TemplateTagger v1.0.0
A Template Matching Tool for Jet Substructure

Mihailo Backović, José Juknevich

Department of Particle Physics and Astrophysics
Weizmann Institute of Science, Rehovot 76100, Israel

Abstract

TemplateTagger is a C++ package for jet substructure analysis with Template Overlap Method. The code operates with arbitrary models within fixed-order perturbation theory and arbitrary kinematics. Specialized template generation classes allow the user to implement any model for a decay of a boosted heavy object. In addition to template overlap, the code provides ability to calculate other template shape and energy flow observables. We describe in detail the structure of the package, as well as provide examples of its usage.

Contents

1 Introduction

Algorithms for tagging of boosted objects necessarily exploit observables sensitive to a parton shower history, including color flow and hadronization. Most jet sub-structure methods can be characterized as jet de-clustering and re-clustering algorithms, with a common feature that they perform the analysis on the entire jet, after showering and hadronization. (see Refs. [1–19] for a review).

Tracking the parton shower history from physical final states to the hard-parton subprocesses often becomes rather involved as the number of QCD emissions is typically very large. Details of hadronization only further complicate the analysis. The Template Overlap Method [20–23], aims to bridge the gap between energy flow of observed jets and partonic configurations calculated at fixed order perturbation theory. The method allows for subjet identification in an infrared-safe way, by providing a mapping between energy-unweighted variables and the template that defines the energy flow distribution.

The primary intended use of the TemplateTagger package is the analysis of jet substructure in High Energy Physics collider data. TemplateTagger allows the user to design a custom boosted jet analysis for a variety of scenarios by using the same basic three-stage approach. First, the user generates sets (or "catalogs") of templates by scanning over a phase space of the parton decay daughters of a massive particle of mass M and transverse momentum p_T. Second, the TemplateTagger code performs template matching on an event-by-event basis whereby candidate signal jets are located in the $\eta - \phi$ space. Finally, the events are tagged using best matched templates as approximate subjet locations. The overlap approach has several important advantages over other jet substructure algorithms commonly used at hadron collider experiments:

- TemplateTagger is model independent. The user is required to define a template model, while the code will efficiently search for the matching subjet-like structures in the jet energy

flow patterns.

- The pattern recognition based approach permits an efficient way of determining the jet topology which takes into account the event kinematics (jet mass, subjet asymmetry, etc.).
- TemplateTagger tools can be used to preserve as much energy flow information as possible, which is particularly useful for events where the energy distribution is all that is available. This information is presented in a well-organized form convenient for a detailed analysis of jet substructure.
- Reconstructed subjets have well defined shapes insensitive or weakly sensitive to pileup and underlying event.

TemplateTagger is a C++ library which provides basic implementation of the Template Overlap Method for jet substructure. We designed the code around the FastJet [24] package of jet algorithms with the aim at easy implementation into existing jet analysis tools. TemplateTagger is also a testbed containing programs and routines for generating template data sets, collecting and analyzing statistics on the performance of template overlap and jet shapes, and visualizing the structure of these events. The package can be downloaded from `http://tom.hepforge.org/`.

In addition to overlap analysis, TemplateTagger provides the necessary tools to analyze a boosted jet using observables constructed out of best matched templates. We provide implementations of various template-based jet shapes and energy flow observables such as Template Planar Flow, Template Angularity, etc.

This manual describes how to download and install TemplateTagger, how to use the main libraries, as well as how to change its configuration for different overlap measures and template catalogs. Finally, the manual shows how to use the sample programs for testing purposes and basic data analysis. In section 2, we briefly introduce the physics behind the commands of TemplateTagger. We give a short description of the TemplateTagger code structure in section 3. Section 4 lists several possibilities for further extensions of the program. Appendices A and B discuss boost-invariant implementations of the template generation and various jet shape observables. Appendix C contains the detailed syntax and functionality of all relevant internal methods.

2 Physics Overview

The current version of TemplateTagger allows the user to study the substructure of massive high-p_T jets for various models. The user defines a model by specifying a catalog of partonic decay configurations, labelled as f, which are taken to represent the decays of a heavy particle of mass M at a given p_T. In addition, one has to specify a functional measure to quantify agreement between the energy flow of a jet and the flow of each template. For each jet candidate, the overlap function is defined as

$$Ov_N = \max_{\{f\}} \exp\left[-\sum_{a=1}^{N} \frac{1}{\sigma_a^2}\left(\epsilon\, p_{T,a} - \sum_{i\in j} p_{T,i}\, F(\hat{n}_i, \hat{n}_a^{(f)})\right)^2\right] \tag{1}$$

where $\{f\}$ is the set of templates defined for the given jet p_T, $p_{T,a}$ are the transverse momenta of the heavy particle (or resonance) decay daughters for the given template, $p_{T,i}$ is the p_T of the ith jet constituent (or calorimeter tower, topocluster, etc.). The parameter ϵ serves to correct for the energy emitted outside the template subcone. The first sum is over the N partons in the template and the second sum is over jet constituents. The kernel functions $F(\hat{n}, \hat{n}_a^{(f)})$ restrict the angular sums to (nonintersecting) regions surrounding each of the template momenta. We provide two concrete implementations of kernels: a Gaussian around each of the directions of the template momenta with normalization $F(0, \hat{n}_a^{(f)}) = 1$,

$$F(\hat{n}_i, \hat{n}_a^{(f)}) = \exp\left[-(\Delta R)^2/(2\omega_a^2)\right], \tag{2}$$

and a normalized step function that is nonzero only in definite angular regions around the directions of the template momenta p_i,

$$F(\hat{n}_i, \hat{n}_a^{(f)}) = \begin{cases} 1 & \text{if } \Delta R < R_a \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where ΔR is the plain distance in the (η,ϕ) plane. The parameters ω_a (R_a) determine the radial scale of the template subjet. Together with the energy resolution scale σ_a, these are the only tunable parameters of the model. A few possible strategies to determine the optimal values of σ_a and R_a are as follows:

- Choose the best parameters according to some optimization criterion (*e.g.,* optimize the tagging efficiency and background rejection) Use the same parameters in every event.
- For each event, make a choice of parameters for which the overlap is maximized. Estimate the stability of the configuration.
- Choose the parameters separately for each template, *e.g.* using a p_T-dependent scale for template matching.

Template overlap provides a mapping of final states j to partonic configurations $f[j]$ at any given order. The best matched template $f[j]$ can be used to characterize the energy flow of the jet, giving additional information on the likelihood that it is signal or background. Furthermore, we can derive additional jet shape information out of $f[j]$ to further increase the rejection power of the method.

It is important to realize that other choices for the functional measure and kernel functions can easily be implemented, and we encourage the reader to explore them. The choice of template parameters is largely dependent on the application of template overlap and the user's preferences. The same is true for the template generation. Typically, the choice of template libraries is dependent upon the emphasis sought. The commands detailed in appendix C give you access to this information.

3 Program Structure and Use

We proceed to discuss the installation of TemplateTagger and execution of the example code. We also discuss the general structure of the code, for the benefit of the user who might wish to read or modify the source code.

3.1 Installation

TemplateTagger runs on any architecture with a modern C++ compiler such as `g++` and an installation of FastJet. For the convenience of Unix and Mac OS X users, we provide Makefile scripts. TemplateTagger depends on FastJet for jet finding and uses the internal FastJet classes for implementation of basic relativistic kinematics. The current version of TemplateTagger requires FastJet version 3.0 or higher. To use the Makefile provided with the code, simply add the location where FastJet is installed, so `fastjet-config` can be found.

To install and run the TemplateTagger follow these steps:

1. In a web browser, navigate to

   http://www.hepforge.org/archive/tom/

2. Download the (current) source tar-ball and extract it.

   ```
   tar -xvf TemplateOverlap-X.Y.Z.tar.gz
   ```

   and replacing `X.Y.Z` with the appropriate version number (currently 1.0.0). This will create a new subdirectory `TemplateOverlap-X.Y.Z` where all the TemplateTagger source files are now ready and unpacked.

