems in many disciplines

Populations and Generations

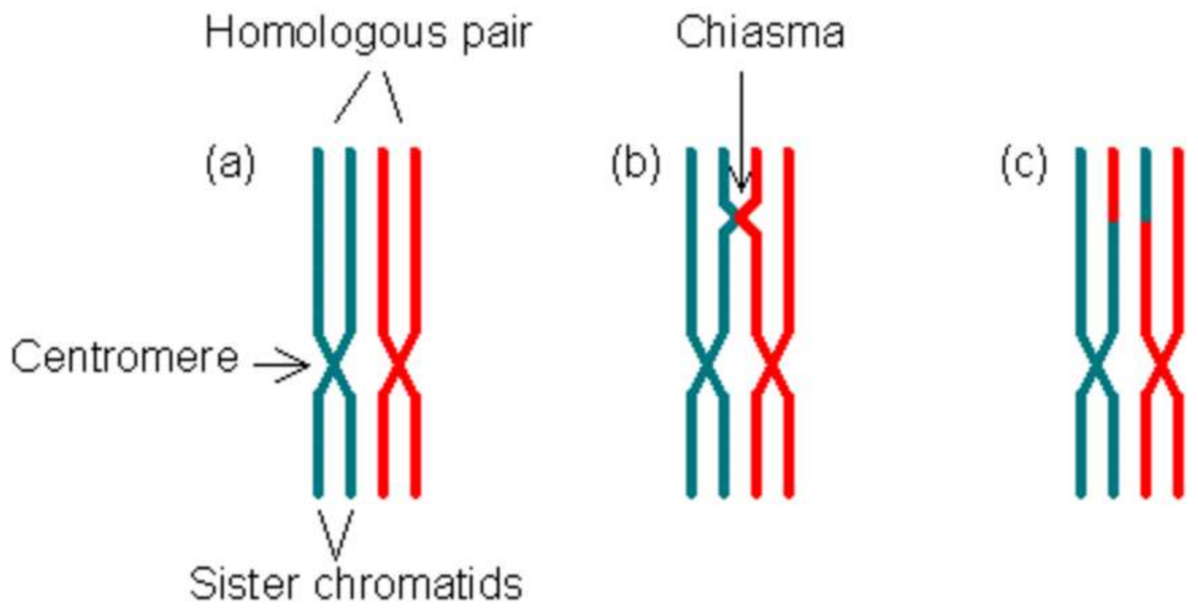
Genetic Programming works by transforming one group of individuals (programs) in generation n into another group of individuals in generation $n+1$. There are typically a few hundred to a few thousand programs per generation.

The initial programs in the 0^{th} generation are generated completely randomly.

Typically the number of individuals in each generation is the same. Usually no duplication is allowed in the 0^{th} generation. Duplication *is* allowed in later generations. (Diversity decreases.)

Gene Cross-over and Mutation

Biological
(DNA)
Cross-over



Mutations in nature change the genetic code for a small region of DNA. Usually are harmful or neutral; occasionally helpful (creates a better/different protein).

Mutations can restore lost (or never present) diversity.

These two processes, combined with natural selection,
drive biological evolution.

Preparatory Steps

To prepare to solve a problem with Genetic Programming, two steps are necessary:

- Define a series of functions
  - Some functions may return a variable or input
  - Other functions may perform an operation
    - $+$, $-$, $>$, $<$ are all “functions”
    - So are IF-THEN-ELSE and DO (FOR) constructs
- Define the fitness of the program. Examples:
  - How many events does it classify correctly?
  - In how many cases does it provide the correct output?
  - How well does it fit the data?

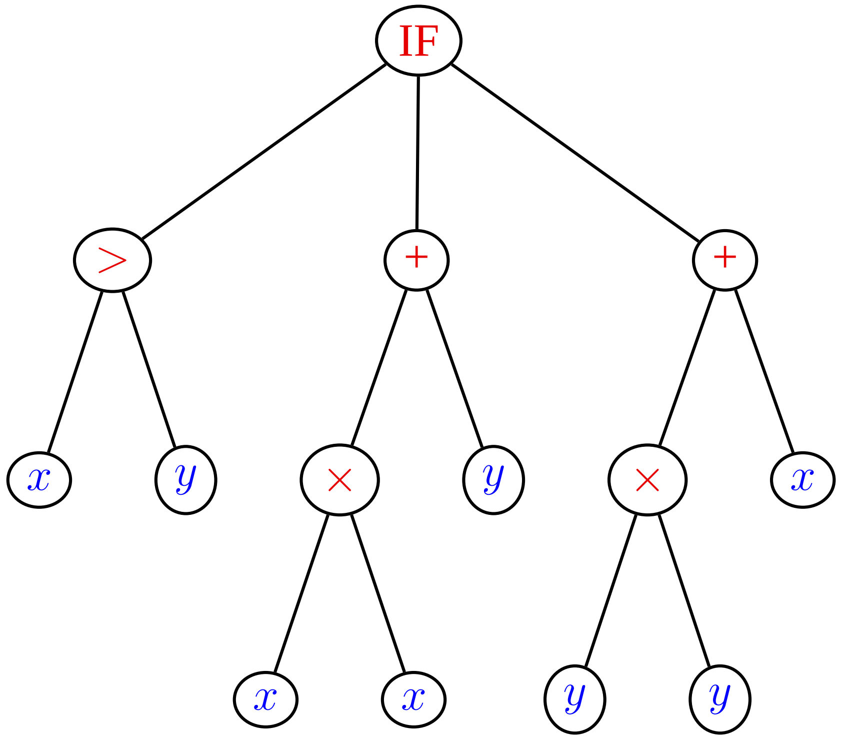
Tree Representation

Genetic Programming fundamentals are easier to illustrate if we adopt a “Tree” representation of a program. An example of this representation:

C code:

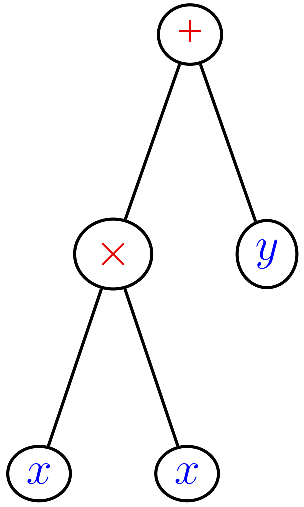
```
float myfunc(float x, float y) {
    float val;
    if (x > y) {
        val = x*x + y;
    } else {
        val = y*y + x;
    }
    return val;
}
```

Program tree



Tree Representation, cont.

From a fraction of our tree, we can see a few things:



Two kinds of “nodes”

