

Faculté des sciences

A machine learning driven approach to the MLM merging algorithm

Presented in order to obtain the Degree

Master [120] en sciences physiques, finalité approfondie

Author : William Puget

Thesis supervisors : Fabio Maltoni, Olivier Mattelaer

Readers : Vincent Lemaitre

Academic Year 2023-2024

Acknowledgements

This work would not have been possible without the availability and time of Olivier Mattelaer. The insights and guidance he provided at our weekly meetings were the necessary ingredients for this master thesis to come together.

I would also like to thank my co-supervisor, Fabio Maltoni and my reader Vincent Lemaitre, for the time they dedicated to this work.

Finally, I would also like to thank my family and my friends, known by the insiders as the *EDIUS*, for their continuous support during my academic curriculum.

Contents

1	Introduction	4
2	Overview of the simulation of a complete event	5
2.1	General Principle	5
2.2	Matrix Elements and event generation	6
2.3	Parton Shower	7
2.3.1	Splitting Functions	8
2.3.2	Sudakov factors	8
2.3.3	Unitarity	9
2.3.4	Veto algorithm	10
2.4	Merging	11
2.4.1	MLM	13
2.4.2	CKKW	14
2.4.3	MLM vs CKKW	14
3	A classifier to perform the veto	15
3.1	Simulation Setup and data handling	15
3.2	Selection of variables	15
3.3	Sample selection	16
3.4	Neural Network structure	16
3.5	Results	17
4	Parton Shower implementation	19
4.1	Limitations	20
5	CKKW using Sudakovs from the MLM fit	22
5.1	Limitations	22
5.2	Overview	23
5.3	Results	23
6	Replacement of the MLM veto by a reweighting	24
6.1	Limitations	26
7	Conclusion	27
8	Annex	28
8.1	Reproduce the plots	28

List of Figures

2.1	Overview of an event	6
2.2	Double counting	12
3.1	Acceptance distribution of the 0 jet sample	16
3.2	Machine learning fit	17
3.3	Acceptance distribution of the 1 jet sample	18
4.1	Distribution of momenta in a splitting	20
4.2	Results of a shower	21
6.1	Machine learning fit	25
6.2	Result from our reweighting	25
6.3	Second result from our reweighting	26

Chapter 1

Introduction

One of the most used tools to perform jet merging at LO is the MLM algorithm. Despite its strengths, the veto it imposes on a significant part of the fully generated events implies that we waste a non-negligible amount of computation. The purpose of this work is to find a possible approach to reduce the amount of wasted computation.

The initial approach, to accomplishing this, was to use a neural network classifier halfway through the simulation instead of selecting the event at the very end, which is currently the situation. To be more precise, we intended to use only the results of the Matrix Element computation to veto event before running the parton shower, while still ensuring that we did not veto events that MLM would have kept. As will be discussed later, this approach was unsuccessful because of the lack of classifying power accessible at this stage in the simulation.

We decided to avoid this problem of event rejection altogether by using the CKKW algorithm, which has a more invasive approach on each event but does not reject any. One of the necessary inputs of CKKW is the Sudakov suppression, this suppression is classically computed analytically (to LLO). We then identified the MLM veto probability as a Sudakov. So we modified our neural network to estimate this rejection probability based only on the output of the ME. We later understood that this rejection probability encompassed more than just a Sudakov which also doomed this second approach. The considerations for this approach are mostly irrelevant for our final result, but, nevertheless, we kept them in this work as a connection to what was presented at the pre-defence of this thesis.

This leads us to our final methodology, simply reweighting each event with a fit of its acceptance probability by MLM as an alternative to actually applying the MLM algorithm.

This work will start by reviewing how numerical simulation of high-energy particle collisions works. We will especially focus on the parton shower and, of course, jet merging. We will detail our first approaches and why they were unsuccessful. This will lead us to our final procedure, which comes with some results that can be seen as proof of concept with some limitations.

Chapter 2

Overview of the simulation of a complete event

2.1 General Principle

The goal of high-energy numerical simulation is to predict what will be observed inside the detector of a particle collider from a given model. So not only are we interested in the cross-section of a given process, say $gg \rightarrow t\bar{t}$, but also in the decay chain of the $t\bar{t}$ and in all radiation emitted during the event.

