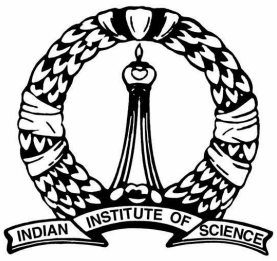
MadAnalysis Recast Code for Dark Matter Production in Association with Bottom Quarks

Masters Thesis

Diptaparna Biswas
5th Year BS-MS
Indian Institute of Science
Bangalore - 560012



Project Guide:

Dr. Jyothsna Komaragiri
Indian Institute of Science
Bangalore - 560012

April 2017

Abstract

The CMS-B2G-15-007 analysis was carried out to investigate the $pp \rightarrow bb+DM$ process, which is a candidate process of direct Dark Matter production. In this particular work, this process is simulated and analyzed for 13 TeV LHC experiment using MadGraph, Pythia 8 and MadAnalysis C++ API. The recast code is written to reproduce the results obtained by the experimentalists in CMS-B2G-15-007 analysis. The result is interpreted within simplified scalar model in terms of the coupling between the mediator and the DM candidate and evaluated based on the MET distribution.

CMS-B2G-15-007 uses a dataset of $2.17fb^{-1}$ of data collected by the CMS experiment in $\sqrt{s} = 13TeV$ proton-proton collisions at LHC. This analysis is sensitive also to DM production processes in association with top quarks. Results are reported as upper limits on the cross section for the b quark and top quark associated production independently, and interpreted within simplified models in terms of the coupling between the mediator and the DM candidate, for different mediator and DM candidate masses. In this particular recasting the mediator mass is chosen to be $100GeV$ and the DM candidate is chosen to be $1GeV$.

Contents

List of Figures

Chapter 1

Introduction

One of the most important questions in particle physics is the understanding of Dark Matter. Although various astrophysical sources point to an invisible gravitationally interacting substance in the Universe, no evidence for such a particle has been seen in direct detection experiments. Given the CERN LHC proton-proton collision energy increase to 13 TeV, and the absence of any clear signatures beyond the standard model of particle physics (SM) in Run-1, the search for the production of DM in 13 TeV proton-proton collisions will be the primary interest for the LHC Run-2.

MadAnalysis is a tool for analyzing particle accelerator data and recasting results obtained in some particle physics experiment. In this project the CMS data for $pp \rightarrow bb + DM$ collision is simulated using MadGraph and further analyzed using the MadAnalysis C++ API. Event samples are generated using MadGraph 5, hadronization is carried out using Pythia 8 and recasting code is written using MadAnalysis 1.3 with ROOT 5 as computational engine.

1.1 Overview of the CMS-B2G-15-007 (bb+DM) Analysis

The process investigated here is a candidate model of Dark Matter production along with bottom quarks[1]. It is experimentally done in LHC experiment and the products are detected using CMS detector. The collision is between two high energy (6.5 TeV) proton beams. In this process bottom and anti-bottom quarks are produced along with some BSM particle, which might be a candidate for Dark Matter. The MET (missing transverse momentum/energy) is calculated for this process and non-zero value of MET may correspond to the momentum carried out by the Dark Matter particles.

$$pp \rightarrow b\overline{b} + \chi\overline{\chi}$$

There are two important sub-processes to simulate -

- p p > b b~ chi chi~
- p p > b b~ chi chi~ j

The chi and chi~ are the candidate Dark Matter particles.

The Feynman diagrams of the involved processes are given below -

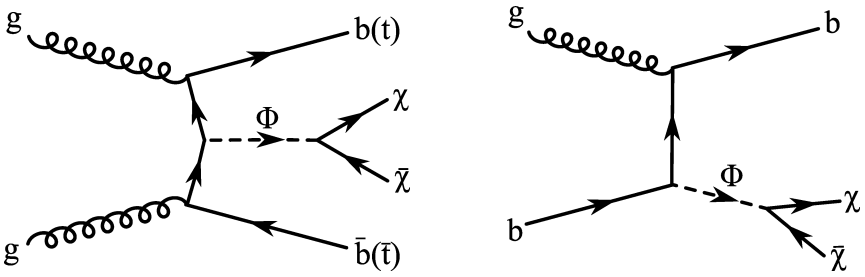


Fig 1: Feynman diagrams involved in the process

1.2 What is a Recast Code?

Recast code is a way to enable theorists and phenomenologists to reproduce the same result obtained by experimentalists using simulated data.

So, for recasting an LHC analysis, we need the following things -

- An event generator
- A detector simulator
- An analysis framework

1.3 Software Setup for this Recasting

For this recasting of CMS-B2G-15-007 analysis, the following softwares has been used -

- Event generator – MadGraph (as MC generator) & Pythia (as hadronizer)
- Detector Simulator – Delphes (with proper CMS card)
- Analysis Framework – MadAnalysis (in expert mode)

The overview of the whole process is shown in the following diagram -

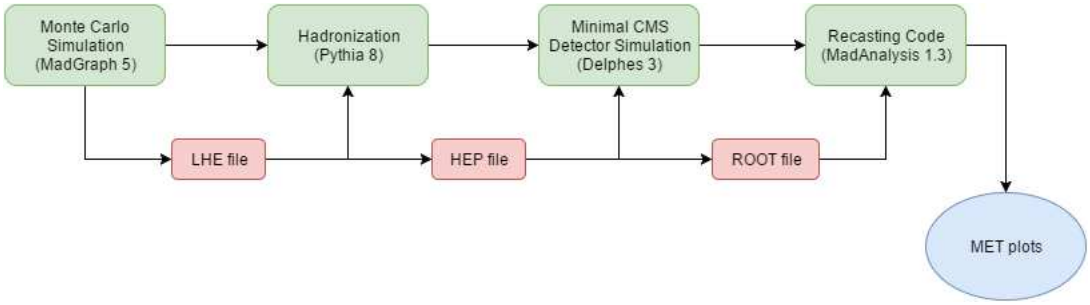


Fig 2: The complete workflow diagram

Recast Plots

Neither, every plot obtained in real analysis can be obtained from simulated data, nor every plot obtained in real analysis is interesting to theorists or phenomenologists.

So, recast plots are limited – only MET distribution (in our case), as this is a Dark Matter Search.

An elaborated explanation for choosing the recast plot is given in the analysis section.

Chapter 2

Event Generation

The Monte Carlo generator used for event generation is MadGraph5_aMC@NLO. It is a framework that aims at providing all the elements necessary for SM and BSM phenomenology, such as the computations of cross sections, the generation of hard events and their matching with event generators, and the use of a variety of tools relevant to event manipulation and analysis. Processes can be simulated to LO accuracy for any user-defined Lagrangian, and the NLO accuracy in the case of QCD corrections to SM processes. Matrix elements at the tree and one-loop level can also be obtained[2].

MadGraph takes inputs in the form of several cards -

- proc_card.dat – description of the processes
- param_card.dat – mass, decay and other model parameters
- run_card.dat – beam energy, pdfset and other collider configuration

proc_card.dat is actually made interactively using the following commands -

```
> import model DMScalar
> generate p p > b b~ chi chi~
> add process p p > b b~ chi chi~ j
```

2.1 Parameter Card

There are two 'unknown' particles involved in the whole process –

- The Dark Matter candidates (χ and $\bar{\chi}$)
- The mediator (ϕ) that splits into χ and $\bar{\chi}$

The mass of phi is set to 100 GeV for this simulation. The mass of χ (and $\bar{\chi}$) is set to 1 GeV.

Other settings in the param_card_default.dat are kept unchanged.

