

Genetic Algorithms and Genetic Programming

Pavia University and INFN

Third Lecture: Application to HEP

Eric W. Vaandering

ewv@fnal.gov

Vanderbilt University

Overview

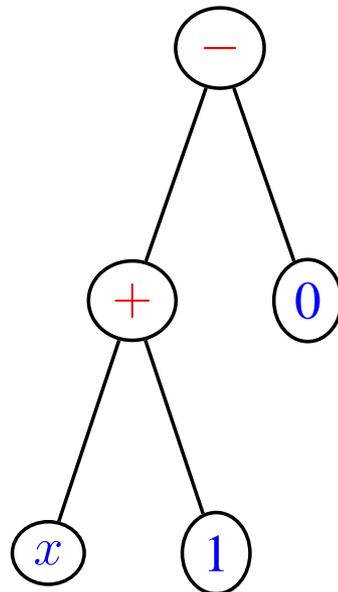
- Machine learning techniques
- Genetic Algorithms
- Genetic Programming
 - Review
 - Application to a problem in HEP

GP Review

Let's quickly review what we've covered the last two times:

Genetic programming transforms, from one generation to the next, a group of programs (proposed solutions to a problem) into a "better" group of programs based on biological models and the concept of fitness.

For our purposes, we've been representing these programs as trees, like this:



Which corresponds to the "program"

$$(x + 1) - 0$$

“Running” the GP

With what we learned last time, we are ready to “run” the GP (*i.e.* find a solution). Recall that these are the steps a GP framework will take:

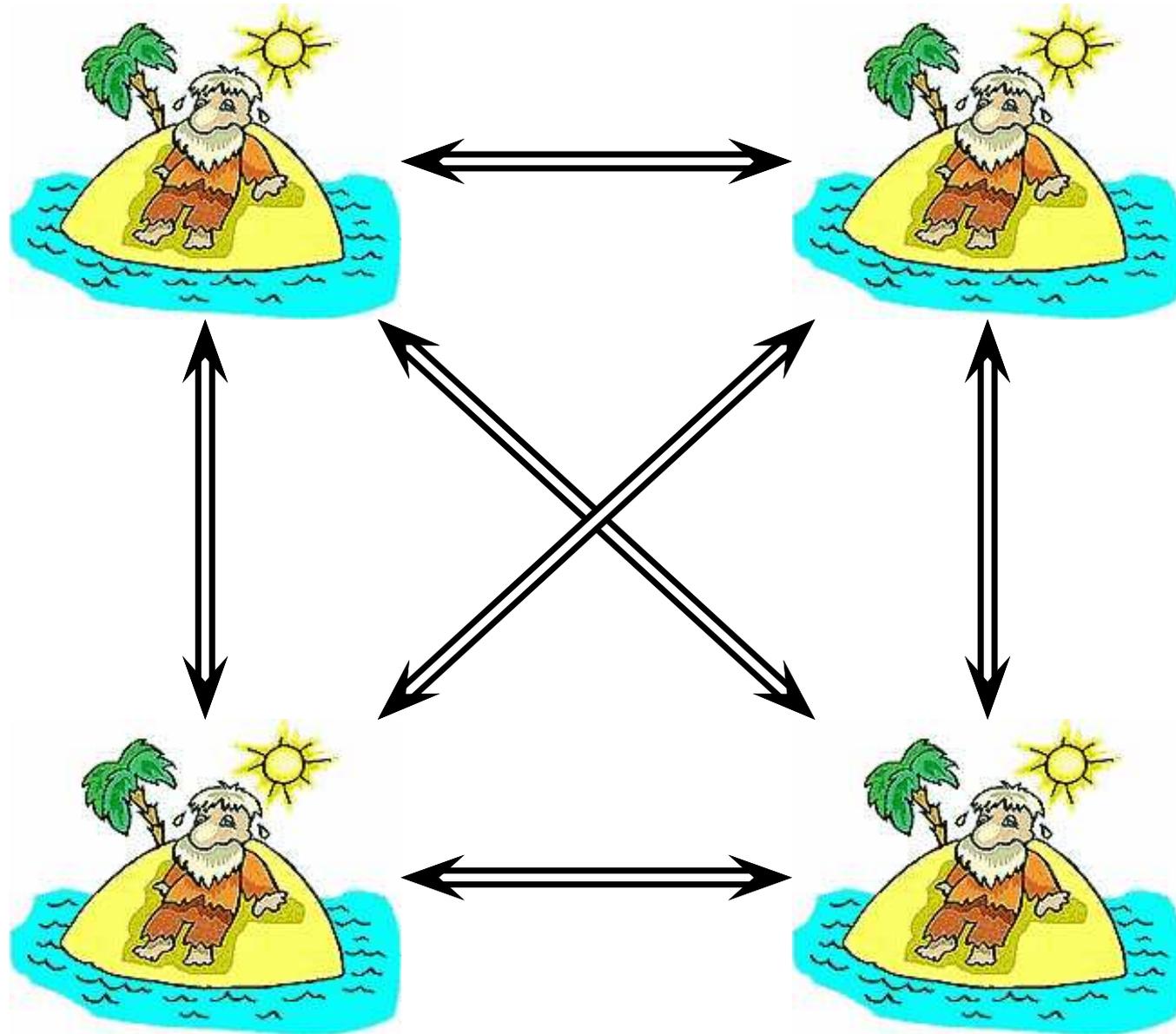
- User has defined functions and definition of fitness
- Generate a population of programs (few hundred to few thousand) to be tested
- Test each program against fitness definition
- Choose genetic operation (copy, crossover, or mutation) and individuals to create next generation
 - Chosen randomly according 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

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



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 a 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. Neural networks are used effectively for *B* flavor tagging at LEP, CLEO, etc.

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

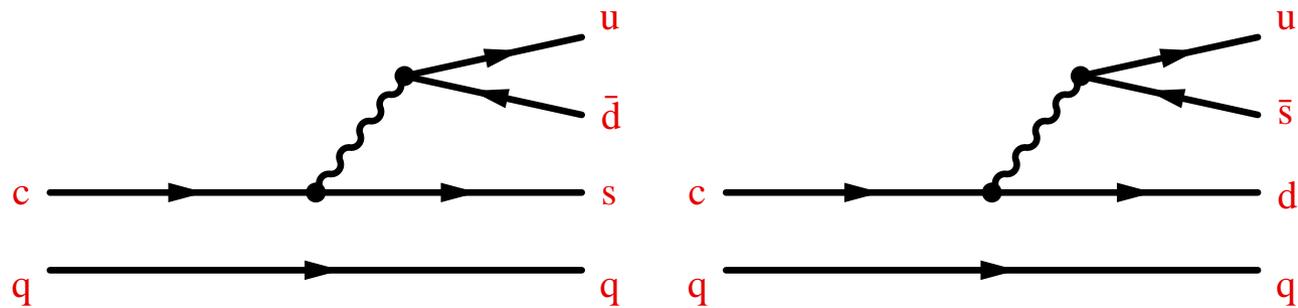
Questions

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 finding good combinations?
- What about units? Can you add a momentum and a mass?
 - All numbers are defined to be unit-less

