# Optimization of Associative Memory

Gradon H. Faulkner

Since the construction of the Large Hadron Collider (LHC), the ATLAS detector is one of the general detectors that is responsible of collecting and interpretation of data that is generated from the collider, which generates around 10 Petabytes of data per year. The ATLAS Trigger and Data Acquisition department (TDAQ) is responsible of developing, implementing, and maintaining techniques that are necessary of manipulating the massive amount of data from the ATLAS detector. Currently, there are three “triggers”, which are implementations of hardware and software, that are responsible of trimming down the amount of data that is stored that has been generated by the collider. A new way of trimming downing the amount of data is the development of an additional trigger, called the FastTracker Trigger, that uses pre-stored particle trajectory information, called patterns. Patterns help match particle trajectory data from a collision to known particle characteristics at the hardware level. Generating patterns is accomplished by a simulation that uses data previously collected from the ATLAS detector to produce patterns that have the highest probability of recognizing particular particles that can be used to explore new physics. Due to the storage of patterns in hardware and the cost of the hardware necessary to implement this trigger, there is a hardware requirement that must be meet with certain efficiency, making the pattern bank generation a vital component of the FastTrack Trigger.

To find a most efficient pattern to collect particular data, researching into methods that optimize the pattern from the simulation would be beneficial in maintaining high efficiency meeting the hardware constraints on the FastTracker Trigger. From previous research, there have three different types of patterns are stored in the trigger that have been developed. These patterns are called the thin patterns, large patterns, and variable resolution patterns. The differences between these patterns are the amount of volume that they represent within the detector and how much noise they misinterpret as accurate data from the detector. More thin patterns are required to make a match while less large patterns are needed but then additional noise is collected. There is a directly proportional relationship between pattern volume and noise collection. Therefore, in order to achieve the best efficiency of the patterns, further research is needed to develop an algorithm to go through each of the patterns generated from the simulation and try to find a more optimized way by of combining each pattern with other patterns to reduce the amount of additional noise that is collected.

## Methodology

Developing an algorithm that would try to find a more optimized way of combing each pattern with other patterns was influenced from the variable resolution patterns, which are made by combining two or more thin patterns together. Combining the thin patterns together to form a variable resolution pattern is achieved by using a hardware characteristic called “don’t care” bits, which determine how much volume that the patterns will occupy. Since the more volume a pattern has, the increase in additional noise increases, reducing the volume of the patterns as much as possible while maintaining the same efficiency lead to the design of a “re-splitting” algorithm. The principle of a re-splitting algorithm is to exploit the “don’t care” bits by setting a threshold of how many each variable resolution pattern can have. Setting a threshold on the variable resolution patterns reduces the volume that they would have if they have more "don't care" bits then the threshold. In order to implement this algorithm, each variable resolution pattern was broken down to look for a more optimized configuration of the thin patterns that are make up the variable resolution pattern. Once a variable resolution pattern separated into all of the thin patterns that define it, the algorithm would try to combine these thin patterns into new variable resolution patterns that would have no more "don't care" bits then the set threshold. The algorithm could separate the original variable resolution pattern into all of it's thin patterns if the combination of any two result with more "don't care" bits than the threshold; or leave the variable resolution pattern as it was before being broken down since it had less "don't care" than the threshold. Once the variable resolution pattern was passed through this algorithm, it would be saved and run the next variable resolution pattern in the pattern bank through this algorithm. When this algorithm was implemented, it had also set a basic structure of optimizing the patterns with a more advance algorithm. Due to the characteristics of the variable resolution patterns being made up of thin patterns, the group of such algorithms called clustering algorithms was chosen to be used for the more advanced algorithm.

Clustering algorithms group objects together such that each group contain objects that are similar to one another. Distinguishing the objects to be thin patterns and the groups of these objects to be variable resolution patterns, the outline of a clustering algorithm was set. The type of clustering algorithm that was implemented was an agglomerative hierarchical algorithm that was based on complete-linkage. Complete-linkage will group together objects that are the furthest apart first, making it ideal for the variable resolution patterns since it would keep combining the patterns together until it got back to the original variable resolution pattern and allow a choice of how to break the pattern apart in the best configuration. Implementing this was done by rearranging the thin patterns in a variable resolution pattern into clusters. These clusters would first be formed by having each thin pattern be a cluster itself. Then the clusters would be compared to each other to determine which two clusters were the most different from one another, which was done by seeing how many “don’t care” bits would be needed to having two clusters merged together. The two clusters that would have the most “don’t care” bits after merging would be the most different clusters. Following this, the two clusters would be merged together into one new cluster, and have the change in volume recorded, along with the total number of current clusters. Repeating this process until only one cluster remained was necessary to determine the next step of the algorithm. This next step was to use the data that was recorded after each merge of clusters and determine which one had the greatest reduction in volume with the smallest increase in the number of patterns in comparison to the original variable resolution pattern. Finally, the configuration that is determine would be made by breaking apart the variable resolution into the desired configuration.

## Results

The implementation of the re-splitting algorithm and the clustering algorithm lead to a higher efficiency of the pattern bank than the pattern bank that was generated from the simulation at a small increase in the number of patterns contained within the pattern bank. For the re-splitting algorithm, the setting of the threshold would modify the pattern bank vastly depending on how the threshold was set. If a low threshold was set, the number of patterns would vastly increase, while setting a high threshold would keep the pattern bank the same. Changing the threshold and running tests to see the various efficiencies of different threshold lead to the conclusion that the pattern bank can be optimized further and that development of a more advanced algorithm would be beneficial. By pursuing a more advance algorithm, such as the clustering algorithm, the results became more clear that it was worth the research into algorithms to optimize the pattern bank. Obtaining this result was from the clustering algorithm running with a small data set of the simulation to test the algorithm that implementation ran correctly with patterns.Despite the success in implementation of the clustering algorithm, the algorithm would crash when it was ran with the simulation due to not enough computation space to handle the full simulation. Further research would benefit from optimizing the implementation of the algorithm and run with a larger computer farm to handle the data. As a result from the small test runs though, the efficiency that was achieved from the clustering algorithm was higher in comparison to the re-splitting algorithm and achieved a much more efficient pattern bank to be used for the trigger system. From these results, further development and optimization of the algorithm will be continue to be undertaken so that the algorithm will run with the simulation, allowing for optimization of the pattern bank generation for the FastTrack Trigger.

## References

Shochet, M., Tompkins, L., Cavaliere, V., Giannetti, P., Annovi, A., & Volpi, G. (2013). Fast tracker (ftk) technical design report. Available from CERN Document Server. Retrieved from http://cds.cern.ch/record/1552953?ln=en