Given that we can factorize this computation depending on the energy scale at each step in the event, we can split our event into multiple computation stages that will each use a specific tool, this is illustrated in Fig. 2.1.

These different steps are not only possible, but are necessary because of the limitations of each scheme employed.

The first step is the event generation using a matrix element (ME) computation. This step will compute the cross-section of the hard process using the full interference (typically $2 \rightarrow 2$, $2 \rightarrow 6$, ... processes). It will use these results to generate the core of the event, i.e., it will give a set of particles in the final state with momenta and a parton history that led to this state. The ME can't simulate the whole event as the computational cost scales really fast with the number of particle in the final state. As such, $2 \rightarrow 8$ processes are generally the limit, and in a typical pp collision at LHC energy, we can expect to have in the 100's of partons in the final state. Secondly, the ME diverges in the soft and collinear limit, for energy far lower than $\sqrt{s}$ but still greater than 1GeV , we need to have another tool.

For these reasons, the second part of the event will be simulated using a parton shower (PS). This shower allows us to split recursively the partons generated by the ME to a low cutoff (1GeV) where QCD becomes non-perturbative and these partons start to rehadronise. The PS is a resummation of collinear approximations, and hence fill the limitation of the ME computation. It is implemented using a Markov chain, so the computational cost per radiation is quite low.

This rehadronisation is once again simulated using a different set of tools. This part of the simulation is performed after everything that will be discussed in this work, we will thus not detail it.

These three steps will give us a whole event, which means a collection of particles in the final and initial states and also in every intermediate state, which gives us a history of every interaction that took place. In this form, events are ready to be

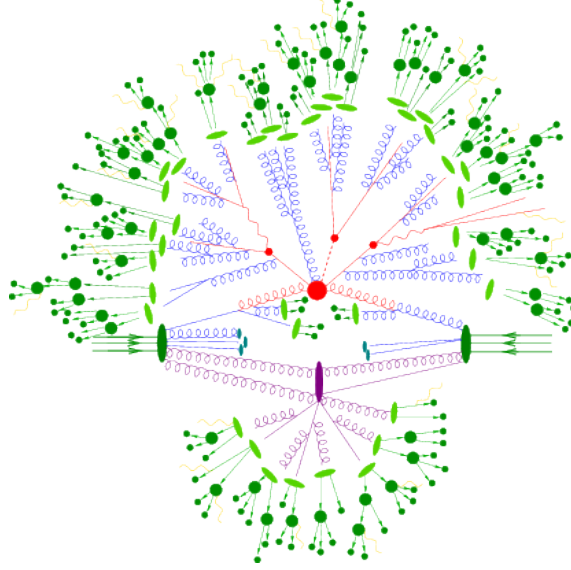


Figure 2.1: This figure[1] shows the different parts of an collider event. **Hard Process**, **Parton Shower**, **Hadronisation**, **Hadron Decays and scattering** These different parts are separated by the energies involved, the hard part take a significant part of the $\hat{s}$ and the Hadronisation only start under around 1Gev . For completeness, we can also see **secondary interactions**.

treated in software used to simulate the behavior of detectors and, ultimately, be compared to experimental data.

It's important to add here that this interaction history is not physical, it would be impossible the build something similar for experimental data as some intermediate states would be in superposition. We are meant to aggregate a large number of those individual events to produce physically relevant results.

2.2 Matrix Elements and event generation

The goal here, is to compute the inclusive cross-section of the hard process. By inclusive we mean that the final state can still decay and radiate additional emission. The master formula (for proton-proton collision) is as follow:

$$\sigma_{inc} = \sum_{a,b} \int dx_1 dx_2 d\Phi_{FS} f_a(x_1, \mu_F) f_b(x_2, \mu_F) \hat{\sigma}_{ab \rightarrow X}(\hat{s}, \mu_F, \mu_R)$$

There is three parts to this formula :

- The phase space intgral $[\int \Phi_{FS}]$
- The parton density functions (PDF) $f_a(x_1, \mu_F)$ and $f_b(x_2, \mu_F)$. Depending on x_1, x_2 the fraction of energy taken by each parton from their respective proton.
- Parton-level cross-section, $\hat{\sigma}_{ab \rightarrow X}(\hat{s}, \mu_F, \mu_R)$ which depend on the energy in the center of mass $\hat{s}$.