Cabibbo Suppressed Decays

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



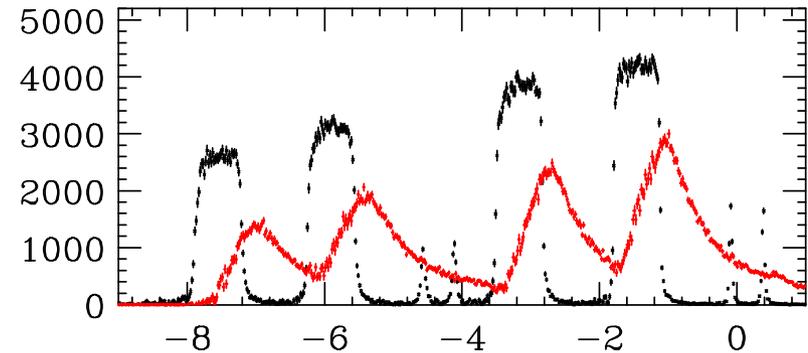
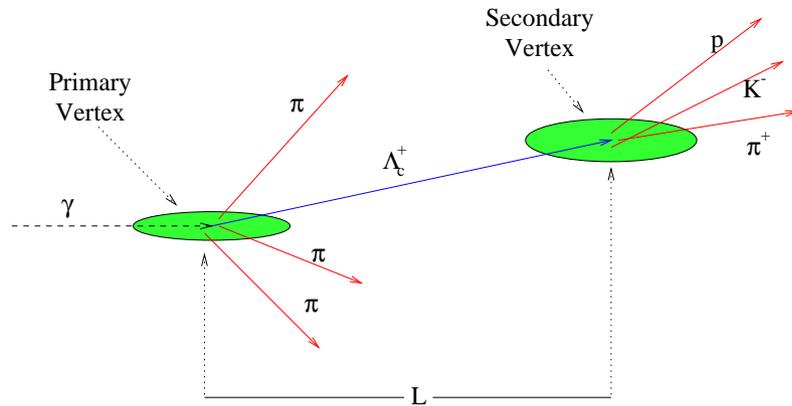
Cabibbo Favored

Doubly Cabibbo Suppressed

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

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

Target and Vertexing



Some details of the FOCUS candidate driven vertexing

- L : Distance between production and decay vertices. l/σ_l , significance of separation
- OoT: Significance of decay being out of target material
- CLS, CLP: Confidence levels 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

Variables and Operators

Give the GP lots of things to try:

Functions (22)		Variables ($D^+ - 35, \Lambda_c^+ - 37$)		
\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	π_{con}	p_T
x^y	NOT	CLS	Track χ^2 's	Σp_T^2
$\sqrt{\quad}$	AND	CLP	OS Vertex CL	m_{err}
log	OR	Iso1	OS $\Delta W(\pi K)$	μ_{max}
$>$	XOR	Iso2	OS CL_μ	TS/NoTS
$<$	IF	#life	Real $(-2, +2)$	REME
$\langle \Rightarrow \rangle$	sin	Pri. OoT	Int $(-10, +10)$	
$f(n)$	cos	$p(\Lambda_c^+)$	0,1	

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

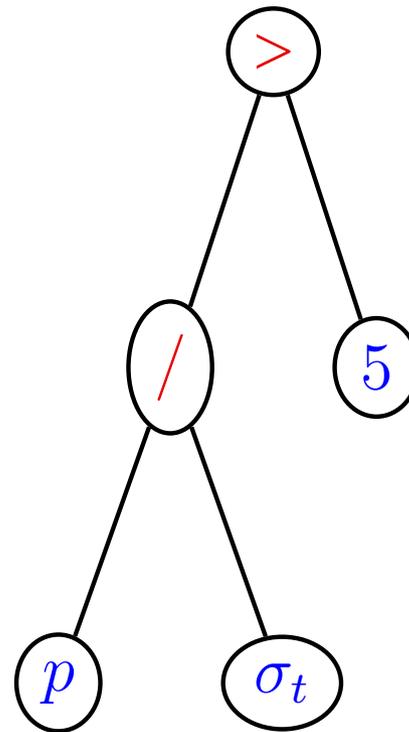
Evaluating the GP

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, which gives a single value
- Select events for which $\text{Tree} > 0$
 - 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).

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.

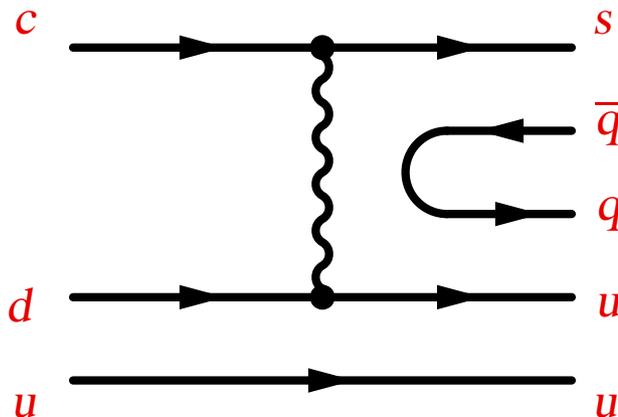


This filter is then applied to each event in my sample and the fitness is determined from the selected events.

$$\Lambda_c^+ \rightarrow \mathbf{p} \mathbf{K}^+ \pi^-$$

The first decay we're looking for is $\Lambda_c^+ \rightarrow p K^+ \pi^-$. There are no observations or limits. Even an observation of $\tan^4 \theta_c$ relative to $\Lambda_c^+ \rightarrow p K^- \pi^+$ is challenging, 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^+ lifetime.)



So, the expected branching ratio will be reduced, maybe by 50%.

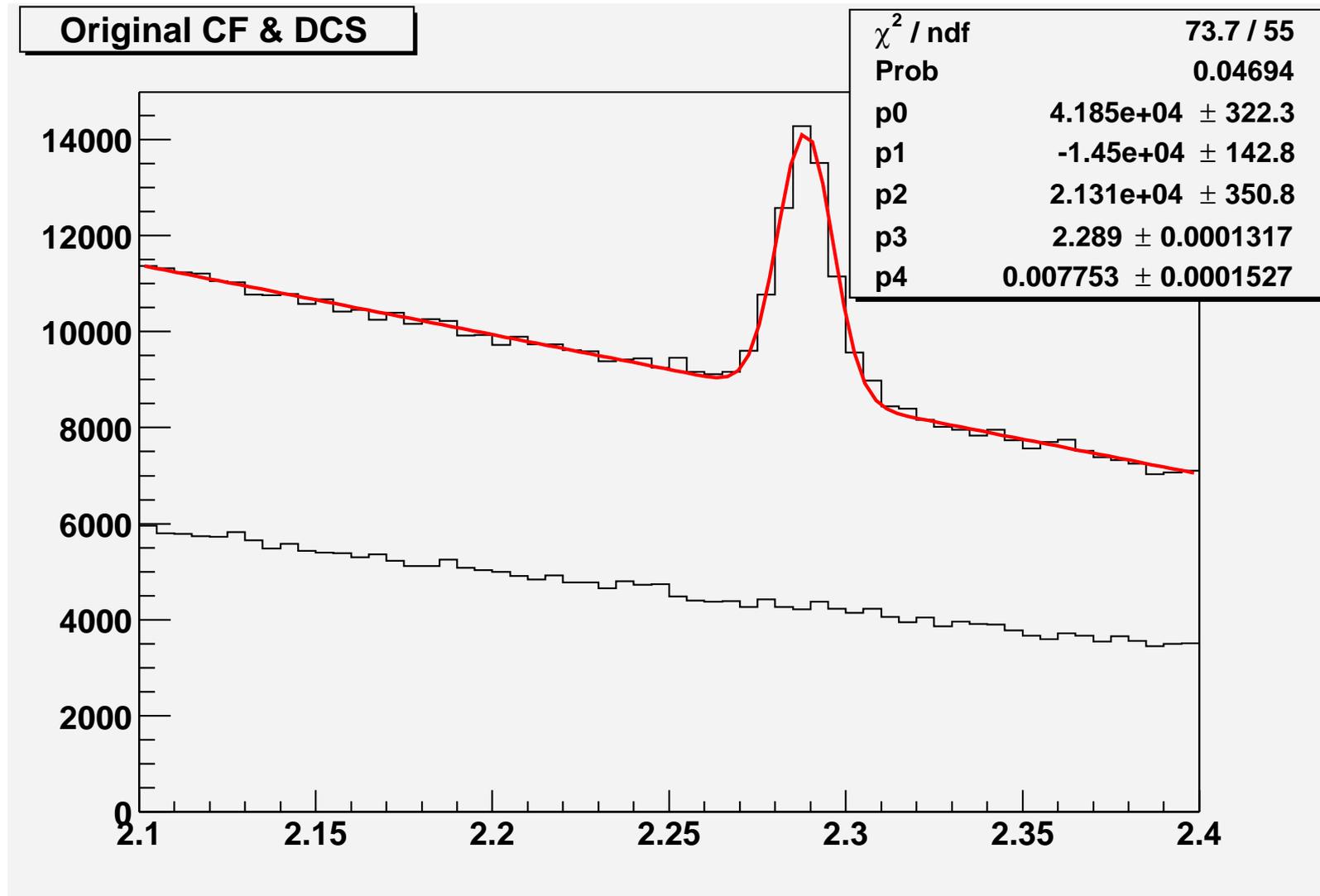
Sample Preparation

FOCUS collected about 6×10^9 events in its one year run. This is way to many events to store, let alone look at with Genetic Programming (about one million is the most I can handle today).