2.2 Run Card

The following changes are made in run_card.dat -

- 10000 = nevents
- 6500 = ebeam1
- 6500 = ebeam2
- lhapdf = pdflabel
- 262400 = lhaid
- 1 = ickkw
- 4 = maxjetflavor
- 20 = xqcut

2.3 Hadronization with Pythia 8

The Pythia program is a standard tool for the generation of events in high-energy collisions between elementary particles, comprising a coherent set of physics models for the evolution from a few-body hard-scattering process to a complex multiparticle final state. Parts of the physics have been rigorously derived from theory, while other parts are based on phenomenological models, with parameters to be determined from data. Currently the largest user community can be found among the LHC experimentalists, but the program is also used for a multitude of other phenomenological or experimental studies. Main tasks performed by the program include the exploration of experimental consequences of theoretical models, the development of search strategies, the interpretation of experimental data, and the study of detector performance. Thereby it spans the whole lifetime of an experiment, from early design concepts for the detector to final presentation of data. In this particular case, however, Pythia is only used for hadronizing the MC sample produced by MadGraph.

For this purpose, Pythia 8 is compiled with HEPMC2 support. The main89.cc example program is used to hadronize the .lhe file (MadGraph output) to .hepmc file. main89 takes a configuration card (.cmnd file), a part of it is given below -

- 9100000:new = MED MED 3 0 0 100 0 0 0 99999
- 9100022:new = DM DM 2 0 0 1 0 0 0 99999
- 9100022:mayDecay = off
- Tune:ee = 3
- Tune:pp = 5

The complete listing of the Pythia 8 configuration card (DM_settings.cmnd) is given in the Appendix I.

The .hepmc file is produced by invoking the following command -

$ ./main89 DM_settings.cmnd pythia_output.hepmc

2.4 Detector Simulation with Delphes 3

DelphesHepMC executable takes a configuration file delphes_card_CMS.tcl and performs detector simulation on the .hepmc file. Information about various MC objects (particles) and Reconstructed objects (jets, rec. particles) are saved in a .root file in form of Delphes trees.

Delphes 3 uses fastjet internally for jet reconstruction. The .root file can be opened using root itself. However, we write the MadAnalysis C++ code to analyze the .root file and produce plot.

The Delphes tree can be produced by invoking the following command -

$ ./DelphesHepMC cards/delphes_card_CMS.tcl DM.root pythia_output.hepmc

This produces a file called DM.root, which contains the Delphes tree and will be used in the further analysis.

Chapter 3

LHE Analysis

LHE (or LHEF) stands for Les Houches Event File. It is the primary output that we get from a MC generator like MadGraph. It contains various kinematic parameters of all the particles involved in the processes along with the description of simulated processes, model parameters and run conditions[3].

LHE Analysis is performed to understand various basic kinematic properties of the produced MC sample. The kinematic variables associated with different particles of the event can be obtained using this method.

3.1 Structure of a LHE File

Every event contains a line (first line), which has the information about number of particles involved in the process and the weight of the process.

The next lines are the descriptions for each particle involved. The kinematic variables associated with each particle is present in tab separated columns -

- Col 1: ID code of the particle. This number determines the identity and charge of the particle.
- Col 2-6: Unused, might be used in future LHE versions.
- Col 7-9: px, py and pz of the particle in GeV.
- Col 10: Energy E of the particle in GeV.
- Col 11: Mass m of the particle in GeV.
- Col 12: Unused, might be used in future LHE versions.
- Col 13: Spin of the particle.

Also, in the beginning of the LHE file, all the details of run_card, param_card and proc_card are also provided.

A small portion of produced LHE file is given below -

```
<event>
 7   1  0.1563300E-03  0.4097958E+02  0.7818608E-02  0.1364061E+00
       21   -1    0    0  503  502  0.00000000000E+00  0.00000000000E+00  0.90938214042E+01  0.90938214042E+01  0.00000000000E+00 0.  1.
       21   -1    0    0  501  503  0.00000000000E+00  0.00000000000E+00 -0.61108611318E+03  0.61108611318E+03  0.00000000000E+00 0.  1.
  9100000    2    1    2    0    0 -0.15299993906E+01  0.15704157034E+02 -0.51267060797E+03  0.52195744923E+03  0.96744328403E+02 0.  0.
        5    1    1    2  501    0  0.15363574118E+00 -0.16580335406E+02 -0.91267437007E+02  0.92880386433E+02  0.47000000000E+01 0. -1.
       -5    1    1    2    0  502  0.13763636495E+01  0.87617837209E+00  0.19457532005E+01  0.53420989276E+01  0.47000000000E+01 0.  1.
  9100022    1    3    3    0    0  0.16089689000E+01  0.46045180892E+02 -0.12738013590E+03  0.13546012877E+03  0.10000000000E+01 0.  1.
 -9100022    1    3    3    0    0 -0.31389682906E+01 -0.30341023858E+02 -0.38529047208E+03  0.38649732046E+03  0.10000000000E+01 0.  1.
<scales pt_clust_1="13000.00000" pt_clust_2="13000.00000" pt_clust_3="13000.00000" pt_clust_4="13000.00000"></scales>
<mgrwt>
<rscale>  2 0.30091215E+02</rscale>
<asrwt>0</asrwt>
<pdfrwt beam="1">  1       21 0.13990494E-02 0.40979585E+02</pdfrwt>
<pdfrwt beam="2">  1       21 0.94013248E-01 0.40979585E+02</pdfrwt>
<totfact> 0.44678675E+04</totfact>
</mgrwt>
<clustering>
<clus scale="    4.975">  1  5  2 -1</clus>
<clus scale="   17.113">  2  4  1 -1</clus>
<clus scale="   96.744">  6  7 -1 -1</clus>
<clus scale="   98.134">  1  4  2 -1</clus>
</clustering>
</event>
```

3.2 LHE Analysis Code

First, the LHE file is opened and read line by line. The text processing advantage of Python is used to extract the actual event part from the LHE file. This list of events is saved in the *events* variable.

```
f = open("unweighted_events.lhe")
line=""
events=[]
currEvent=""
addLine=False
for line in f:
        if line.strip()=='<event>':
                currEvent=""
                addLine=True
        elif line.strip().startswith("<scales"):
                events.append(currEvent.strip())
        addLine=False
        elif addLine:
        currEvent+=line.strip()+'\n'
f.close()
```

Then various particles can be extracted by matching the PDG-id.

```
phi=[s for s in event.split('\n') if s.split()[0]=='
    9100000']
chi=[s for s in event.split('\n') if s.split()[0]=='
    9100022']
chibar=[s for s in event.split('\n') if s.split()[0]=='
    -9100022']
```

Whether a particular particle really exists can be checked using the if statement on list. An empty list is evaluated *False* in Python.

```
if phi:
        mass_phi.append(float(phi[0].split()[10]))
        px=float(phi[0].split()[6])
        py=float(phi[0].split()[7])
        pz=float(phi[0].split()[8])
        e=float(phi[0].split()[9])
        p=TLorentzVector(px,py,pz,e)
        pt_phi.append(p.Pt())
        eta_phi.append(p.Eta())
```

Finally, these can be plotted using TH1F class given by ROOT library on TCanvas and saved as .png files.

The complete code listing for LHE Analysis is given in Appendix II.