Multiple tterms also depend on μ_F , the factorisation scale and μ_R the renormalisation scale.

The parton-level cross-section

This will be the cross-section of the hard part of the event ($gg \rightarrow t\bar{t}$ as an example).

The computation is as usually a serie in perturbation theory :

$$\hat{\sigma}_{incl} = \sigma^{Born} \left(1 + \frac{\alpha_s}{2\pi} \sigma^{(1)} + \left(\frac{\alpha_s}{2\pi} \right)^2 \sigma^{(2)} + \dots \right)$$

If we only compute the σ^{Born} term, we speak about Leading order computation (LO) if we add the term in $\sigma^{(1)}$ we speak about next leading order (NLO) and so on for later terms (NNLO, NNNLO,...). Each order will be a correction to the previous one and will reduce the dependency on the factorization and renormalization scales, which are not physical. But higher orders will be more costly to compute. So, depending on the computational resources available and the necessary level of precision needed for an analysis, a given order can be chosen.

For the rest of this work, will be restricted to LO.

Parton desity functions

These functions give the probability of having a given parton (gluon, each type of quark) at a given energy from a proton actually participating in the main interaction. There is no analytical derivation for this probability, it is a fit made on experimental data.

The phase space integration

This integration is done numerically because it is generally faster, and there is no guarantee of an analytical solution, especially for non-SM theory. The phase space is of high dimensionality ($\sim 3n$ with n the number of particles in the initial and final states). This high dimensionality makes Monte Carlo integration the most efficient integration technique. The exact details of the integration are outside the scope of this work.

With this, we can generate a set of hard events, to reduce the scale of each of these events to the rehadronisation scale, we have to give them to the PS.

2.3 Parton Shower

With our event generator, we computed an inclusive cross-section because it is only for the production of the final state X from the initial states a and b. It does not constrain what may happen beyond X, we could, and for an event at LHC, will, see any number of particles emerge from X, and this final state could be in multiple kinematic configurations. The role of the parton shower will be to create, from this inclusive cross-section, an exclusive one. The PS will thus be a Markov chain that must conserve both the probability (unitarity) and the total momentum of the X state. It will accomplish this by splitting recursively the partons until it reaches a low energy cutoff (Q_0), where the hadronisation takes place.

A parton shower implementation contain 4 key ingredients, the splitting functions, the no-emission probability, the veto algorithm, and the splitting kinematics. The splitting function contains our physical information, they are derivated in the

soft and collinear approximations. They will encode the probability to split a parton and will depend on the fraction of energy (z) taken by the splitter. This, alone, would not make our new configuration exclusive, we need a no-emission probability to restrict further splitting. From these probabilities, the PS must choose a value of z at which the splitting occurs (there is also a probability that no splitting occurs), this is done using the veto algorithm. Having determined the energy of both children (z and $1 - z$), we can then construct their kinematics.

2.3.1 Splitting Functions

The splitting functions simply gives us the probability for a splitting to occur to one parton at a given value of the fraction of energy taken by the splitter. The four splitting function are

$$\begin{aligned} P_{q \rightarrow qg}(z) &= C_F \frac{1+z^2}{1-z} \\ P_{q \rightarrow gq}(z) &= C_F \frac{1+(1-z)^2}{z} \\ P_{g \rightarrow qq}(z) &= T_R \left[z^2 + (1-z)^2 \right] \\ P_{g \rightarrow gg}(z) &= C_A \left[\frac{z}{1-z} + \frac{1-z}{z} + z(1-z) \right] \end{aligned}$$

A formal derivation of theses functions can be found in Chap. 17 of the Peskin [2]. These splitting functions allow us to define the cross section for a $n + 1$ process based on the cross section for n , we can give as example the a $q\bar{q}g$:

$$d\sigma_{q\bar{q}g} \approx \sigma_{q\bar{q}} \sum_{partons} C_F \frac{\alpha_s}{2\pi} \frac{d\theta^2}{\theta^2} dz \frac{1+(1-z)^2}{z}$$

We can repalce the θ differnetial by any other parameter with $\theta^2 \propto \rho$. This ρ is called the evolution variable and as the name indicate, will be the parameter around wich our PS will evolve, this will become clear when we describe the veto algorithm. For our implementation, detailed later, we used $p_{\perp}^2 = z^2(1-z)^2\theta^2 E^2$ instead of the angle.

