Genetic Programming and HEP

Vanderbilt Department of Physics and Astronomy

Nuclear and Particle Physics Seminar

1 November 2004

Eric W. Vaandering

ewv@fnal.gov

Vanderbilt University

and

FOCUS Collaboration

Overview

- Machine learning techniques
- Introduction to Genetic Programming
  - Populations and Generations
  - Mutation and Crossover
  - Fitness and Natural Selection
- Genetic Programming applied to doubly Cabibbo suppressed decays
  - $D^+ \to K^+\pi^+\pi^-$
  - $\Lambda_c^+ \to pK^+\pi^-$

Genesis

I first learned of this technique from our VAMPIRE colleauges in the VU Medical Center.

They work in the Human Genetics program and use this technique to study gene expression levels to predict which type of breast cancer a person is susceptable to (among other things).

The fact that their technique is called Genetic Programming and that they are Geneticists is completely unrelated, a pure coincidence.

But, I thought maybe I could apply this technique to high energy physics.

Machine Learning

There has been a long interest in teaching machines to "automatically" solve problems, given the broad parameters of the possible solutions.

For all but the simplest problems, exhaustive or completely random searches are impractical. There are numerous attempts to automatically find solutions: neural nets, simulated annealing, expert systems, etc.

To find the best solution, maybe we should take a clue from biology and the evolutionary process. (→ Genetic Algorithms)

Since we will use computer programs to implement our solutions, maybe the *form* of our solution should *be* a computer program.

Combined, these last two points form the basis of Genetic Programming

Genetic Programming

"How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?"

— Attributed to Arthur Samuel, 1959
(Pioneer of Artificial Intelligence,
coined term "machine learning")

"Genetic programming *is* automatic programming. For the first time since the idea of automatic programming was first discussed in the late 40's and early 50's, we have a set of non-trivial, non-tailored, computer-generated programs that satisfy Samuel's exhortation: 'Tell the computer what to do, not how to do it.' "

— John Holland, University of Michigan, 1997
(Pioneer of Genetic Algorithms)

Genetic Programming

Definition:

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.

- Applies a model of biological evolution to program "discovery"
- Pioneered by John Koza in 1989
- Seminal reference: *Genetic Programming: On the Programming of Computers by Natural Selection* (Koza, 1992)
- Since 1992, more than 3,000 papers applied to a wide range of problems

Programming Assumptions

Normally when we program, we assume a number of guidelines:

- Correctness: The solution works perfectly
- Consistency: The problem has one preferred solution
- Justifiability: It is apparent why the solution works
- Certainty: A solution exists
- Orderliness: The solution proceeds in a orderly way
- Brevity: Every part of the solution is necessary, a shorter solution is better (Occam's Razor)
- Decisiveness: We know when the solution is complete

Genetic Programming *requires* that *all* of these assumptions be discarded

GP principles

In fact, in Genetic Programming:

- Correctness: A solution may be “good enough”
- Consistency: Many very different solutions may be found
- Justifiability: It may be very unclear how or why a solution works
- Certainty: A perfect solution may never be found
- Orderliness: A solution may be very disorganized
- Brevity: Large parts of the solution may do nothing
- Decisiveness: We may never know if the best solution has been found

Populations and Generations

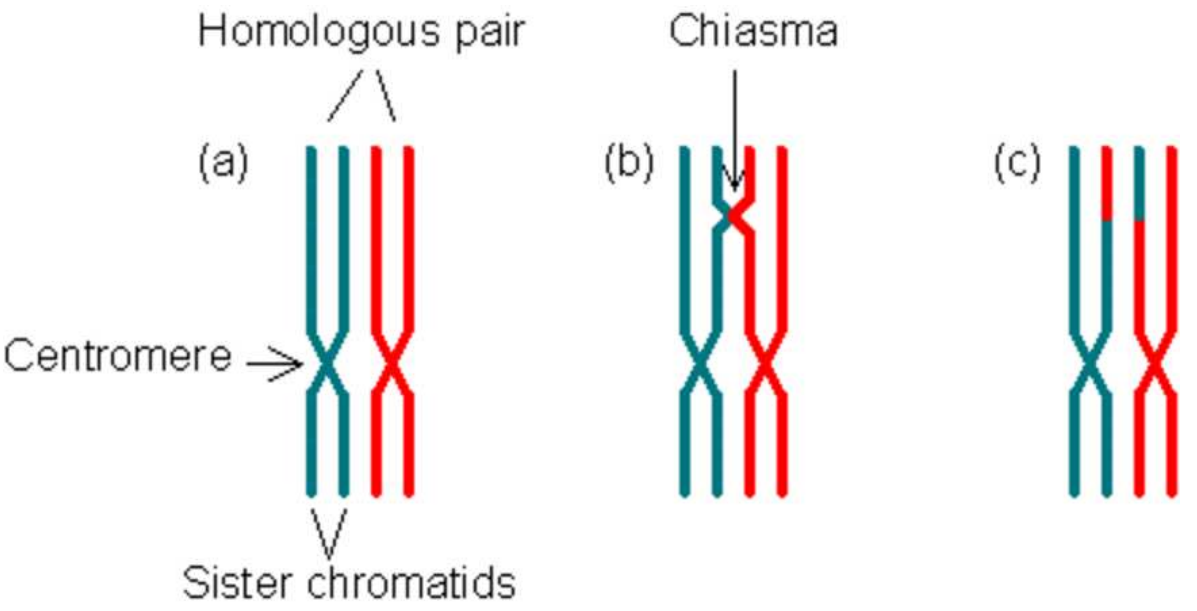
Genetic Programming works by transforming one group of individuals (programs) in generation n into another group of individuals in generation $n + 1$. There are typically a few hundred to a few thousand programs per generation.

Typically the number of individuals in each generation is the same. Usually no duplication is allowed in the 1^{st} (or 0^{th}) generation. Duplication *is* allowed in later generations. (Diversity decreases.)

There are GP implementations where change is not generational, but adiabatic. In these implementations, when a new individual is created, an old one is usually "killed," keeping the population size the same.

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?

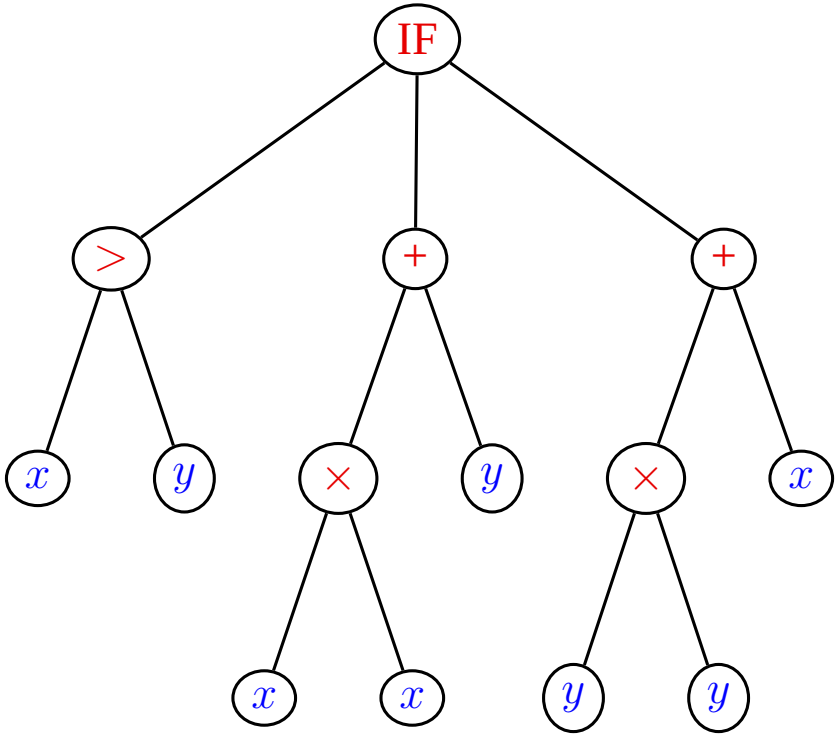
Tree Representation

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

C code:

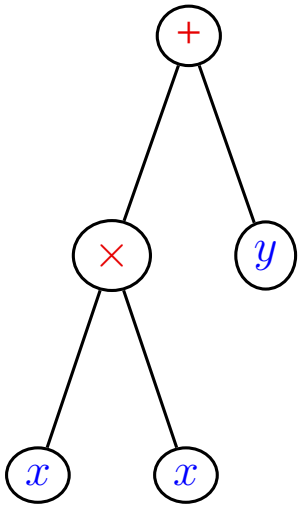
```
float myfunc(float x, float y) {
    float val;
    if (x > y) {
        val = x*x + y;
    } else {
        val = y*y + x;
    }
    return val;
}
```

Program tree



Tree Representation, cont.

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



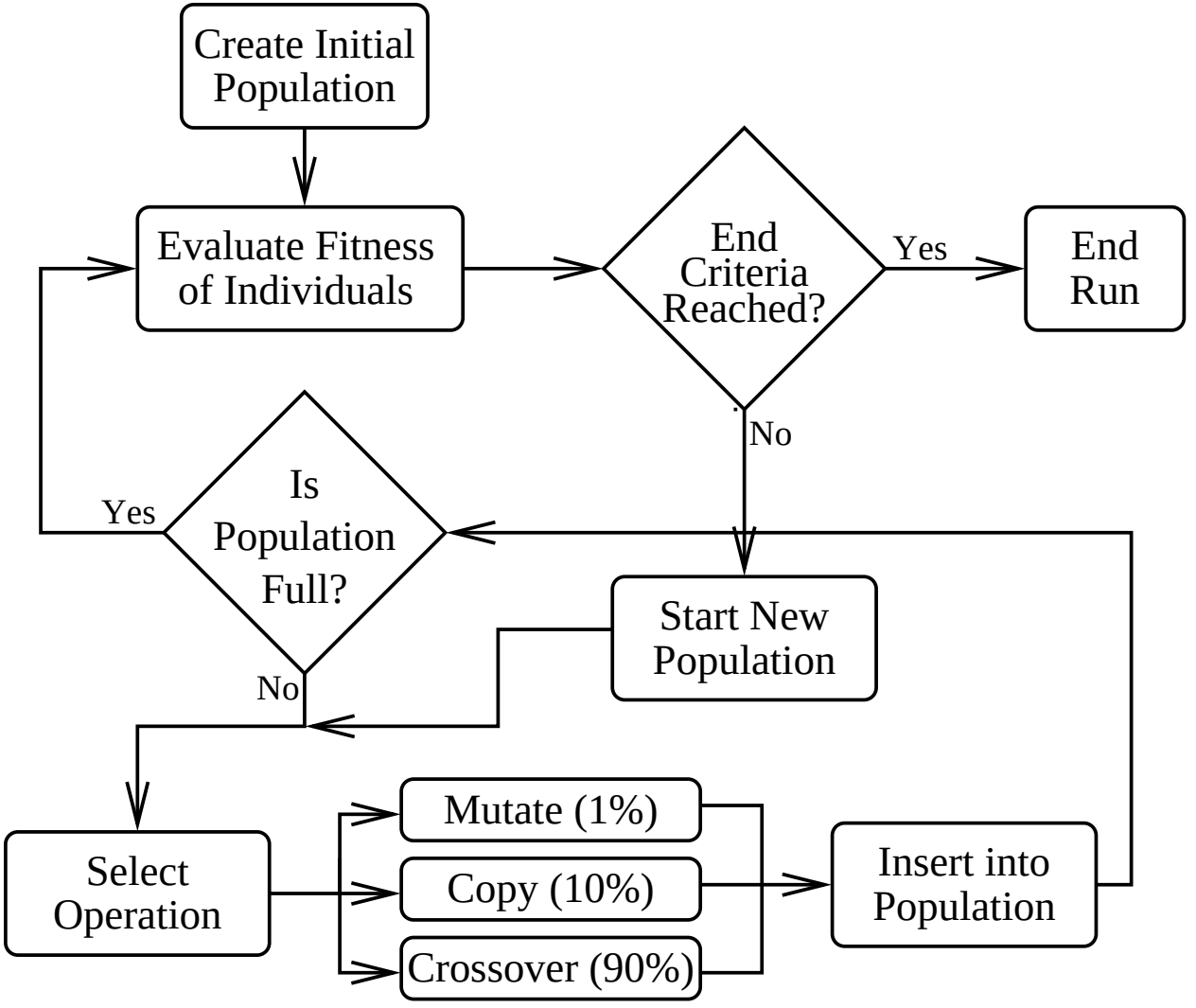
Two kinds of "nodes"

- 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:

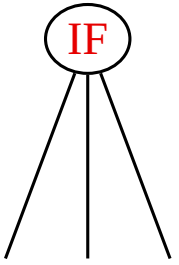
- IF (float)
- $x + (y > x)$
- Divide by zero (if we use division)

Genetic Programming Process



Building a tree

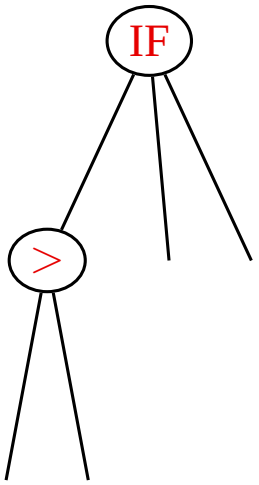
Trees are *randomly* built up one node at a time.