3.3 LHE Analysis Plots

The following plots are obtained using the above method -

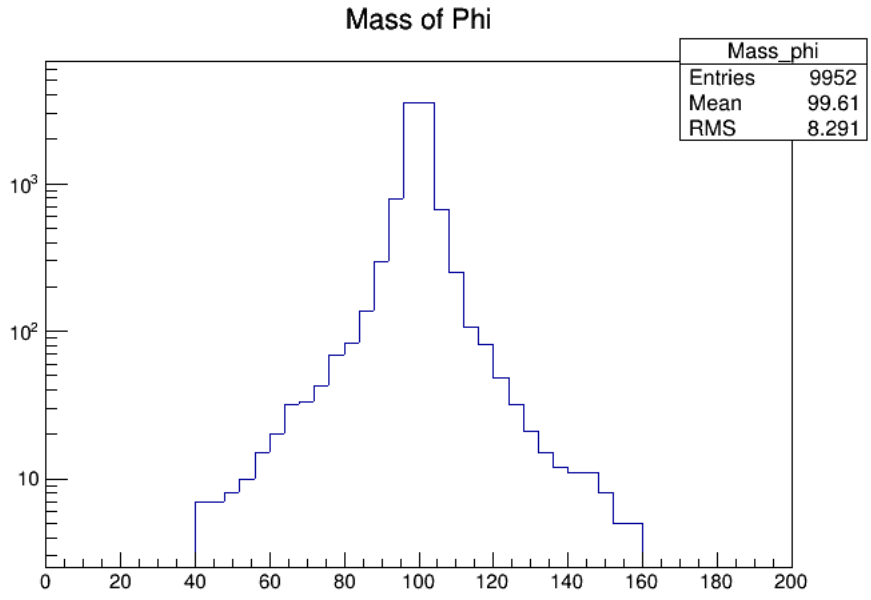


Fig 3: Histogram showing the mass distribution of ϕ

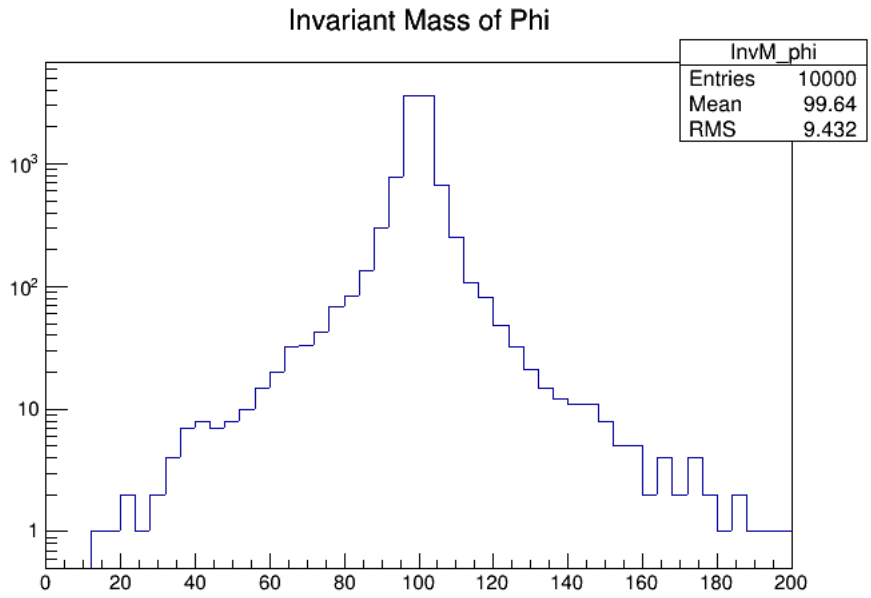


Fig 4: Histogram showing the distribution of invariant mass of ϕ

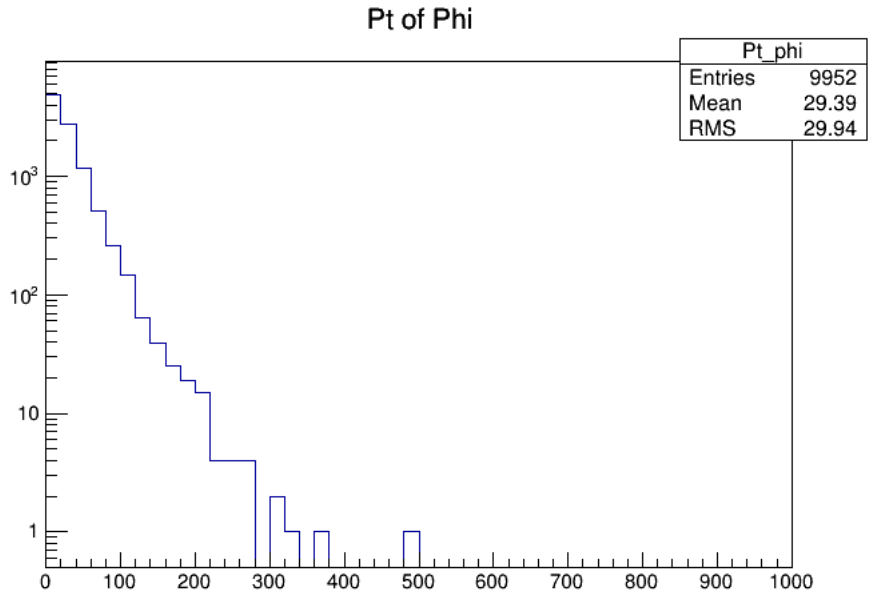


Fig 5: Histogram showing the distribution of transverse momentum of ϕ

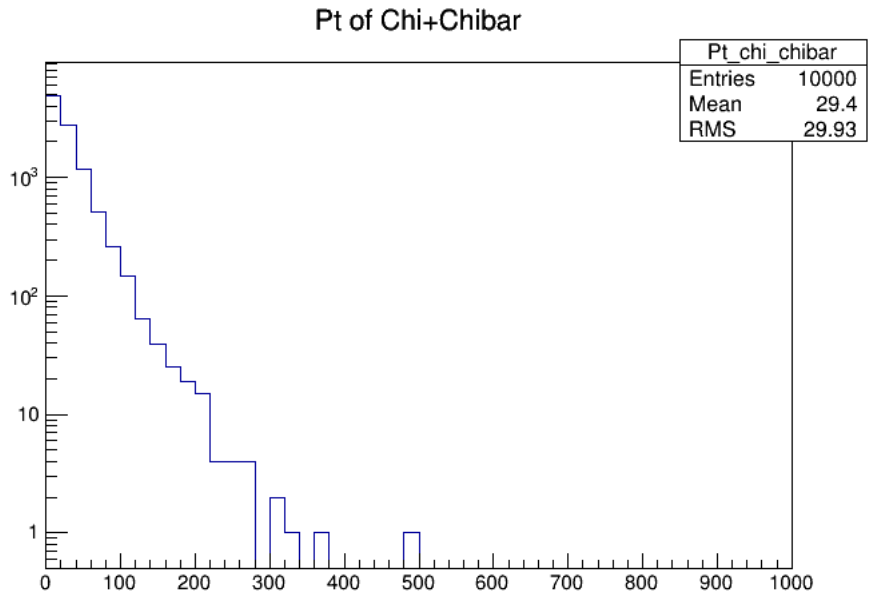


Fig 6: Histogram showing the distribution of net transverse momentum of χ and $\bar{\chi}$

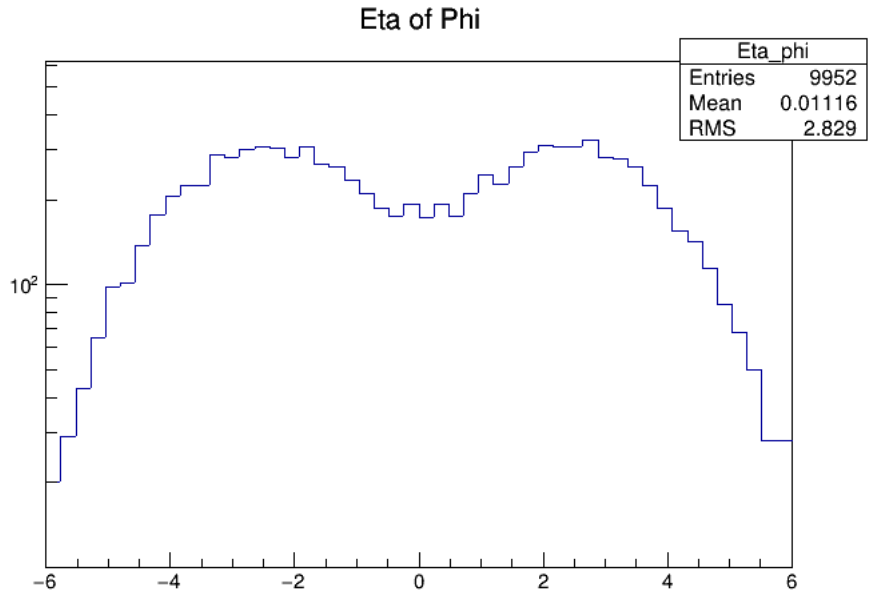


Fig 7: Histogram showing the distribution of η of ϕ

3.4 Conclusions from LHE Analysis Plots

From the plots obtained by LHE Analysis, we can conclude the following -

- The actual mass and the invariant mass of the mediator (ϕ) are indeed identical.
- The transverse momentum of the mediator (ϕ) is equal to the sum of the transverse momenta of the dark matter candidates (χ and $\bar{\chi}$) produced from the mediator.
- The mass distribution of the mediator has a peak at $100GeV$, which is the mass point we had chosen for ϕ.
- Two peaks of the η distribution of ϕ lies well within -4 and 4.