Now, we can compute the inclusive branching probability,

$$\mathcal{P}_{incl} = \int_{\rho_{min}}^{\rho_{max}} \frac{d\rho}{\rho} \frac{\alpha_s}{2\pi} \int_{z_{min}}^{z_{max}} dz P_{ij}(z)$$

2.3.2 Sudakov factors

We have established the probability of emitting a parton with $\rho \in [\rho_{min}, \rho_{max}]$ and $z \in [z_{min}, z_{max}]$. This is not sufficient, because this emission is not exclusive, our current model does not forbid the emission of another parton previously to this one. This is unacceptable as we would not maintain unitarity (which will be demonstrated just after this) and we could potentially generate an infinite cascade of partons. Fixing this requires us to have a probability of no-emission between two scales. We can define a differential inclusive probability : $d\mathcal{P}_{incl}(\rho) = \frac{d\rho}{\rho} \frac{\alpha_s}{2\pi} \int_{z_{min}}^{z_{max}} dz P_{ij}(z)$. By conservation of probability, we must have $\mathcal{P}_{no-em}(\rho) = 1 - \mathcal{P}_{incl}(\rho)$, we will show that

this linear relationship does not hold if we want a no-emission probability between two scales : $\mathcal{P}_{no-em}(\rho_{max} > \rho > \rho_{min})$.

$$\mathcal{P}_{no-em}(\rho_{max} > \rho > \rho_{min}) = \mathcal{P}_{no-em}(\rho_{max} > \rho > \rho_1) \cdot \mathcal{P}_{no-em}(\rho_1 > \rho > \rho_{min})$$

We can split this interval into an arbitrary large number of interval and get :

$$\lim_{n \rightarrow \infty} \prod_{i=0}^{n-1} \mathcal{P}_{no-em}(\rho_i > \rho > \rho_{i+1})$$

With the limit, the $(\rho_i > \rho > \rho_{i+1})$ is an infinitesimal range and we can reuse the linear relationship, we get :

$$\begin{aligned} \lim_{n \rightarrow \infty} \prod_{i=0}^{n-1} (1 - \mathcal{P}_{incl}(\rho_i > \rho > \rho_{i+1})) &= \lim_{n \rightarrow \infty} \exp\left(-\sum_{i=0}^{n-1} \mathcal{P}_{incl}(\rho_i > \rho > \rho_{i+1})\right) \\ &= \exp\left(-\int_{\rho_{min}}^{\rho_{max}} d\mathcal{P}_{incl}(\rho)\right) \end{aligned}$$

This no emission probability is referred to as the Sudakov form factor : $\mathcal{P}_{no-em}(\rho_1, \rho_2) := \Delta(\rho_1, \rho_2)$. Finally, we have the probability that a parton has its hardest splitting at ρ :

$$d\mathcal{P}_{hardest}(\rho) = d\mathcal{P}_{incl}(\rho) \cdot \Delta(\rho_{max}, \rho)$$

2.3.3 Unitarity

The unitarity of the PS means that the probability of having any number of splitting must be 1. If P_k is the probability to have k ordered splitting, then $\sum_{k=0}^{\infty} P_k = 1$. The differential probability to have 1 ordered splitting from a leg at a scale t_1 will simply be the differential probability to emit a parton at t_1 multiplied by two Sudakov, which will respectively forbid emission above t_1 and below t_1 :

$$dP_1(t_1) = \Delta(Q^2, t_1) dp(t_1) \Delta(t_1, Q_0^2)$$

We can then find the probability to have 2 ordered splitting at t_1 and t_2 :

$$dP_2(t_1, t_2) = \Delta(Q^2, t_1) dp(t_1) \Delta(t_1, t_2) dp(t_2) \Delta(t_2, Q_0^2) \mathcal{H}(t_1 - t_2)$$