3. Move to the resulting directory (`cd TemplateOverlap-X.Y.Z`) and compile one of the examples

   ```
   cd TemplateOverlap-X.Y.Z/
   g++ -Wall -O2 example.cc  -o example \
                       `${FASTJETLOCATION}/fastjet-config --cxxflags --libs`
   ```

   The code can also be compiled with the provided `Makefile` in environments where `make` is available.

4. The previous step compiles the `example` program which illustrate the basic functionality of TemplateTagger. The program reads a test event file `jet.dat` and a template file `template2b.dat` . To execute the test program, type

   ```
   ./example jet.dat template2b.dat
   ```

   The example code will write output to the terminal, `stdout`, and should read:

   ```
   Hardest jet: pt, y, phi =  324.9   0   0, mass = 125.788
   The best-matched templates are: (Ov2 = 0.968626)
   pt, y, phi =  220.425 0.259875 0.0501516, mass = 0.
   pt, y, phi =  105.432 -0.525 6.17819, mass = 0.
   ```

Now that you have seen the `example` application (and perhaps even compiled and run it), you might be wondering how it works. The following explanation will provide you with a basic understanding of the code, but the deeper implications will only become apparent after you have finished reading the rest of the tutorial.

3.2 The Algorithm

The core of the TemplateTagger package is a numerical implementation of the template matching algorithm of Eq. 1. The two primary components of every template matching process are:

- **Source event (j):** A jet containing full information about the constituents or calorimeter energy deposition.
- **Template (f):** A signal template which serves to construct a comparative measure.

The goal of template overlap analysis is to identify events j with high match to templates f. Events with high match have a higher likelihood of being signal. The TemplateTagger package performs the analysis in a sequence general enough to be applicable in a wide variety of HEP analyses:

1. Generate sets (or "catalogs") of large number of N-body templates which uniformly cover the phase space of a massive particle decay of mass M at a given p_T. We suggest that templates be generated in the lab frame by solving all the available kinematical constraints, as in Section A. For a realistic analysis, it is usually necessary to generate several sets of templates for different values of p_T and dynamically determine which template set is appropriate based on jet p_T. [1] Alternatively, templates can be generated in the rest frame of the event and boosted to the lab frame on an event-by-event basis. Note however, that this method comes with a significant increase in computation load, as millions of four momenta will typically have to be boosted for each event.

2. For each template, calculate a measure $d(j, f)$ to quantify the match (how similar the energy flow of the template is to that particular region in the flow of the observed event) of the template and the event

$$d(j,f) = \exp\left[-\sum_{a=1}^{N} \frac{1}{2\sigma_a^2} \left(\sum_{i\in\Omega} E_i^{(j)} F_N(\Omega, r) - E_a^{(f)} \right)^2 \right], \tag{4}$$

   where $F_N(\Omega, r)$ is a kernel function and r is the resolution scale parameter which determines the width and the shape of the kernel.

3. For each template f and event j, store the measure $d(f, j)$ in the result matrix $\mathbf{R}$. The result matrix is analogous to the output of many image pattern recognition algorithms. Fig. 1 shows an example. The points represent a complete template set at a fixed p_T and M of a two body boosted Higgs decay. The color map represents the value of $d(f, j)$ for each template state. The regions of high $d(f, j)$ are where most of the event p_T was deposited.

4. For every event j, find the maximum value

$$Ov_N = \max_{\{f\}} d(j,f), \tag{5}$$

   where f refers to maximizing over the entire set of templates. We refer to Ov_N as the "peak overlap." Similarly, we refer to the template f_{max} which maximizes $d(f, j)$ as the peak template.

[1] Choosing a template set based on jet p_T is inappropriate in a pileup environment.

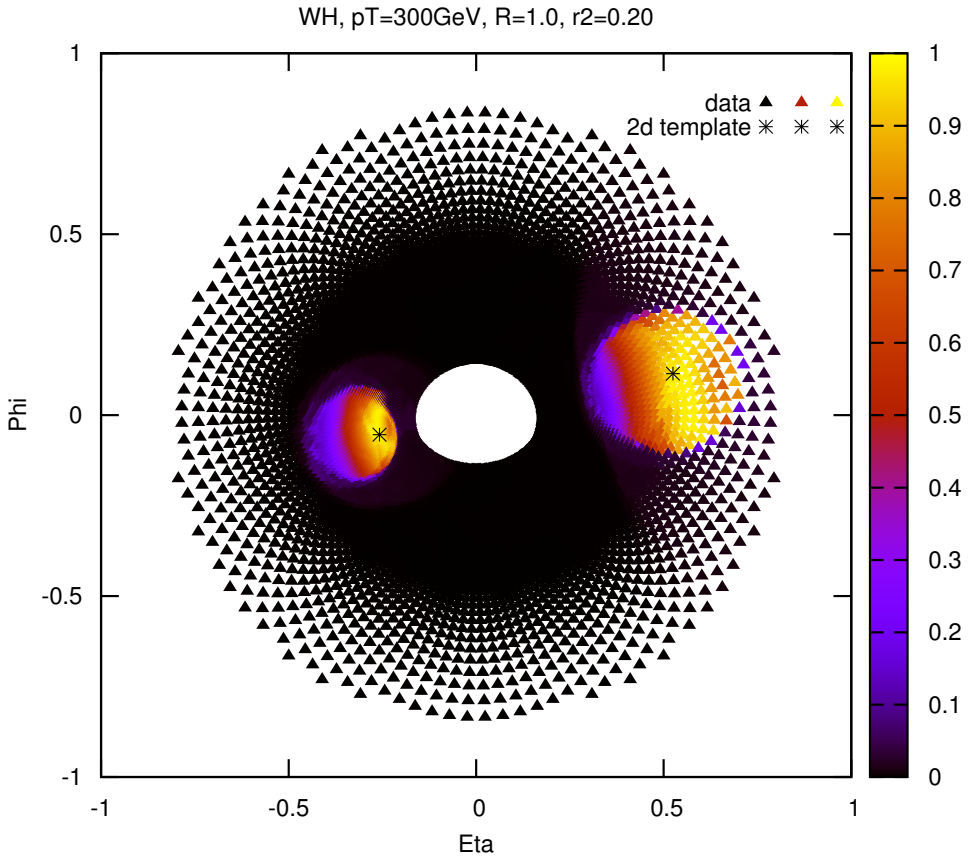

Figure 1: Energy flow reconstruction from 2-particle templates for a single boosted Higgs event. The points show angular positions of a highest p_T template parton (two-particle templates). Note that the other parton is uniquely determined by energy conservation. The color map shows the overlap score of the template parton at various positions in (η, ϕ). The region around $\eta = \phi = 0$ is not covered due to the kinematic constraint of $\Delta R_{b\bar{b}} > 2m_h/p_T^h$.

5. The previous three steps are repeated as many times as necessary using different values of N, *e.g.* for a boosted Higgs, $N = 2, 3$. Fig. 2 shows a typical matching process for a Higgs jet with $p_T = 300$ GeV analyzed with both 2- and 3-particle templates.