Chapter 4

Analysis with MadAnalysis

The MadAnalysis 5 program provides to the user a platform including a wide class of functionalities allowing one to perform sophisticated physics analyses. Even if the existing possibilities are rather large, implementing an analysis within the MadAnalysis 5 framework always follows the same steps, each of these steps being linked to one or several of the key features of the program. These key features[4] are briefly described in the following sections.

4.1 Sample Declaration

Sample declaration in MadAnalysis is done in terms of Datasets.

When implementing an analysis within the framework of MadAnalysis 5, the first task which is asked of the user is to indicate the Monte Carlo event files to be processed on run-time. In general, a full Monte Carlo event generation requires the simulation of several physical processes, such as the signal under consideration and the associated sources of background. Furthermore, for the processes with the highest cross sections, more than one event sample may have to be generated in order to get enough statistical significance. The user then requires these samples, describing the same physics, to be treated on the same footing. In MadAnalysis 5, this can be performed in a very natural way. The event files to be merged are gathered under the form of an object dubbed a dataset. When the analysis is effectively executed by the C++ core of the program, datasets, i.e., collections of event files, are processed rather than the event files individually.

4.2 The Content of the Events

Particles and multiparticles (along with the reconstructed objects) make up an event in MadAnalysis.

According to the conventions of the different event formats introduced above, the particles described in the events are defined, in an unambiguous way, through

their Particle Data Group identifier (PDG-id). Similarly, the algorithms generated by MadAnalysis 5 are widely based on this concept to distinguish the various particle species. The drawback is obviously that this identifier is most of the time non-intuitive for the user and can even become unreadable when the set of particles included in the events becomes large. For instance, the particle content of hadron-level events consists in 12 hundreds of different particles, each of them being identified by a different PDG-id. In the spirit of the MadGraph program, MadAnalysis 5 alleviates this issue by allowing us to associate a particle label to a given PDG-id. Hence, instead of representing an electron and a positron by the integer numbers 11 and -11, one can create the (more intuitive) labels e- and e+ and associate them to the PDG-ids 11 and -11, respectively. It is also possible to collect labels together through the concept of multiparticles. Hence, one could define a label e referring to both the electron and the positron. In order to implement an analysis in an efficient way, it is recommended to the user to define, in a first step, a series of particle and multiparticle labels facilitating the readability (and then the validation) of the analysis.

4.3 Definition of the Analysis

An analysis is defined in terms of several selection cuts. They are either control selection or region selection cuts.

Implementing the analysis is the core task of the user. It consists in defining the histograms that have to be generated and the selection cuts that need to be applied. These two types of objects, i.e. histograms and cuts, are uniquely dubbed selections. It can be noted that even if asking for the generation of several histograms is a commutative operation, the ordering of the selection cuts is in contrast important. Applying the cuts in a different order indeed leads to the production of different (intermediate) histograms and efficiency tables.

4.4 Running the Analysis

When an analysis is run, it is done in terms of several jobs.

After having created a series of dataset objects and defined the analysis to be performed, i.e. having implemented the histograms and selection cuts of interest, the user needs to execute the analysis on the datasets. Contrary to the previous steps which rely on the Python module of MadAnalysis 5, this task is built, for efficiency reasons, upon a C++ code which is generated by MadAnalysis 5 and denoted as a job. After MadAnalysis 5 creates the job, it has to be executed by the SampleAnalyzer kernel for each of the predefined datasets. Once the analysis has been performed, the results are subsequently imported in the Python module and re-interpreted by MadAnalysis 5.

4.5 Display of the Results

The result is displayed in terms of report in interactive mode.

The generated results can be collected and displayed in a synthetic report. This report is given either under the html format, as a webpage, or as a latex document that has to be further compiled. The report contains the histograms which have been generated and the efficiency tables related to the implemented selection cuts. Furthermore, a signal over background table is generated provided the datasets defined by the user have been tagged as signal and background samples.

However, in expert mode the result is generally saved or plotted using the help of different classes of the ROOT library.

Chapter 5

Recasting of bb+DM Analysis

For the production of DM at the LHC, simplified mediator models have been recently agreed upon by the ATLAS and CMS Collaborations to facilitate the comparison of the results and the subsequent combination of the search data. In such case a mediator, ϕ, between the Standard Model particles and the DM particles, χ, would be too heavy to be directly produced at the LHC, an Effective Field Theory description of its interaction would suffice. In case the mass of this mediator is in reach of the LHC, this production can be described by the production of $bb\phi$, with the subsequent decay $\phi \to \chi\bar{\chi}$.

As mediators of the scalar and pseudoscalar types are expected to have Yukawa-like couplings to the Standard Model quarks, the final state missing transverse energy (MET) with heavy flavor (HF) quarks represents an important signature to probe at hadronic colliders.

The strategy to search for this signature relies on the reconstruction of MET as well as the identification of jets originating from the fragmentation and hadronization of bottom quarks. The description of the MET observable is controlled by using dedicated control regions in data. In the $bb\phi$ three-body production process, one of the accompanying b quarks is often not reconstructed due to its small transverse momentum or large production angle. The dominant signature of the signal is then a large amount of E miss T with one b jet, owing to detector acceptance and minimum jet pT requirements. Besides from DM+bb production, this search is also sensitive to possible contributions from DM+tt production, where the b quarks are produced in top quark decays. In order to target also the possibility that the two b quarks from DM+bb or DM+tt are visible in the detector, this analysis includes a second category for the MET+bb jets final state. Due to the soft momenta of the two b quarks, an additional jet with large transverse momentum and not originating from heavy-flavor quarks, is allowed to retain events with initial state gluon radiation.

As mentioned earlier, MadAnalysis, in general, can be used in two modes -

- Interactive mode (using Python shell)
- Expert mode (writing C++ code)

Recasting a LHC analysis is done in expert mode. Each analysis is implemented as a class. MadAnalysis C++ API provides PHYSICS, SORTER etc. services and various classes to work easily with event objects. Complete ROOT support is automatically included. No additional header file or manual compiler configuration is needed.

5.1 Basic Structure of an Analysis

An analysis written using MadAnalysis expert mode has 3 member functions -

- bool cms_pp_bbdm::Initialize(const MA5::Configuration& cfg, const std::map<std::string, std::string>& parameters)
- void cms_pp_bbdm::Finalize(const SampleFormat& summary, const std::vector<SampleFormat>& files)
- bool cms_pp_bbdm::Execute(SampleFormat& sample, const EventFormat& event)

Initialize and Finalize are called only once, during the start and the end of the analysis run respectively.

Execute is called for each event. Most of the implementation is actually inside this function.

MadAnalysis automatically creates EventFormat objects from .lhe/.hep/.root file(s) and passes its reference to the Execute method while calling.

5.2 Signal Regions and Selection Cuts

To make an analysis easier to understand/implement, it is often divided into various Signal Regions.

The Signal Regions are generally disjoint sets and selected using event properties. Each Signal Region has its own set of plots.

To efficiently distribute events among Signal Regions, various bounds or comparison on event properties are needed. These rules are called Selection Cuts.

The sequence of the number of events surviving the incremental set of applied Selection Cuts is called Cutflow.

5.3 Preselection Plots

Before going into the details of CMS-B2G-15-007, using MadAnalysis interactive mode, several preselection plots are made to understand the sample, which also helps to validate the analysis.