The $\mathcal{H}$ is the Heaviside function that impose the order $t_1 > t_2$. We can then infer the probability of k splittings :

$$dP_k(t_1, \dots, t_k) = \Delta(Q^2, Q_0^2) \prod_{l=1}^k dp(t_l) \mathcal{H}(t_{l-1} - t_l)$$

$$P_k = \int dP_k(t_1, \dots, t_k) = \Delta(Q^2, Q_0^2) \frac{1}{k!} \left[\int_{Q_0^2}^{Q^2} dp(t) \right]^k$$

We can just sum over all k to get the total probability :

$$\sum_{k=0}^{\infty} P_k = \Delta(Q^2, Q_0^2) \sum_{k=0}^{\infty} \frac{1}{k!} \left[\int_{Q_0^2}^{Q^2} dp(t) \right]^k = \Delta(Q^2, Q_0^2) \exp\left[\int_{Q_0^2}^{Q^2} dp(t) \right]$$

Of course, the Sudakov : $\Delta(Q^2, Q_0^2) = \exp\left[-\int_{Q_0^2}^{Q^2} dp(t)\right]$, and we get that $\sum_{k=0}^{\infty} P_k = 1$. The Unitarity of the PS can only be guaranteed if we have Sudakovs.

2.3.4 Veto algorithm

With the splitting functions and the Sudakovs, we can determine the cross-section of a given event. But what we want is to generate the event in the first place. Thus, we must establish a procedure to find a value of z at which the splitting occurs. We use a proxy for z , which is our evolution variable that we call t here, by analogy with time dependent processes. For each parton we would look for a t which will be bound by the scale at which the parton appeared (t_{max}) and Q_0 , the low energy scale. We would then use the ratio t/Q_0 to determine z .

Say that the probability that a parton branches at t is described by $f(t)$, this would be our branching probability. This branching can only happen if no branching has happened before. This forces us to use a probability that the branching happens only at $t : P(t)$. We can thus establish an equation that depends on the probability that no branching happens until $t, N(t)$. This is quite similar to a radioactive disintegration :

$$\begin{aligned} P(t) &= -\frac{dN}{dt} = f(t)N(t) \\ &= f(t)N(0)\exp\left[-\int_0^t f(u)du\right] \end{aligned}$$

This is while assuming that ordering variable started at $t = 0$. If we know the inverse of the primitiv of $f(t)$ ($F(t)$), then solving for t is straightforward, Assuming $N(0) = 1$:

$$\int_0^t P(u)du = N(0) - N(t) = 1 - \exp\left(-\int_0^t f(u)du\right) = 1 - R$$

We get that $t = F^{-1}(F(0) - \ln R)$, with R a random number between 0 and 1. We could then just generate a random number and get our splitting.

In the contrary, if we do not know F^{-1} or it is too computationally expensive to find it, we would instead use the veto algorithm.

This algorithm is based on the use of an easier function $g(t)$ with $f(t) \leq g(t), \forall t \geq 0$. It proceeds as follows: first, a value of t is generated as $t_i = G^{-1}(G(t_{i-1}) - \ln R)$. We know that $g(t) \geq f(t) \geq 0$, it means that $G(t)$ is growing and $t_i \geq t_{i-1}$. We then verify if $f(t_i)/g(t_i) \geq R$, if true we keep this t_i , if not, we pick a t_{i+1} . We can show that this algorithm produce the desired result by looking at the probability of branching after n iterations, $P_n(t)$.

At the first iteration, no t_i has yet been rejected, so we have,

$$P_0(t) = \exp\left[-\int_0^t g(u)du\right]g(t)\frac{f(t)}{g(t)} = f(t)\exp\left[-\int_0^t g(u)du\right]$$

If this t_1 is rejected, with probability $1 - f(t_1)/g(t_1)$, a t_2 is selected and if accepted we would get the probability :

$$P_1(t) = \int_0^t dt_1 \exp\left[-\int_0^{t_1} g(u)du\right]g(t_1)\left\{1 - \frac{f(t_1)}{g(t_1)}\right\}\exp\left[-\int_{t_1}^t g(u)du\right]g(t)\frac{f(t)}{g(t)}$$

We find here the four expected parts, the probability of selecting t_1 , the rejection of this t_1 , the probability to select t_2 after this and the acception probability of t_2 .