Root node ‘IF’ has 3 args.

Building a tree

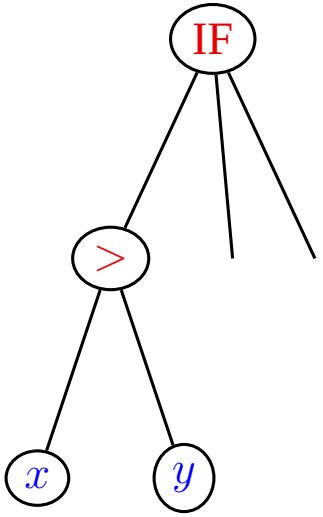
Trees are *randomly* built up one node at a time.



Root node ‘IF’ has 3 args.
‘>’ chosen for 1st arg.

Building a tree

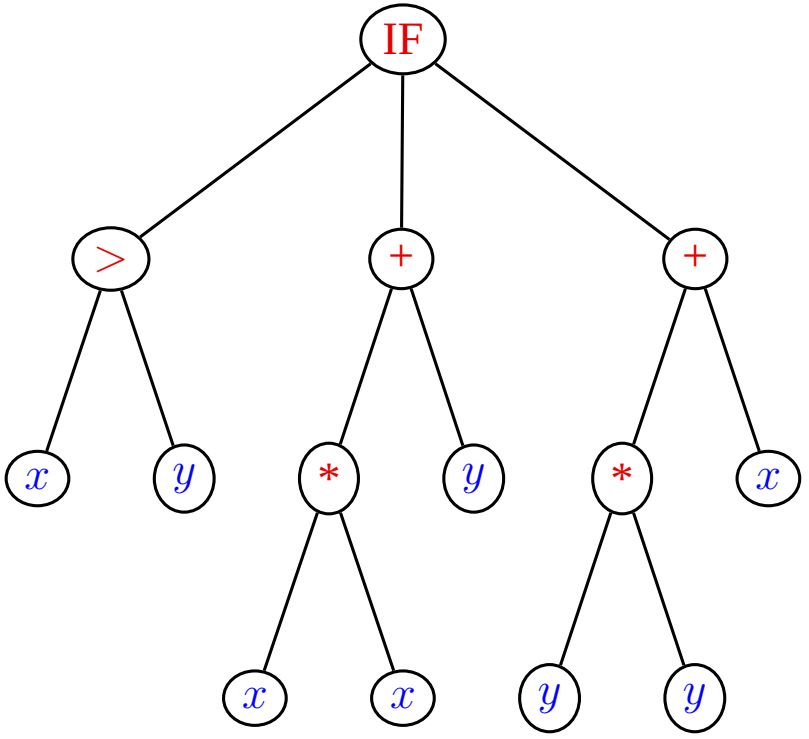
Trees are *randomly* built up one node at a time.



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

Building a tree

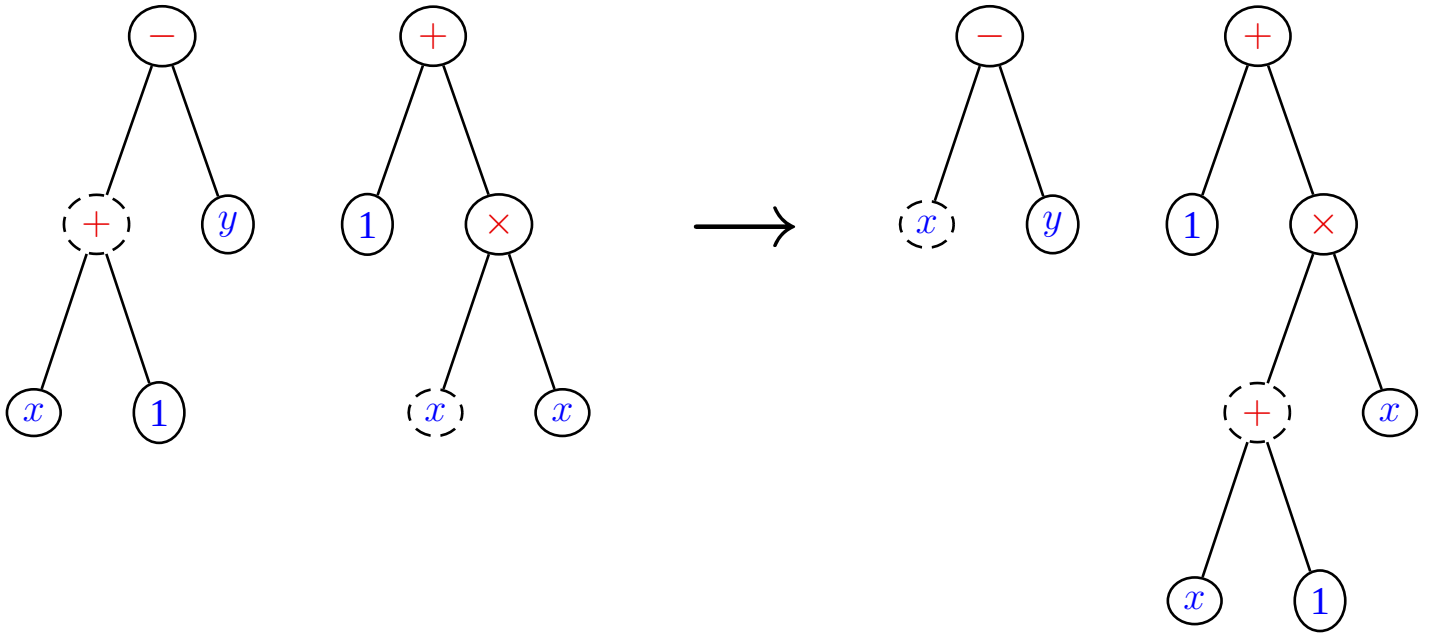
Trees are *randomly* built up one node at a time.



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

Crossover (Recombination)

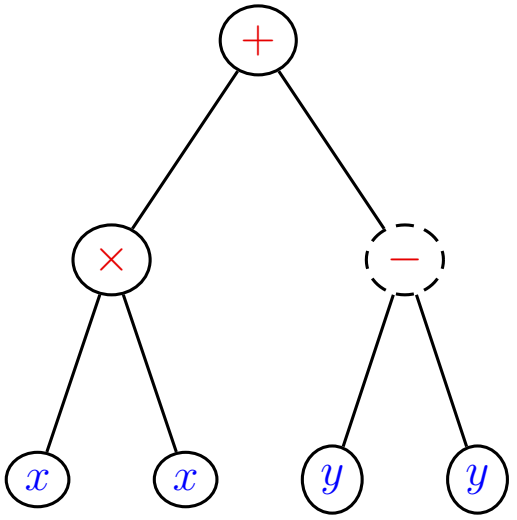
Two programs and crossover points within them are chosen. Sub-trees are removed and swapped between trees, giving two new “children”



We hope to 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

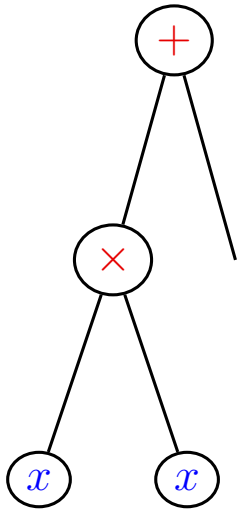
Occasionally we want to introduce a mutation into a program or tree.



Pick a parent & mutation point

Mutation

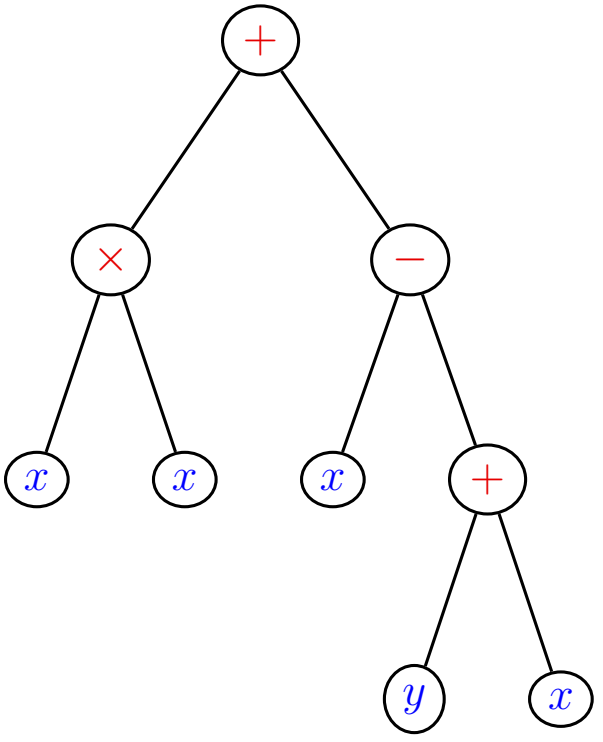
Occasionally we want to introduce a mutation into a program or tree.



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.

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 less than desired depth, pick terminals for all nodes at desired depth.

So far we've mimiced *how* organisms reproduce.
The other half of the problem is *why* they reproduce.

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.

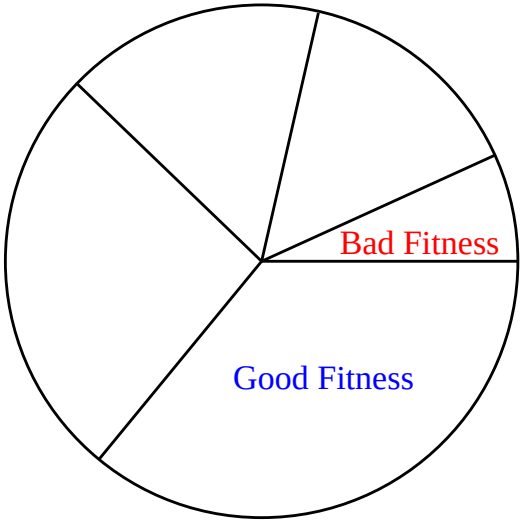
- Organisms with serious deformities are still-born or die at a young age
- Faster, stronger, or longer lived organisms will produce more offspring

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

- The better 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. (Otherwise we are engaging in a random search.)

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

Tournament Selection

Another type of selection is also used to implement survival of the fittest. In tournament selection, a number of individuals (two or more) are selected randomly. The most fit from that group is selected to reproduce. The process is repeated to find a mating partner (if needed).

We see this behavior in nature too...