The preselection plots are given below -

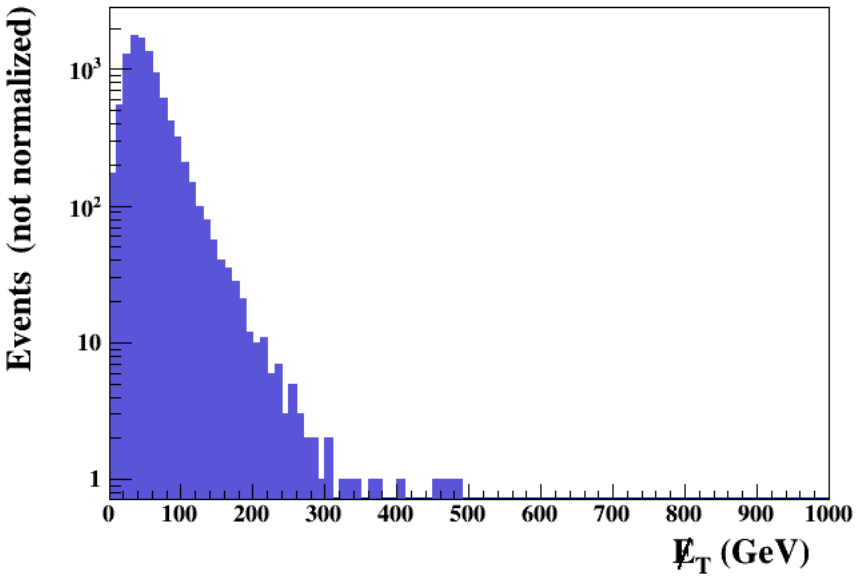


Fig 8: Histogram showing the MET distribution of the events

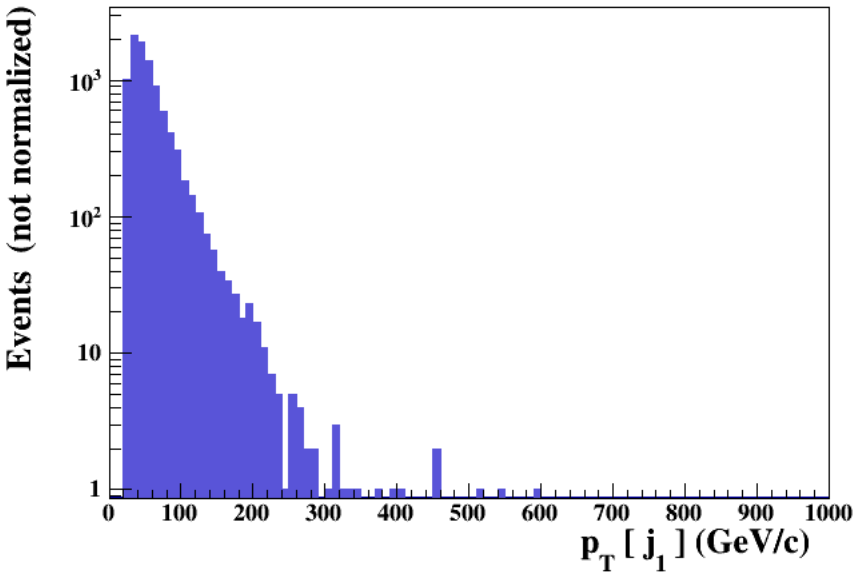


Fig 9: Histogram showing the distribution of the transverse momentum of the leading jet

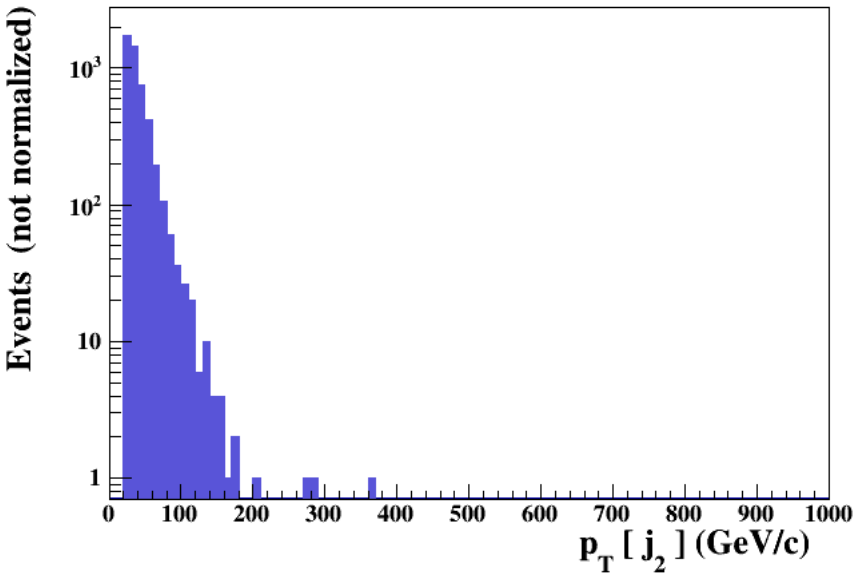


Fig 10: Histogram showing the distribution of the transverse momentum of the second leading jet

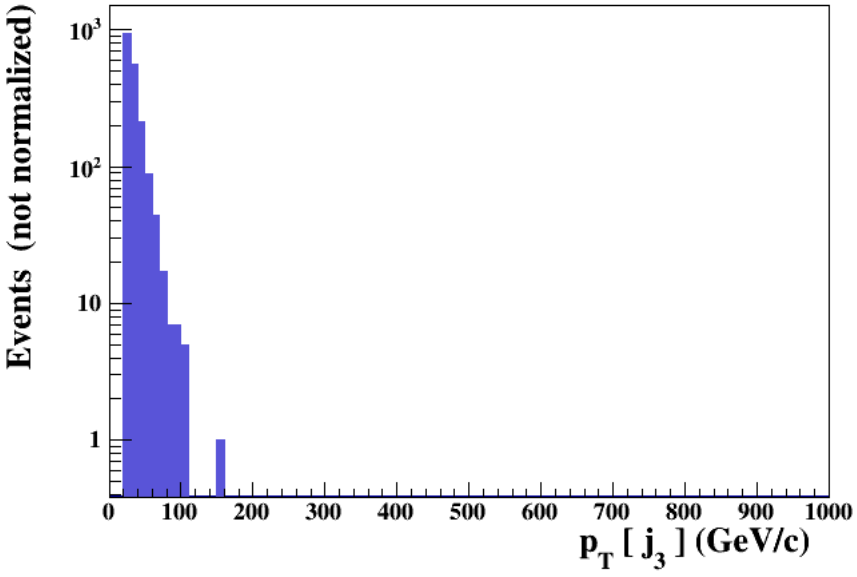


Fig 11: Histogram showing the distribution of the transverse momentum of the third jet

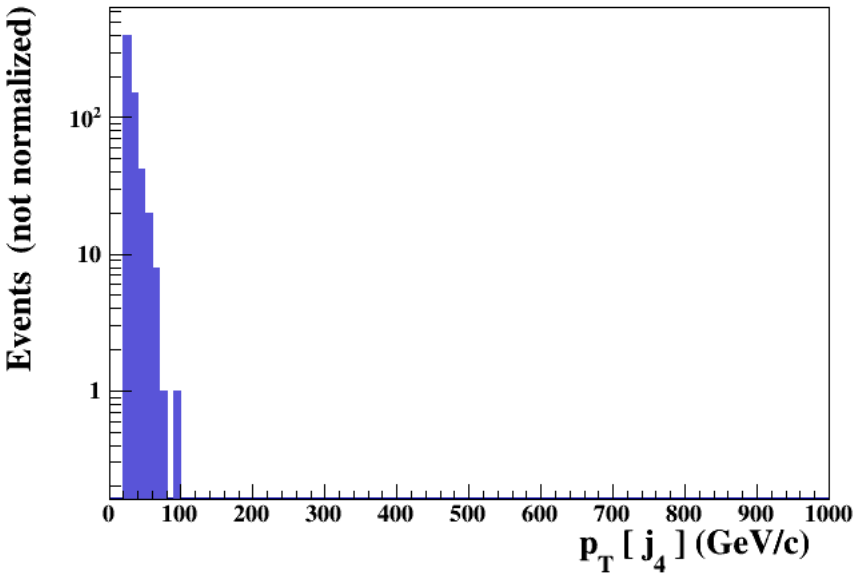


Fig 12: Histogram showing the distribution of the transverse momentum of the fourth jet