We see that the integrals in the exponentials together integrate from 0 to t and gives us $P_0(t)$, the other factors will simplify to give us :

$$P_1(t) = P_0(t) \int_0^t dt_1 [g(t_1) - f(t_1)]$$

For $P_2(t)$, we would have three exponentials that would still give us $P_0(t)$, we would still have the $g(t_1)(1 - f(t_1)/g(t_1))$ that give us the integral of $g(t_1) - f(t_1)$, on top of that, we would have a similar term but in $g(t_2) - f(t_2)$:

$$\begin{aligned} P_2(t) &= P_0(t) \int_0^t dt_1 [g(t_1) - f(t_1)] \int_{t_1}^t dt_2 [g(t_2) - f(t_2)] \\ &= P_0(t) \frac{1}{2} \left(\int_0^t [g(u) - f(u)] du \right)^2 \end{aligned}$$

The final acceptance probability will just be the sum of the $P_i(t)$:

$$\begin{aligned} P(t) &= \sum_{i=0}^{\infty} P_i(t) = P_0(t) \sum_{i=0}^{\infty} \frac{1}{i!} \left(\int_0^t [g(u) - f(u)] du \right)^i \\ &= f(t) \exp \left[- \int_0^t g(u) du \right] \exp \left[\int_0^t [g(u) - f(u)] du \right] \\ &= f(t) \exp \left[- \int_0^t f(u) du \right] \end{aligned}$$

showing that we indeed recover the desired probability while generating a t even if $f(t)$ is not integrable.[3] section 4.2

2.4 Merging

Now, we have the matrix element for the hard part and the partons shower that will treat the soft part of the events.

We could simply generate a process without extra jets with the ME and then let the PS generate all the jets. This description is sufficient for some analysis, but if we are interested in distributions of the jets themselves (a jet $p_{\perp}$ distribution for example), the PS description of the jet will be inaccurate. The PS does not handle well hard radiation, and so our jet distribution will not be correct in this region. To improve this, we can generate, with the ME, the base process + n jets, this will improve the description of those n jets in the hard region. This consideration alone will lead to double counting, as we have two concurrent descriptions of the first n jets, one from the PS and one from the ME. This is illustrated in Fig 2.2.

Multiple procedures exist to address this problem, here we will only be interested in multi-jet merging at LO. The term merging is used here to indicate that we will use a merging scale, ρ_{ms} , which will divide the phase-space into two. Above this merging scale, the jets should come from the ME and below it, from the PS, exploiting each of them at their respective best scale.

To be more practical, we will see the general outlook of an “ideal” merging algorithm for a process $I \rightarrow F + n$, with $n = 1, 2, \dots, N$. For each event generated by our event generator, we will :

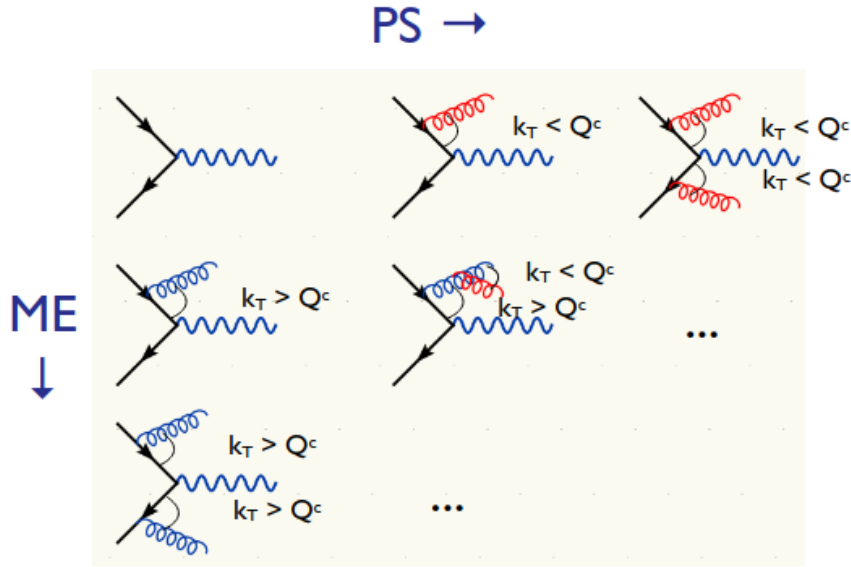


Figure 2.2: On the horizontal axis, we have the number of partons generated with the PS, and on the vertical axis, with the ME. We can see that a final state with a given number of parton can be generated with different combinations of ME and PS. We control the final state of the ME, in the sense that we can ask the generator to create the final states that we wish. But we have no such control over the PS, the rising diagonals on the graph show configurations that have the same number of partons in the final state. If we simply sum those contributions, we expose ourselves to double counting.