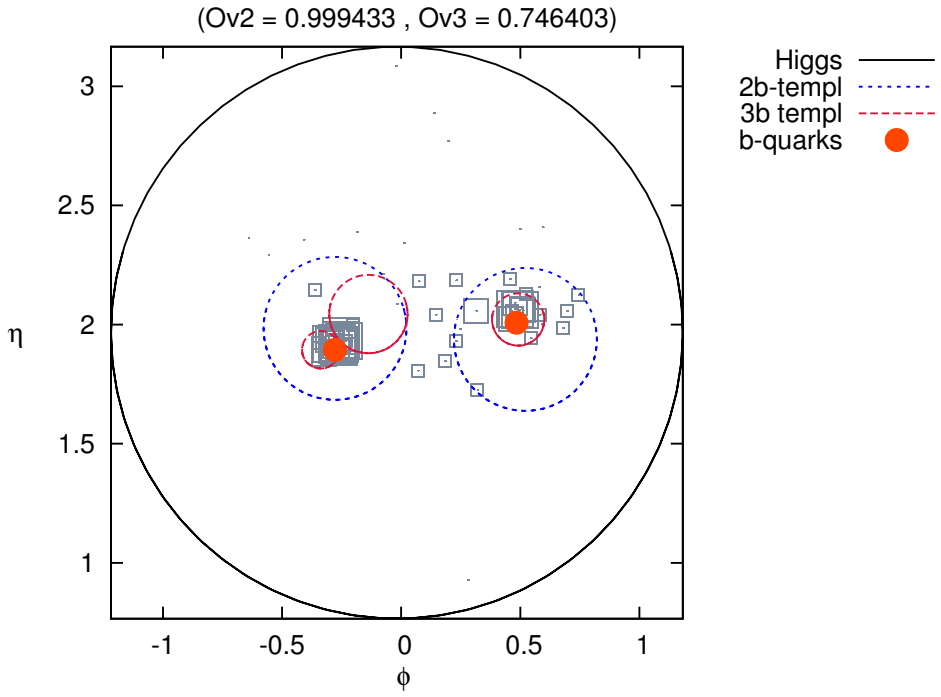

Figure 2: Event displays for a typical Higgs jet with invariant mass near 125 GeV. The blue and red circles indicate the region spanned by the best matched templates with $N = 2$, 3, respectively, using `CONE` and `DefaultMeasure`. In this and subsequent event displays, the particles are shown in grey cells of variable size, and the marker area for each cell is proportional its scalar transverse momentum. The solid red dots are the positions of b-quarks in the hard process.

3.3 Efficient Generation of Template Libraries

The TemplateTagger package provides routines which generate template libraries for a given model. The N-particle template libraries in the $\eta - \phi$ space can be represented using $3N$-dimensional tables with equidistant grids in the η, ϕ, and p_T variables. The construction of such tables would typically proceed with the help of Monte Carlo data to determine the size of p_T steps and the minimum number of templates required to maximize signal efficiency while maintaining sufficient background rejection power.

The function `TemplateBuilder` in `build_template.cc` contains the full specification of how to carry out the generation of templates in the lab frame. According to the method described in Appendix A, a "template definition" should include parameters such as template p_T, mass and jet cone radius R as well as the number of template patrons N. The function call is

```
void TemplateBuilder (std::ofstream & File,
                      const fastjet::PseudoJet & axis,
                      const double etaMax,
                      const double phiMax,
                      const double minPt,
                      const int nEta,
                      const int nPhi,
                      const int nPt,
                      const double R,
                      const int mode),
```

where `File` is the output file into which the code writes the template catalog and `axis` is the four vector of the event jet axis, providing relevant parameters like template mass and p_T. `etaMax` and `phiMax` are the maximum value of η and ϕ of a template parton relative to the jet axis, while `minPt` is the minimum p_T of a template parton (for infra-red safety). `nEta`, `nPhi` and `nPt` are the number of steps in η, ϕ and p_T respectively for the template generation. `R` is the anti-k_T jet cone used to cluster the template patrons. If the patrons can not be clustered into a jet of radius `R`, the template is rejected. Finally, `mode` is one of the entries of the enumerated `TemplateModel`:

```
enum TemplateModel {TOP, HIGGS2, HIGGS3, ...};
```

The current implementation of TemplateTagger contains three default modes:

- `TOP`: Three body t decay. The top template model generates template states to cover three-particle phase space for top decay, $t \to b+W \to b+q+\bar{q}$, with the constraint $(p_q+p_{\bar{q}})^2 = M_W^2$. To construct these states, the algorithm uses a sequential scan over four angles. We take these to be the rapidities and azimuthal angles that define the b and a W's daughter in the lab frame, defined relative to the direction of the top jet axis.
- `HIGGS2`: Two-body Higgs decay. Two angles define the two-body state of the daughter particles. We choose these to be the rapidity and azimuthal angle of the first daughter in the lab frame defined relative to the Higgs direction.
- `HIGGS3`: Three-body Higgs decay. Four angles and one energy define the three-body state of the daugher particles. We take these to be the the rapidities and azimuthal angles that define the b and a $\bar{b}$ directions in the lab frame, defined relative to the Higgs direction. The remaining variable is the p_T of the leading parton.

The `TemplateBuilder` routine surveys the kinematically-allowed template configurations by fixing the total four momentum to `axis` and then taking possible values of energies ($p_{T,i}$) and the angles ($\hat{p}_i$) within the bounded interval as defined by η_{max}, ϕ_{max} and p_T^{min}. The number of variables depends on the number of degrees of freedom of the configuration. The domain of the $3N-4$ independent variables, $\hat{p}_1, \cdots, \hat{p}_{N-1}$ and $p_{T,1}, \cdots p_{T,N-2}$, that define a template is divided into a uniform grid, according to a fixed interval, and the remaining transverse momentum $p_{T,N-1}$ is obtained by applying the restrictions of conservation of energy-momentum. The resulting groups which include negative p_T are automatically discarded. An additional restriction of

$$p_N = P - \sum_{i=1}^{N-1} p_i, \tag{6}$$

imposes the condition $P = (p_T, 0, 0, E_J)$.

3.4 Choosing the Kernel

The optimal choice of the template matching kernel depends on the analysis strategy and the amount of information the user has about the signal and the background. A reasonable choice of the kernel width typically assumes at least some kinematic properties or jet shapes of the signal. In fact, optimal choice will be different for different signals. For example, an analysis searching for a Higgs decaying into $b\bar{b}$ pairs is likely to make very different assumptions about jet substructure from a data analysis which looks for $t\bar{t}$ events in the all-hadronic 6-jet mode. It is thus both interesting and important to look at the substructure of a jet using a variety of kernels and kernel parameters. Table 1 lists the kernels available in the default implementation of TemplateTagger.

The kernel function $F(\Omega, f)$ should be a sufficiently smooth function of the angles for any template state f in order to ensure infra-red safety. For instance, the kernel could be defined as a Gaussian around each of the template momenta, which we provide with the option `GAUSSIAN`. Alternatively, we may choose F to the a normalized step function that is nonzero only in definite angular regions around the directions of the template momenta p_a. This is the default option in TemplateTagger and is implemented as an option `CONE`.

The TemplateTagger package also allows the user to choose from a variety of template matching strategies to fix the energy resolution scales. A fast and simple template matching can be performed using a single resolution scale (at one fixed cone radius or Gaussian width). This is the setting of the `FIXED` option and is the default option in TemplateTagger. Alternatively, a more sophisticated choice can also take into account more complex jet shapes. Indeed, the optimal width is not necessarily the same for every jet in an event, as low momentum subjets tend to have wider angular profiles. As an implementation of this feature, we propose varying cone schemes for template matching. The scheme is similar to the fixed cone scheme, except that we allow for different cone radii to be associated to each template particle.

Both the kernel and the energy resolution scale are set by variables in Table 1.

Enum	Default option	Alternate option
`Jet_shape_scheme`	`CONE`	`GAUSSIAN`
`Resolution_scale_scheme`	`FIXED`	`VARIABLE`

Table 1: Members of the `Jet_shape_scheme` and `Resolution_scale_scheme` enums which define the choice of kernel function.

3.5 Finding the Best Matched Templates

The Template Overlap approach locates templates in the $\eta - \phi$ space which have higher overlap than all of their neighbors in a template catalog. The algorithm of TemplateTagger is also able to process complicated energy flow patterns, *e.g.* when templates contain an arbitrarily large number of particles. `MatchingMethod` class provides the implementation of the algorithm which is responsible

for the Template Overlap analysis as a whole. The most important part of the `MatchingMethod` declaration (in the "matching.hh" header file) is the constructor of the `MatchingMethod` class

```
MatchingMethod(const string & templateFile, TemplateDefinition templDef)
```

The constructor arguments have the following meaning:

- `templateFile` A string containing the name of the template-catalog file.
- `templDef` An object which collects several settings to use for template matching. See Section 4.4 for more details.