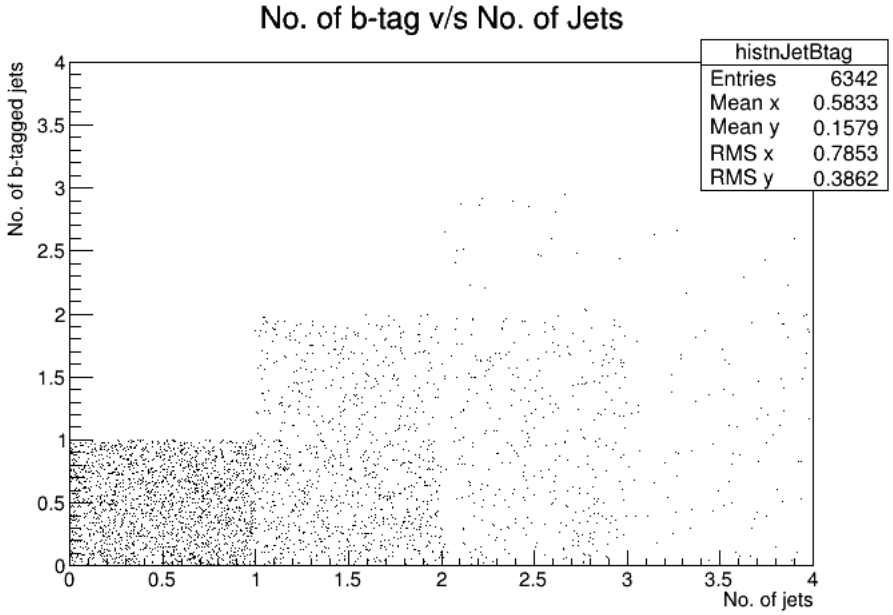


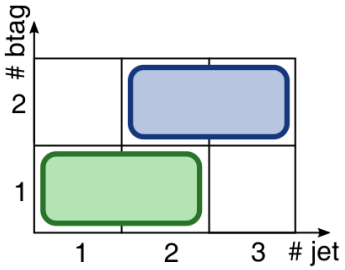
Fig 13: Scatter plot showing the relationship between total number of jets and number of b-tagged jets per event

5.4 Details of CMS-B2G-15-007

Events in the signal region are selected by MET triggers and are required to have an offline reconstructed MET $>$ 200 GeV.

Events that contain at least one reconstructed and identified isolated electron, muon, tau-lepton or photon candidates with pT $>$ 10 GeV (e and μ), pT $>$ 15 GeV (γ), pT $>$ 18 GeV (τ) are rejected.

Two exclusive categories, based on the number of b-tagged jets in the events, are defined as below -



In the single b-tagged category (SR1) one jet with pT $>$ 50 GeV and satisfying tight quality selections (charged hadron fraction $>$ 0.1, neutral hadron fraction $<$ 0.8) is required.

- An additional jet is allowed if its $pT > 30GeV$.
- Both jets have to fulfill the $\Delta\phi(jet, MET) > 0.5$ requirement.
- Exactly one of the two jets has to pass the b-tagging requirement.

The double b-tagged category (SR2) contains events with two jets with $pT > 50GeV$, or with an additional third jet with $pT > 30GeV$.

- The leading jet is required to pass the same quality criteria as in SR1.
- Exactly two of the three jets are required to be b-tagged.
- All the jets have to fulfill the $\Delta\phi(jet, MET) > 0.5$ requirement.

The double b-tagged category (SR2) is introduced to recover part of the efficiency of the DM production in association to top quarks, where two genuine, high-pT quarks are present.

These details are summarized in the following table -

	SR 1	SR 2
MET	> 200 GeV	
Jet 1	pT > 50 GeV Δφ(jet,MET) > 0.5 charged hadron fraction > 0.1, neutral hadron fraction < 0.8	
Jet 2	only if Δφ(jet,MET) > 0.5 pT > 30 GeV	Δφ(jet,MET) > 0.5 pT > 50 GeV
Jet 3	veto	only if Δφ(jet,MET) > 0.5 pT > 30 GeV
b-tag	1 CSV medium WP	2 CSV medium WP
leptons **τ** **γ**	veto veto veto	

Fig 14: Selection cuts for the signal regions

5.5 The Recasting Code

Various parts of the recasting code is listed below according to their purposes. Most of this code is part of the Analyze() function. The complete program listing for cms_pp_bbdm.h and cms_pp_bbdm.cpp is given in Appendix III.

5.5.1 Keeping Track of Event Weights

```
double myEventWeight;

if (Configuration().IsNoEventWeight())
        myEventWeight = 1;
else if (event.mc()->weight() != 0)
        myEventWeight = event.mc()->weight();
else {
        WARNING << "Found one event with a zero weight.
            Skipping..." << endmsg;
        return false;
}

Manager()->InitializeForNewEvent(myEventWeight);
vector<const RecJetFormat*> SignalJets;
TLorentzVector pTmiss = event.rec()->MET().momentum();
double MET = pTmiss.Pt();
```

5.5.2 Find the Jets

```
size_t noOfJets = event.rec()->jets().size();
```

```
for (size_t i = 0; i < noOfJets; i++) {
        const RecJetFormat * CurrentJet = &(event.rec()->
            jets()[i]);
        SignalJets.push_back(CurrentJet);
}
```

5.5.3 Find Event for Any Electron with pt>10

```
bool noElectron = true;

for (size_t i = 0; i < event.rec()->electrons().size(); i
    ++) {
        const RecLeptonFormat * CurrentElectron = &(event
            .rec()->electrons()[i]);
        if (CurrentElectron->pt() > 10.0 &&
            CurrentElectron->isolated())
                noElectron = false;
}
```

- Similar code is added for finding isolated muon, tauon and photon according the the conditions discussed above.

5.5.4 Applying Common Cuts

```
SORTER->sort(SignalJets, PTordering);

// Cut: MET > 200
if (MET <= 200)
        return false;

// Cut: No of Jets <= 3
if (noOfJets > 3)
        return false;

// Cut: At least 1 jet
if (noOfJets < 1)
        return false;

// Cut: 1st Jet has pt > 50
if (SignalJets[0]->pt() <= 50)
        return false;

// Cut: 1st Jet has delta_phi with MET > 0.5
if (SignalJets[0]->dphi_0_pi(pTmiss) <= 0.5)
        return false;
```

5.5.5 Signal Region 1 Conditions

```
bool SR1 = true;

if (noOfJets == 3)
        SR1 = false;

if (noOfJets == 2) {
        if (SignalJets[1]->pt() <= 30 || SignalJets[1]->
            dphi_0_pi(pTmiss) <= 0.5)
                SR1 = false;
        if (SignalJets[0]->btag() && SignalJets[1]->btag
            ())
                SR1 = false;
}

if (noOfJets == 1)
        if (!SignalJets[0]->btag())
                SR1 = false;
```

5.5.6 Signal Region 2 Conditions

```
bool SR2 = true;

if (noOfJets < 2)
        SR2 = false;

if (noOfJets == 2) {
        if (SignalJets[1]->pt() <= 50 || SignalJets[1]->
            dphi_0_pi(pTmiss) <= 0.5)
                SR1 = false;
        if (!(SignalJets[0]->btag() && SignalJets[1]->
            btag()))
                SR2 = false;
}

if (noOfJets == 3) {
        if (SignalJets[2]->pt() <= 30 || SignalJets[2]->
            dphi_0_pi(pTmiss) <= 0.5)
                SR1 = false;
        size_t noBtaggedJets = 0;
        for (size_t i = 0; i < 3; i++)
                if (SignalJets[i]->btag())
                        noBtaggedJets++;
        if (noOfJets != 2)
                SR2 = false;
```

}

5.5.7 Filling Histograms

```
// Cut: No lepton or photon
if (!(noElectron && noMuon && noTau && noPhoton))
        return false;

// Fill the histograms
if (SR1)
        MET_SR1->Fill(MET, myEventWeight);
if (SR2)
        MET_SR2->Fill(MET, myEventWeight);

return (SR1 || SR2);
```

The code is compiled and built using the Makefile in the Build directory of the analysis using the following commands -

```
$ source setup.sh
$ make
```

This creates an executable called MadAnalysisJob, which is run like this -

```
$ ./MadAnalysisJob ../Input/input.txt
```

The input.txt file contains the path to the .root file produced by Delphes.