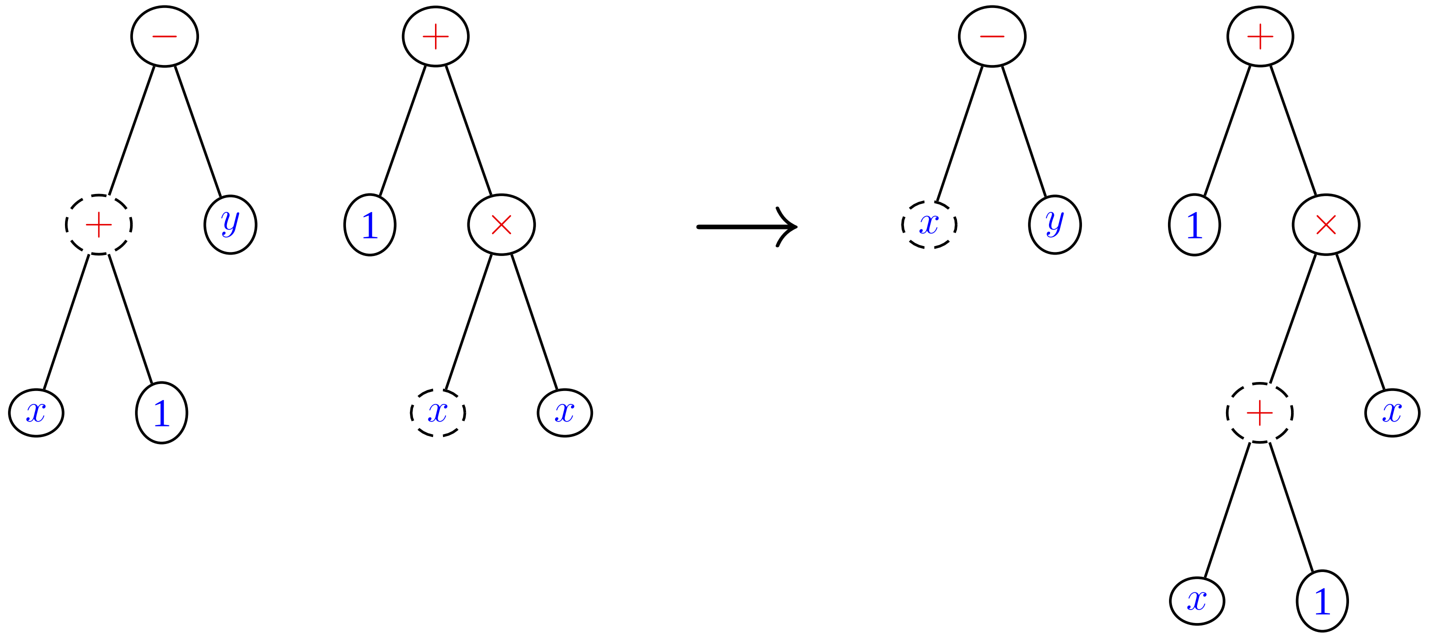
- There are functions (IF, $>$, $+$, $*$)
- There are “terminals” (x, y)
- A function can have any number of arguments (IF has three, $\sin x$ has one)

If we allow *any* function or terminal at any position, then all operations must be allowed:

- IF (float), $x + (y > x)$
- Divide by zero (if we use division)
- I do this by using floats. True $\equiv 1$, False $\equiv 0$

Crossover (Recombination)

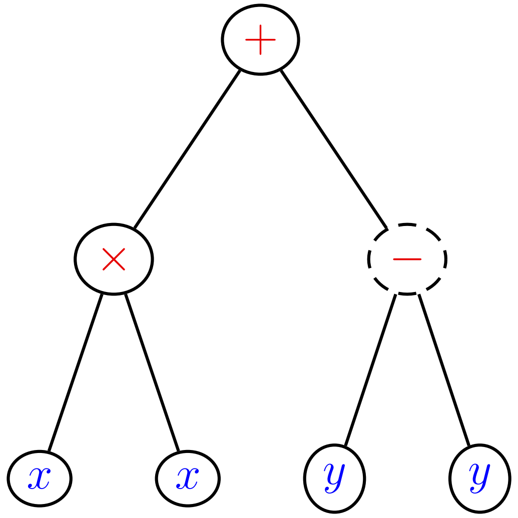
Two programs and crossover points within them are chosen. Sub-trees are removed and swapped between trees, giving two new "children"



It may combine the best aspects of both parents into one child (of course, we are just as likely to end up with the worst aspects in one child).

Mutation

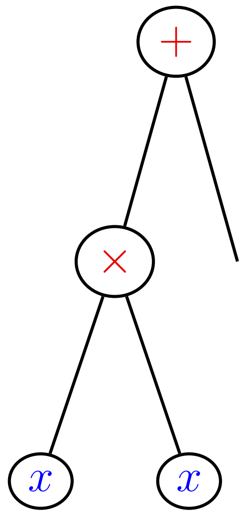
Occasionally we want to introduce a mutation into a program or tree.



Pick a parent & mutation point

Mutation

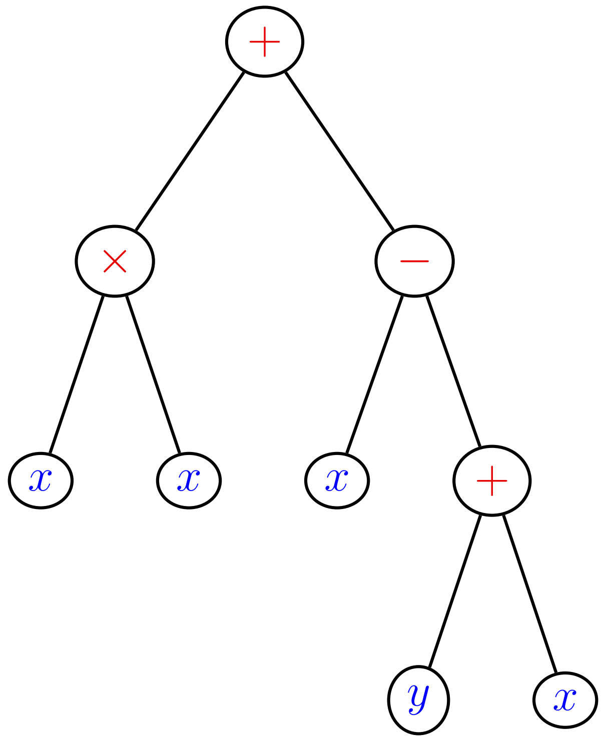
Occasionally we want to introduce a mutation into a program or tree.



Pick a parent & mutation point
Remove the subtree

Mutation

Occasionally we want to introduce a mutation into a program or tree.



Pick a parent & mutation point
Remove the subtree
Finish the new subtree as if it were a "root" tree

Mutation can often be very destructive in Genetic Programming
Remember, both crossover and mutation are random processes.

Survival of the Fittest

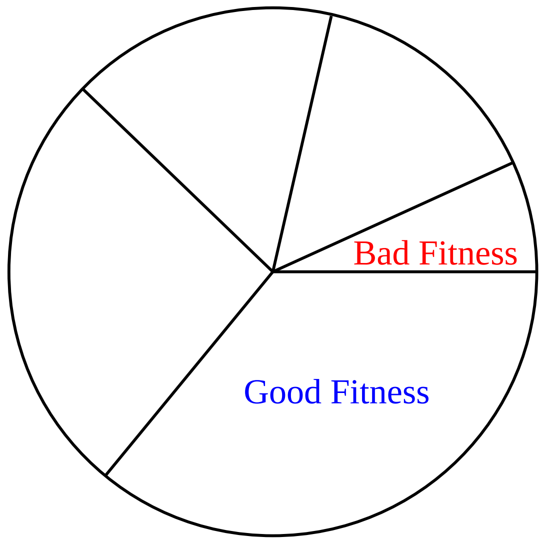
In nature, we know that the more fit an organism is for it's environment, the more likely it is to reproduce. This is one of the basic tenets of evolutionary theory.

The Genetic Programming method mimics this by determining a *fitness* for each individual. Which individuals reproduce is based on that fitness.

- The better (lower) the fitness, the better the solution
- The problem *must* allow for inexact solutions. There may be a single *correct* solution, but there must be a way to distinguish between increasingly incorrect solutions.

Reproduction Probabilities

To select which individuals are chosen to help populate the next generation, they are randomly chosen according to their fitness. The standard method is called “fitness proportionate,” sort of a roulette wheel where the size of the slot is proportional to the fitness.



- The best individual is *most likely* to be chosen
- The best individual is *not guaranteed* to be chosen
- The worst individual *may* be chosen

Running the GP

Putting it all together, we are ready to “run” the GP (find a solution).

- User has defined functions and definition of fitness
- Generate a population of programs (few hundred to few thousand) to be tested
- Test each program against fitness definition
- Choose genetic operation (crossover/mutation) and individuals to create next generation
  - Chosen randomly according to fitness
- Repeat process for next generation
  - Often tens of generations are needed to find the best solution
- At the end, we have a large number of solutions; we look at the best few

Application to HEP

OK, so all this is interesting to computer scientists, but how does it apply to physics, specifically HEP?

In FOCUS, we typically select interesting (signal, we hope) events from background processes using cuts on interesting variables. That is, we construct variables *we* think are interesting, and then require that an event pass the AND of a set of selection criteria.

Instead, what if we give a Genetic Programming framework some variables we think might be interesting, and allow *it* to construct a filter for the events?

- If an AND of cuts is the best solution, the GP can find that

What’s it good for?