Throughout the data reconstruction process, the number of events has been reduced several times. The last time by me. But, we keep as many events as possible so that the GP has room to work. (To find out what makes a good $\Lambda_c^+ \rightarrow pK^- \pi^+$ or $\Lambda_c^+ \rightarrow pK^+ \pi^-$ decay.) This initial process was done with hard cuts as explained before.

When I begin this process, the Cabibbo favored decay is clearly visible, but we have no hope of observing the doubly Cabibbo suppressed decay without further purification of the data.

After skim (pre-GP) signals



Lower histogram is DCS candidates

Starting the GP

At this point, I start the GP constructing filters for the data. It runs on 40 CPUs for 40 generations with 2,000 individual/generation. This means it will look at 3.2 million possible solutions to our problem.

This may seem like a lot, but remember the possible search space is $\gg 10^{100}$, so in reality it will just sample a very small amount of the space. I will rely on the evolution of the GP to guide itself into the right search space.

While I'm ultimately interested in what comes out of the 40th generation, it helps us understand what's happening if we look earlier too.

Size (Parsimony) Pressure

An aside: I apply a penalty to the fitness for each node in the tree. This is to try to keep the tree sizes small. This is *not* recommended by the literature:

- How do you know the right penalty?
- Maybe you miss out on some subtree that needs to grow over time
- Not really shown to help

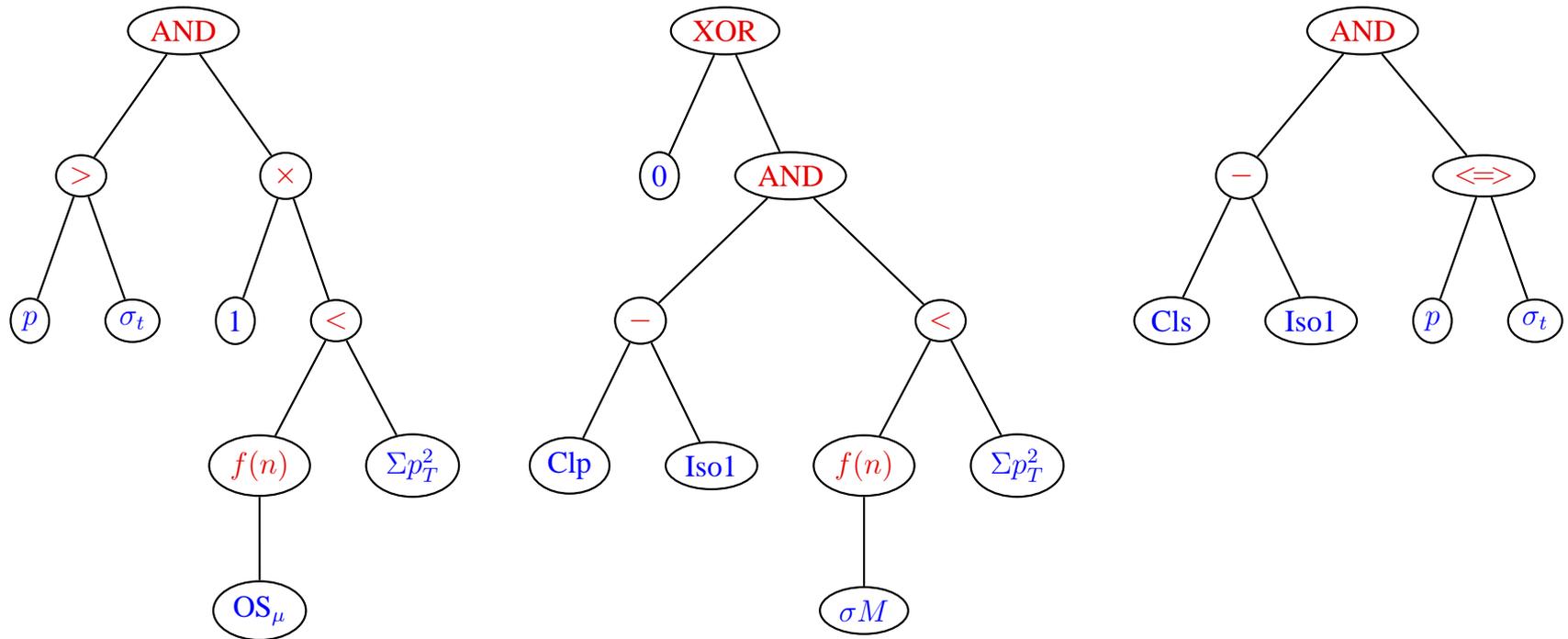
Never the less, I still do it. Why?

- Reduces execution time
- Worry about training the GP to recognize individual events
- Better chance to understand final trees

The penalty I assess is 0.5% per node. That is, a 101-node tree has its fitness increased by 50% over single node tree that performs as well.