The simplest way of performing template matching with TemplateTagger consists in constructing an object of this class at the beginning of the data analysis code and then running its `getOv()` member function on each jet. Fig. 3 shows a schematic representation of the algorithm.

The `MatchingMethod` loads the template catalog from a file using the default constructor. The member function `getOv` then reads a `fastjet::PseudoJet` and passes the input through a sequence of template matching functions which processes the following sequence:

- Load an input event and a test configuration (template)
- Perform a template matching procedure by using the `TemplateOverlap` function `matchTemplate` with either of the two matching methods described before.
- Normalize the output of the matching procedure so that unity means perfect match.
- Localize the template with the highest matching probability
- Return the maximum value of overlap and best matched configuration (maximum overlap template).

The `getOv` function returns the results of the maximization procedure in the format of `temple_t` (defined in Section 4).

where the first element of the ordered pair is the value of the overlap and the second one contains a vector of template four momenta. The user can pass the result of `getOv` to several `FunctionOfPseudoJets` which compute jet observables from the momenta in the best matched template.

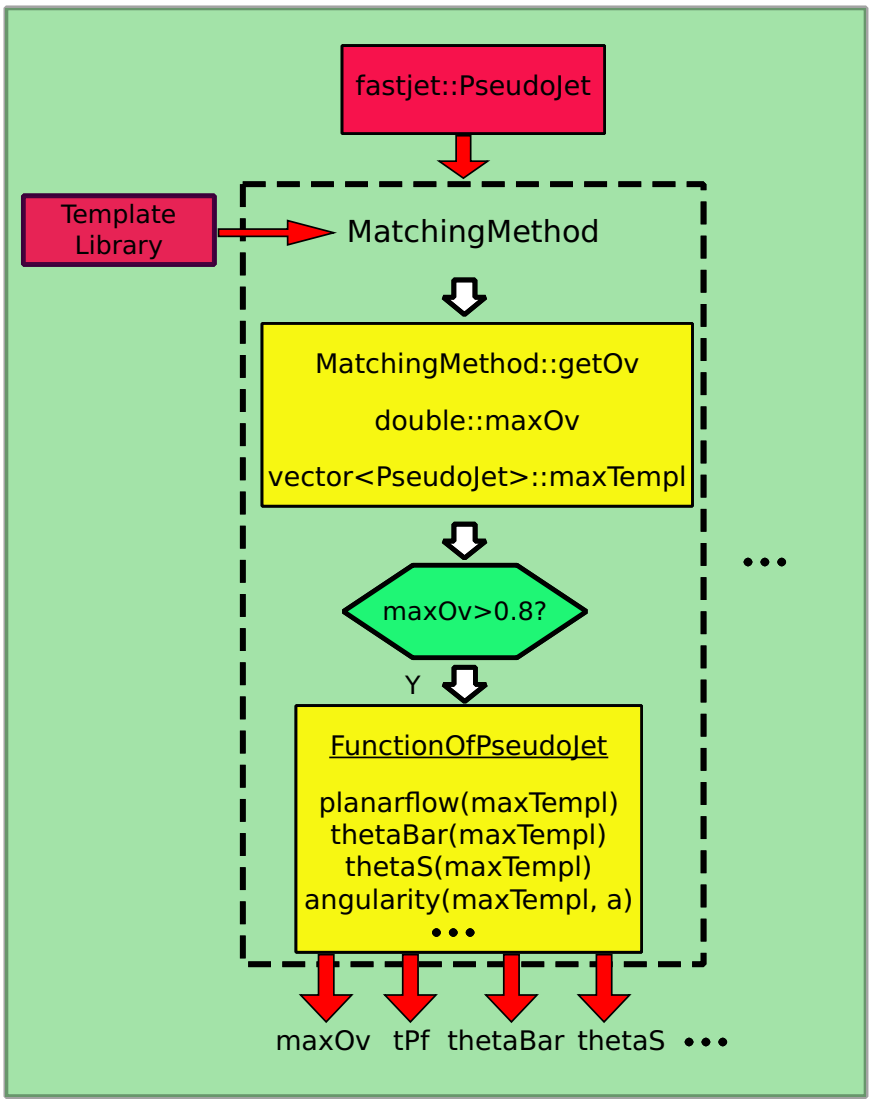

Figure 3: The structure of a typical TemplateTagger analysis. A `MatchingMethod` gets input from a template catalog file using the default constructor `MatchingMethod::MatchingMethod`. The method `getOv` reads a `fastjet::PseudoJet` and passes the input through a sequence of template matching functions that are run sequentially. The output of `getOv` is then passed to several `FunctionOfPseudoJets` that compute jet observables based on angular distributions of the partons in the best matched template.

3.6 Sample Program

An `example` program in `short.cc` provides the basic functionality of TemplateTagger. More specifically, the example shows how the `MatchingMethod` class can be used to perform basic substructure analysis with the template overlap method.

`example` requires only one input file: the template catalog. For the purpose of the example, consider a catalog of two-body templates stored in a file `template2bf.dat`. the command line call for `example` is

```
./example  template2bf.dat
```

Below, we give a detailed description of the relevant code snippets.

1. Include the appropriate header files. The core functionalities of TemplateTagger are contained in the file `matching.hh`.

   ```
   #include "matching.hh"
   #include "fastjet/FunctionOfPseudoJet.hh"
   using namespace TemplateOverlap;
   ```

2. Declare run parameters, event and result containers, as well as the match method. Set the jet cone radius $R = 1.0$ and the template sub-cone radius $r = 0.2$ as well as $\sigma_a = p_T^a/3$, where p_T^a is the transverse momentum of the a^{th} template parton. All settings are put into the `TemplateDefinition` object.

   ```
   /// Set Run Parameters
   double R = 1.2; // anti-kt parameter
   double R2 = 0.40; // template subcone radius
   double sigma = 0.333; // template Gaussian width
   myParams = TemplateDefinition(R2, sigma);
   ```

3. Define the event to be analyzed. For the purpose of the example, we hard-coded an event.

   ```
   /// An event with three-particles
   std::vector<fastjet::PseudoJet> particles;
   particles.push_back( fastjet::PseudoJet( 112.0, -19.8, -56.1, 126.9 ));
   particles.push_back( fastjet::PseudoJet( 110.6,  13.9,  25.3, 114.4 ));
   particles.push_back( fastjet::PseudoJet( 102.3,   5.9,  30.8, 107.1 ));
   ```

4. Define the template matching instance which uses the template catalog of `template2bf.txt` and the parameters stored in `myParams`.

   ```
   /// Create an instance of MatchingMethod for the analysis.
   /// The ctor also loads the templates
   MatchingMethod myCone(argv[1], myParams);
   ```

5. Analyze the event. `temp.first` contains the value of maximum overlap. `temp.second` contains the best matching template.

```
 /// Find the best matched template
templ_t result = myCone.getOv(jets[0]);
std::vector<fastjet::PseudoJet> maxTempl = result.second;
double maxOv = result.first;
```

The `getOv` method performs the core functions of the Template Overlap Method. As such, we deem it important to provide a more detailed discussion of its structure.

1. Let us check the `getOv` function. First define the matching method and make a copy of the source event:

```
/// Defining TemplateOverlap parameters
int match_method = CONE;

 /// Source jet to matrix
jet_t jet_mat;
myJet.copyTo(jet_mat);
```

   The `jet_t` typedef is a vector of particles, that collects basic kinematic information about the constituents in a jet, *i.e.* their η, ϕ and p_T.