- Replace or supplement *cuts*
- Allow us to include *indicators* of interesting decays in the selection process
  - These indicators can include variables we can’t cut on (too low efficiency)
- Can form correlations we might not think of
  - Has already had this benefit

Questions

When considering an approach like this, some questions naturally arise:

- How do we know it’s not biased?
- The tree can grow large with useless information.
- Does it do as well as normal cut methods do?
- Is it evolving or randomly hitting on good combinations?
- What about units? Can you add a momentum and a mass?
  - All numbers are defined to be unit-less

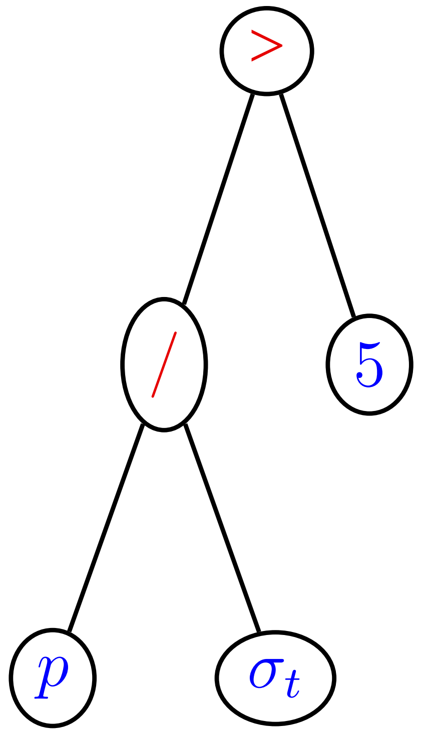
Evaluating the GP

We choose to work with doubly Cabibbo suppressed decay $D^+ \to K^+\pi^+\pi^-$ vs. $D^+ \to K^-\pi^+\pi^+$. Two nearly identical decays $\to$ less stringent systematics.
For each program the GP framework suggests, we have to tell the framework how good the program is:

- All functions must be well defined for all input values, so $> \to 1$ (true) or 0 (false), $\log$ of neg. number, etc.
- Evaluate the tree for each event, which gives a single value
- Select events for which Tree > 0
  - Initial sample has as loose cuts as possible
- Return a fitness to framework
- $\propto \sqrt{S_{\text{Pred-DCS}} + B_{\text{DCS}}} / S_{\text{Pred-DCS}}$ (framework wants to minimize)
- S are predicted signals. B is from fit to DCS BG (masking out signal region).

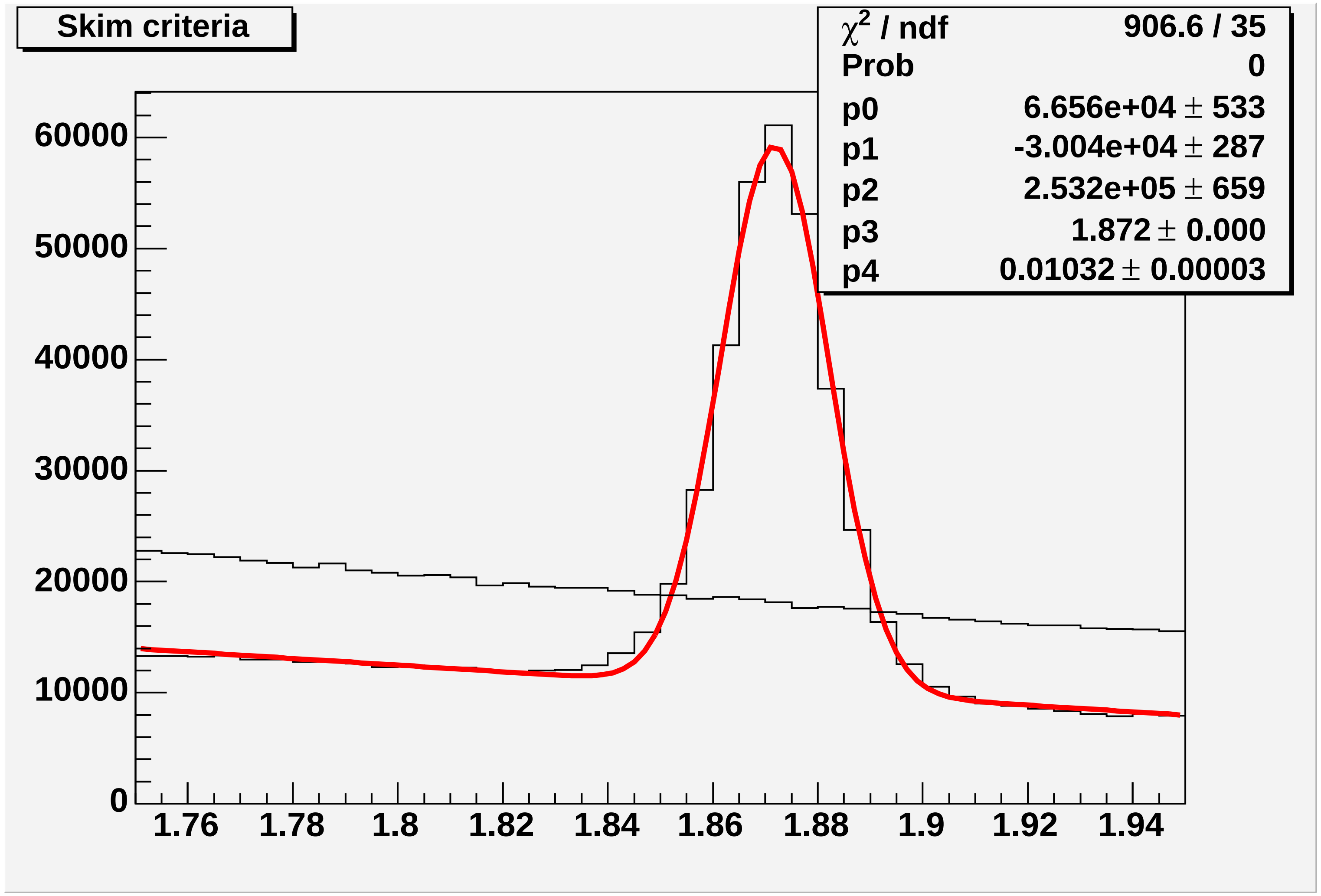
An Example Tree

Let's look at a simple tree. This one will require that the momentum (p) divided by the time resolution (σ_t) is greater than 5.