Chapter 6

Conclusion

MET plots for the SR1 is shown below. The first one is obtained in the original CMS experiment. The second one is created by running the MadAnalysis recast code.

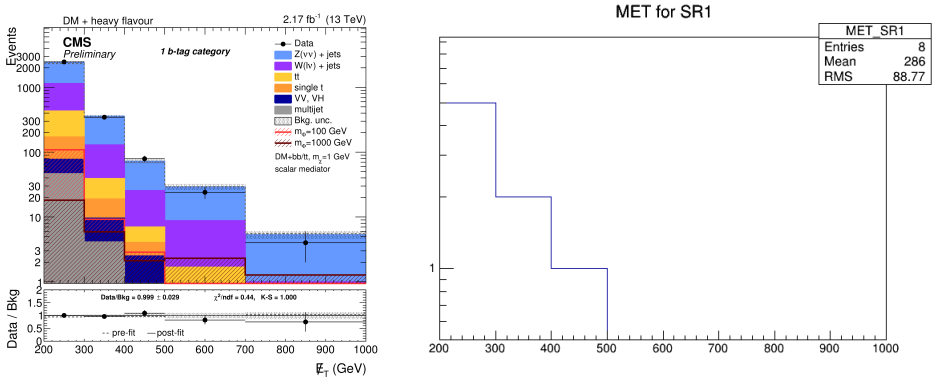


Fig 15: The experimental (left) and the recasted (right) plots for the MET distributions in SR 1

The solid red line in the left figure corresponds to the chosen mass point, i.e. $m_\chi = 1GeV$ and $m_\phi = 100GeV$.

Unfortunately, due to limited number of events in the generated sample, there are only 8 entries in the recasted MET plot for SR1. Also because of the same reason, no meaningful recasted plot was obtained for SR2.

According to the CADI line of this analysis[5], the cross section for our selected mas point is -

$$\sigma_{NLO} = 1.263 \pm 5.970 \times 10^{-3} pb$$

So, out integrated luminosity will be -

$$L = \frac{N_{events}}{\sigma_{NLO}} = 2.92 \times 10^3 pb^{-1} = 2.92 fb^{-1}$$

The integrated luminosity of the of the LHC experiment dataset is $2.17 fb^{-1}$. So, our scaling factor will be $\frac{2.17}{2.92} = 0.743$.

This means that the number of events obtained in the each of the signal regions has to be scaled down by a factor of 0.743, which further reduces the number of entries in the histogram.

Acknowledgement

I am really grateful to my MS thesis advisor Dr. Jyothsna Komaragiri to let me work on the recasting project. She helped me in every possible way while performing the recasting of CMG-B2G-15-007 analysis. I would also like to thank Siew Yan from University of Malaya, who was involved in the original CMS analysis and helped me by providing valuable inputs regarding the sample generation process.

Bibliography

[1] Search for Dark Matter produced in association with bottom quarks - http://cms-results.web.cern.ch/cms-results/public-results/preliminaryresults/B2G-15-007/index.html

[2] MadGraph5_aMC@NLO tutorial - https://indico.cern.ch/event/555228/

[3] A Hitchhiker's guide to High Energy Physics - http://www.phy.duke.edu/~dmb60/the-guide/

[4] An Introduction to MadAnalysis5 and Recasting LHC Results - https://madanalysis.irmp.ucl.ac.be/wiki/Ma5PadStepByStep

[5] CADI line for CMS-B2G-15-007 Analysis (restricted) - http://cms.cern.ch/iCMS/analysisadmin/cadilines?line=B2G-15-007

Appendix I

DM_settings.cmnd

```
! main89mlm.cmnd (modified)

! Specify statistics parameters.
Main:numberOfEvents        = -1 ! number of events
    generated
! Tell Pythia that LHEF input is used
Beams:frameType            = 4

! Use same PDFs and alpha_s as in ME calculation.
!PDF:pSet                  = LHAPDF5:CT10.LHgrid
!SpaceShower:alphaSvalue   = 0.118
!TimeShower:alphaSvalue    = 0.118

! Specify jet matching parameters for MLM
! Note: Some of these settings might be read directly
    from the input LHE file.
JetMatching:merge          = on
JetMatching:scheme         = 1
JetMatching:qCut           = 30.0
JetMatching:nJetMax        = 4
! Be more forgiving with momentum mismatches.
Check:epTolErr             = 1e-2

! Subruns for MLM jet matching
LHEFInputs:nSubruns        = 1
Main:subrun                = 0
9100000:new                = MED MED 3 0 0 100 0 0 0 99999
9100022:new                = DM DM 2 0 0 1 0 0 0 99999
9100022:mayDecay           = off
Tune:ee                    = 3
Tune:pp                    = 5
Beams:LHEF                 = unweighted_events.lhe
```

Appendix II

LHA_Analysis.py

```
from ROOT import TCanvas, TH1F, TLorentzVector

f = open("unweighted_events.lhe")
line=""
events=[]
currEvent=""
addLine=False
for line in f:
    if line.strip()=='<event>':
        currEvent=""
        addLine=True
    elif line.strip().startswith("<scales"):
        events.append(currEvent.strip())
        addLine=False
    elif addLine:
        currEvent+=line.strip()+'\n'
f.close()

print(len(events))

mass_phi=[]
pt_phi=[]
eta_phi=[]
invm_phi=[]
pt_chi_chibar=[]
for event in events:
    phi=[s for s in event.split('\n') if s.split()[0]=="
        9100000"]
    chi=[s for s in event.split('\n') if s.split()[0]=="
        9100022"]
    chibar=[s for s in event.split('\n') if s.split()
        [0]=="-9100022"]
```

```
    if phi:
        mass_phi.append(float(phi[0].split()[10]))
        px=float(phi[0].split()[6])
        py=float(phi[0].split()[7])
        pz=float(phi[0].split()[8])
        e=float(phi[0].split()[9])
        p=TLorentzVector(px,py,pz,e)
        pt_phi.append(p.Pt())
        eta_phi.append(p.Eta())

    px1=float(chi[0].split()[6])
    py1=float(chi[0].split()[7])
    pz1=float(chi[0].split()[8])
    e1=float(chi[0].split()[9])
    px2=float(chibar[0].split()[6])
    py2=float(chibar[0].split()[7])
    pz2=float(chibar[0].split()[8])
    e2=float(chibar[0].split()[9])
    p1=TLorentzVector(px1,py1,pz1,e1)
    p2=TLorentzVector(px2,py2,pz2,e2)
    p=p1+p2
    invm_phi.append(p.M())
    pt_chi_chibar.append(p.Pt())

hist_mass_phi=TH1F("Mass_phi","Mass of Phi",50,0,200)
for i in mass_phi:
    hist_mass_phi.Fill(i)

hist_pt_phi=TH1F("Pt_phi","Pt of Phi",50,0,1000)
for i in pt_phi:
    hist_pt_phi.Fill(i)

hist_eta_phi=TH1F("Eta_phi","Eta of Phi",50,-6,6)
for i in eta_phi:
    hist_eta_phi.Fill(i)

hist_invm_phi=TH1F("InvM_phi","Invariant Mass of Phi"
    ,50,0,200)
for i in invm_phi:
    hist_invm_phi.Fill(i)

hist_pt_chi_chibar=TH1F("Pt_chi_chibar","Pt of Chi+Chibar
    ",50,0,1000)
for i in pt_chi_chibar:
    hist_pt_chi_chibar.Fill(i)
```

```
c=TCanvas()
c.SetLogy()
hist_mass_phi.Draw()
c.SaveAs("mass_phi.png")
hist_pt_phi.Draw()
c.SaveAs("pt_phi.png")
hist_eta_phi.Draw()
c.SaveAs("eta_phi.png")
hist_invm_phi.Draw()
c.SaveAs("invm_phi.png")
hist_pt_chi_chibar.Draw()
c.SaveAs("pt_chi_chibar.png")
```