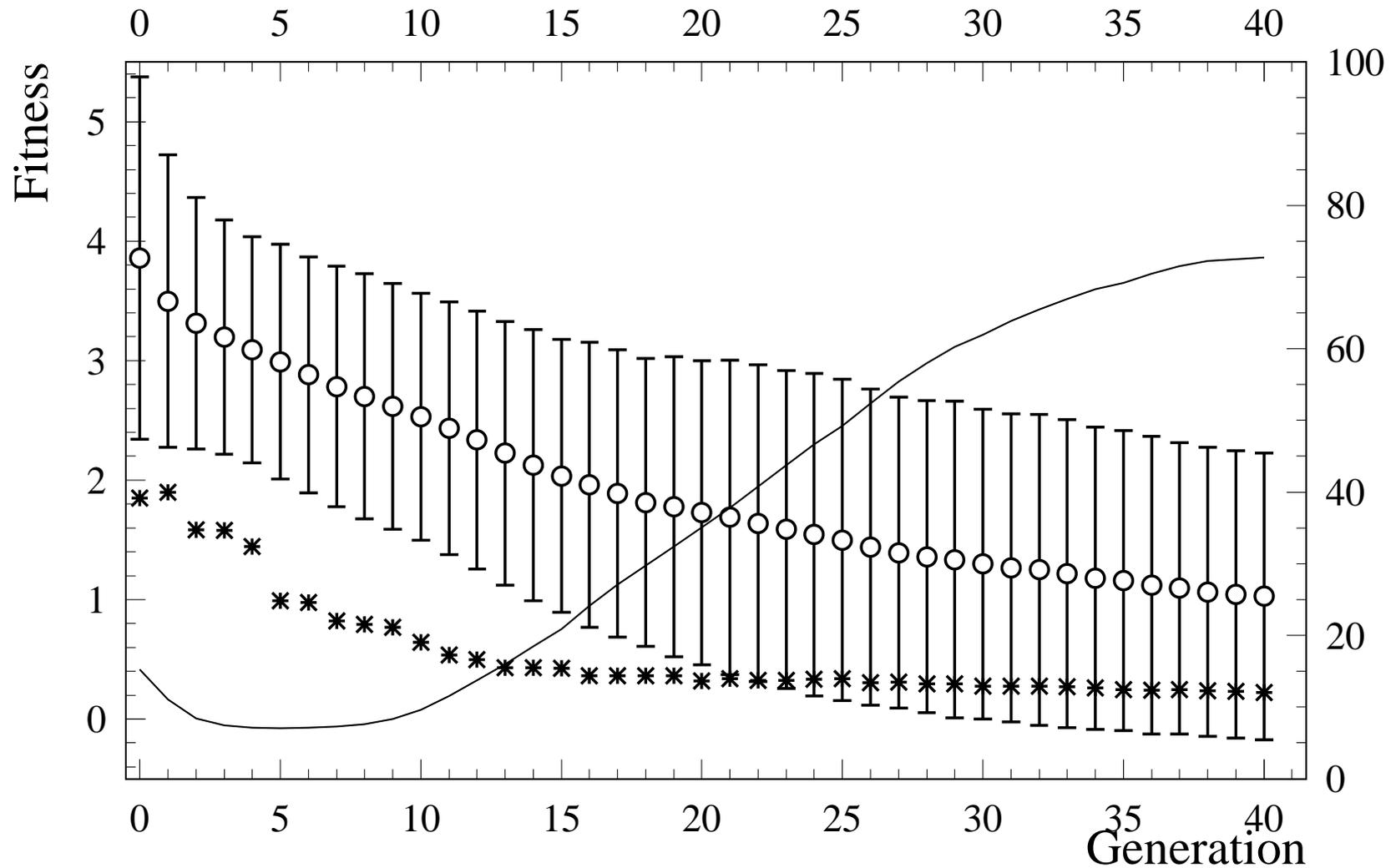
Best Trees for 4th Generation

Final trees are difficult to understand, early ones are smaller.



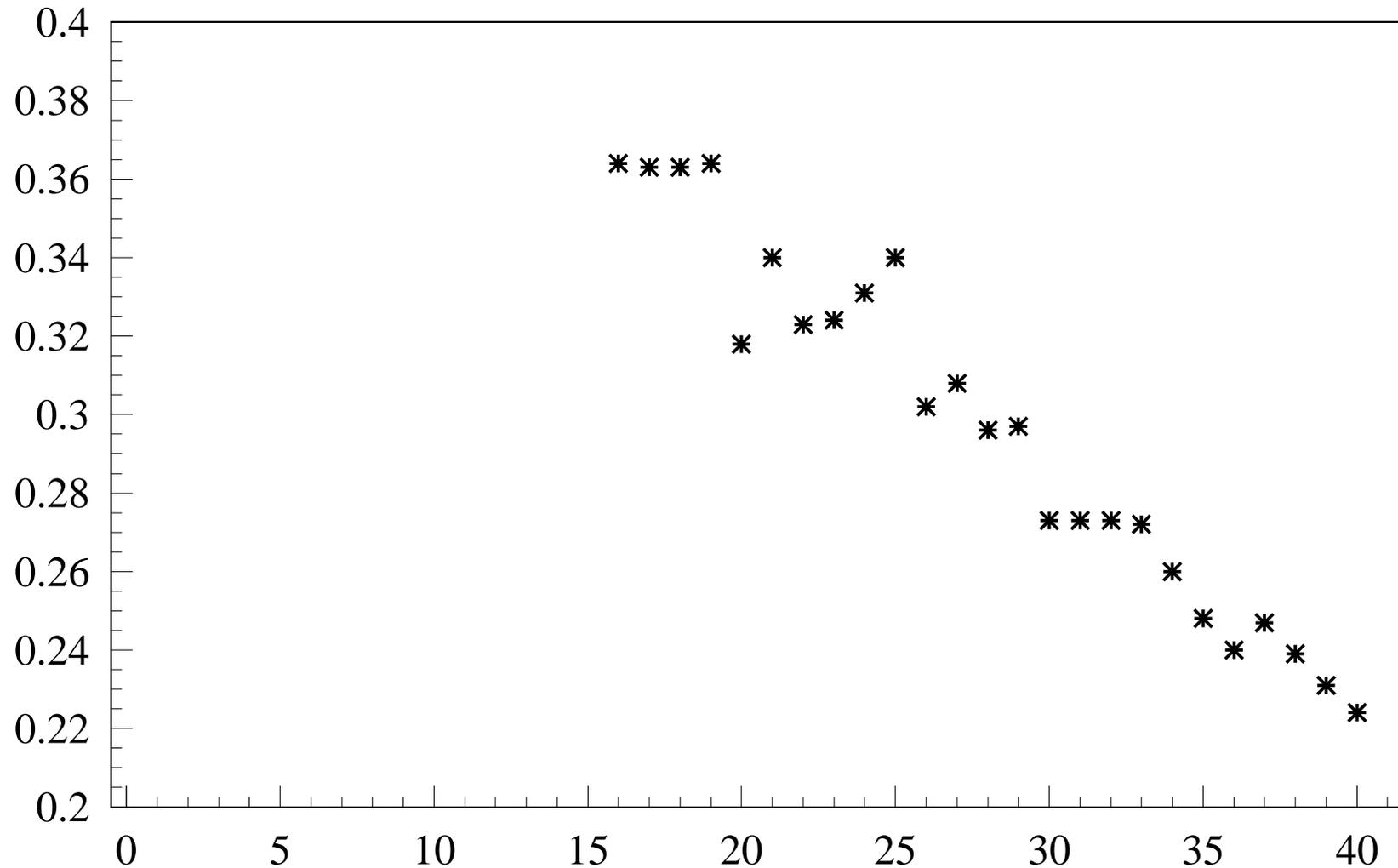
These are the best three trees; best tree at left
(0 XOR) and (1 ×) are worthless
($f(n)$ is a neural net sigmoid threshold function)

Λ_c^+ Evolutionary Trajectory



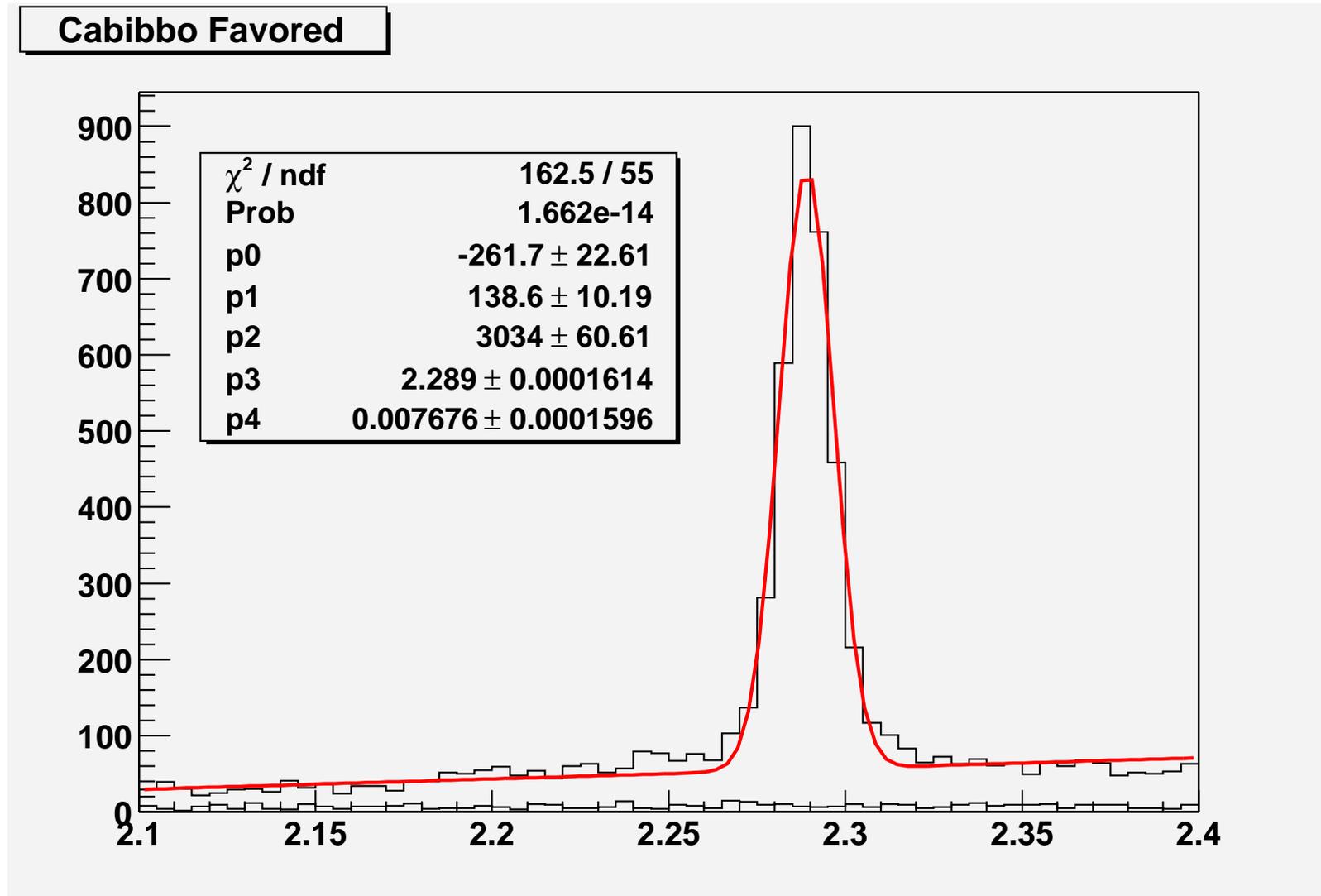
Circles: average, Stars: best, Line: avg. size