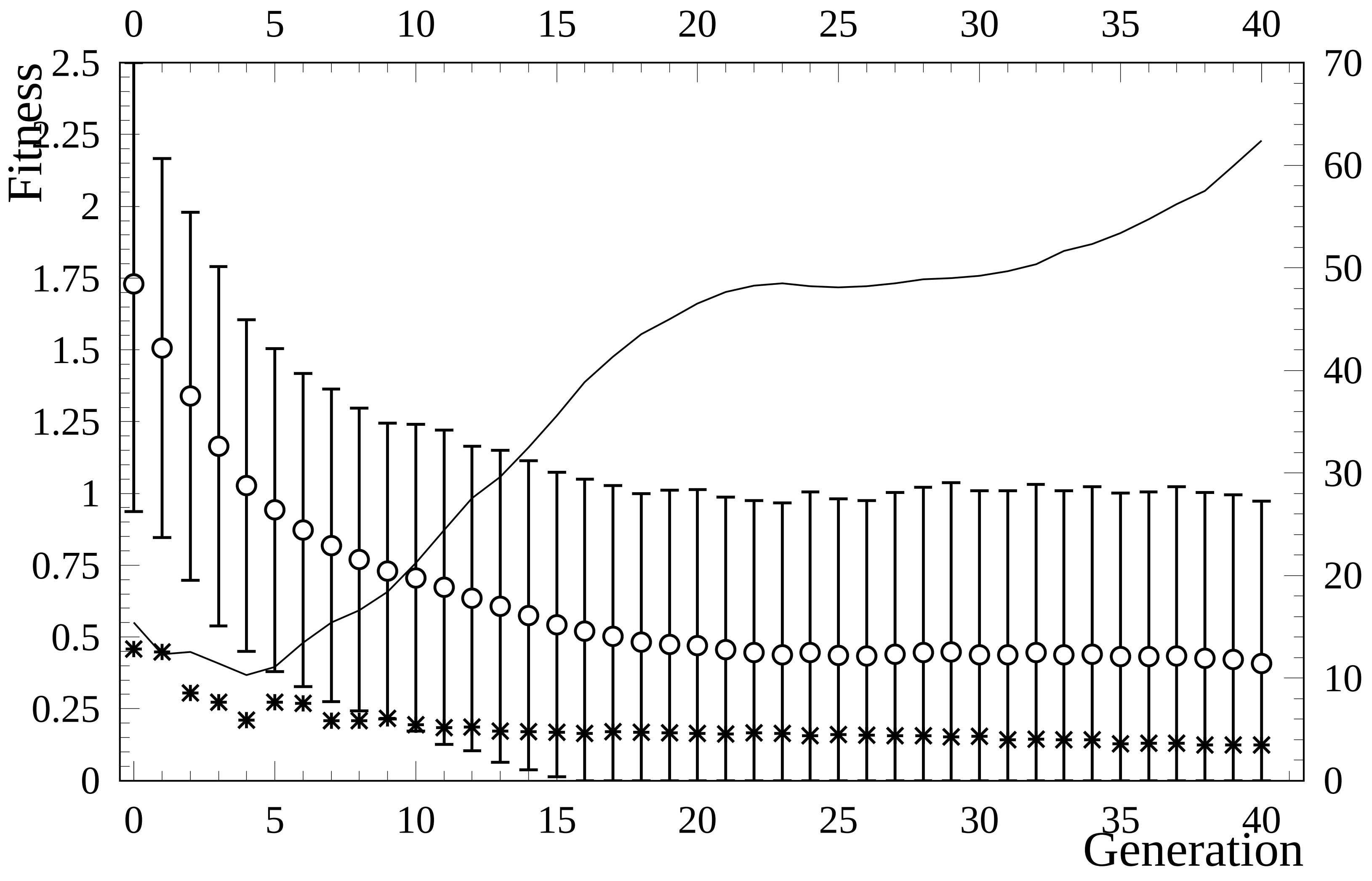
This filter is then applied to each event in my sample and the fitness is determined from the selected events.

After skim (pre-GP) signals



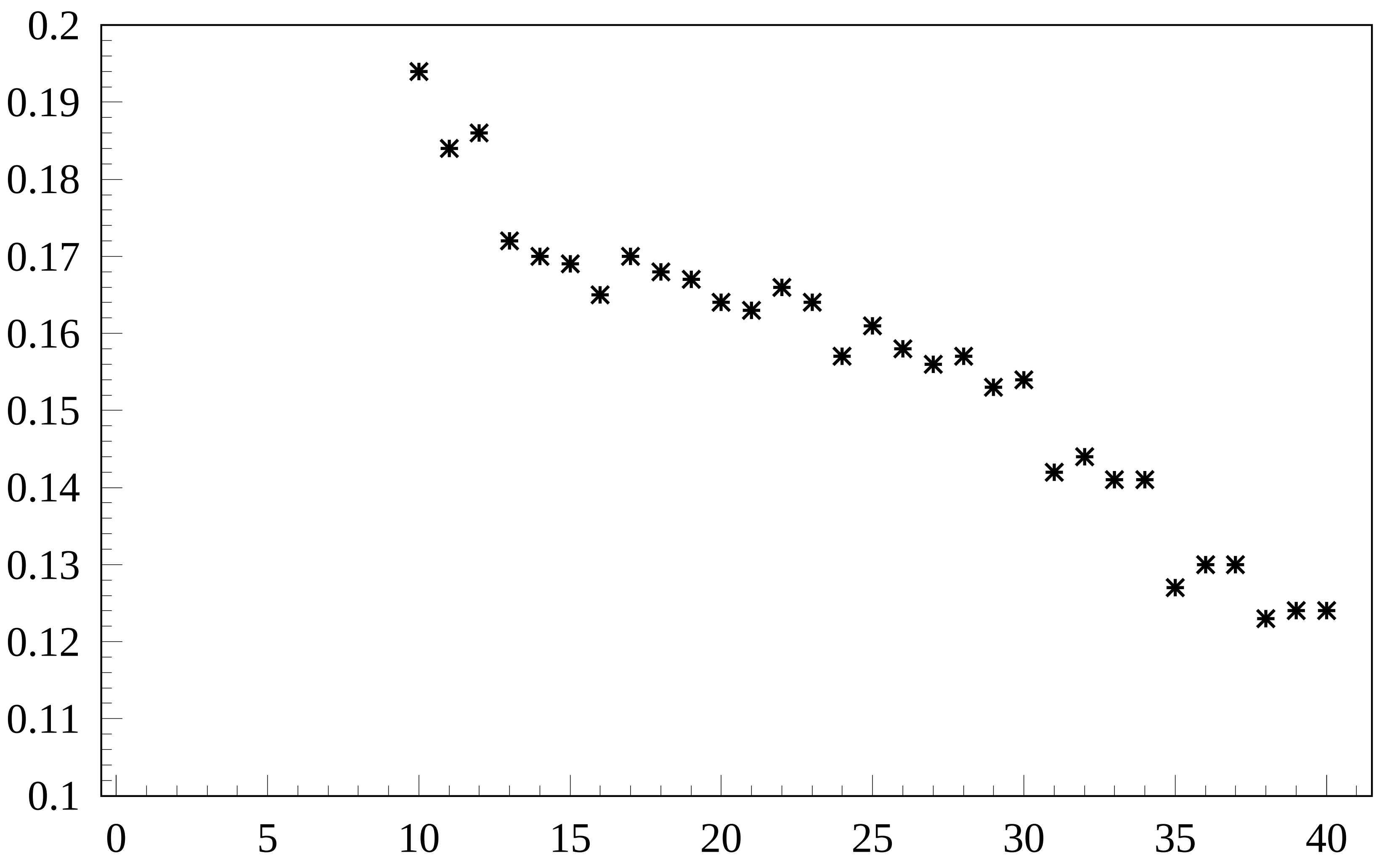
Fit shows $D^+ \rightarrow K^-\pi^+\pi^+$ normalizing mode
"Linear" histogram is DCS candidates

Evolutionary Trajectory



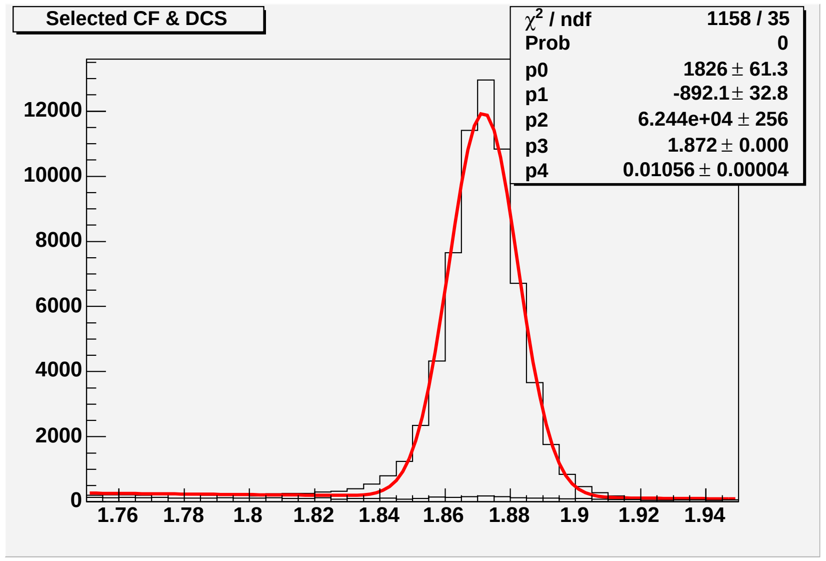
Circles: average, Stars: best, Line: avg. size