Tournament

Reward

Running the GP

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

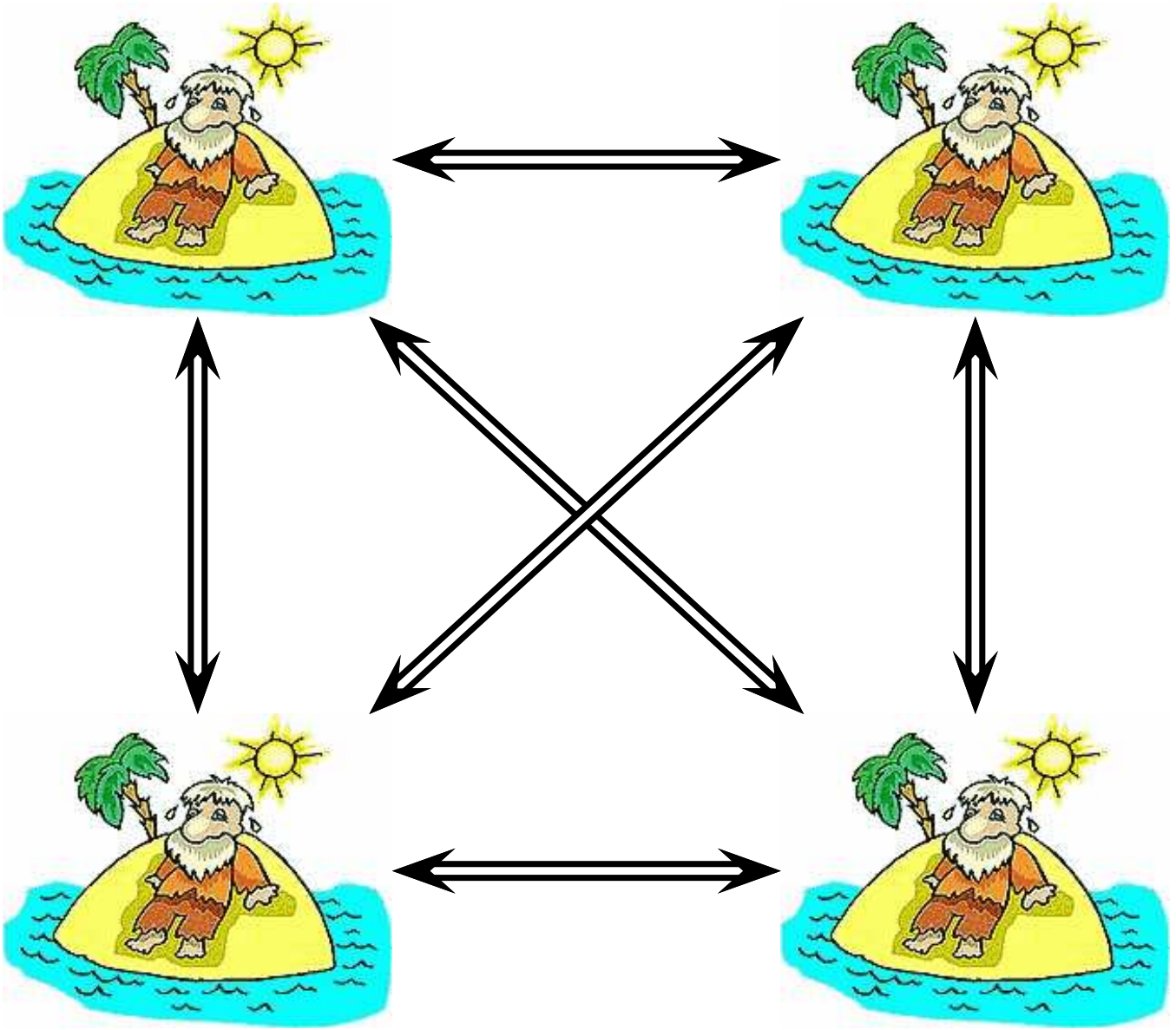
- User has defined functions and definition of fitness
- Generate a population of programs (few hundred to few thousand) to be tested
- Test each program against fitness definition
- Choose genetic operation (crossover/mutation) and individuals to create next generation
  - Chosen randomly according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

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 quite robust

Parallelizing the GP



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 the variables we think are 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

We already have some experience with these types of methods. *E.g.*, neural networks are used effectively for B flavor tagging by several experiments.

What's it good for?

- Might 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
- Hope to produce a more effecient selection mechanism

An Easy Problem

Fitness

Model

A Difficult Problem

Fitness

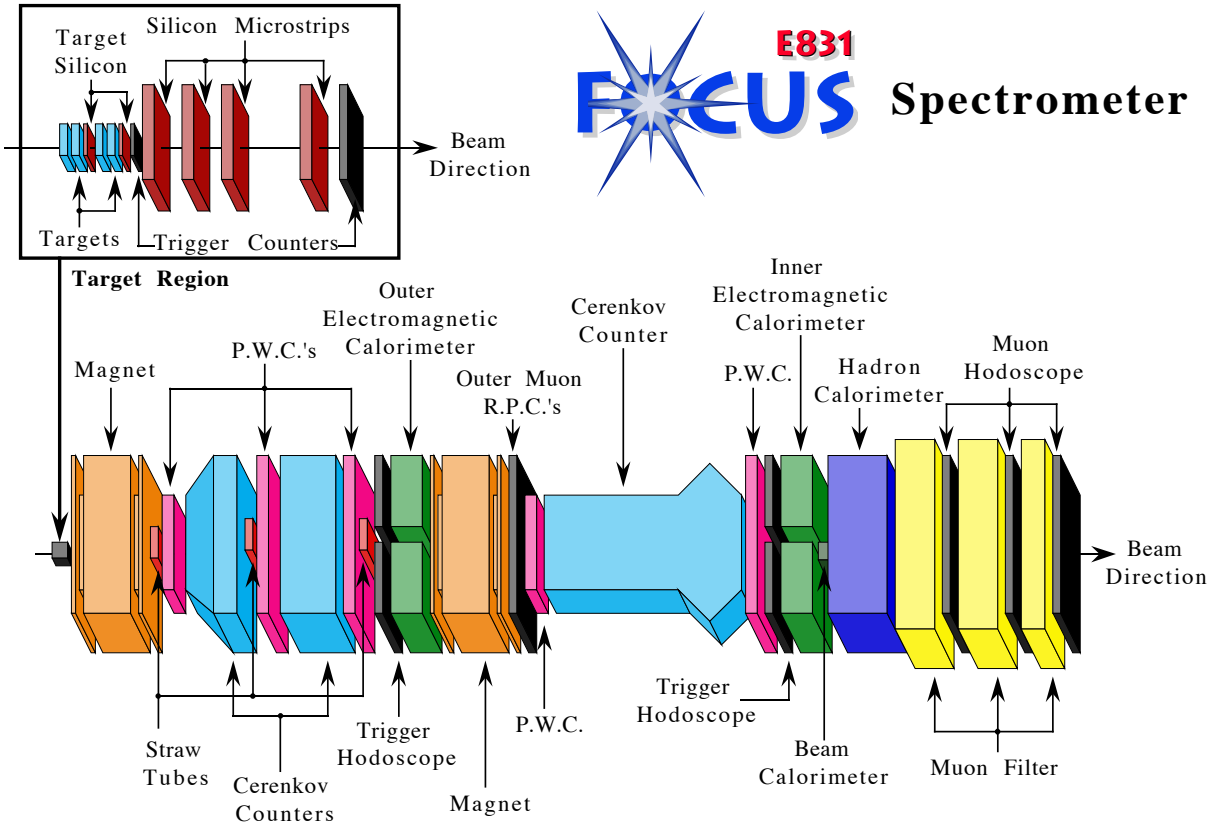
Model

Questions

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

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

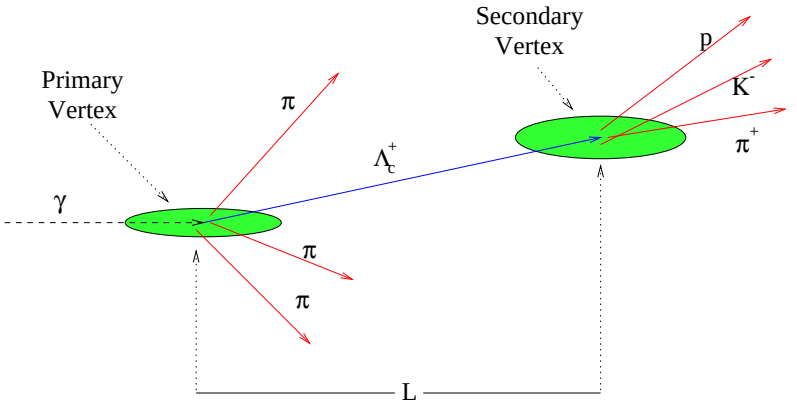
FOCUS Spectrometer

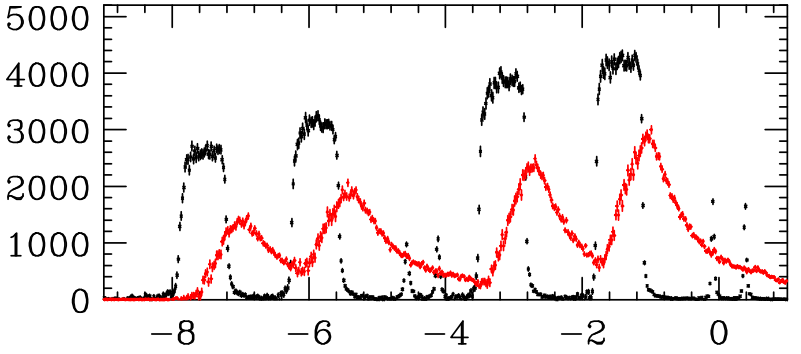


Highlights:

- Segmented target
- Silicon vertexing
- MWPC tracking
- Threshold Čerenkov
- EM/hadronic calorimeters
- Muon detectors

Target and Vertexing





Some details of the FOCUS candidate driven vertexing

- L: Distance between production and decay vertices. ℓ/σ_ℓ, significance of separation
- CLS, CLP: CLs of decay and production vertices
- Iso1: CL that tracks from decay vertex are consistent with production vertex
- Iso2: CL that other tracks (incl. from production vertex) are consistent with decay vertex
- OoT: Significance of decay being out of target material

Variables and Operators

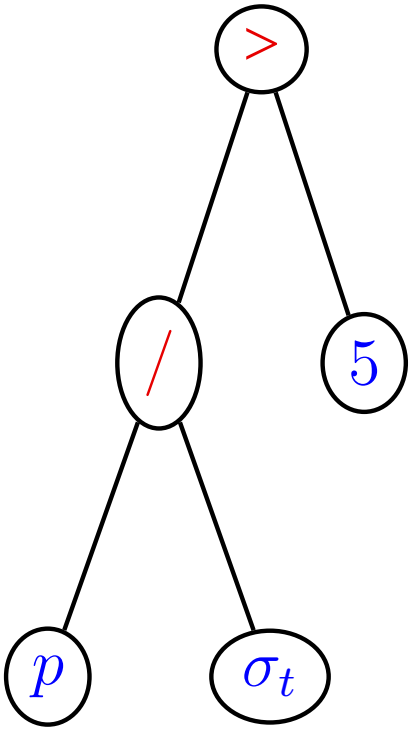
Give the GP lots of things to try:

Functions (22)		Variables (D^+–35, Λ_c^+–37)		
$\times$	sign	ℓ	$\Delta W(\pi p)$	
$/$	negate	σ_ℓ	$\Delta W(Kp)$	
$+$	max	ℓ/σ_ℓ	$\Delta W(\pi K)$	σ_t
$-$	min	OoT	$\pi_{\rm con}$	p_T
x^y	NOT	CLS	Track χ^2's	Σp_T^2
$\sqrt{}$	AND	CLP	OS Vertex CL	$m_{\rm err}$
log	OR	Iso1	OS $\Delta W(\pi K)$	$\mu_{\rm max}$
$>$	XOR	Iso2	OS CL$_\mu$	TS/NoTS
$<$	IF	#life	Real $(-2,+2)$	REME
$<=>$	sin	Pri. OoT	Int $(-10,+10)$	
$f(n)$	cos	$p(\Lambda_c^+)$	0,1	

E.g.: 80 nodes (40 func., 40 var.) $\rightarrow 40^{22} \times 40^{37} = 3.3 \times 10^{94}$ combinations.
Just one topology of many (as big as 340).

An Example Tree

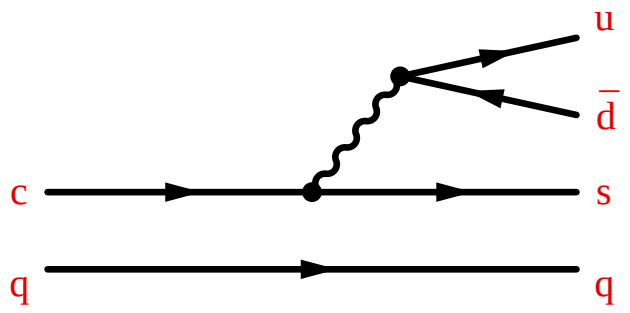
Let's look at a very simple tree. This one gives **1** when the momentum (p) divided by the time resolution (σ_t) is greater than 5, gives **0** otherwise. (This is just a cut.)



