Pattern Matching in the STT

Michael Papenbrock

Department for Physics and Astronomy

September 5th, 2017
Novosibirsk, Russia

DyTER - Dynamic Track and Event Reconstruction

What is the idea?

- Focus on hyperons (displaced vertices)
- Break away from traditional event-based reconstruction
- Generate tracks and events dynamically from continuous data stream
- Use track and vertex information in event building

$\rightarrow$ Track reconstruction and event building as an interdependent process

- Write highly modularised code

DyTER - Dynamic Track and Event Reconstruction

What is our approach?

- Use SttCellTrackFinder as basis and develop it further
- Implement longitudinal momentum reconstruction (W. Andersson)
- Investigate detector signatures of hyperons in detail to guide development (J. Regina)
- Investigate possibilities using highly parallelised framework (B. Andersson, J. Nordström)
- Implement and test algorithms for complete time-based simulation/reconstruction chain (D. Steinschaden)

Question: Could pattern matching be of some use?

Pattern Matcher: Questions and Ideas

Questions

- Is it feasible with the STT and hyperons?

  $\rightarrow$ How many patterns will there be?
- What are the benefits?

Ideas

- Lightweight testing ground for time-based data processing
- Pre-clustering (procedure suitable for FPGAs)
- Augment SttCellTrackFinder with pattern matching algorithms or vice versa
- Stand-alone track finder using machine learning

Pattern Matcher: Concept

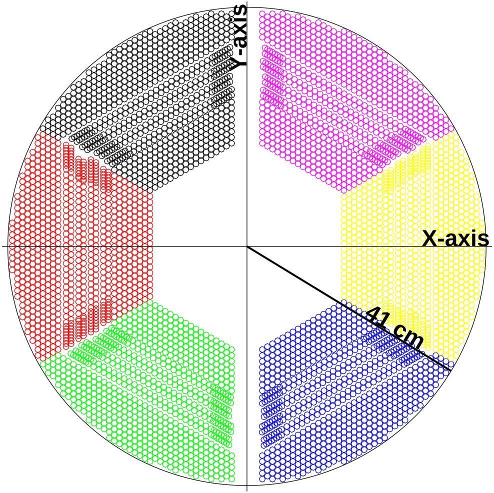
! Simple pattern counter unfeasible for large event numbers $\rightarrow$ more sophisticated concept was needed

- Divide STT into 6 sectors
- Simulate desired channel (here: $\Lambda\bar{\Lambda}$ at 7 GeV, pandaroot rev. 30040)
- Store pattern as std::set of tube IDs
- Determine and store complementary information
- Merge duplicate/similar patterns
- Start matching

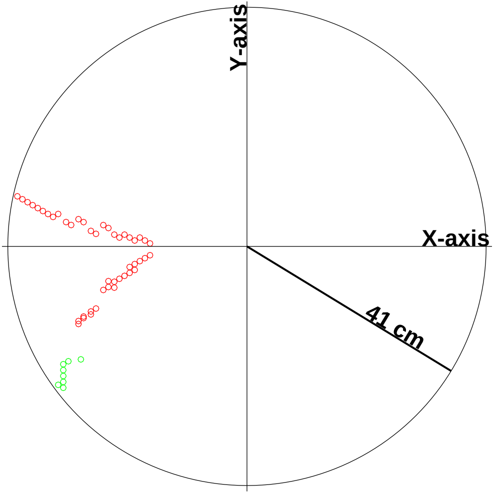
Closer look: Pattern

- tubeIDs
- momenta
- timeStamps
- sectorID
- count

Pattern Matcher: Concept



Pattern Matcher: Concept



Pattern Matcher: Database Generator

- Generate events for desired channel (use ideal track finder)
- Identify patterns as tubeIDs for hits corresponding to a track
- Extract complementary information (e.g. momentum, sectorID, etc.)
- Store data as ROOT TTree

Attention

- TTree will be filled with duplicate patterns!