- **1** Compute the inclusive cross-section for our $F + n$ and generate the event above the ρ_{ms} , a cut is necessary to prevent singularities anyway.
- **2** Find scales to compute the no-emission probability and other scale dependent ratios (α_S and the PDF). This will make the
- **3** Multiply the cross-section in 1) by the factors in 2)
- **4** Run the shower below ρ_{ms}

In practice, it is not direct to implement this algorithm. This led to the creation of multiple algorithms, each with their own (dis)advantages. We will survey the two algorithms that are of interest in this work, MLM and CKKW.

2.4.1 MLM

This description of MLM and CKKW is based on [4]. Note that some other variation can be found in the literature such as the use of other jet clustering algorithm, difference in the cuts, ...

- **1** We generate parton-level configuration using ME for all parton multiplicities $n \leq N$. Some kinematic cut will be applied by the generator :

$$p_{\perp}^{part} > p_{\perp}^{min}, |\eta_{part}| < \eta_{max}, \Delta R_{jj} > R_{min}$$

with $\Delta R_{ij}^2 = \Delta \eta_{ij}^2 + \Delta \phi_{ij}^2$, this cut could be replaced by a direct $k_{\perp}$ -cut as in MadGraph.

- **2** We define a $k_{\perp}$ -measure as : $d_{ij} = \Delta R_{ij}^2 \min(p_{\perp i}^2, p_{\perp j}^2)$.

We then use a $k_{\perp}$ -clustering algorithm, allowing only physically possible clustering, to get a single Feynman diagram.

The $k_{\perp}$ -values are used as scale for the computation of the strong coupling constant at each vertex. So we have a factor $\alpha_S(k_{\perp 1})\alpha_S(k_{\perp 2}) \dots \alpha_S(k_{\perp n})/(\alpha_S(Q))^{n-2}$. We must here divide by the factor used by the ME generator which computes α_S only at the hard scale Q .

- **3** The events are showered with a standard PS. The upper veto for the shower is given by the hard scale Q . After the end of the PS, a jet cone algorithm is applied to the final partons. The reclustered jets will be defined by a cone size, a maximum pseudo-rapidity and a minimum transverse energy. These 3 parameters are generally related to the kinematic cuts from step 1. We would have $R_{clus} = R_{min}$, $\eta^{clus} = \eta_{max}$ and the energy $E_{\perp}^{clus} = p_{\perp}^{min}$, this $E_{\perp}^{clus}$ is our merging scale.. Although the work sticks to this choice of values, we note that it is not compulsory and can change in other implementations.
- **4** We will proceed to a matching of the tree-level parton with the PS jets. So, starting from the hardest parton, we take the jet that is closest to it, if the distance between the center of the jet and the parton is smaller than $1.5 \times R_{clus}$, the parton and the jet match. The matched jet is no longer considered, and the matching continues for the other partons. The event matches if each parton matches to a jet. Non-matching events are simply rejected.

- **5** Matched events with $n < N$ are also rejected if they have at least an extra jet. This prevents the double counting as these jets that are already present in the $n+1$ sample. Of course, events with $n = N$ are allowed to have extra jets as there is no $N + 1$ sample, but it is still required that the hardest (in $p_\perp$) of these extra jets is softer than the softest matched jet.

The veto performed on the events removes the need for the explicit Sudakov that was suggested in the “ideal” algorithm. It avoids the need for direct computation of those quantities, but it also means that their treatment is not exact. Indeed, only the final state of the ME generation is taken for the matching, we thus lose information on the intermediate states.

The main advantage of this algorithm is that it is simple, both to implement and conceptually. It does not require a dedicated shower implementation and can just be added at the end of the computation.