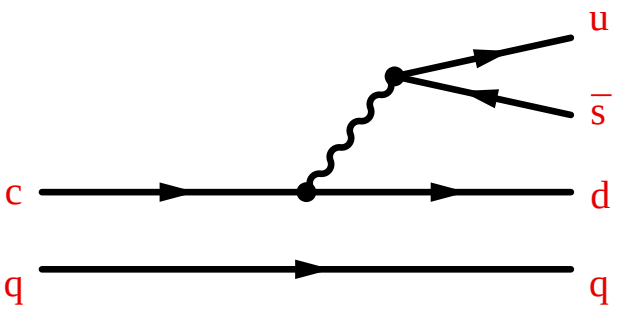
This filter is then applied to each event in my sample and the fitness is determined from the selected events. (The **1**s.)

Cabibbo Suppressed Decays

Doubly Cabibbo suppressed decays can only be observed in charm. Both W vertices are Cabibbo suppressed.



Cabibbo Favored



Doubly Cabibbo Suppressed

Doubly Cabibbo suppressed decays are chosen for this application since the final state particles are often identical (*e.g.*, $D^+ \to K^-\pi^+\pi^+$ vs. $D^+ \to K^+\pi^+\pi^-$). This eliminates many possible sources of systematics arising from inexact modeling of what the GP is doing.

Expected relative branching ratios: $\sim \tan^4\theta_c \approx 0.25\%$.

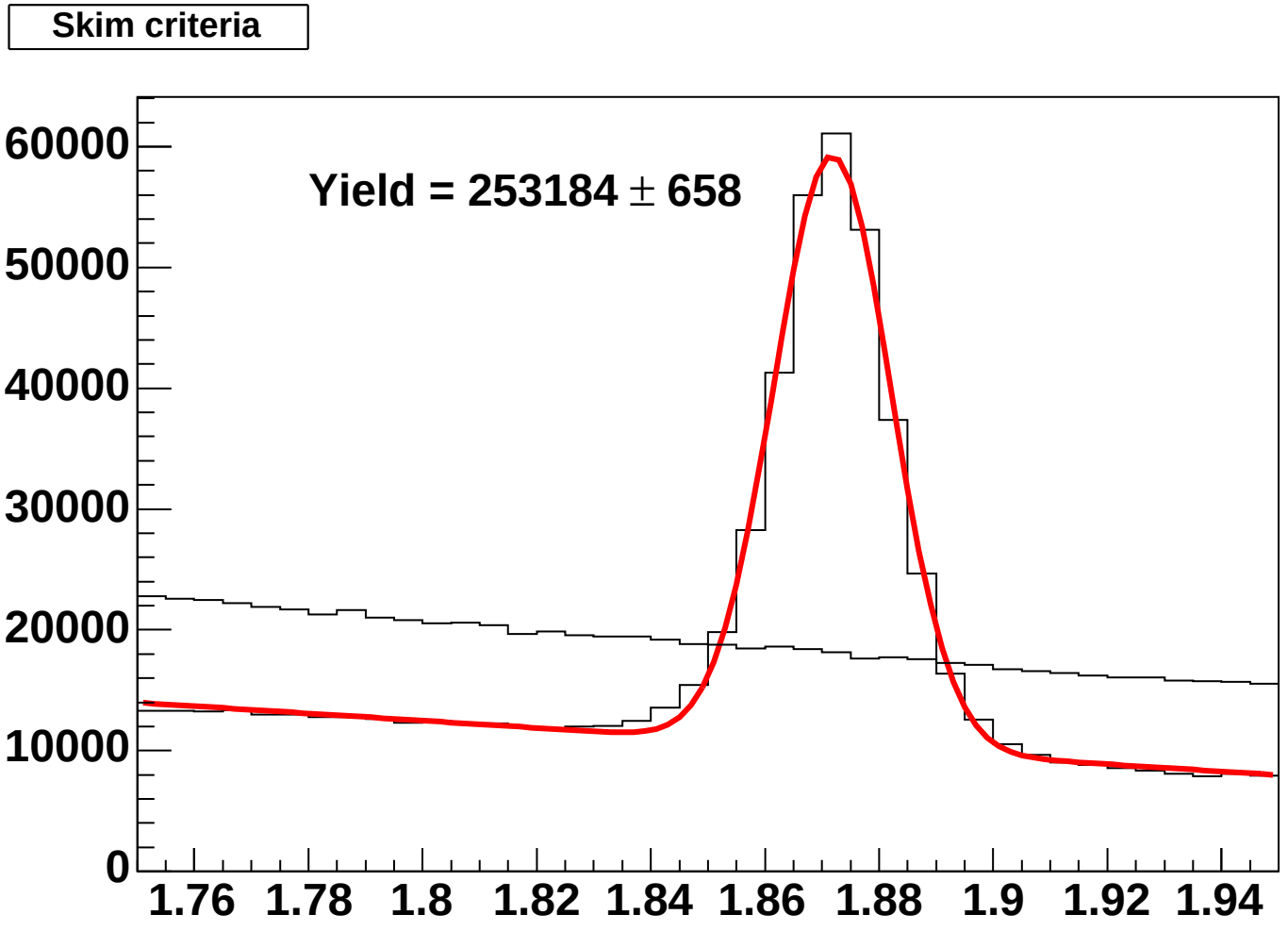
$D^+ \to K^+\pi^+\pi^-$

We first want to check this method on a known doubly Cabibbo suppressed decay. The first such decay discovered, $D^+ \to K^+\pi^+\pi^-$ has been known for many years now and it's branching ratio, relative to $D^+ \to K^-\pi^+\pi^+$, is reasonably well known. This branching ratio is surprisingly large (about $3\tan^4\theta_c$). The PDG value is $0.75 \pm 0.16\%$ relative to $D^+ \to K^-\pi^+\pi^+$.

Using the same data I do, a recently published FOCUS branching ratio measurement and Dalitz analysis finds about 200 events in this mode and measures the relative branching ratio as $0.65 \pm 0.08 \pm 0.04\%$

Can a GP analysis reproduce the known branching ratio? Can it improve upon the errors of a "traditional" analysis?

After skim (pre-GP) signals



Fit shows $D^+ \to K^-\pi^+\pi^+$ normalizing mode

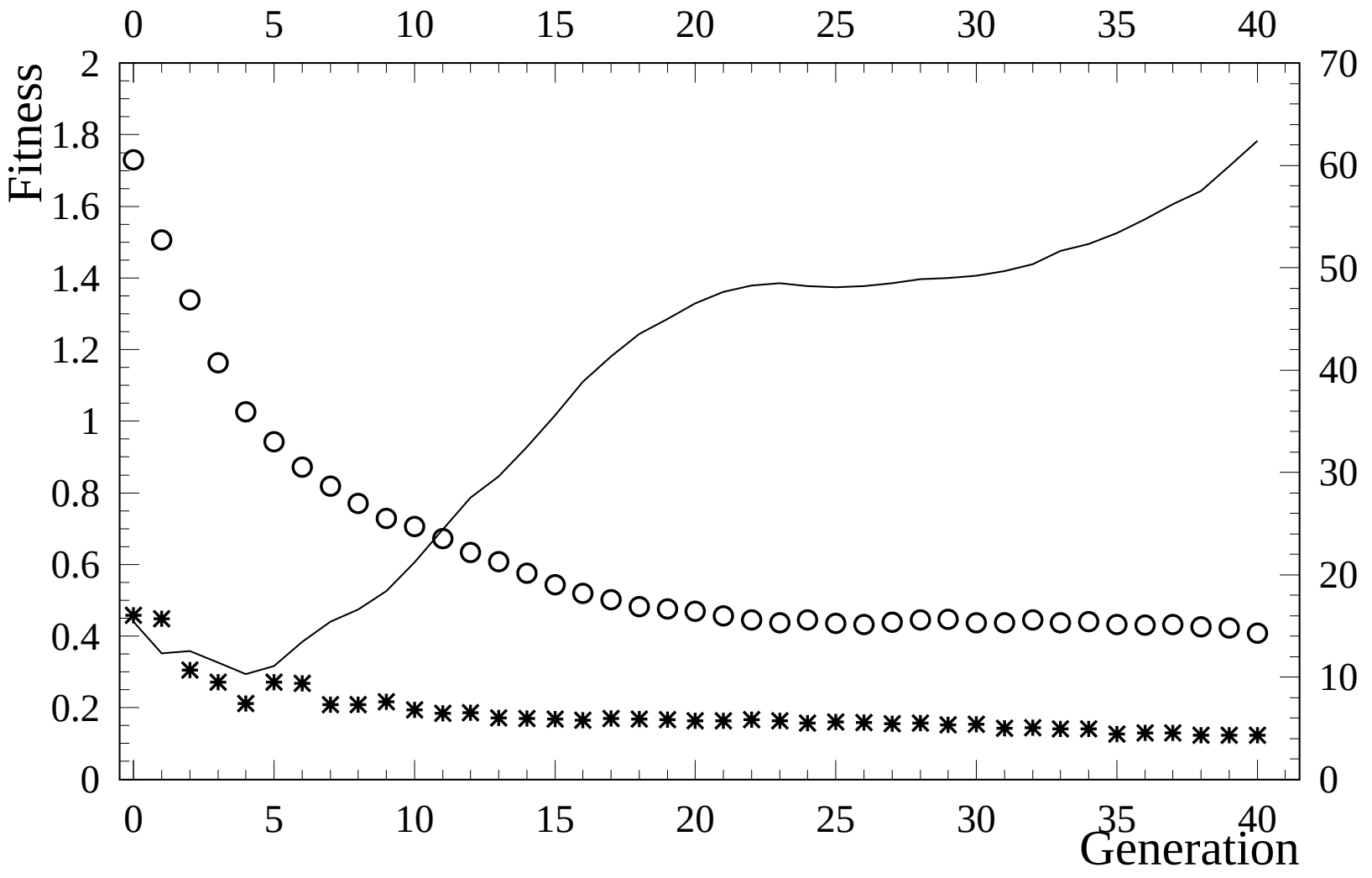
"Linear" histogram is DCS candidates

Evaluating the GP

For each program the GP framework suggests, we have to tell the framework how good the program is:

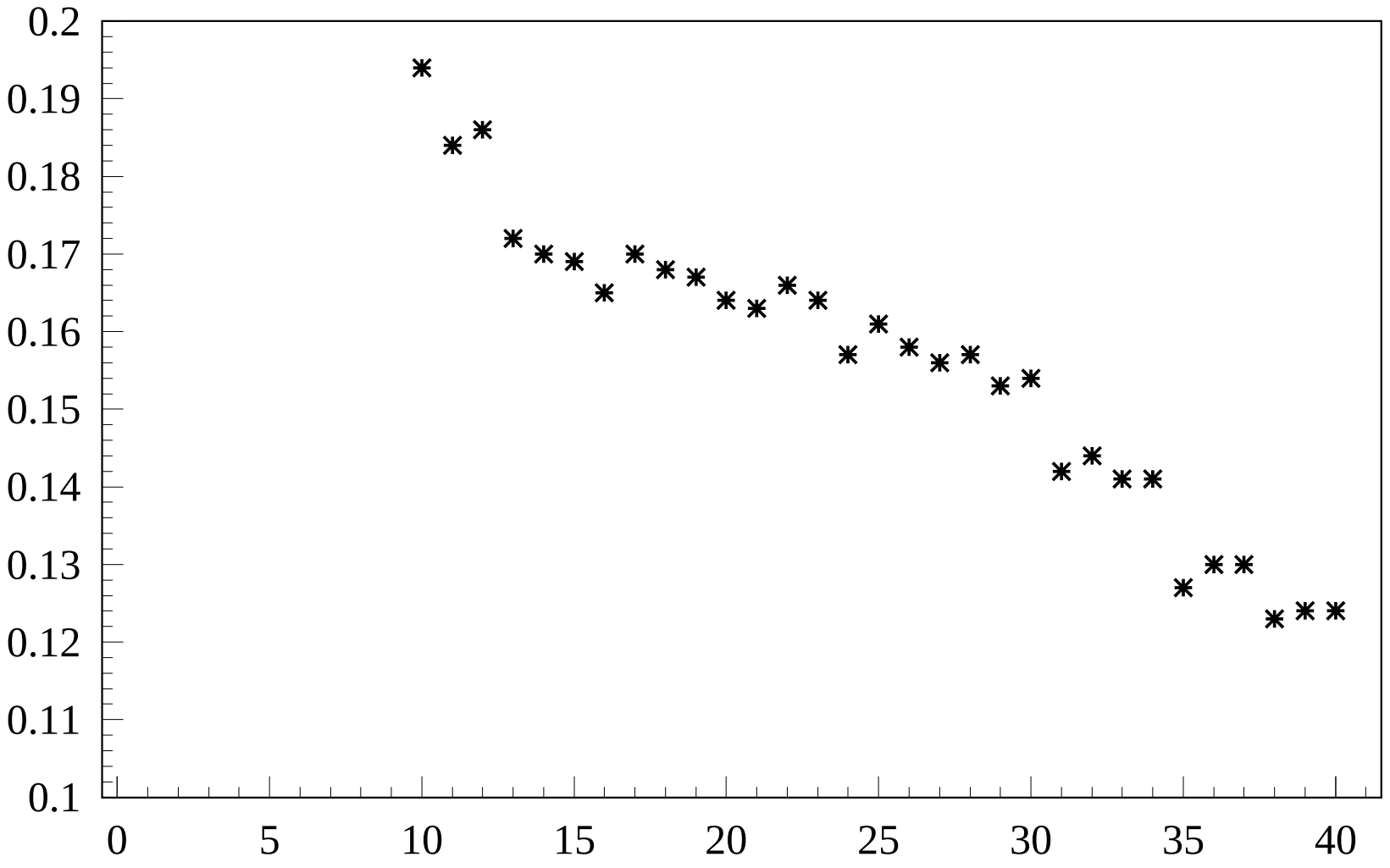
- All functions must be well defined for all input values, so $> \rightarrow 1$ (true) or 0 (false), $\log$ of neg. number, etc.
- Evaluate the tree for each event $\rightarrow$ a single value
- Select events for which Value of tree > 0
  - Initial sample has as loose cuts as possible
- Return a fitness to framework
- Could be $\propto \sqrt{S+B}/S$ (framework wants to minimize)
- In this case S is from CF mode scaled down to expected/measured DCS level. B is from fit to DCS BG (masking out signal region if appropriate).