2. Next it creates the matrix that will store the results for each template location:

```
/// Create the result matrix
 vector<double>  result;
```

3. Use the `TemplateOverlap` function `matchTemplate` to search for matches between a template and a input event

```
/// Do the matching and normalize
matchTemplate(jet_mat, _templates, result, match_method);
```

   the arguments are naturally the input event j (`jet_mat`), the templates f (`_templates`), the result $\mathbf{R}$ (`result`), and the `match_method`

4. Use the `TemplateOverlap` function `maximize` to find the maximum overlaps (as well as their template directions) in the result array:

```
/// Localizing the best match with minMaxLoc
double maxVal; int maxLoc;
maximize(result, maxVal, maxLoc );
```

   the function calls as arguments:

- `result`: the source array
- `maxVal`: variable to save the maximum value in the array, *i.e.* the maximum overlap
- `maxLoc`: the point location of the maximum values in the array, *i.e.* the best match template

5. Convert to `PseudoJet` and return the result

```
vector<PseudoJet> peak_template = ConvertToPseudoJet(_templates[_maxLoc], jet);
return std::make_pair<double, vector<PseudoJet> >(_maxVal, peak_template);
```

In summary, the output of `getOv` for any `PseudoJet` is the value of the overlap Ov_N, and also the identity of the peak template as a `vector` of `PseudoJet`s.

4 Miscellaneous Tools

4.1 Data types

We have defined two new data types, `templ_t` and `jet_t` for convenience. Here we briefly list the new data types for completeness and clarity, as they appear throughout the TemplateTagger code. We typically use `templ_t` to store the results of the overlap analysis for each event. The `double` value typically holds the maximum overlap, while the vector of `PseudoJet` holds the best matching template. The definitions are

1. `typedef std::pair<double, std::vector<fastjet::PseudoJet> > templ_t;`

2. `typedef std::vector<SingleParticle> jet_t;`

4.2 `FastJet` plugin

With the advent of FastJet 3.0+, it has become straightforward to write wrappers for jet analysis tools around the suite of tools available in FastJet. The FastJet bare class `FunctionOfPseudoJet<T>` provides a common interface for jet measurements. For the convenience of the user, we provide a class `Noverlap`, defined in `TemplateTagger.hh`, which performs all functions to the TemplateTagger code within the FastJet framework. The class wraps the core Template Tagger code to provide the `fastjet::FunctionOfPseudoJet` interface for convenience in larger analyses. See `matching.hh` for definitions of Ov_N and the constructor options. The relevant methods of the class are

1.
```
Noverlap Noverlap( double R2, double sigma, const string & templateFile
                , Resolution_scale_scheme variableCone, Jet_shape_scheme mode)
```

Constructor for the `Noverlap` class. `R2` is the template sub cone radius and `templateFile` is the file containing the template catalog, `sigma` is the fraction of template parton p_T^a used in σ_a, `variableCone` is the scaling rule for the sub cone radius (`FIXED` or `VARIABLE`), and `mode` determines the kernel function (`CONE` or `GAUSSIAN`).

2. `PseudoJet templ_t result(const PseudoJet& jet) const`

   This function returns the results of the overlap analysis in the format of a pair of values, the maximum value of overlap and the best matched template. See the discussion of `getOv` for more information.

FastJet 3.0+ also provides a common base class for jet manipulation: `Transformer`. Transformers can remove particles, re-arrange substructure, or tag/reject jets. The class `TemplateTagger` in `TemplateTagger.hh` implements a generic template overlap code described in the previous sections.

The `TemplateTagger` class derives from `Transformer`, and can be constructed using a pointer to a `Selector` class derived from `FunctionOfPseudoJet<templ_t>` which contains a value for the template kernel width, and the name of a file containing the catalog of templates. A simple example illustrates the implementation of TemplateTagger within the FastJet architecture:

```
#include "TemplateTagger.hh"
// ...
/// Set up the template tagger
SharedPtr<Noverlap> cone(new Noverlap(R2, sigma, argv[1], FIXED, CONE));
SelectorMassRange(minMass,maxMass) selector;
TemplateTagger tagger(cone.get(), selector, R2, ovcut);
/// Now tag the leading jet using template tagger
PseudoJet tagged_jet = tagger(jets);
```

We adopt the convention that if a given jet does not satisfy $Ov_N >$`ovcut` the result of the transformer is a jet whose 4-momentum is zero. The `pieces()` of the resulting tagged jet correspond to the subjets that were associated to the best matched template:

```
std::vector<fastjet::PseudoJet> subjets = tagged.pieces();
```

Additional structural information related to the value of the maximum overlap value and the best matched template is easily accessible. For instance:

```
cout << "(Ov2 = " << tagged.structure_of<fastjet::TemplateTagger>().ov() <<")"
      << endl << endl;
 cout << " The best-matched templates are: "<< std::endl;
 PrintJets(tagged.structure_of<fastjet::TemplateTagger>().maxTempl()); ,
```

displays the value of maximum overlap and the four momenta of the best matched template.

4.3 Jet and Template Moments

TemplateTagger allows a user to calculate additional substructure observables. The FastJet 3 base class `fastjet::FunctionOfPseudoJet<double>` provides a common interface for the calculation.

The following `FunctionOfPseudoJets` are available (all defined in `TemplateTagger.hh`). Each function returns a `double` value. Examples of their use can be found in `jet_shapes.cc`.

`planarflow()`: Calculates planar flow of the jet using longitudinally boost-invariant quantities

`angularity(int a)`: Calculates the angularity τ_{-a} of the jet

`thetaS()`: Calculates the angle between the softest template particle and the jet axis.

`thetaBar()`: Calculates an energy-unweighted distance between the template particles

`Stretch()`: Calculates the imbalance in ΔR between the peak two-body template and the two leading subjets inside the jet.

`Area()`: Calculates the template area, *i.e.* the area in $\eta - \phi$ space projected by the cones around the directions of the template particles.

4.4 `TemplateDefinition`

The `TemplateDefinition` class keeps track of all modes and parameters used during the jet clustering and template matching processes. As such, it serves all the `MatchingMethod` program elements from one central settings record. The user is allowed to access and change these settings to modify the template matching behavior. The complete list of methods and arguments is as follows.

```
 /// Parameters that define TemplateOverlap
class TemplateDefinition {
private:
   double _subConeRadius;     // subCone radius
   double _sigma;  // Gaussian energy resolution relative to parton pT
   Resolution_scale_scheme _subConeMode; //Fixed or varying subcones
   Jet_shape_scheme _mode; // Functional measure
   double _minPtParton; //For infrared safety, partons are not too soft

public:
   TemplateDefinition(const double subConeRadiusIn=0.20,
            const double sigmaIn=0.33,
            Resolution_scale_scheme variableConeIn=FIXED, Jet_shape_scheme modeIn = CONE,
             const double minPtPartonIn = 10.) :
   _subConeRadius(subConeRadiusIn),  _sigma(sigmaIn),
   _subConeMode(variableConeIn), _mode(modeIn) ,_minPtParton(minPtPartonIn) {}
    // Returns the value of the template sub cone radius
   double r() const {return _subConeRadius;}
   //Returns the fraction of template parton p_T used as \sigma_a.
   double sigma() const {return _sigma;}
    //Returns the status value for the overlap calculation mode,
   //i.e. fixed or varying cone
   Resolution_scale_scheme variableCone() const {return _subConeMode;}
   //Returns the kernel function mode
   //i.e. Gaussian or Cone
```

```
   Jet_shape_scheme mode() const {return _mode;}
   //Returns the minimum p_T of the template parton with smaller transverse momentum.
   double minPtParton() const {return _minPtParton;}
};
```

Acknowledgments

We thank Gilad Perez for introducing us to this topic and for a strong support in all stages of this project. We are very grateful to L. Levinson, P. Choukroun and the rest of the managers of Weizmann Institute physics computing farm for their flexibility and enormous help with large scale calculations. We thank S.J. Lee and R. Alon for the help with the early versions of the code and their suggestions and Gavin Salam for reading the code and sending valuable feedback. We also benefited greatly from discussions with P. Sinervo J. Winter, F. Spano, E. Duchovni and O. Silbert. Finally, we would like to thank the CERN Theory group for their hospitality. This work is supported by

A Properties of Templates

Template Overlap Method is a systematic framework aimed to identify kinematic characteristics of an boosted jet. A typical template configuration consists of a model template, f, calculated in perturbation theory, which describes a "prong-like" shape of the underlying hard subprocess of a jet. Template construction typically employs prior theoretical knowledge of the signal kinematics and dynamics, as well as possible experimental input. For instance, Higgs 2-particle templates are sets of 2 four momenta which satisfy kinematic constraints of a boosted Higgs decay etc.