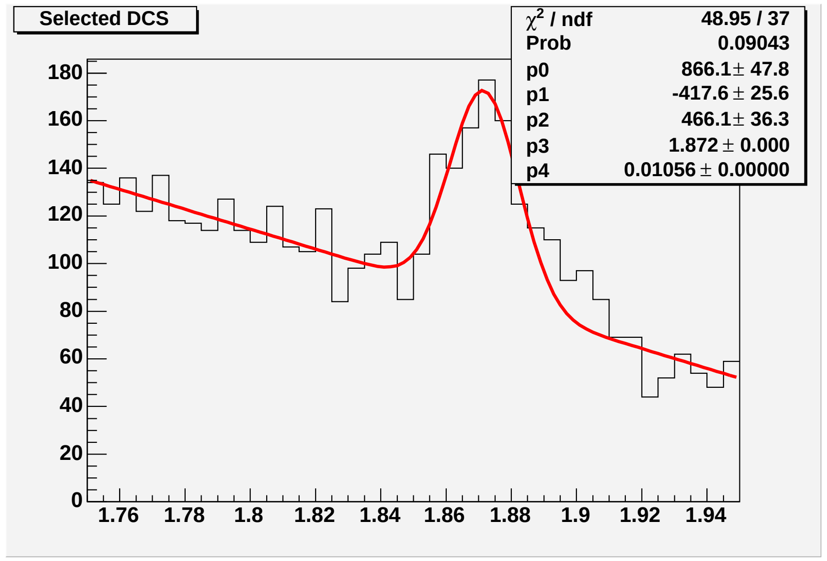
Expansion of best trees



Stars are the best tree, still evolving at generation 40

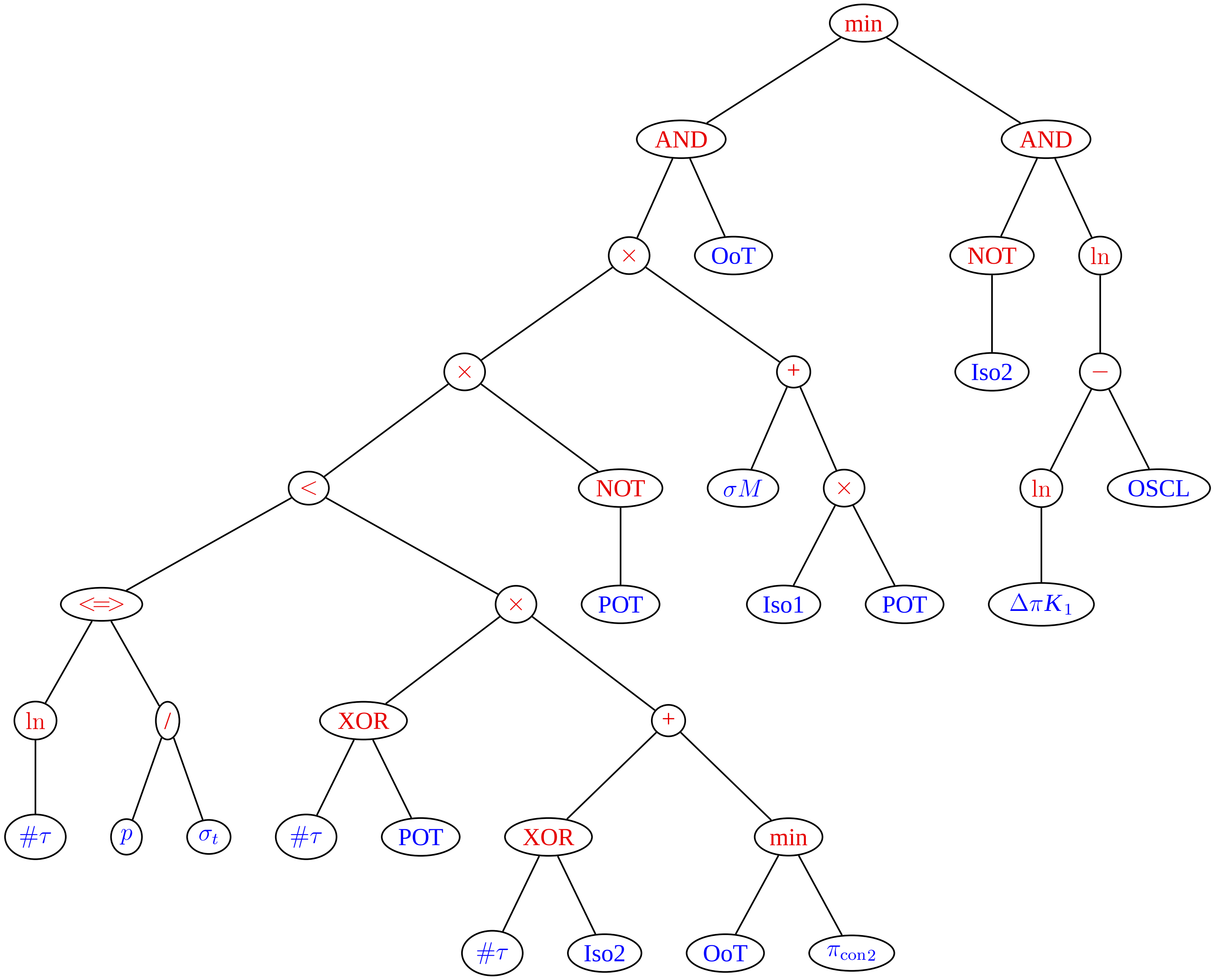
CF and DCSD signals





- Retains 62K of 253K original CF events
- DCS background reduced a factor > 150
- DCS mass and width are fixed to CF values

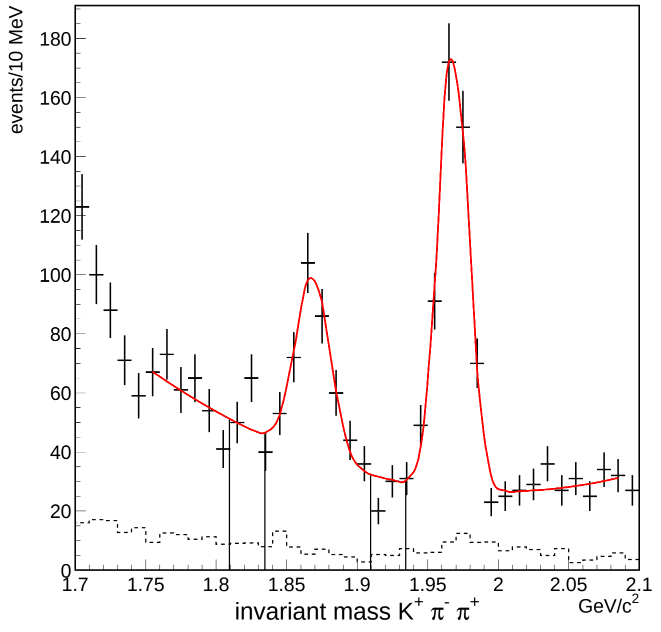
Best tree (40 generations)



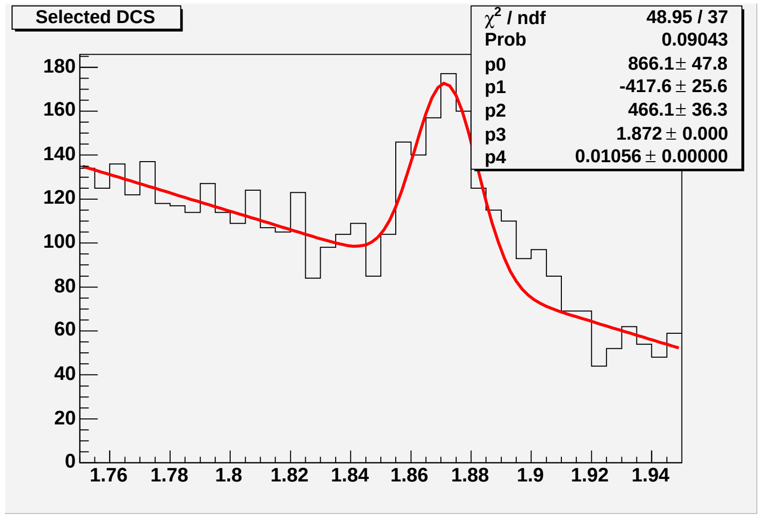
Comparison with Cut Method

How does this compare with our normal method?