Expansion of best trees



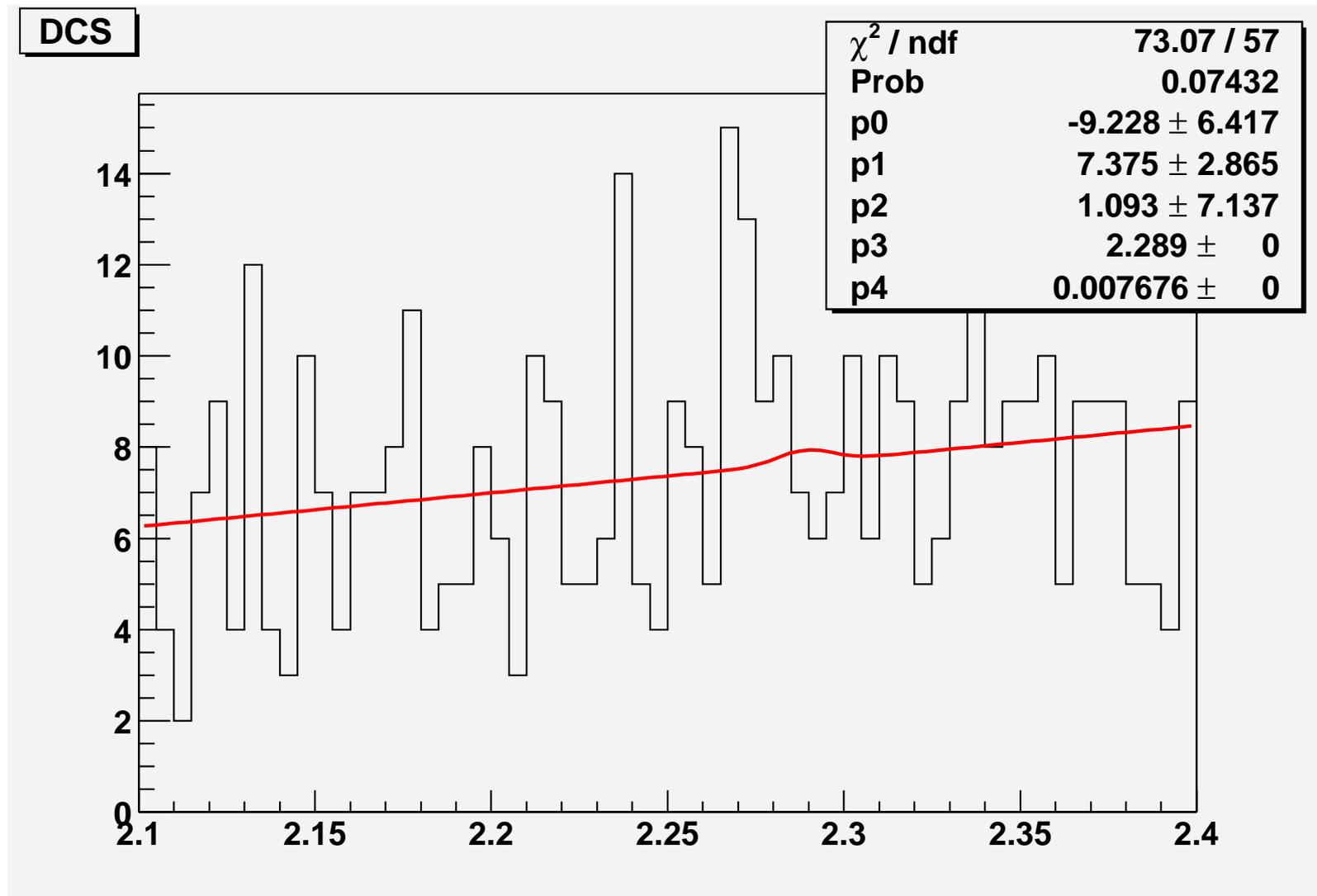
Stars are the best tree, still evolving at generation 40

$\Lambda_c^+ \rightarrow p K^- \pi^+$ (CF) signal



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

$\Lambda_c^+ \rightarrow p K^+ \pi^-$ (DCS) signal

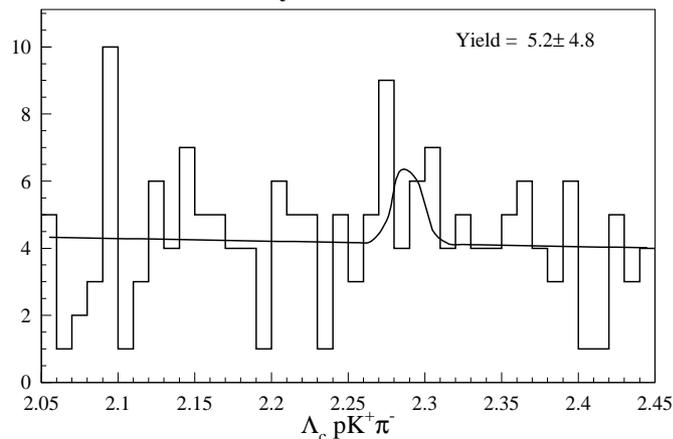
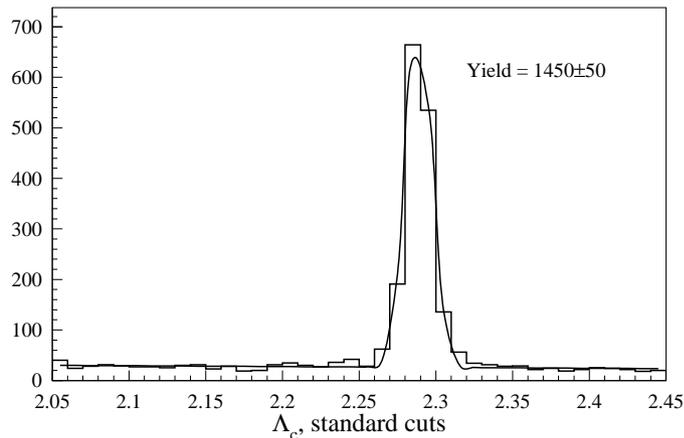


Mass and width are fixed to CF values

Comparison with Cut Method

How does this compare with our normal method?