The simplest template configurations are the ones describing the kinematics of two-body processes such as the decay of SM Higgs or W/Z bosons into quark-antiquark pairs. These are easily dealt with by assuming the rest frame of the parent particle and producing two decay products with equal and opposite, isotropically-selected momenta and magnitude, subject to energy conservation. The problem of a N-body decay subtracts four constraints from the decay products' $3N$ degrees of freedom: three for overall conservation of momentum and one for energy[2]. The final states can therefore be found on a $(3N-4)$-dimensional manifold in the multi-particle phase space. Note that the dimensionality of the template space increases rapidly with additional patrons. For instance, the two body templates require only two degrees of freedom, while a corresponding four body template space is already eight dimensional.

The question of which kinematic frame the templates should be generated in requires careful consideration. Authors of Ref. [25] argued that a search for the global maximum of Ov_N could be too computationally intensive. To improve the computation time, the template states were generated in the Higgs rest frame using a Monte Carlo routine, and then boosted into the lab frame. While this method worked sufficiently well for tagging a highly-boosted object(*i.e.* a 1 TeV Higgs jet), it

[2]For our purpose, a template is an object with no other properties other than its four-momenta.

introduced residual algorithmic dependence and a certain sense of arbitrariness in the jet shape. At lower p_T the Monte Carlo approach samples mainly the templates within the soft-collinear region, leaving other regions of phase space unpopulated. An enormous number of templates is required to adequately cover the phase space at $p_T \sim O(100\text{ GeV})$, thus fully diminishing the motivation for a Monte-Carlo approach. The simplest and most robust choice is then to generate templates directly in the lab frame and then rotate them into the frame of the jet axis. The result is a well covered template phase space in all relevant boosted frames. In addition, the lab frame templates result in a significant decrease in computation time as a much smaller number of templates are needed.

We proceed to show how to generate the phase space for 2- and 3-parton final states as well as how to generalize the results to arbitrary N.

A.1 The case of 2-body templates

First, we summarize our notation and conventions. The model template consists of a set of four vectors, $p_1, \cdots, p_N$, on the hyperplane determined by the energy-momentum conservation,

$$\sum_i p_i = P,\ P^2 = M^2, \tag{7}$$

where M, P are the mass and four momentum of a heavy boosted particle, i.e. the Higgs. For simplicity, we treat all template particles to be massless. We work in an (η, ϕ, p_T) space, where η is pseudorapidity, ϕ azimuthal angle and p_T transverse momentum. Without loss of generality, we can assume that the template points in the x direction ($\eta = \phi = 0$). The templates are distributed according to

$$p_i = p_{T,i}(\cos\phi_i, \sin\phi_i, \sinh\eta_i, \cosh\eta_i),\ i = 1, 2, 3 \tag{8}$$

subject to the constraint

$$\sum_{i=1}^{N} p_i = P = (p_T, 0, 0, E_J) \tag{9}$$

with $E_J = \sqrt{M^2 + p_T^2}$. We find it useful to define unit vectors by

$$\hat{p}_i = (\cos\phi_i, \sin\phi_i, \sinh\eta_i, \cosh\eta_i),\ i = 1, 2, \tag{10}$$

so that $p_i = p_{T,i}\,\hat{p}_i$.

Phase space for the 2-body decay processes is characterized by particularly simple kinematic parameters. To illustrate, first note that the 2-particle templates are uniquely determined by one single four momentum, p_1 subject to the condition

$$(P - p_1)^2 = 0. \tag{11}$$

Writing $p_1 = p_{T,1}\hat{p}_1$, we can solve for $p_{T,1}$ in terms of the angles of the first parton

$$p_{T,1} = \frac{M^2}{2(P \cdot \hat{p}_1)}. \tag{12}$$

We see that a 2-particle template is therefore completely determined in terms of the unit vector $\hat{p}_1$ as follows:

$$p_1 = \frac{M^2}{2(P \cdot \hat{p}_1)} \hat{p}_1 \tag{13}$$

$$p_2 = P - p_1. \tag{14}$$

Note that we can represent such a template as a point $(\hat{\eta}, \hat{\phi})$ in $\eta - \phi$ plane. These are the two degrees of freedom, in accordance with the general result that the dimensionality of the N template space is $3N - 4$.

A.2 The case of 3-body templates

A space of five degrees of freedom allows for 3-particle templates to differ from one another in more than one way. The 3-particle templates are determined by two four momenta, p_1 and p_2, subject to the constraint,

$$(P - p_1 - p_2)^2 = 0. \tag{15}$$

Using $p_1 = p_{T,1}\hat{p}_1$ and $p_2 = p_{T,2}\hat{p}_2$, we can solve for $p_{T,2}$ in terms of the angles of first two partons and $p_{T,1}$,

$$p_{T,2} = \frac{M^2 - 2P \cdot p_1}{2(P \cdot \hat{p}_2 - p_1 \cdot \hat{p}_2)}. \tag{16}$$

A general 3-particle template is then completely specified by $p_{T,1}$ and two unit vectors (or, equivalently, four angles) $\hat{p}_1$ and $\hat{p}_2$.

A.3 Extension to arbitrary N

A generalization to an arbitrary number of particles is straight-forward. Proceeding as above, the N-particle templates are determined by $p_1, \cdots, p_{N-1}$ subject to the constraint,

$$(P - \sum_{i=1}^{N-1} p_i)^2 = 0. \tag{17}$$

Using $p_i = p_{T,i}\hat{p}_i$, we can now solve for $p_{T,N-1}$ in terms of the $p_1, \cdots, p_{N-2}$ and $\hat{p}_{N-1}$,

$$p_{T,N-1} = \frac{M^2 + 2\sum_{i<j}^{N-2} p_i \cdot p_j - 2\,P \cdot \sum_i^{N-2} p_i}{2\,(\hat{p}_{N-1} \cdot P) - 2\,\hat{p}_{N-1} \cdot \sum_i^{N-2} p_i} \tag{18}$$

For the special cases of $N = 2$ and $N = 3$, this formula reduces to the above results .

B Substructure and jet shapes

The TemplateTagger code contains implementations of several jet shape observables in addition to the Template Overlap Method. Jet shapes are inclusive, infrared-safe observables which are smooth functions of the energy distribution within jets. They are constructed as weighted sums over the four-momenta of the constituents of a jet and reveal details about its inner structure, shedding light on its partonic origin.

- A set of such jet shape observables is given by the class of angularities τ_a of a jet, defined by

$$\tau_a \equiv \frac{1}{2E_J} \sum_{i \in J} |\mathbf{p_T^i}| e^{-\eta_i (1-a)}, \tag{19}$$

  where a is a parameter taking values $-\infty < a < 2$, the sum is over all the particles in the jet, E_J is the jet energy, $\mathbf{p}_T$ is the transverse momentum relative to the jet direction, and $\eta = -\ln \tan \theta/2$ is the pseudorapidity relative to the jet direction.

  Angularities, τ_a, are able to distinguish between QCD jets and other two-body decays. Almeida *et al.* [23] showed that the discriminating power of angularities is owned to the fact that the decays of color neutral objects are democratic, sharing energy symmetrically, whereas QCD events with same mass are typically asymmetric.