- From hep-ex/0407014, measured BR of $D^+ \to K^+\pi^+\pi^-$
- Not a direct comparison, not optimized on $S/\sqrt{S+B}$



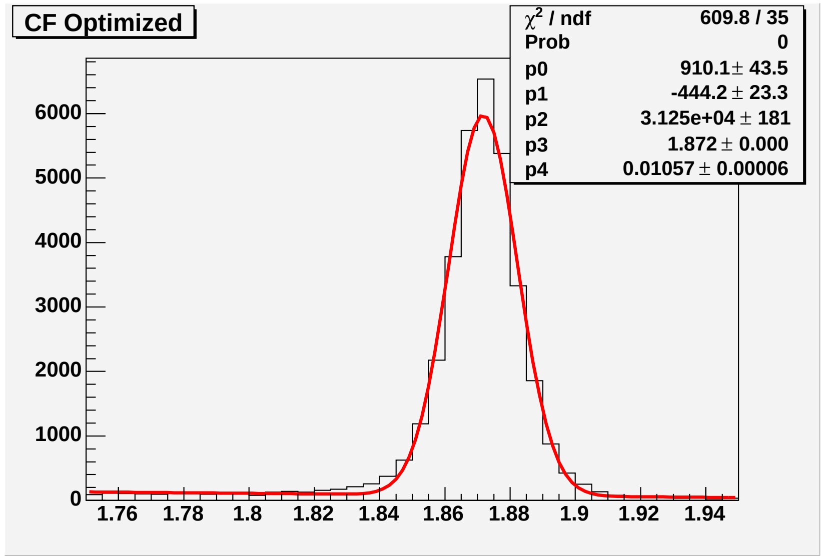
- Similar signal to noise
- Cuts: Yield $= 189 \pm 24$ events
- GP: Yield $= 466 \pm 36$ events

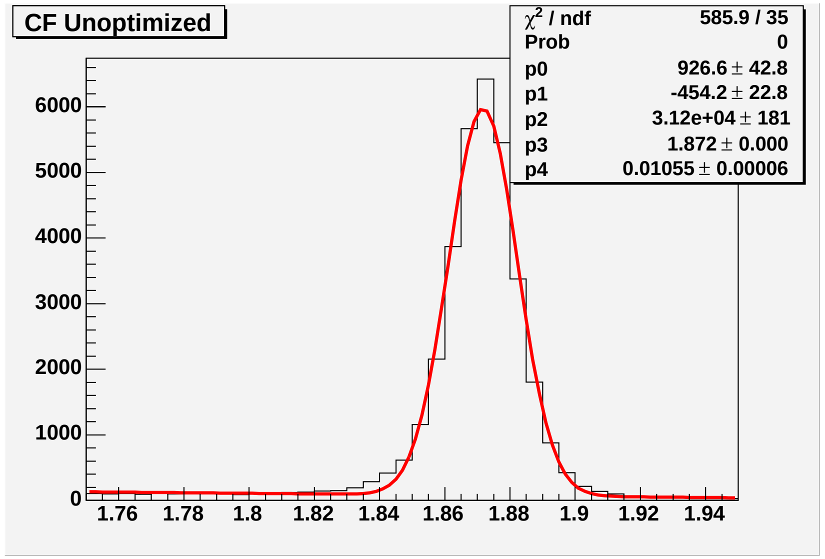


D_s^+ also shown

What about bias?

We put in a penalty (0.5%) for each node to make sure added nodes are valuable. Then, to evaluate bias, we optimize on only half the events (at left).





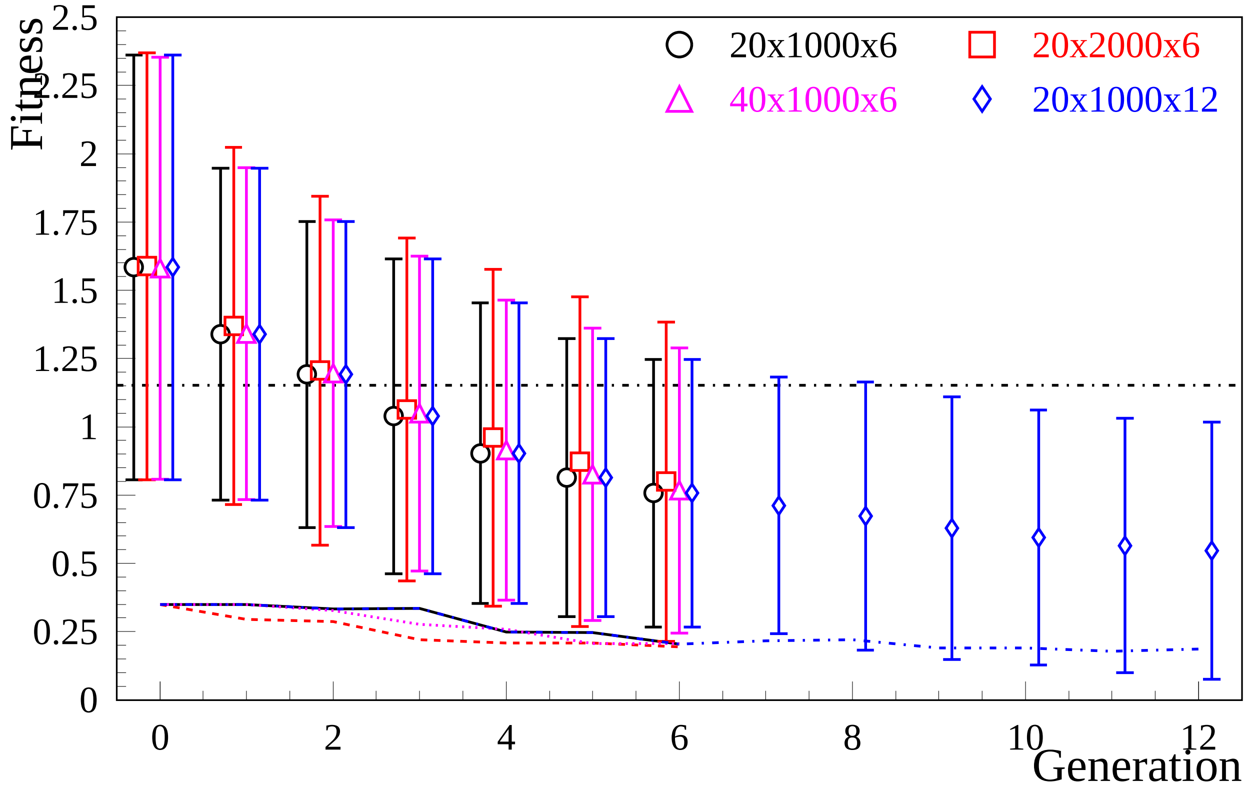
31250 ± 180 events

31200 ± 180 events

No evidence of selection induced bias here.
Doubly Cabibbo suppressed distributions are also similar: 2135 (optimized) vs. 2123 (unoptimized) events in whole plot.