→ Identify and merge identical patterns

→ Bonus: Identify and count "similar" patterns (e.g. 90 % match)

Pattern Matcher: Merging

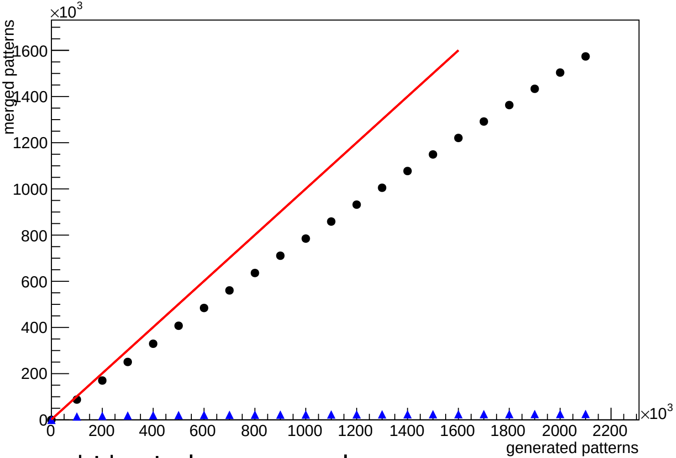
Before merging

Count	tubeIDs	sectorID	etc.	
1	1,2,3,4,5	1	...	
1	2,3,4,5,6	1	...	
1	10,11,12,13,14	3	...	
1	1,2,3,4,5	1	...	

After merging

Count	tubeIDs	sectorID	etc.	
2	1,2,3,4,5	1	...	
1	2,3,4,5,6	1	...	
1	10,11,12,13,14	3	...	

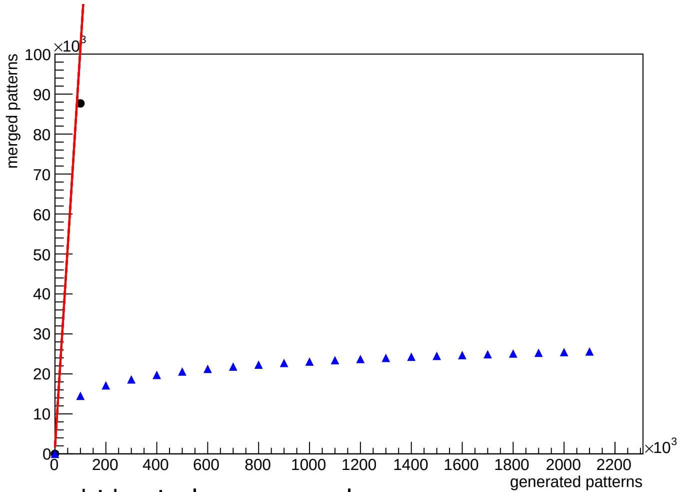
Pattern Matcher: Merging



black: merged identical patterns only

blue: merged 90% similar patterns

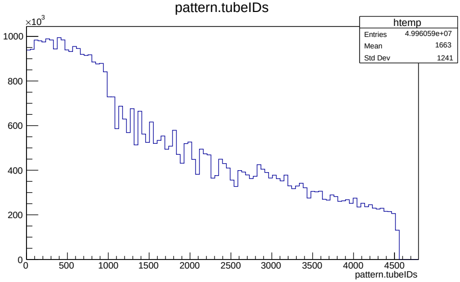
Pattern Matcher: Merging

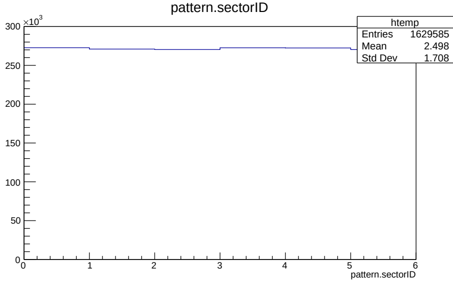


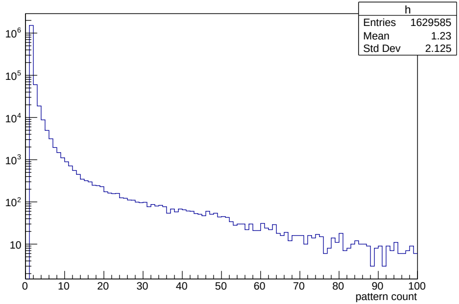
black: merged identical patterns only

blue: merged 90% similar patterns

Pattern Matcher: Merging







Pattern Matcher: Matching Algorithm

- Same principle as merging
- Compare patterns against database using std::set_intersection
- Find full or partial matches in incoming data
- "Ideal" matching ratio currently under investigation