- Planar Flow (Pf) [26–28] is another useful jet substructure observable. We defined the default TemplateTagger implementation of Pf in terms pseudorapidity $\eta = -\ln \tan(\theta/2)$, the azimuthal angle ϕ and the transverse momentum p_T:

$$Pf = \frac{4 \det \mathbf{I}}{(\mathrm{tr}\, \mathbf{I})^2}, \tag{20}$$

  where $\mathbf{I}$ is defined by,

$$\mathbf{I} = \frac{1}{m_J} \sum_i p_T^i \begin{pmatrix} (\Delta\eta_i)^2 & \Delta\eta_i \Delta\phi_i \\ \Delta\eta_i \Delta\phi_i & (\Delta\phi_i)^2 \end{pmatrix} \tag{21}$$

  with m_J the jet mass, p_T^i is the transverse energy of particle i in the jet. Here, $(\Delta\eta_i, \Delta\phi_i) = \vec{c}_i - \vec{J}$, where $\vec{J} = (\eta_J, \phi_J)$ is the location of the jet and $\vec{c}_i$ is the position of a cell or particle with transverse momentum p_T^i. Notice that the Pf definition of Eq. 20 is invariant under boosts along the beam axis.

  Planar flow describes the way energy is deposited on the plane transverse to the jet axis. It peaks at zero for linear energy depositions and is close to unity for uniform energy configurations. For instance, two-pronged jets, such as leading order QCD jets, are expected to leave two cores of energy resulting in average low planar values of planar flow. On the other hand, three-prong jets coming from hadronic decays of boosted tops, are expected to have a rather uniform planar flow distribution. Thus planar flow can be used to separate massive boosted QCD jets from top jets.

- Angular correlations of the template momenta which can otherwise be concealed in the numerical values of the peak overlap are of particular value. For instance, the angular distribution of a

jet radiation can be measured with the variable $\bar{\theta}$, defined as

$$\overline{\theta} = \sum_i \sin \Delta R_{iJ},$$

where ΔR_{iJ} is the distance in the $\eta - \phi$ plane between the i^{th} template momentum and the jet axis. When measured using three-boy templates, the variable $\bar{\theta}$ characterizes the difference in angular ordering in our peak templates between the signal and background. Notice that for highly boosted jets, the 2-body version of $\overline{\theta}$ simply reduces to the angle between the two templates.

- Template Stretch is a pileup insensitive observable sensitive to the mass difference between a jet and the peak template. First introduced in Ref. [29] template stretch is defined as:

$$S_{b\bar{b}}^{(f)} = \frac{\Delta R_{b\bar{b}}}{\Delta R_f}.$$

where ΔR_f is the distance between the peak two-body template momenta and $\Delta R_{b\bar{b}}$ is the distance between the two b-tagged sub-jets. A generalization of S to non-b-tagged jets and other kinematic configurations is straightforward.

C TemplateTagger Classes and Commands

C.1 Classes

C.1.1 `SingleParticle`

`SingleParticle` is a helper class for `MatchingMethod` whose aim is to contain minimum information about a particle in an event or a template, mainly its energy (or transverse momentum), rapidity and azimuthal angle. When the particle is associated with a template, there are two variables that contain additional, non-kinematics information: the template parton's width parameter `radius` (i.e. radius of the template sub cone), and its energy resolution `sigma`.

```
 /// A helper class for MatchingMethod
class SingleParticle {

public:

  // Constructors.
  SingleParticle(  double pTIn = 0., double yIn = 0., double phiIn = 0.)
  :  pT(pTIn), y(yIn), phi(phiIn), mult(1), isUsed(false), radius(0.), sigma(0.) { }
  SingleParticle(const SingleParticle& ssj) :  pT(ssj.pT),
    y(ssj.y), phi(ssj.phi), mult(ssj.mult),
    isUsed(ssj.isUsed), radius(ssj.radius), sigma(ssj.sigma) { }
  SingleParticle& operator=(const SingleParticle& ssj) { if (this != &ssj)
    {  pT = ssj.pT; y = ssj.y; phi = ssj.phi;
    mult = ssj.mult; isUsed = ssj.isUsed;
```

```
    radius =ssj.radius; sigma=ssj.sigma;} return *this; }

  // Properties of particle.
  double pT, y, phi;
  int    mult;
  bool   isUsed;
  double radius; //For templates
  double sigma;

  double deltaR2(const SingleParticle & other) const {
    double  dPhi = abs(phi - other.phi );
    if (dPhi > M_PI) dPhi = 2. * M_PI - dPhi;
    double  dEta =  y - other.y;
    return (dEta * dEta + dPhi * dPhi );
  }

};
```

C.1.2 `MeasureFunctor`

`TemplateTagger` provides a bare class to define custom overlap functions: `MeasureFunctor`. This provides the user with some flexibility to specify different kernel functions or to build new ones for customized template analyses. The `MeasureFunctor` provides the minimum bare class from which other overlap functions can be derived. The most important part of the class declaration looks as follows.

```
class MeasureFunctor {
protected:
  MeasureFunctor() {}
public:
  virtual double distance(const SingleParticle& particle,const SingleParticle& axis)=0;
  virtual double numerator(const SingleParticle& particle,const SingleParticle& axis)=0;
  std::vector<double> subOverlaps(const jet_t & particles, const jet_t& axes);
  double overlap(const jet_t & particles, const jet_t& axes);
};
```

C.1.3 `DefaultMeasure`

The `DefaultMeasure` implements a cone-based template matching with the function

$$F(\hat{n}_i, \hat{n}_a^{(f)}) = \begin{cases} 1 & \text{if } \Delta R < R_a \\ 0 & \text{otherwise} \end{cases} , \tag{22}$$

By default, this function finds all particles within a cone of radius R from the template parton and calculates the unweighted sum of the particles p_T. By default, a fixed cone radius R is used. This can be modified via the `TemplateDefinition::set_varying_cone()`. This option corresponds to varying cone; note that for the varying cone mode to work the user needs to specify a model for cone scaling rule (energy profile). See `eShape` for more details.

```
class DefaultMeasure : public MeasureFunctor {
private:
  TemplateDefinition _templDef;
public:
  DefaultMeasure() {}
  virtual double distance(const SingleParticle& particle, const SingleParticle& axis){
         return std::sqrt(particle.deltaR2(axis)); }
  virtual double numerator(const SingleParticle& particle, const SingleParticle& axis){
         double deltaR = std::sqrt(particle.deltaR2(axis));
         if (deltaR > _templDef.r()) return 0.0;
         return particle.pT;   }  };
```

C.1.4 `GaussianMeasure`

Similar to `DefaultMeasure`, but uses a Gaussian kernel function

$$F(\hat{n}_i, \hat{n}_a^{(f)}) = \exp\left[-(\Delta R)^2/(2\omega_a^2)\right],$$

to add the p_T of each particle in a jet. It has two constructors with the same arguments as `DefaultMeasure`.

```
class GaussianMeasure : public MeasureFunctor {
private:
  TemplateDefinition _templDef;
public:
  GaussianMeasure() {}

  virtual double distance(const SingleParticle& particle, const SingleParticle& axis) {
         return std::sqrt(particle.deltaR2(axis));}
  virtual double numerator(const SingleParticle& particle, const SingleParticle& axis) {
        double etaNow = particle.y;
        double phiNow = particle.phi;
        double rNow = particle.radius;
        double weight = exp(-(etaNow*etaNow+phiNow*phiNow)/(2 * rNow * rNow));
         return (particle.pT * weight);}
};
```

C.2 Functions

C.2.1 `matchTemplate`

The `matchTemplate` function is used to locate patterns inside the observed energy distributions within jets which have good overlap ("match") to the set of templates. The algorithm can handle a variety of complicated patterns, *e.g.*, when the templates have more than the minimum number of partons or when there are additional kinematical constraints, such as the W mass in a hadronically decaying top quark. The algorithm is implemented in the `matchTemplate` function whose prototype looks like this:

```
void matchTemplate(jet_t & jet,
                   const  vector<jet_t> & templates,
                   vector<double>  & result,
                   int match_method);
```

The function arguments have the following meaning:

- `jet`: The jet being analyzed. It must be a vector of `SingleParticle`.
- `templates`: Comparison template catalog. It must not have more particles than the source event.
- `result`: Map of comparison results. It must be a one-dimensional vector of `double`. After the search, its dimension is `templates.size()`.
- `match_method`: Parameter specifying the functional measure or comparison method. It can take values available in `Jet_shape_scheme` . See Table 1 for details.