Appendix III

cms_pp_bbdm.h

```
#ifndef analysis_cms_pp_bbdm_h
#define analysis_cms_pp_bbdm_h

#include "SampleAnalyzer/Process/Analyzer/AnalyzerBase.h"


namespace MA5 {

class cms_pp_bbdm: public AnalyzerBase {

INIT_ANALYSIS(cms_pp_bbdm,"cms_pp_bbdm")

public:
        virtual bool Initialize(const MA5::Configuration&
            cfg, const std::map<std::string, std::string
           >& parameters);
        virtual void Finalize(const SampleFormat& summary
            , const std::vector<SampleFormat>& files);
        virtual bool Execute(SampleFormat& sample, const
            EventFormat& event);

private:
        TH1F *MET_SR1;
        TH1F *MET_SR2;

};

}

#endif
```

cms_pp_bbdm.cpp

```
#include "SampleAnalyzer/User/Analyzer/cms_pp_bbdm.h"

using namespace MA5;
using namespace std;

//
   ------------------------------------------------------------------------------

// Initialize
// function called one time at the beginning of the
    analysis
//
   ------------------------------------------------------------------------------

bool cms_pp_bbdm::Initialize(const MA5::Configuration&
    cfg, const std::map<std::string, std::string>&
    parameters) {
        // Information on Analysis
        INFO<<"Analysis: pp -> bb~ + DM"<<endmsg;

        cout << "BEGIN Initialization" << endl;
        // Declaring all signal regions

        // Declaring preselection cuts

        // Other cuts

        MET_SR1=new TH1F("MET_SR1","MET for SR1"
            ,8,200,1000);
        MET_SR2=new TH1F("MET_SR2","MET for SR2"
            ,8,200,1000);
        cout << "END Initialization" << endl;
        return true;
}

//
   ------------------------------------------------------------------------------

// Finalize
// function called one time at the end of the analysis
//
   ------------------------------------------------------------------------------
```

```
void cms_pp_bbdm::Finalize(const SampleFormat& summary,
    const std::vector<SampleFormat>& files) {
        cout << "BEGIN Finalization" << endl;

        TCanvas c;
        c.SetLogy();
        MET_SR1->Draw();
        c.SaveAs("MET_SR1.png");
        MET_SR2->Draw();
        c.SaveAs("MET_SR2.png");
        cout << "END Finalization" << endl;
}

//
    ------------------------------------------------------------------------

// Execute
// function called each time one event is read
//
    ------------------------------------------------------------------------

bool cms_pp_bbdm::Execute(SampleFormat& sample, const
    EventFormat& event) {

        // Keeping track of event weights.
        double myEventWeight;
        if (Configuration().IsNoEventWeight())
                myEventWeight = 1;
        else if (event.mc()->weight() != 0)
                myEventWeight = event.mc()->weight();
        else {
                WARNING<< "Found one event with a zero
                    weight. Skipping..." << endmsg;
                return false;
        }
        Manager()->InitializeForNewEvent(myEventWeight);

        vector<const RecJetFormat*> SignalJets;

        TLorentzVector pTmiss = event.rec()->MET().
            momentum();
        double MET = pTmiss.Pt();

        // find the jets
        size_t noOfJets = event.rec()->jets().size();
        for (size_t i = 0; i < noOfJets; i++) {
```

```
        const RecJetFormat * CurrentJet = &(event
            .rec()->jets()[i]);
        SignalJets.push_back(CurrentJet);
}

// find event for any electron with pt>10
bool noElectron = true;
for (size_t i = 0; i < event.rec()->electrons().
    size(); i++) {
        const RecLeptonFormat * CurrentElectron =
            &(event.rec()->electrons()[i]);
        if (CurrentElectron->pt() > 10.0 &&
            CurrentElectron->isolated())
                noElectron = false;
}

// find event for any muon with pT>10
bool noMuon = true;
for (size_t i = 0; i < event.rec()->muons().size
    (); i++) {
        const RecLeptonFormat * CurrentMuon = &(
            event.rec()->muons()[i]);
        if (CurrentMuon->pt() > 10 && CurrentMuon
            ->isolated())
                noMuon = false;
}

// find event for any tau with pT>18
bool noTau = true;
for (size_t i = 0; i < event.rec()->taus().size()
    ; i++) {
        const RecTauFormat * CurrentTau = &(event
            .rec()->taus()[i]);
        if (CurrentTau->pt() > 18 && CurrentTau->
            isolated())
                noTau = false;
}

// find event for any photon with pT>15
bool noPhoton = true;
for (size_t i = 0; i < event.rec()->photons().
    size(); i++) {
        const RecPhotonFormat * CurrentPhoton =
            &(event.rec()->photons()[i]);
        if (CurrentPhoton->pt() > 15 &&
            CurrentPhoton->isolated())
```

```
                noPhoton = false;
}

SORTER->sort(SignalJets, PTordering);

// Applying the Cuts

// Cut: MET > 200
if (MET <= 200)
        return false;
// Cut: No of Jets <= 3
if (noOfJets > 3)
        return false;
// Cut: At least 1 jet
if (noOfJets < 1)
        return false;
// Cut: 1st Jet has pt > 50
if (SignalJets[0]->pt() <= 50)
        return false;
// Cut: 1st Jet has delta_phi with MET > 0.5
if (SignalJets[0]->dphi_0_pi(pTmiss) <= 0.5)
        return false;

// Check if the event satisfies SR1 conditions
bool SR1 = true;
if (noOfJets == 3)
        SR1 = false;
if (noOfJets == 2) {
        if (SignalJets[1]->pt() <= 30 ||
            SignalJets[1]->dphi_0_pi(pTmiss) <=
            0.5)
                SR1 = false;
        if (SignalJets[0]->btag() && SignalJets
            [1]->btag())
                SR1 = false;
}
if (noOfJets == 1)
        if (!SignalJets[0]->btag())
                SR1 = false;

// Check if the event satisfies SR2 conditions
bool SR2 = true;
if (noOfJets < 2)
        SR2 = false;
if (noOfJets == 2) {
        if (SignalJets[1]->pt() <= 50 ||
```

```
                SignalJets[1]->dphi_0_pi(pTmiss) <=
                0.5)
                    SR1 = false;
            if (!(SignalJets[0]->btag() && SignalJets
                [1]->btag()))
                    SR2 = false;
        }
        if (noOfJets == 3) {
            if (SignalJets[2]->pt() <= 30 ||
                SignalJets[2]->dphi_0_pi(pTmiss) <=
                0.5)
                    SR1 = false;
            size_t noBtaggedJets = 0;
            for (size_t i = 0; i < 3; i++)
                if (SignalJets[i]->btag())
                    noBtaggedJets++;
            if (noOfJets != 2)
                SR2 = false;
        }

        // Cut: No lepton or photon
        if (!(noElectron && noMuon && noTau && noPhoton))
            return false;

        // Fill the histograms
        if (SR1)
            MET_SR1->Fill(MET, myEventWeight);
        if (SR2)
            MET_SR2->Fill(MET, myEventWeight);

        return (SR1 && SR2);
}
```