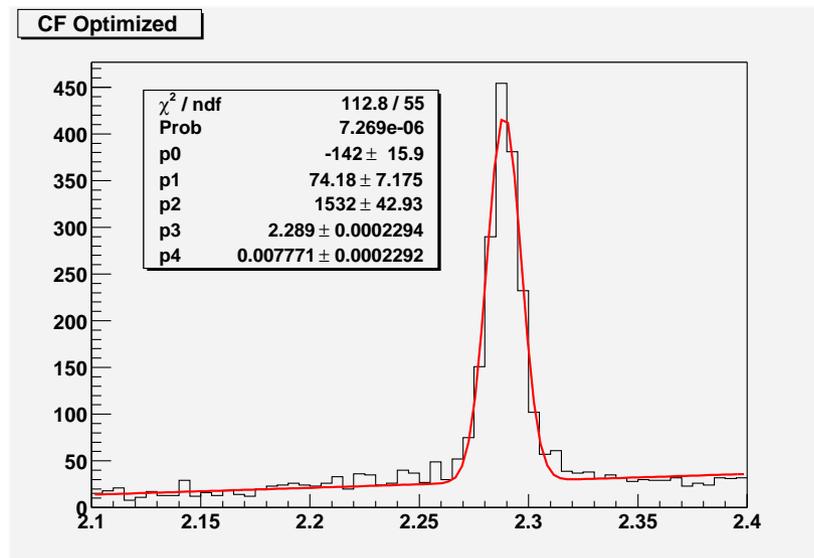
- Looked for DCSD before by maximizing S/\sqrt{B} with cut trees and scans



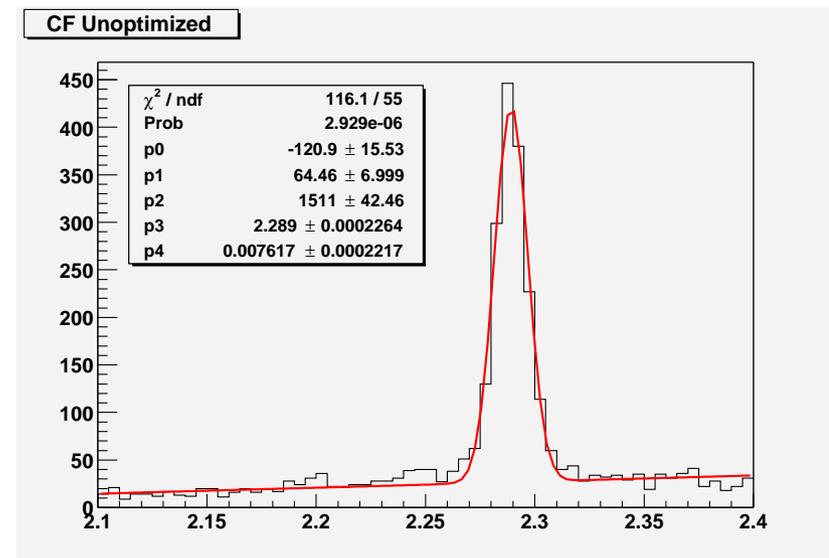
- Important quantity: $\sigma Y_{\text{DCS}}/Y_{\text{CF}}$
- Cuts: Yield = 1450, $\sigma Y = 4.8$
- GP: Yield = 3030, $\sigma Y = 7.1$
- $\sigma Y/Y$ for GP = 2.3×10^{-3}
- $\sigma Y/Y$ for cuts = 3.3×10^{-3}
- GP method is $\sim 50\%$ better, but still need luck or significant improvements to observe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

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).



1532 ± 43 events

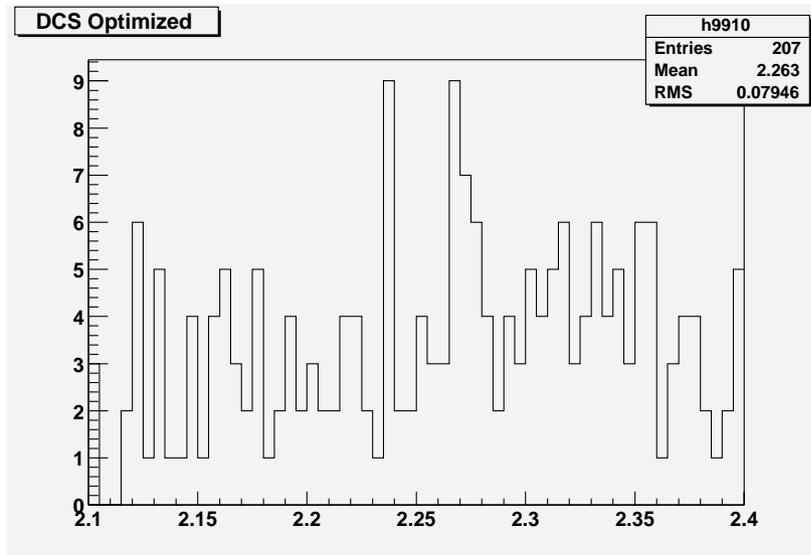


1511 ± 42 events

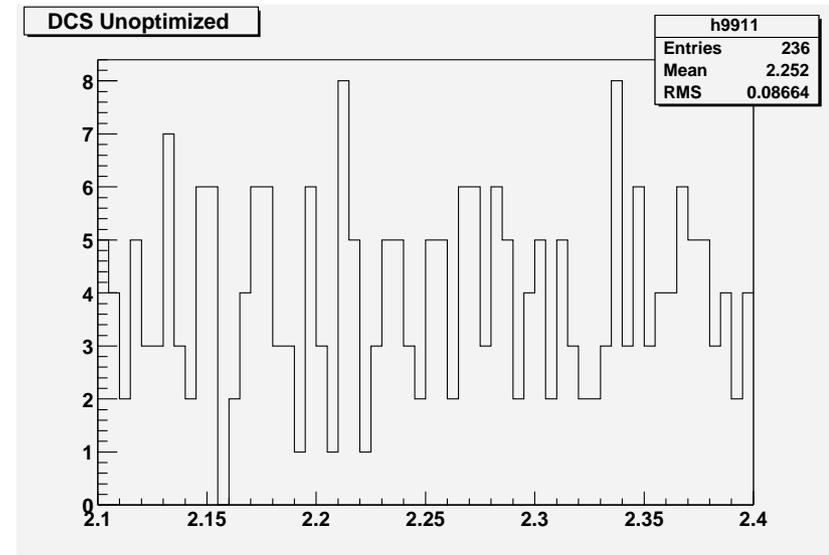
There are slightly more $\Lambda_c^+ \rightarrow pK^- \pi^+$ candidates in the events used to optimize, but completely consistent with statistics.

Bias checks, continued

What about wrong sign, or DCS, distributions? (BG is linear fit across entire range.)