Evolutionary Trajectory



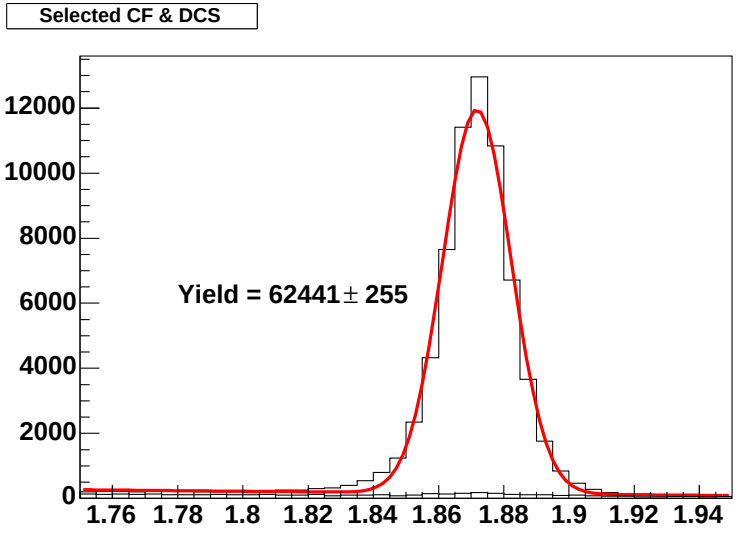
Circles: average, Stars: best, Line: avg. size

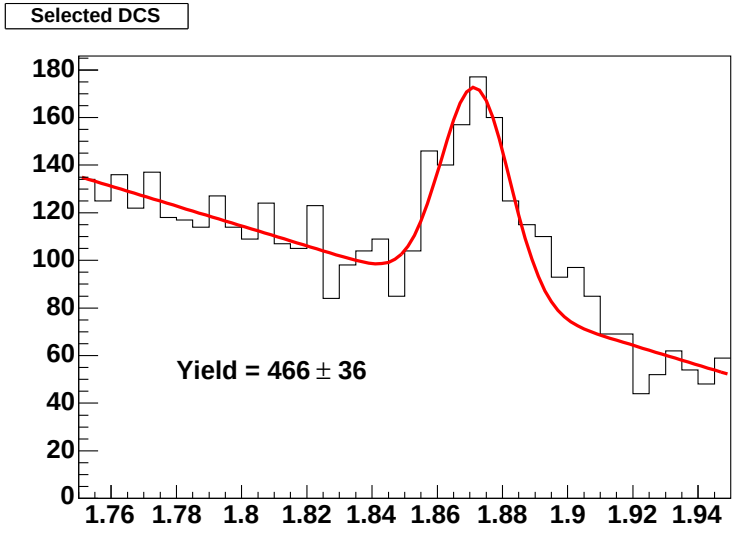
Expansion of best trees



Stars are the best tree, still evolving at generation 40

CF and DCSD signals



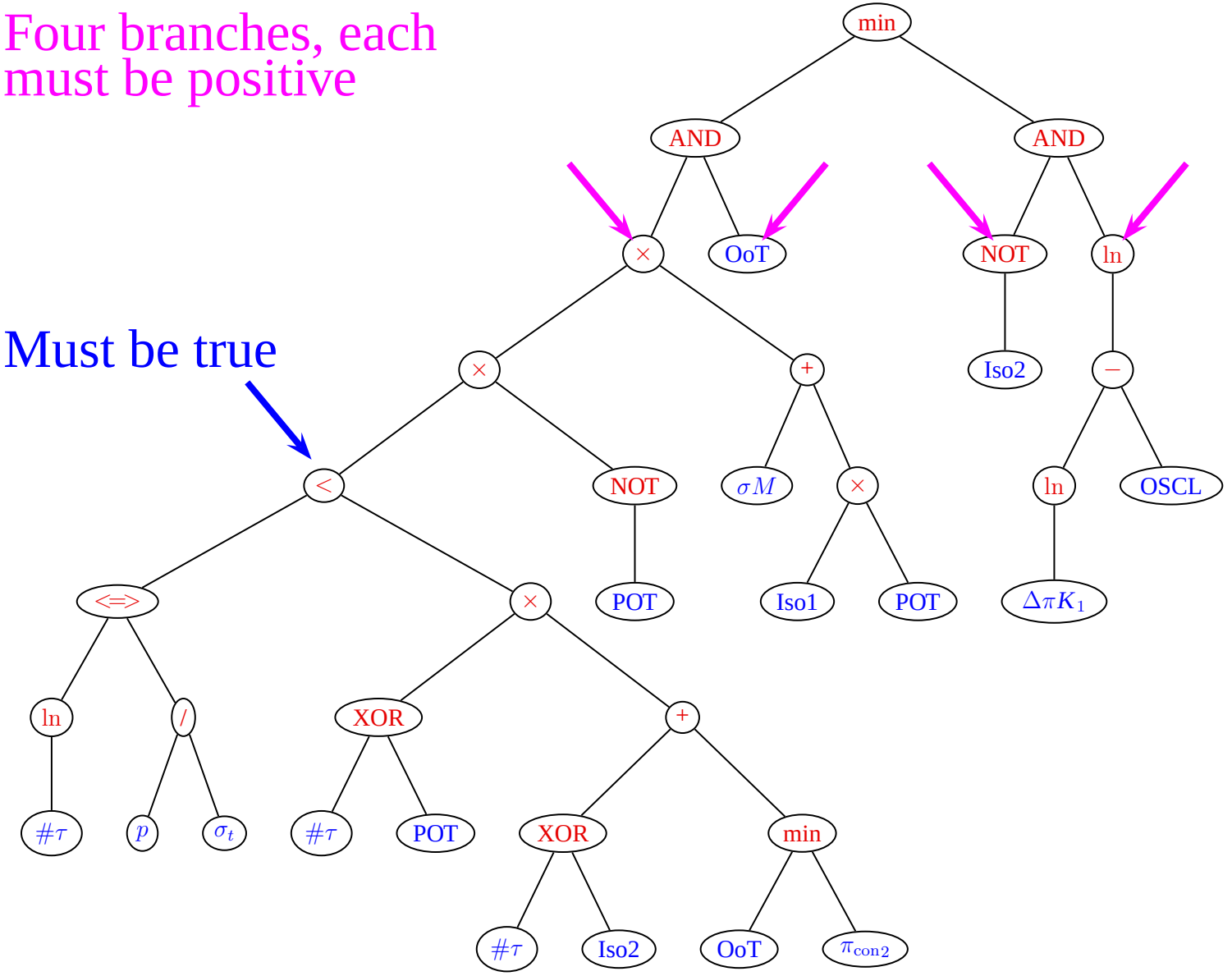


- Retains 62K of 253K original CF events
- DCS background reduced a factor > 150
- DCS mass and width are fixed to CF values

Best tree (40 generations)

Four branches, each must be positive

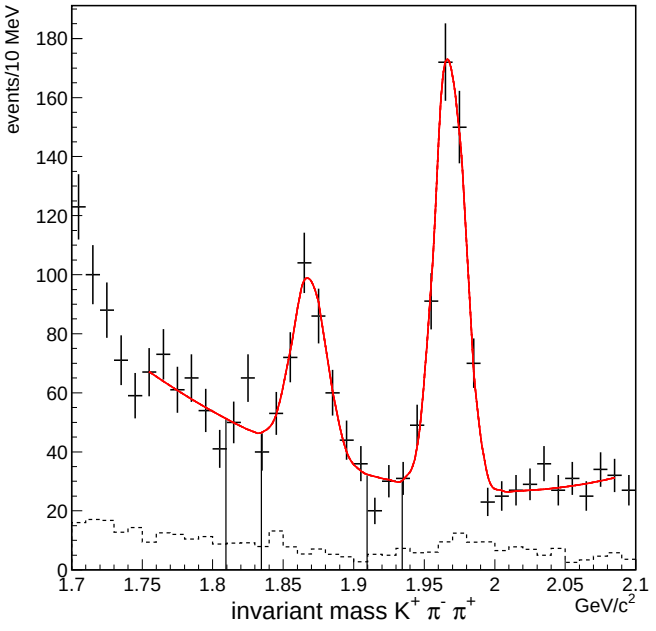
Must be true



Comparison with Cut Method

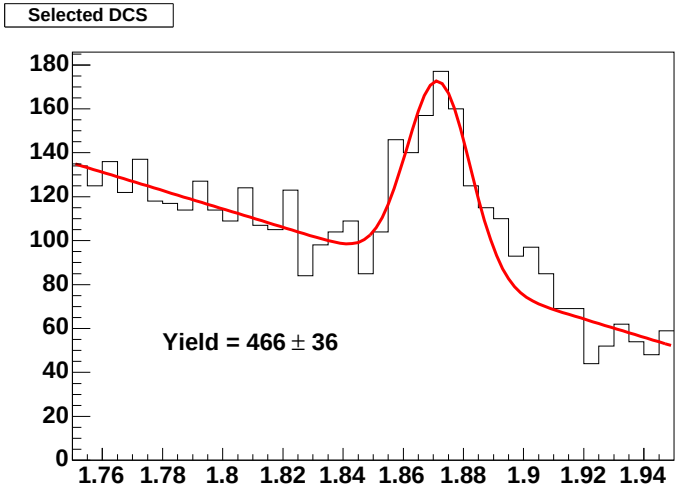
How does this compare with our normal method?