Tuning GP parameters

Start: 20 CPUs, 1000 trees/CPU, 6 gen. Doubled each parameter



- Points: avg & RMS, dots: best
- More generations is only clear improvement
- Plots in analysis section from 20x1500x40

Conclusions

This method shows promise, but there are some caveats

- More challenging for modeling
- Perhaps best used where statistical errors dominate
- Trees are very complex and any attempt to understand the whole thing may be pointless

However

- Worthwhile to try to understand parts of trees
- Combination CLP - Iso1 occurred often
  - Now being used in other analyses
- Even simpler trees do better than the cuts they suggest

We think this novel method at least deserves further exploration

Thank you for listening

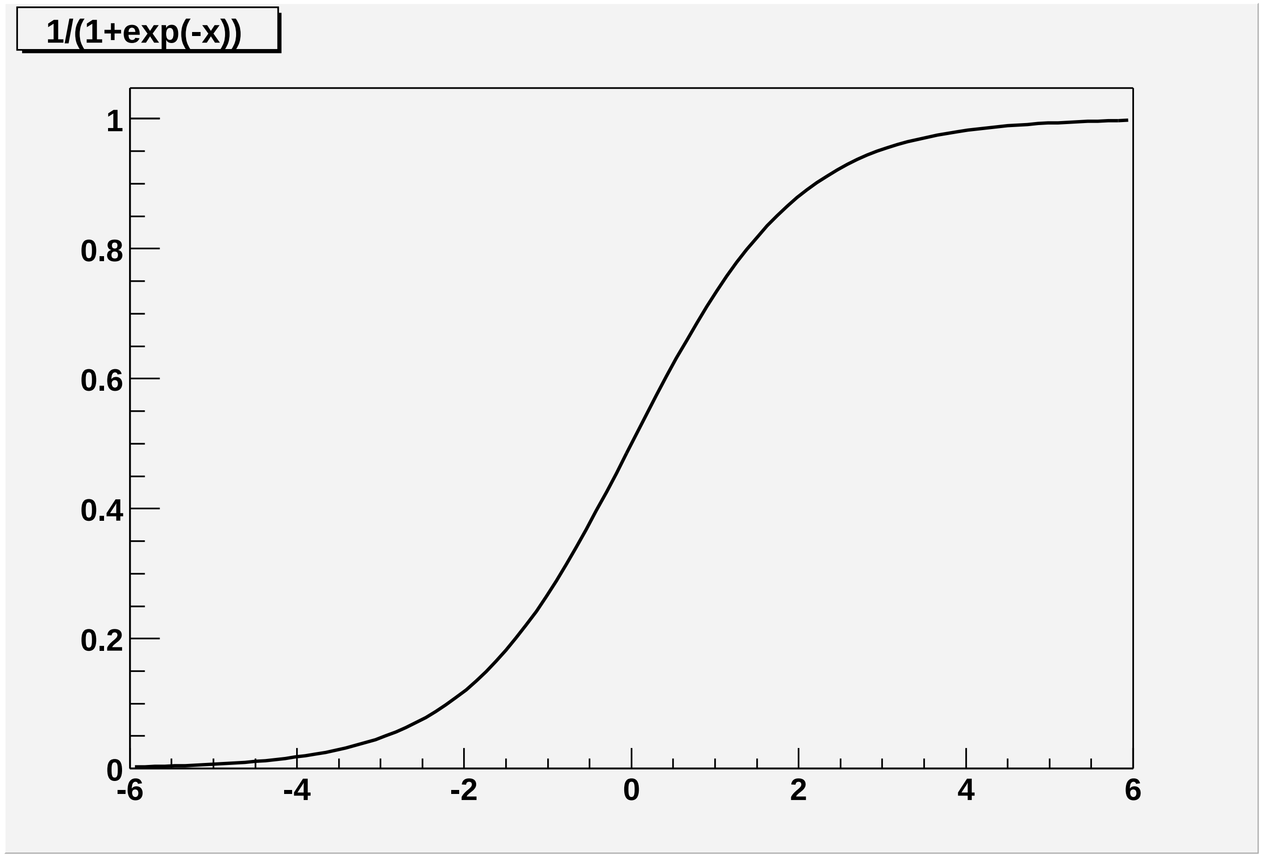
Backup slides

Backup slides

$f(n)$ function

A threshold function used in neural networks:

$$f(n) = \frac{1}{1+e^{-n}}$$

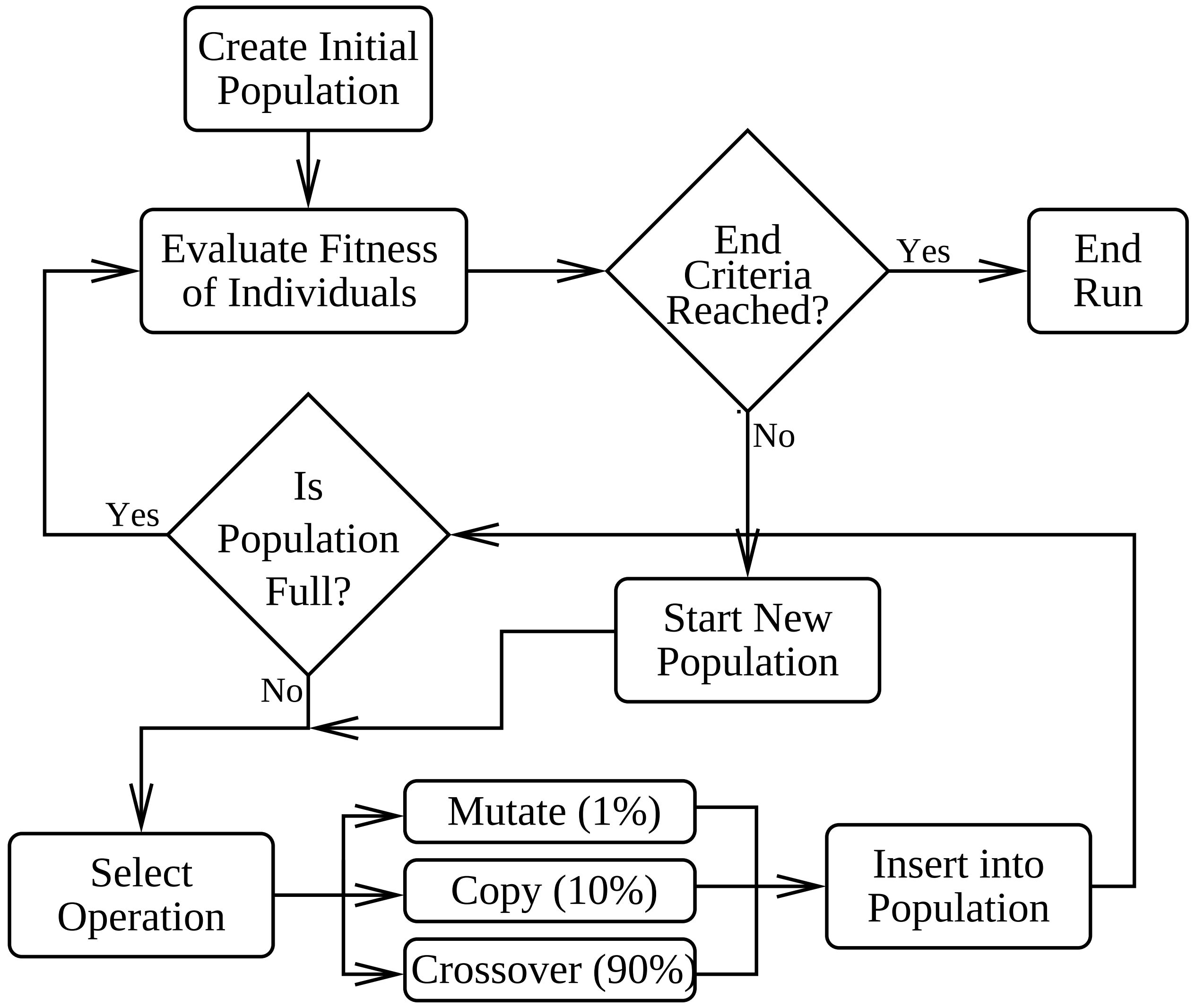


SW Mechanics & Conclusions

Is interfacing to an existing experiment’s code difficult?

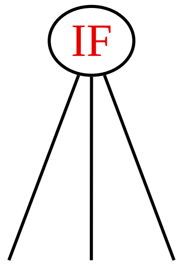
- Genetic programming framework
  - C language based **lilgp** from MSU Garage group
  - Modified for parallel use (**PVM**) by Vanderbilt Med Center group
  - Parallel version allows sub-population exchange
- Physics variables start with standard FOCUS analysis
  - Write **HBOOK** ntuples, convert to Root Trees
  - Write a little C++ code to access Trees, fill and fit histograms (using **MINUIT**) and return the fit information to the lilgp framework
- This is actually pretty easy

Genetic Programming Process



Building a tree

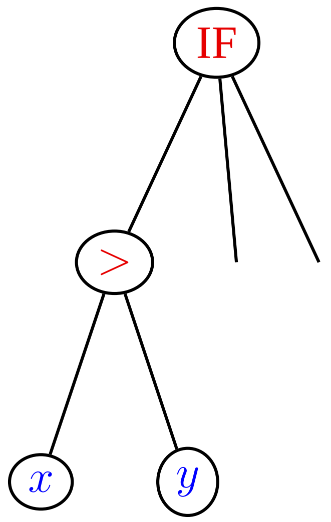
Trees are randomly built up one node at a time.