207 events

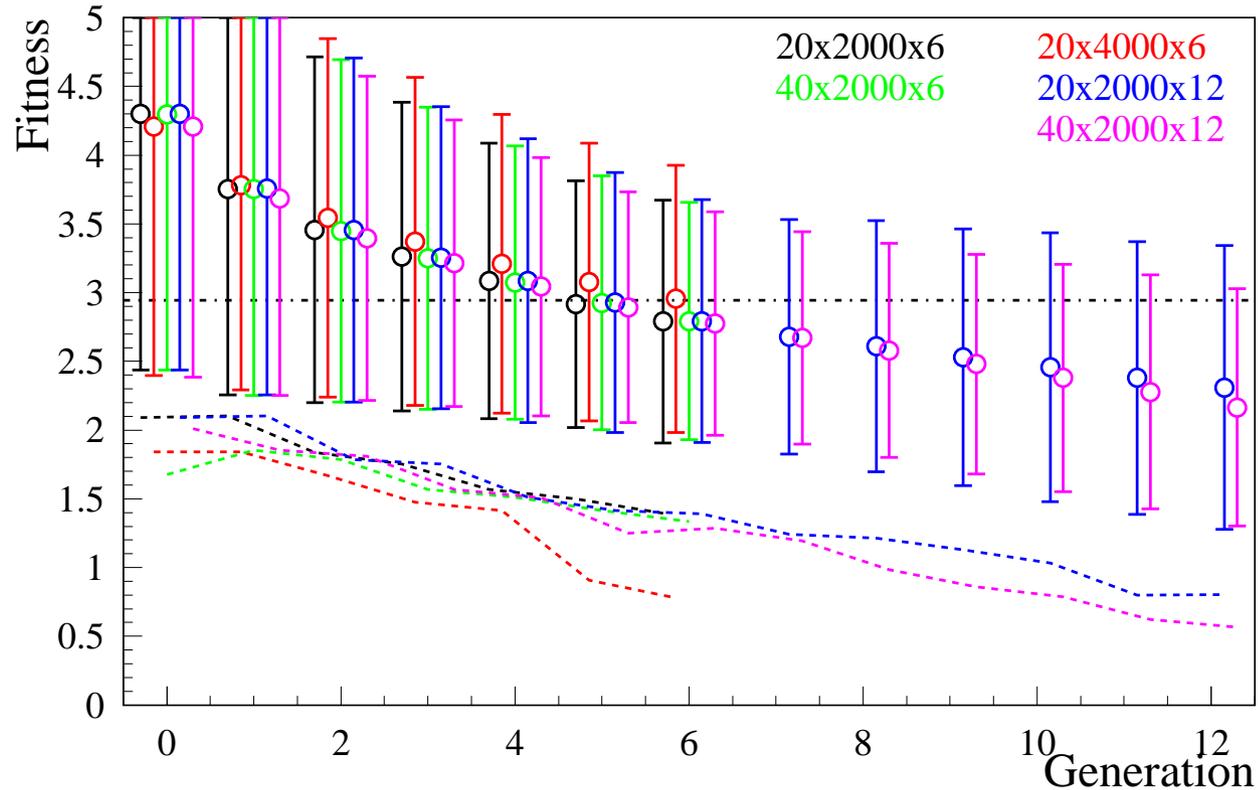


236 events

$236 - 207 = 29 \pm 21$ events difference between modes. Would need to cross-check this by optimizing on the other half of events, changing random numbers, etc. before concluding there is a problem.

Tuning GP parameters

Start: 20 CPUs, 2000 trees/CPU, 6 gen. Doubled each parameter



- Points: avg & RMS, dots: best
- More generations seems to be best
- Clearly evolution is occurring (not random success)

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

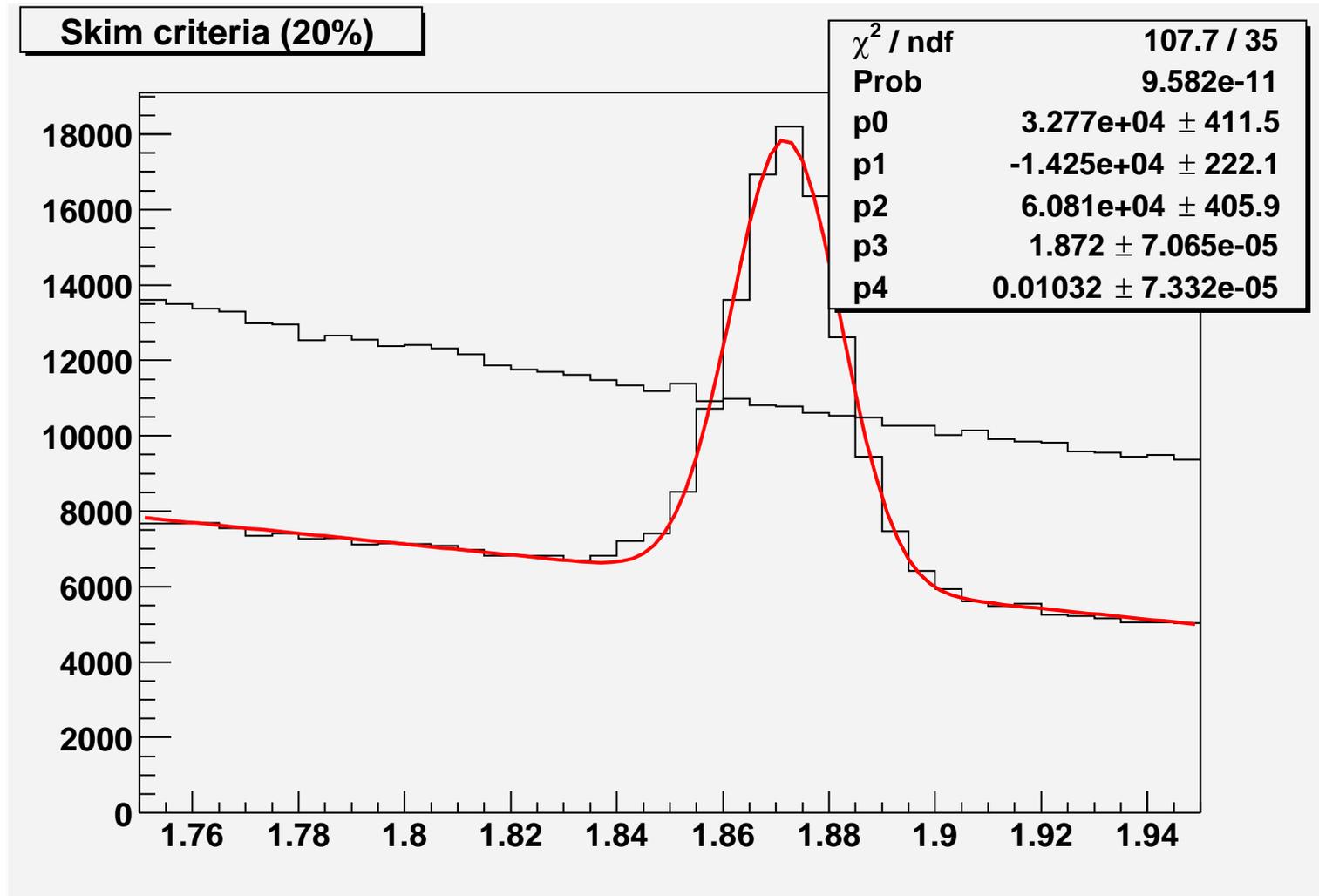
While $\Lambda_c^+ \rightarrow p K^+ \pi^-$ is the analysis we're pursuing with this method, $D^+ \rightarrow K^+ \pi^+ \pi^-$ is a useful check.

There is too much data to optimize on all of it, so we choose to optimize just on 20% of the data.

This branching ratio has been measured and is surprisingly large (about $3 \tan^4 \theta_c$). The PDG value is $0.75 \pm 0.16\%$ relative to $D^+ \rightarrow K^- \pi^+ \pi^+$.