Considerations

- ROOT TTree philosophy: write once, never touch $\rightarrow$ less than ideal for merging/sorting of database

? Possible other solutions: PostgreSQL, FairDB(?)

- Very simple algorithm $\rightarrow$ Lightweight (code), good for testing purposes
- Value not only as stand-alone track finder

$\rightarrow$ Possible hybrid solution with SttCellTrackFinder

Where do we go from here?

- Implement complementary data (momentum, time stamps, etc.) in database

$\rightarrow$ Investigate how well these data can be "guessed" from pattern

- Test sector-less database
- Implement and test time-based processing (e.g. using discreet time windows)
- Use findings to complement SttCellTrackFinder
- Explore machine learning possibilities (possible future project)

Appendix: Virtualisation with Vagrant and Ansible

Vagrant

- "Tool for building and managing virtual machine enviroments in single workflow"
- Use pre-existing "boxes" to quickly set up VM
- Single configuration file

Ansible

- Workflow automation, e.g. installing and configuring packages or system components
- Playbook defines what should be added/configured on a system
- Roles set up packages, install software, etc.

Appendix: Virtualisation with Vagrant and Ansible

Vagrantfile (Ruby)

```ruby
# disk = './workspace.vdi'

Vagrant.configure(2) do |config|

  config.vm.box = "bento/ubuntu-16.04"
  config.vm.box_check_update = false
  config.vm.network "private_network", ip: "192.168.33.102"
  config.vm.provider "virtualbox" do |vb|
    vb.gui = true
    vb.cpus = 4
    vb.memory = "4096"
    vb.name = "panda"
    vb.customize ["modifyvm", :id, "--vram", "128"]
  end

  config.vm.synced_folder "../../ansible", "/provisioning"

  config.vm.provision "ansible_local" do |ansible|
    ansible.provisioning_path = "/provisioning"
    ansible.inventory_path = "/provisioning/hosts"
    ansible.playbook = "panda.yml"
    ansible.verbose = "v"
    ansible.limit = "panda"
  end

  config.vm.provision :reload

end
```

Appendix: Virtualisation with Vagrant and Ansible

ansible playbook (YAML)

```yaml
---
- hosts: panda
  remote_user: "vagrant"
  become: true

  roles:
    - { role: update }
    - { role: desktop }
    - { role: pandaroot }

...
```

panda role (YAML)

```yaml
- name: create panda directory
  file:
    path: /panda
    state: directory
    mode: 0777

- name: get fairsoft
  become_user: vagrant
  git:
    repo: https://github.com/FairRootGroup/FairSoft
    dest: /panda/source/fairsoft
    version: may16p1

- name: create fairsoft installation directory
  become_user: vagrant
  file:
    path: /panda/build/fairsoft
    state: directory
    mode: 0755

- name: compile fairsoft
  become_user: vagrant
  shell: /provisioning/roles/pandaroot/files/installFairSoft.sh > /provisioning/logs/fairsoftinstall.log

- name: get fairroot
  become_user: vagrant
  git:
    repo: https://github.com/FairRootGroup/FairRoot.git
    dest: /panda/source/fairroot
    version: v-16.06b
```

Appendix: Virtualisation with Vagrant and Ansible

- Setup available for PANDA virtual machine

What's in it?

- Xubuntu as base system
- Complete pandaroot toolchain

What's not?

- Development tools
- TORQUE
- Docker (either to install components or replace full VM)

Thank you for your attention!