- From PLB 601 10–19, measured BR of $D^+ \to K^+\pi^+\pi^-$
  - Rel. BR — PLB: $0.65 \pm 0.08 \pm 0.04$, GP: 0.76 ± 0.06
- Not a perfect comparison, not optimized on $S/\sqrt{S+B}$



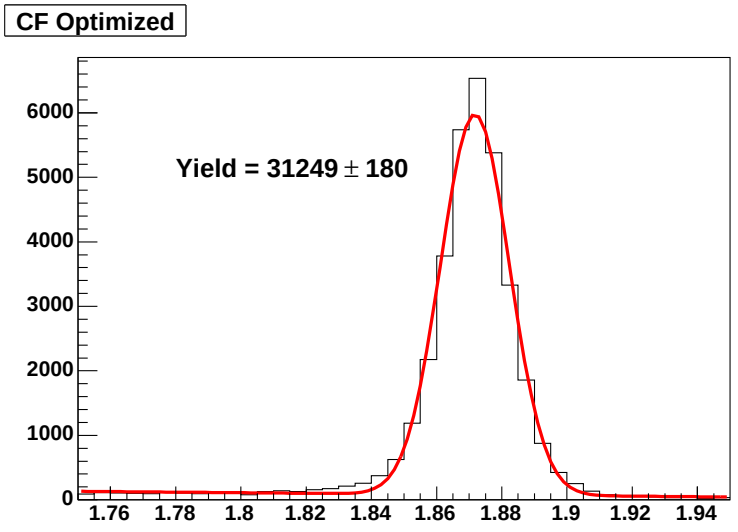
D_s^+ shown on right

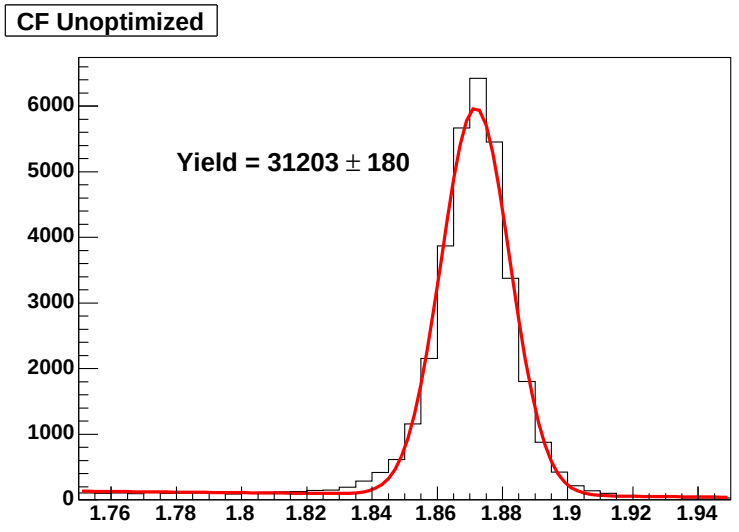
- Similar signal to noise
- Cuts: Yield $= 189 \pm 24$ events
- GP: Yield $= 466 \pm 36$ events



What about bias?

Try to reduce by putting in a penalty (0.5%) for each node (make sure added nodes are valuable).
Then, to test for bias, optimize on only half the events (left).





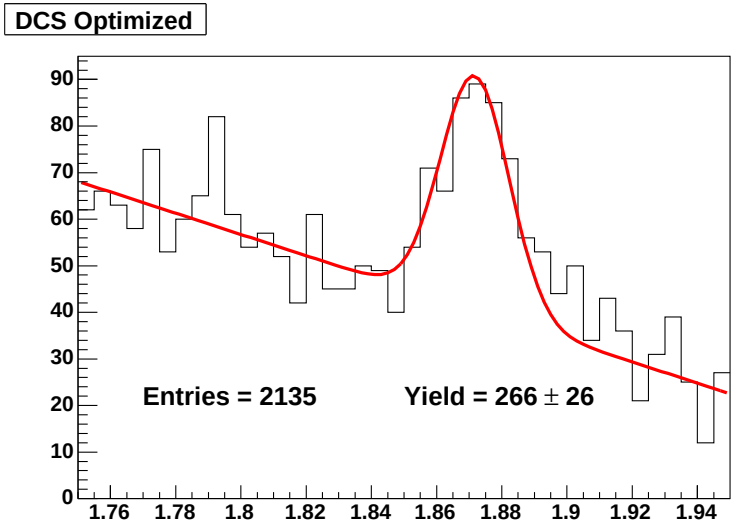
31250 ± 180 events

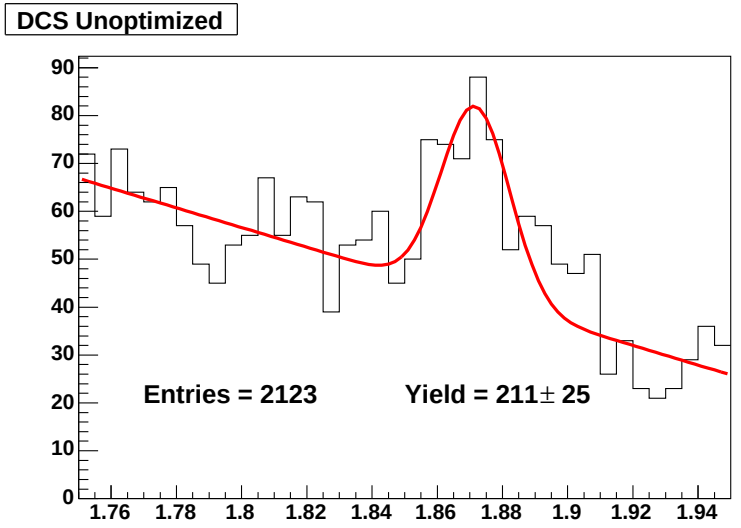
31200 ± 180 events

No evidence of selection induced bias here.

Bias, continued

Look at the same plots for doubly Cabibbo suppressed events.

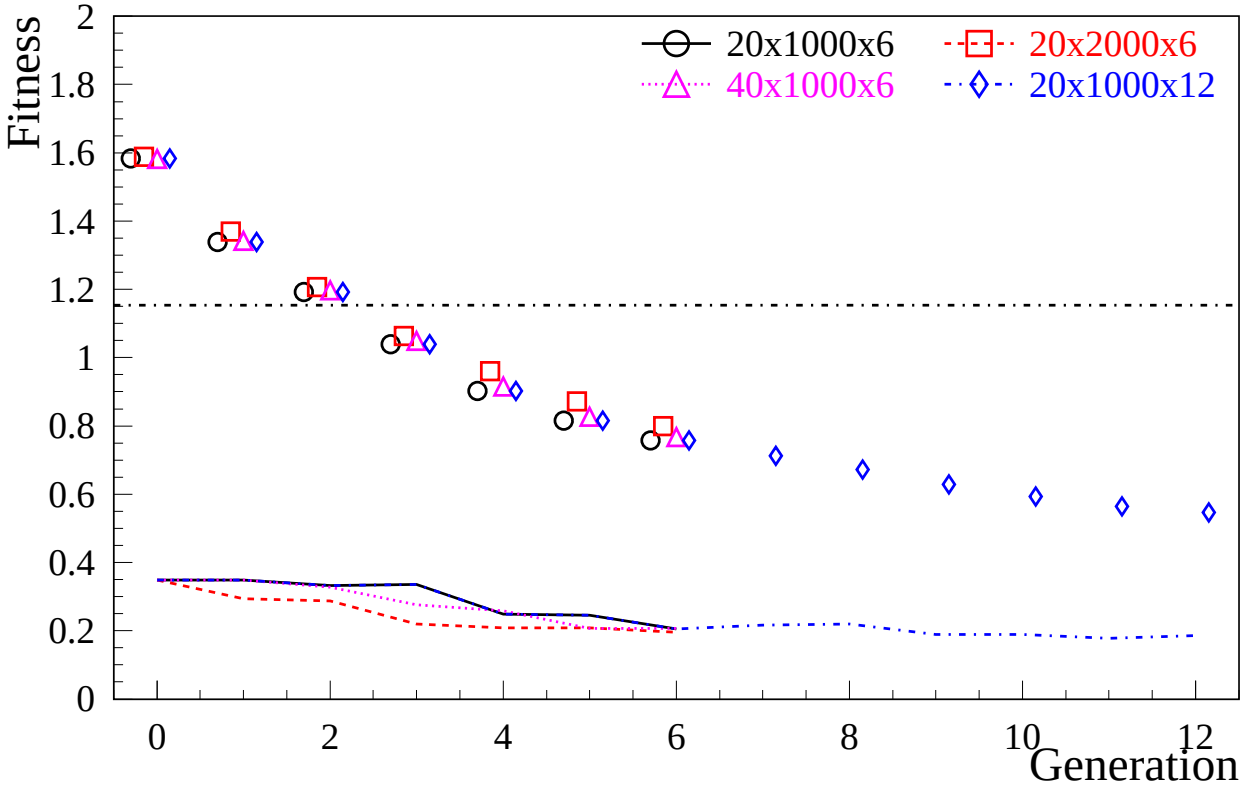




Doubly Cabibbo suppressed distributions are also similar: 2135 (optimized) vs. 2123 (unoptimized) events in whole plot. (Remember optimization is blind to signal region, so differences there not due to GP.)

Tuning GP parameters

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



- Cleaner starting sample for these studies
- Points are average, dotted line is best
- More generations is only clear improvement
- Plots in analysis section use 20x1500x40

Data MC comparisons

Since the two decays are nearly identical, what is important is that the efficiency of the tree for CF and DCS modes is the same. What need not be known is the *absolute* efficiency on a single mode. But, we can study this with Monte Carlo.

For the 40^{th} generation program, the MC efficiencies are:

	CF Eff. (%)	DCS Eff. (%)
Skim cuts	5.76 ± 0.01	5.57 ± 0.01
GP Selection	1.43 ± 0.01	1.41 ± 0.01
GP/Skim	24.91 ± 0.12	25.29 ± 0.12

But, we can look at GP/Skim for the CF mode in the data.

Data MC comparisons

Since the two decays are nearly identical, what is important is that the efficiency of the tree for CF and DCS modes is the same. What need not be known is the *absolute* efficiency on a single mode. But, we can study this with Monte Carlo.

For the 40^{th} generation program, the MC efficiencies are:

	CF Eff. (%)	DCS Eff. (%)
Skim cuts	5.76 ± 0.01	5.57 ± 0.01
GP Selection	1.43 ± 0.01	1.41 ± 0.01
GP/Skim	24.91 ± 0.12	25.29 ± 0.12

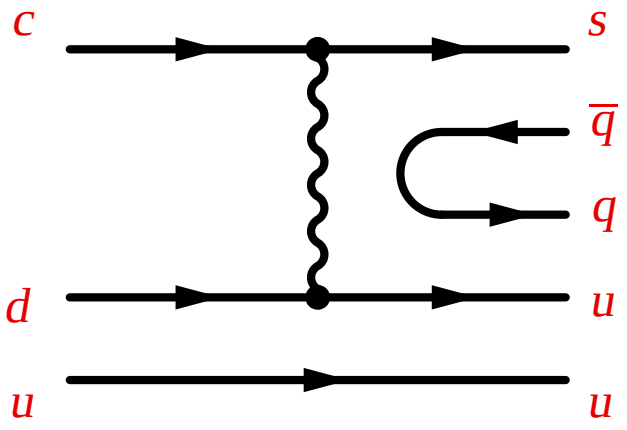
But, we can look at GP/Skim for the CF mode in the data. We find $24.66 \pm 0.08\%$ (vs. $24.91 \pm 0.12\%$).

So, not only do the GP efficiencies for MC agree, the one place we can compare the GP on MC and data, it also comes out very close.