Root node 'IF' has 3 args.

Building a tree

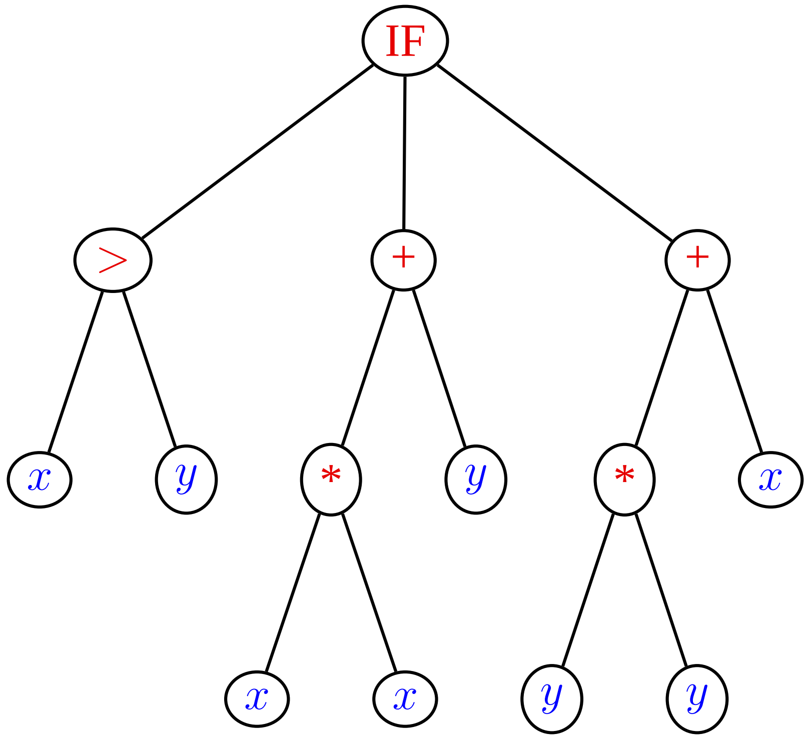
Trees are randomly built up one node at a time.



Root node 'IF' has 3 args.
'$>$' chosen for 1st arg.
x and y terminate '$>$'

Building a tree

Trees are randomly built up one node at a time.



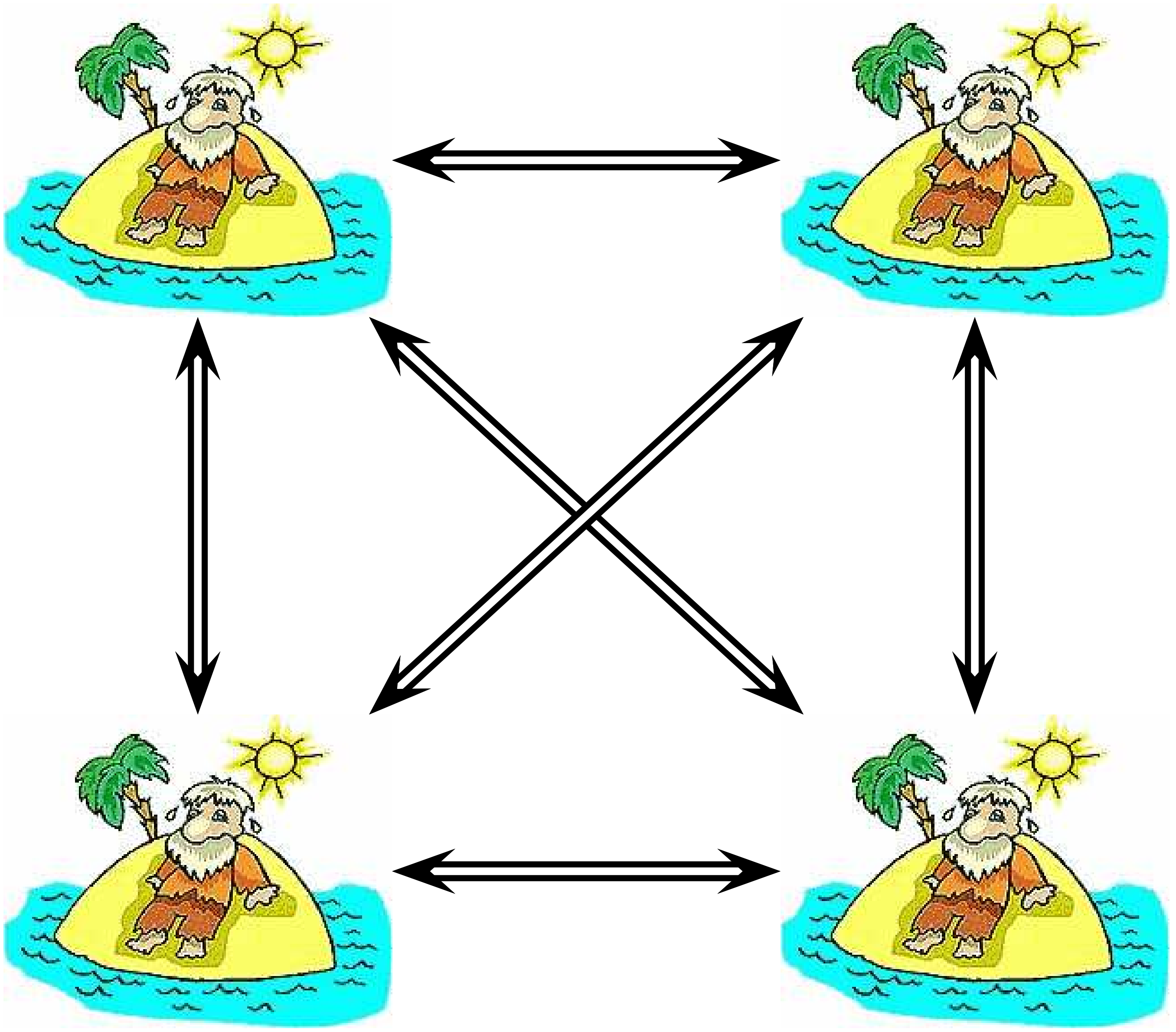
Root node ’IF’ has 3 args.
’$>$’ chosen for 1st arg.
x and y terminate ’$>$’
Remaining branches grown
Tree is complete
(all branches terminated)

Parallelizing the GP

Each test takes a while (10–60 sec on a 2 GHz P4) so spread over multiple computers

- Adopt a South Pacific island type model
    - A population on each island (CPU)
    - Every few generations, migrate the best individuals from each island to each other island
- Lots of parameters to be tweaked, like size of programs, probabilities of reproduction methods, exchanges, etc.
    - None of them seem to matter all that much, process is robust

Parallelizing the GP



Practical considerations

Obviously, a tree can grow nearly infinite in size. This is usually undesirable. There are ways to control this:

- Set limits on number of nodes
- Set limits on depth of nodes
- Create initial topologies of specified depth

A common approach is to allow half of the initial population to grow completely randomly and to create the other half at a range of (shallow) depths. In the latter case, pick functions for all nodes $<$ desired depth, pick terminals for all nodes at desired depth.

Data vs. MC comparisons

Since these decays are nearly identical, what is important is that the efficiency of the tree for CF and DCS modes is the same (or modeled by our MC). What is less important is the *absolute* efficiency on a single mode.

But studies of absolute tree efficiencies are encouraging. We can study

$$\frac{\epsilon_{\text{tree}}}{\epsilon_{\text{original selection}}}$$

for both the MC and the normalization mode. We typically find deviations $\lesssim 10\%$.

Highly dependent on the quality of the simulation package.