C.2.2 `maximize`

Finds the global maximum in the result array and its position.

```
void maximize(const vector<double>  & result, double & maxVal, int & maxLoc );
```

The parameters have the following meaning:

- `result`: input array
- `maxVal`: pointer to the returned maximum value; `NULL` is used if not required.
- `maxLoc`: pointer to the returned maximum location

C.2.3 `ConvertToPseudoJet`

For internal use. Converts a `jet_t` to `PseudoJet`.

```
std::vector<fastjet::PseudoJet> ConvertToPseudoJet(const jet_t& particles);
```

C.2.4 `ConvertToMat`

For internal use. Converts a `PseudoJet` to `jet_t`.

```
jet_t ConvertToMat(const fastjet::PseudoJet& jet);
```

C.2.5 `overlapDistance`

Calculates the overlap between two jets or templates in the η, ϕ, p_T space, assuming one jet is a "template". `R0` is the template sub-cone radius.

```
double overlapDistance (jet_t & jet1, jet_t & jet2, double R0) ;
```

C.2.6 `reset`

For internal use. Clear internal flags in a jet for reuse.

```
void reset(jet_t & jet);
```

C.2.7 `eShape`

In some applications, one would like to have a more realistic model for the energy profile of a template parton than simply a fixed cone. TemplateTagger provides with a simple scaling rule for the template subcone radius that draws information from jet shape measurements at the LHC. To compute the energy profile of a template parton, the numerical values for the integrated jet shape measured by ATLAS are fit in different regions of jet p_T. `eShape` returns the radius, r, that is needed to contain 80% of the transverse momentum in a cone of radius r around the template parton direction.

```
double eShape (double x)
{
  double aux =  0.422258 - 0.00377161* x + 0.0000174186 * x*x -
  3.50639e-8 * x *x*x + 2.53302e-11 * x*x*x*x;

  return aux;
}
```

References

[1] S.D. Ellis, J. Huston, K. Hatakeyama, P. Loch, and M. Tonnesmann. Jets in hadron-hadron collisions. *Prog.Part.Nucl.Phys.*, 60:484–551, 2008.

[2] A. Abdesselam, E. Bergeaas Kuutmann, U. Bitenc, G. Brooijmans, J. Butterworth, et al. Boosted objects: A Probe of beyond the Standard Model physics. *Eur.Phys.J.*, C71:1661, 2011.

[3] A. Altheimer, S. Arora, L. Asquith, G. Brooijmans, J. Butterworth, et al. Jet Substructure at the Tevatron and LHC: New results, new tools, new benchmarks. *J.Phys.*, G39:063001, 2012.

[4] Gavin P. Salam. Towards Jetography. *Eur.Phys.J.*, C67:637–686, 2010.

[5] Pran Nath, Brent D. Nelson, Hooman Davoudiasl, Bhaskar Dutta, Daniel Feldman, et al. The Hunt for New Physics at the Large Hadron Collider. *Nucl.Phys.Proc.Suppl.*, 200-202:185–417, 2010.

[6] Tilman Plehn and Michael Spannowsky. Top Tagging. *J.Phys.*, G39:083001, 2012.

[7] Jesse Thaler and Ken Van Tilburg. Maximizing Boosted Top Identification by Minimizing N-subjettiness. *JHEP*, 1202:093, 2012.

[8] Chunhui Chen. New approach to identifying boosted hadronically-decaying particle using jet substructure in its center-of-mass frame. *Phys.Rev.*, D85:034007, 2012.

[9] Zhenyu Han. Tracking the Identities of Boosted Particles. *Phys.Rev.*, D86:014026, 2012.

[10] Ilya Feige, Matthew D. Schwartz, Iain W. Stewart, and Jesse Thaler. Precision Jet Substructure from Boosted Event Shapes. *Phys.Rev.Lett.*, 109:092001, 2012.

[11] Tilman Plehn, Gavin P. Salam, and Michael Spannowsky. Fat Jets for a Light Higgs. *Phys.Rev.Lett.*, 104:111801, 2010.

[12] Tilman Plehn, Michael Spannowsky, and Michihisa Takeuchi. How to Improve Top Tagging. *Phys.Rev.*, D85:034029, 2012.

[13] Tilman Plehn, Michael Spannowsky, and Michihisa Takeuchi. Boosted Semileptonic Tops in Stop Decays. *JHEP*, 1105:135, 2011.

[14] Jesse Thaler and Ken Van Tilburg. Identifying Boosted Objects with N-subjettiness. *JHEP*, 1103:015, 2011.

[15] Yanou Cui, Zhenyu Han, and Matthew D. Schwartz. W-jet Tagging: Optimizing the Identification of Boosted Hadronically-Decaying W Bosons. *Phys.Rev.*, D83:074023, 2011.

[16] Anson Hook, Martin Jankowiak, and Jay G. Wacker. Jet Dipolarity: Top Tagging with Color Flow. *JHEP*, 1204:007, 2012.

[17] Davison E. Soper and Michael Spannowsky. Finding top quarks with shower deconstruction. 2012.

[18] David E. Kaplan, Keith Rehermann, Matthew D. Schwartz, and Brock Tweedie. Top Tagging: A Method for Identifying Boosted Hadronically Decaying Top Quarks. *Phys.Rev.Lett.*, 101:142001, 2008.

[19] Jason Gallicchio and Matthew D. Schwartz. Seeing in Color: Jet Superstructure. *Phys.Rev.Lett.*, 105:022001, 2010.

[20] Leandro G. Almeida, Ozan Erdogan, Jose Juknevich, Seung J. Lee, Gilad Perez, et al. Three-particle templates for a boosted Higgs boson. *Phys.Rev.*, D85:114046, 2012.

[21] Leandro G. Almeida, Raz Alon, and Michael Spannowsky. Structure of Fat Jets at the Tevatron and Beyond. *Eur.Phys.J.*, C72:2113, 2012.

[22] Leandro G. Almeida, Seung J. Lee, Gilad Perez, Ilmo Sung, and Joseph Virzi. Top Jets at the LHC. *Phys.Rev.*, D79:074012, 2009.

[23] Leandro G. Almeida, Seung J. Lee, Gilad Perez, George Sterman, and Ilmo Sung. Template Overlap Method for Massive Jets. *Phys.Rev.*, D82:054034, 2010.

[24] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. The Anti-k(t) jet clustering algorithm. *JHEP*, 0804:063, 2008.

[25] Leandro G. Almeida, Ozan Erdogan, Jose Juknevich, Seung J. Lee, Gilad Perez, et al. Three-particle templates for a boosted Higgs boson. *Phys.Rev.*, D85:114046, 2012.

[26] Jesse Thaler and Lian-Tao Wang. Strategies to Identify Boosted Tops. *JHEP*, 0807:092, 2008.

[27] Leandro G. Almeida, Seung J. Lee, Gilad Perez, George F. Sterman, Ilmo Sung, et al. Substructure of high-p_T Jets at the LHC. *Phys.Rev.*, D79:074017, 2009.

[28] Guy Gur-Ari, Michele Papucci, and Gilad Perez. Classification of Energy Flow Observables in Narrow Jets. 2011.

[29] Mihailo Backović, Jose Juknevich, and Gilad Perez. Boosting the standard model higgs signal with the template overlap method. xxx.