$\Lambda_c^+ \longrightarrow \boldsymbol{pK^+\pi^-}$

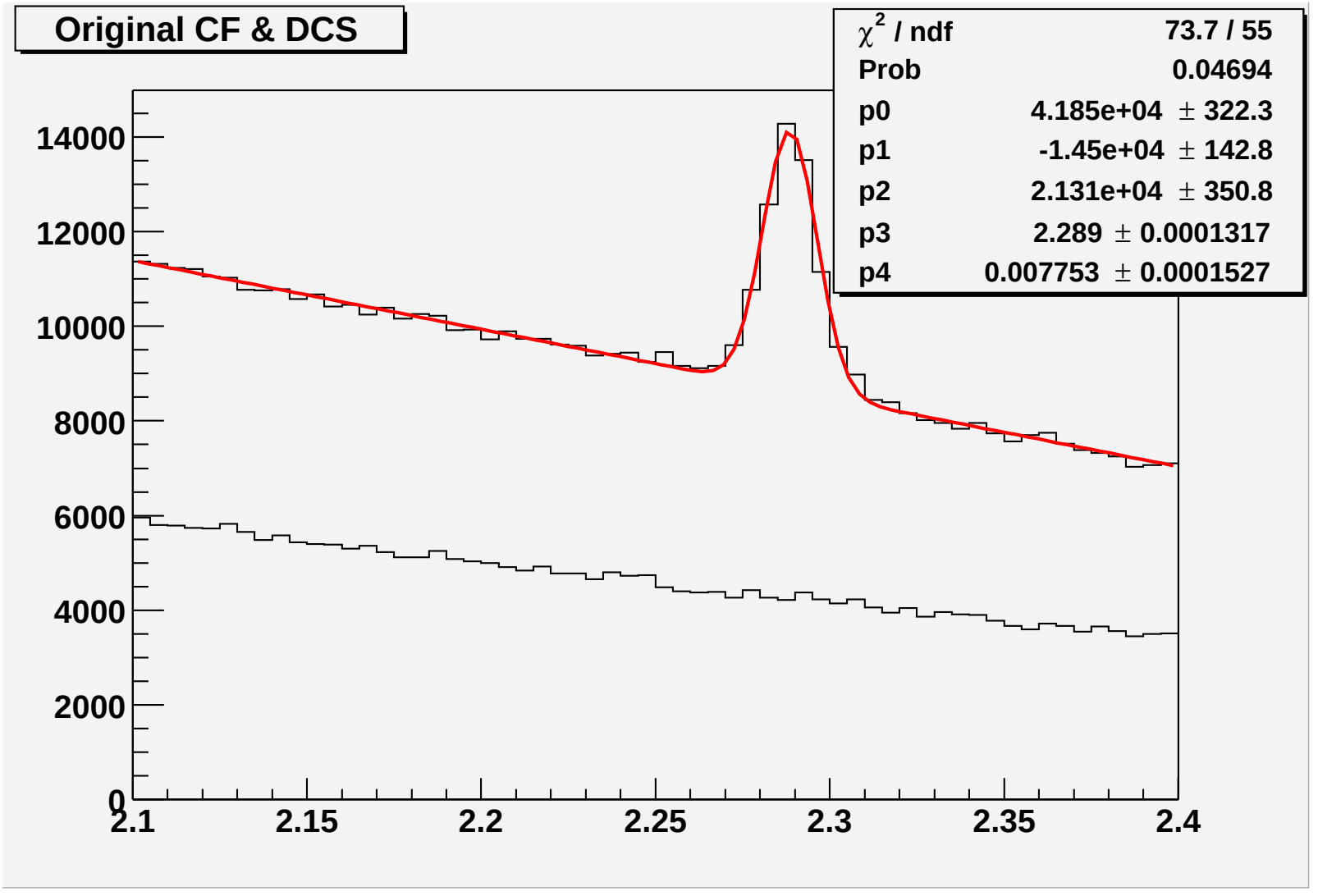
The next decay we consider is $\Lambda_c^+ \longrightarrow pK^+\pi^-$. There are no observations or limits. Even an observation of $\tan^4\theta_c$ relative to $\Lambda_c^+ \longrightarrow pK^-\pi^+$ is challenging for FOCUS, but there is a complication.

The Cabibbo favored mode has an W-exchange contribution while the DCS decay does not. (This contribution is also why the Λ_c^+ lifetime is about one half the Ξ_c^+ (csu) lifetime.)



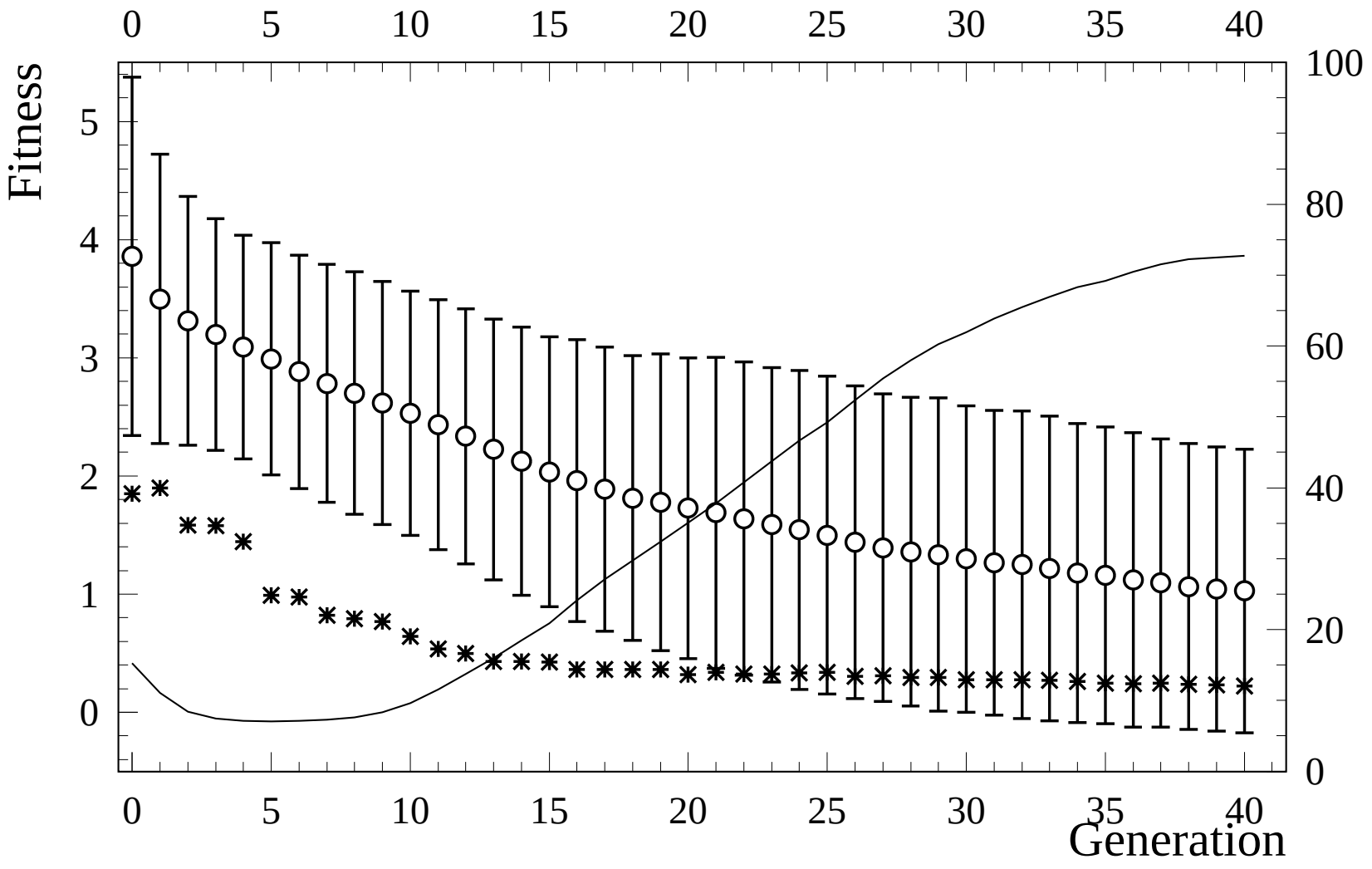
So, the expected branching ratio will be reduced.

After skim (pre-GP) signals



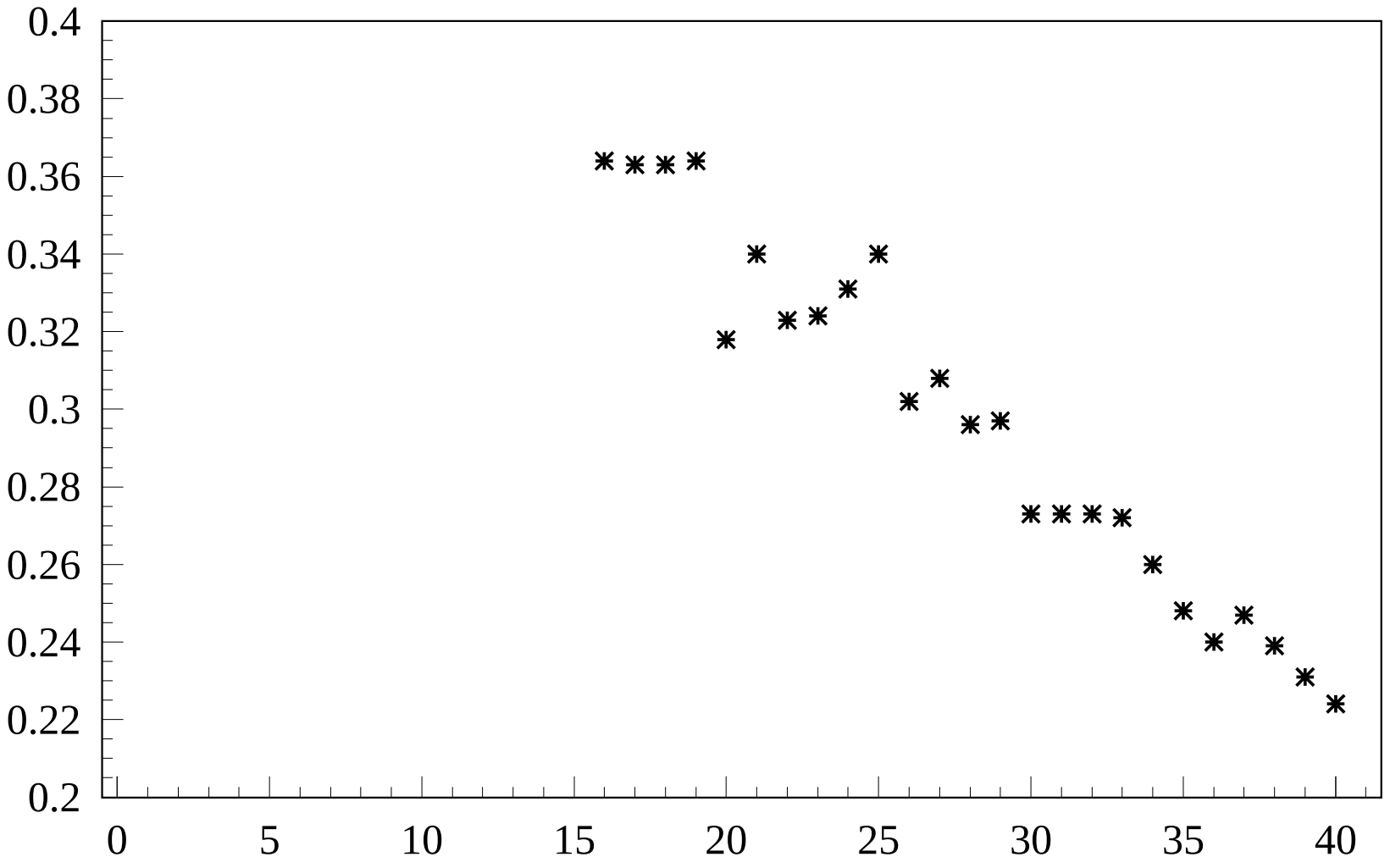
Lower histogram is DCS candidates

Λ_c^+ Evolutionary Trajectory



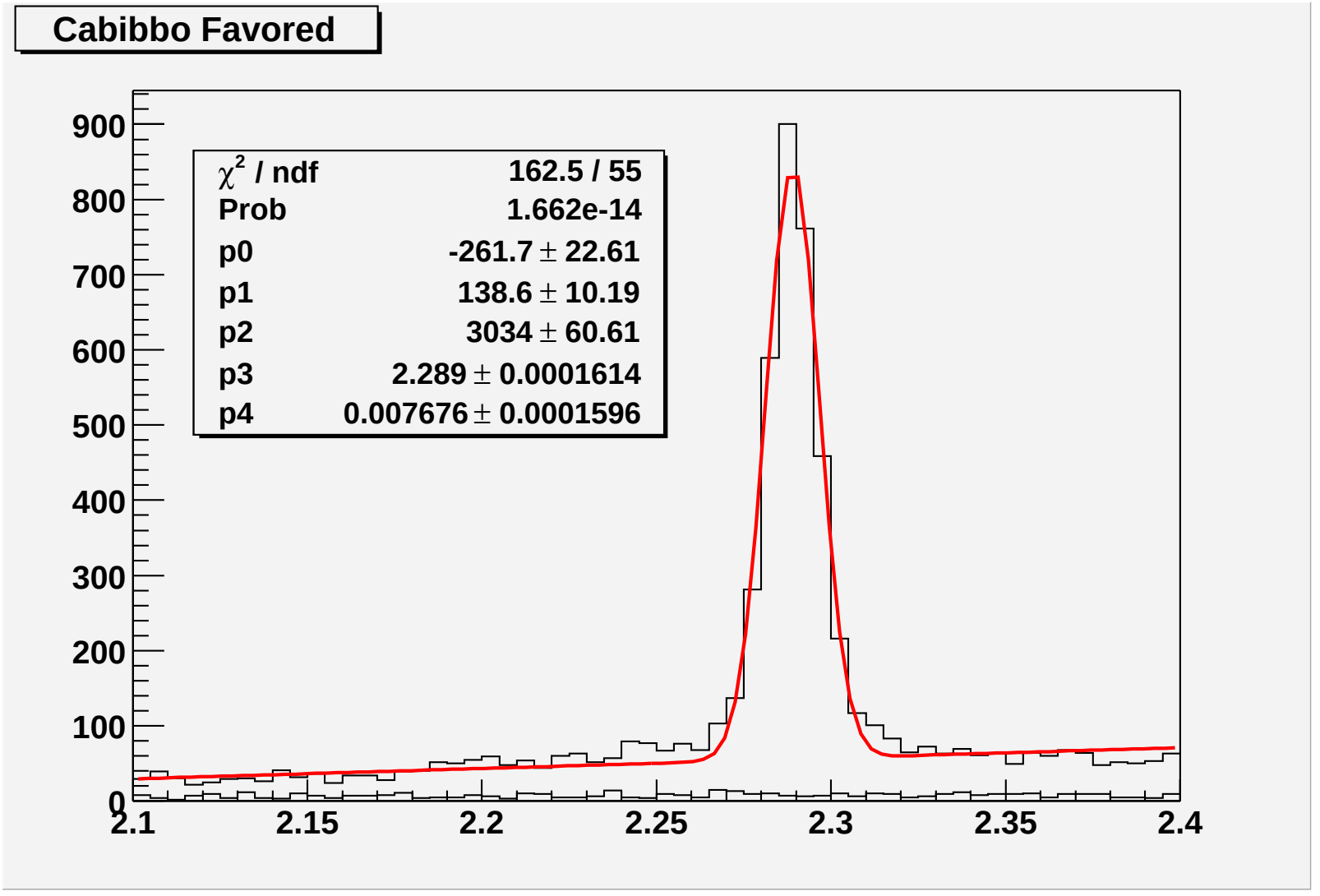
Circles: average, Stars: best, Line: avg. size