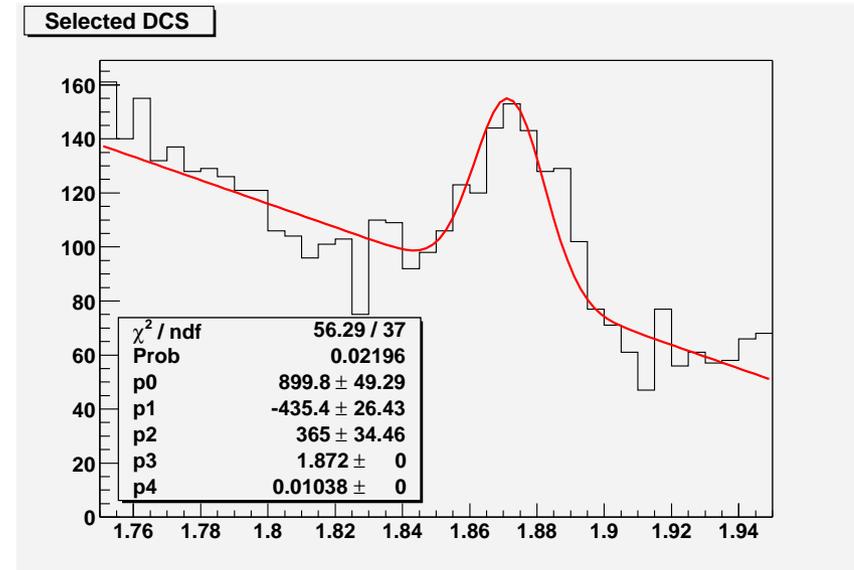
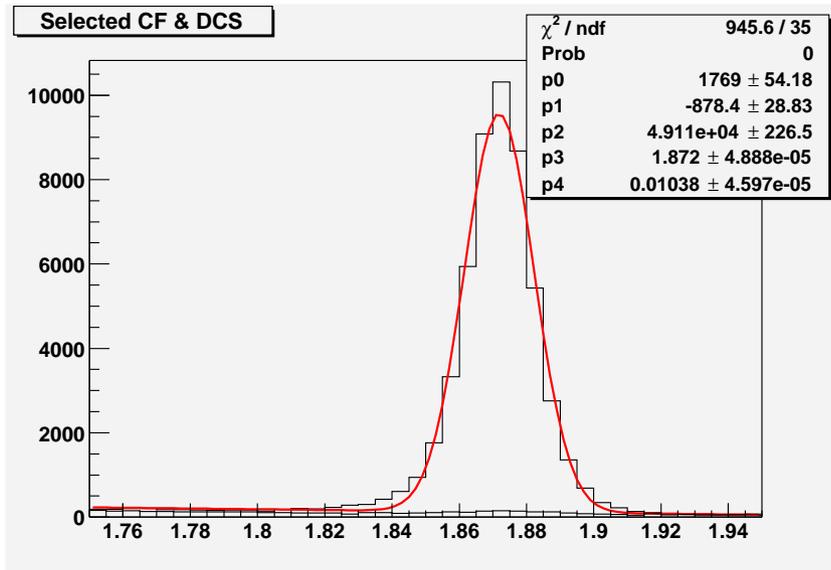
A FOCUS branching ratio measurement and Dalitz analysis is underway with about 200 events in this mode.

Starting point



Just the 20% that the GP sees. BG is higher than signal
60,810 \pm 410 events in $D^+ \rightarrow K^- \pi^+ \pi^+$ signal

After 10 generations



The full sample: CF on left, DCS on right
 365 ± 35 events in $D^+ \rightarrow K^+ \pi^+ \pi^-$ signal

Data 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. What need not be known is the *absolute* efficiency on a single mode.

For the program which generated this, the MC efficiencies are:

	CF Eff. (%)	DCS Eff. (%)
Skim cuts	6.99 ± 0.01	6.71 ± 0.01
GP Selection	1.12 ± 0.01	1.11 ± 0.01
GP/Skim	16.03 ± 0.08	16.58 ± 0.09

Data 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. What need not be known is the *absolute* efficiency on a single mode.

For the program which generated this, the MC efficiencies are:

	CF Eff. (%)	DCS Eff. (%)
Skim cuts	6.99 ± 0.01	6.71 ± 0.01
GP Selection	1.12 ± 0.01	1.11 ± 0.01
GP/Skim	16.03 ± 0.08	16.58 ± 0.09

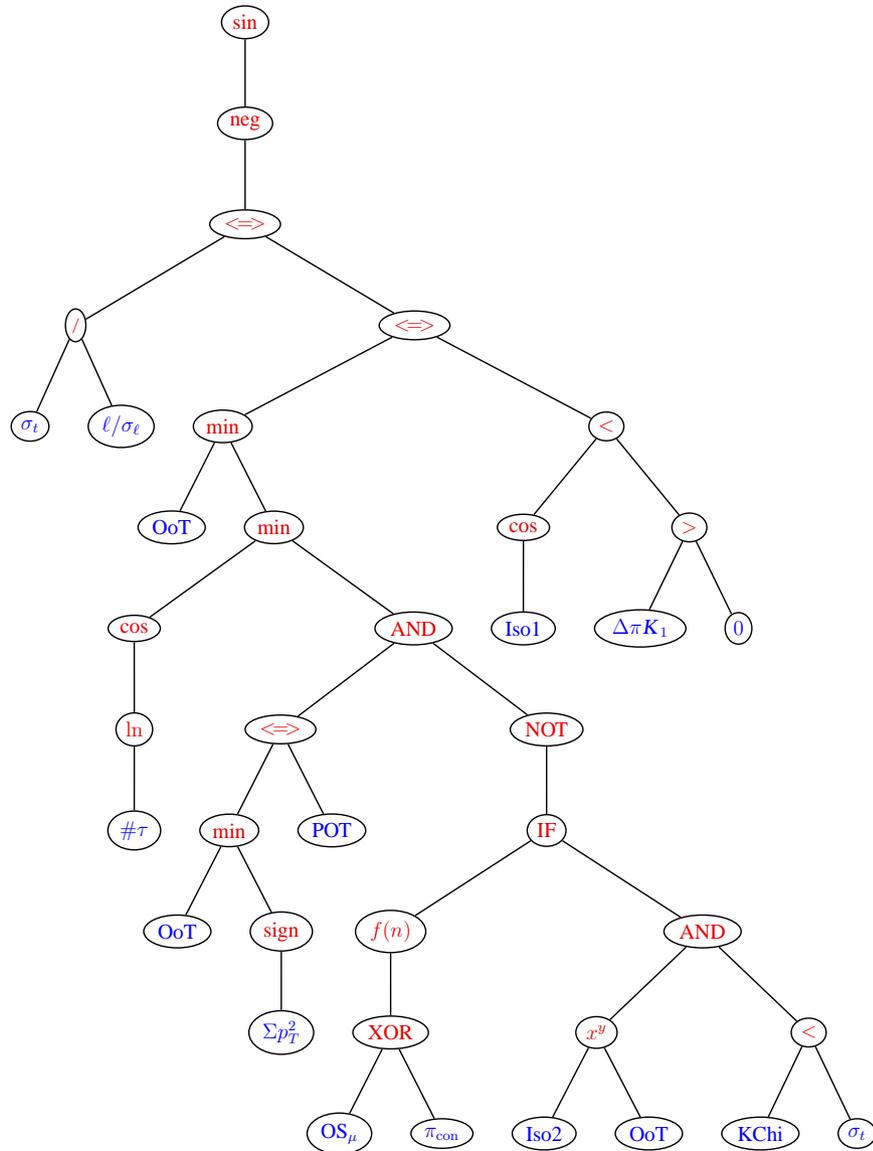
But, we can look at GP/Skim for the CF data.

We find $16.15 \pm 0.08\%$.

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 right.

Good modeling is important if you need to understand the GP in detail.

10th generation result



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

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

Homework

Ok, now for the not-so-fun part. I'm supposed to give you a homework assignment. We'll make it easy:

- On the “Resources” slide are several web pages with software
- Look around and pick a software package in your favorite language
 - C++, C, Java
 - Perl, Ruby, Python
 - FORTRAN?!?!, LISP, others
- Most or all of these will have example problems and code
- Pick one you like, change the evolutionary parameters and try changing the problem too

The idea is just to get your feet wet a little.

I also want to hear your impressions, because I've only used one of these frameworks (`lilgp`).

GA & GP Resources

There is a lot of information on the web about Genetic Algorithms and Programming:

- <http://www.aic.nrl.navy.mil/galist/> — Genetic Algorithms
- <http://www.genetic-programming.org/> — John Koza

Software frameworks for both GA and GP exist in almost every language (most have several)

- http://www.genetic-programming.com/coursemainpage.html#_Sources_of_Software
- <http://zhar.net/gnu-linux/howto/> – GNU AI HowTo (GA/GP/Neural nets, etc.)
- <http://www.grammatical-evolution.org> — GA–GP Translator

Backup slides

Backup slides

$f(n)$ function

A threshold function used in neural networks:

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