Expansion of best trees



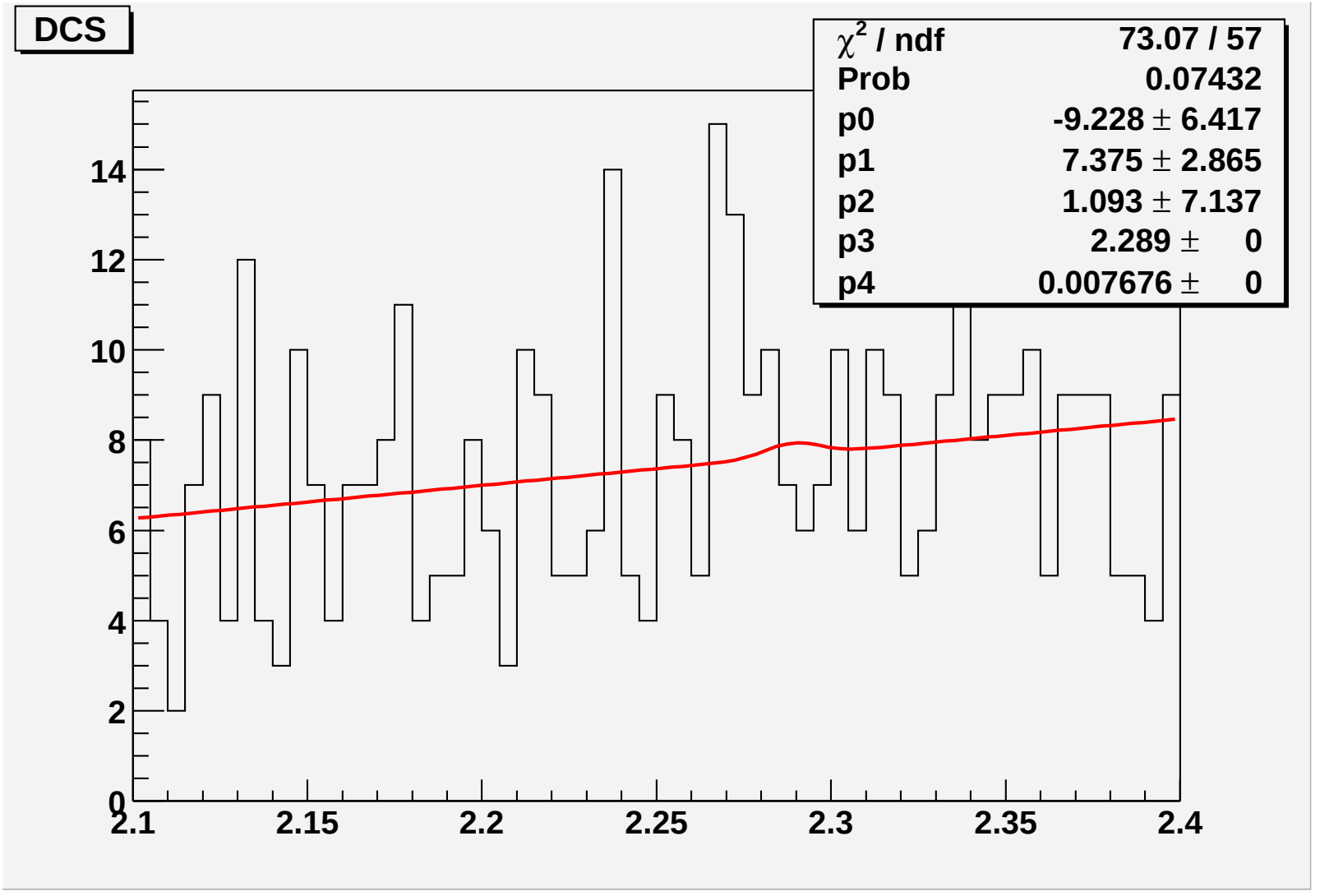
Stars are the best tree, still evolving at generation 40

$\Lambda_c^+ \to pK^-\pi^+$ (CF) signal



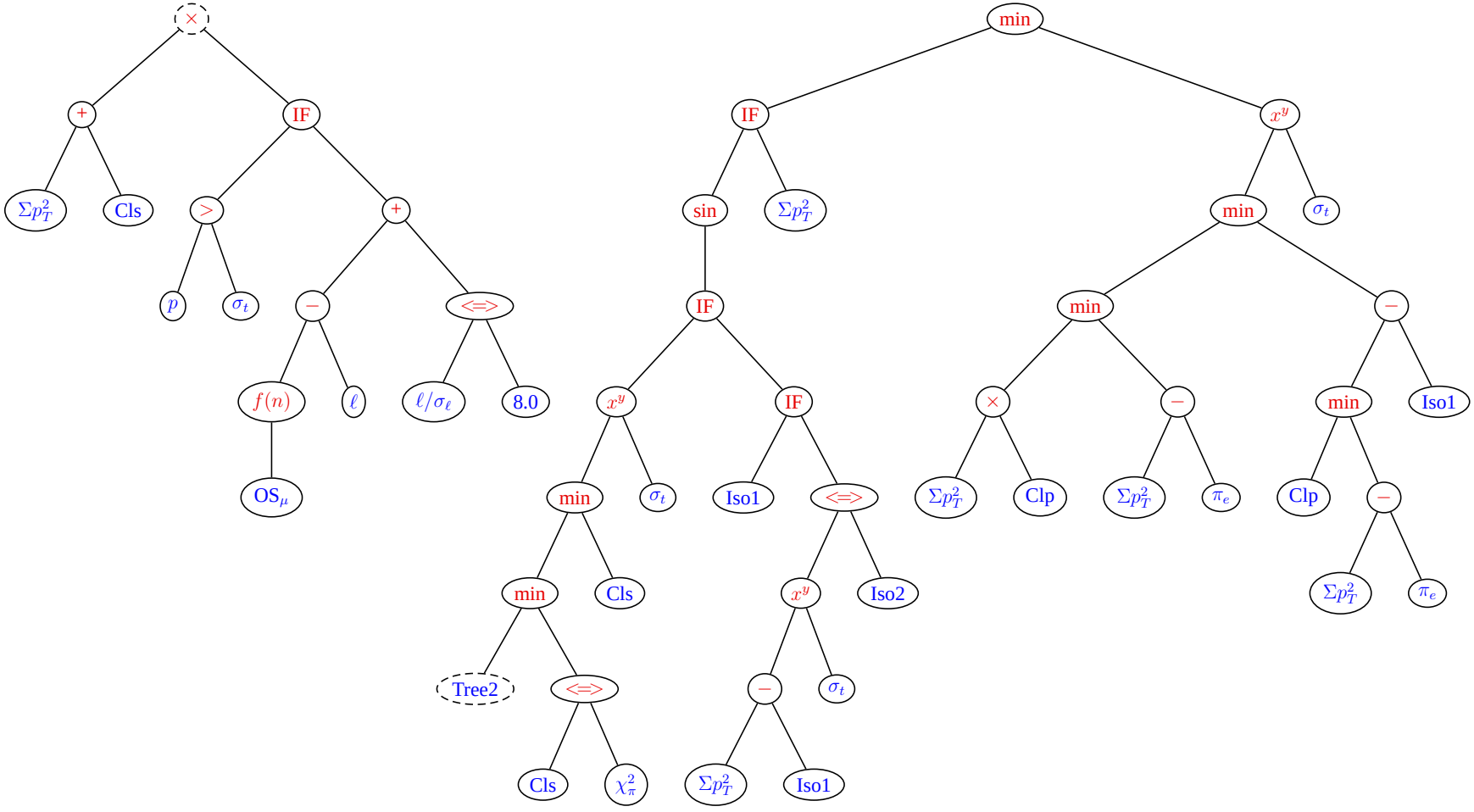
Retains 3,000 of 21,300 original events, lower is DCS

$\Lambda_c^+ \to pK^+\pi^-$ (DCS) signal



Mass and width are fixed to CF values

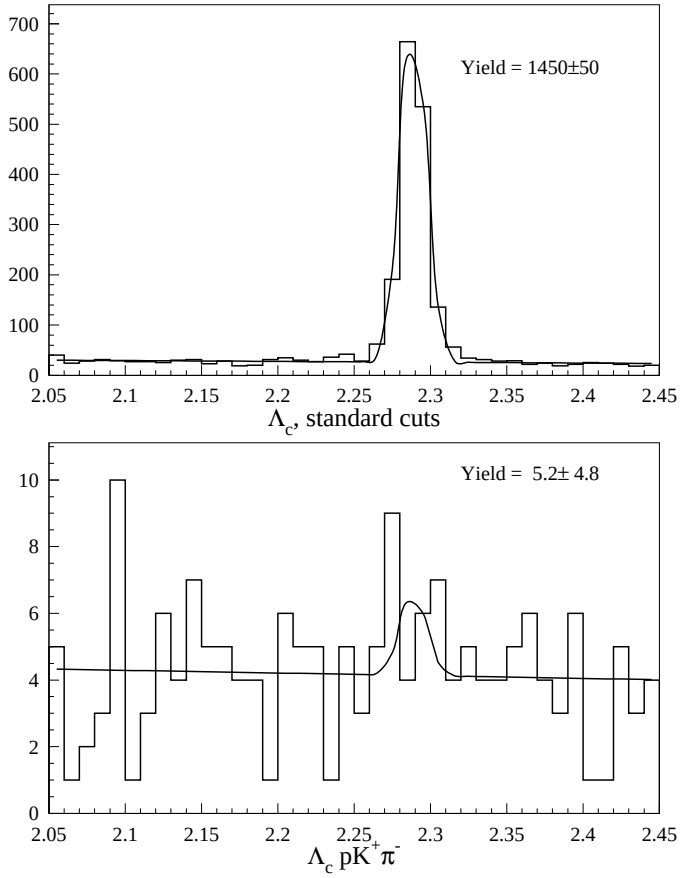
Best tree (40 generations)



Comparison with Cut Method

How does this compare with our normal method?

- I've looked for $\Lambda_c^+ \to pK^+\pi^-$ before by maximizing $S/\sqrt{S+B}$ with manual "hill-climbing."



- Important quantity: $\sigma Y_{\mathrm{DCS}}/Y_{\mathrm{CF}}$
- Cuts: $\sigma Y = 4.8$, Yield $= 1450$
- GP: $\sigma Y = 7.1$, Yield $= 3030$
- $\sigma Y/Y$ for GP $= 2.3 \times 10^{-3}$
- $\sigma Y/Y$ for cuts $= 3.3 \times 10^{-3}$
- GP method is $\sim$ 50% better, but still need luck or significant improvements to observe $\Lambda_c^+ \to pK^+\pi^-$

Conclusions

This method shows promise, but there are some caveats

- May be more challenging for modeling
  - Our Monte Carlo seems up to the task
- 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 deserves further exploration

Backup slides

Backup slides

$f(n)$ function

A threshold function used in neural networks:

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