2.4.2 CKKW

- **1** An internal separation cut : $k_{\perp 0}$ is chosen in $k_\perp$ -measure which is our merging scale. The parton-level configuration is generated using ME while remaining above the merging cut.
- **2** Again similarly to MLM, a $k_\perp$ -algorithm is used to recluster the event, we also impose that we only recluster physically possible combinations. The $k_\perp$ -values are used as scale for the computation of the strong coupling constant. So we have a factor $\alpha_S(k_{\perp 1})\alpha_S(k_{\perp 2})\dots\alpha_S(k_{\perp n})/(\alpha_S(Q))^{n-2}$. We must here divide by the factor used by the ME generator, which computes α_S only at the hard scale Q .
- **3** We will reuse the branching history to reweight the matrix element with Sudakov factors. For each internal line at a scale $k_{\perp i}$ that goes to a scale $k_{\perp j}$ we get a factor : $\Delta(k_{\perp i}, k_{\perp j})$. For an external line, we have $\Delta(k_{\perp i}, k_{\perp 0})$. This overall sudakov factor is necessary because we restricted radiation to $k_{\perp 0}$.
- **4** A vetoed parton shower is then applied to the event. This parton shower will veto the generation of parton that would have been above the merging scale. We cannot directly start the shower at this limit because the cut is not in the same variable as the evolution variable. So, instead, when a generated parton is harder than our merging scale, we simply remove it and continue its showering from the scale at which the vetoed parton was generated.
- **5** Matched events with $n < N$ are also rejected if they have at least an extra jet. This prevents the double counting as these jets that are already present in the $n+1$ sample. Of course, events with $n = N$ are allowed to have extra jets as there is no $N + 1$ sample, but it is still required that the hardest (in $p_\perp$) of these extra jets is softer than the softest matched jet.

The veto performed on the events removes the need for the explicit Sudakov that was suggested in the “ideal” algorithm. It avoids the need for direct computation of those quantities, but it also means that their treatment is not exact. Indeed, only the final state of the ME generation is taken for the matching, we thus lose information on the intermediate states.

The main advantage of this algorithm is that it is simple, both to implement and conceptually. It does not require a dedicated shower implementation and can just be added at the end of the computation.

2.4.3 MLM vs CKKW

We will mention again that, as it will become important later, the MLM and CKKW merging algorithms don't approach the computation of the Sudakov in the same way. CKKW will apply the analytical Sudakovs for each line in the tree-level diagram. This analytical Sudakov is not perfect, as the PS would not have computed the Sudakov in the same fashion. A solution to this is the CKKW-L merging algorithm.

MLM, for its part, with its matching, can only have the correct Sudakovs for the partons in the final state, it has no (direct) information on the intermediate states.

Chapter 3

A classifier to perform the veto

The goal here, is to move, at least partially, the MLM selection process from after the PS to before the PS. This could avoid computing the PS on events that would be discarded, and thus saving on computational resources. The main constraint is that we do not want to reject events that the MLM algorithm would have kept, as this would modify the physical result. We will still keep running MLM, and so our procedure can keep events that should have been vetoed. To try to archive this, we used a neural network (NN).

3.1 Simulation Setup an data handling

We selected $pp \rightarrow t\bar{t}$ merged with $t\bar{t} + j$ as our process. This is quite “complete” as it presents both initial and final state radiations, has a dependence on the PDF, and is a common occurrence at LHC. These events have been fully simulated using MadGraph[5] and Pythia[6] (MadGraph has a built-in interface with Pythia). The data space for our machine learning is the 4-momenta of every parton at tree-level in an event, the target is the veto applied by the MLM algorithm.

3.2 Selection of variables

With the output from the ME generator, we have access to the 4-momentas of every partons at tree-level and to two scales used by MadGraph internally. ME generators, to facilitate comparison with LHC data, generate events with (nearly) total independence in the z axis. We can then exclude p_z momentum from our analysis. We are thus left with p_x , p_y and E .

The MLM algorithm selection is based on a $k_\perp$ measure both for jet clustering and for the matching, extra jets are also removed if they are too hard. And so the main dependence we expect is in p_x and p_y . We will thus use the composite : $p_\perp^2 = p_x^2 + p_y^2$ which could make us lose a marginal quantity of information but makes the NN training easier. We also use the two internal scales from MadGraph (which are related to the energy of the partons). A similar fit is performed in Chapter 6 using p_{x,p_y} and the energy for each parton, and training the network was indeed more difficult.

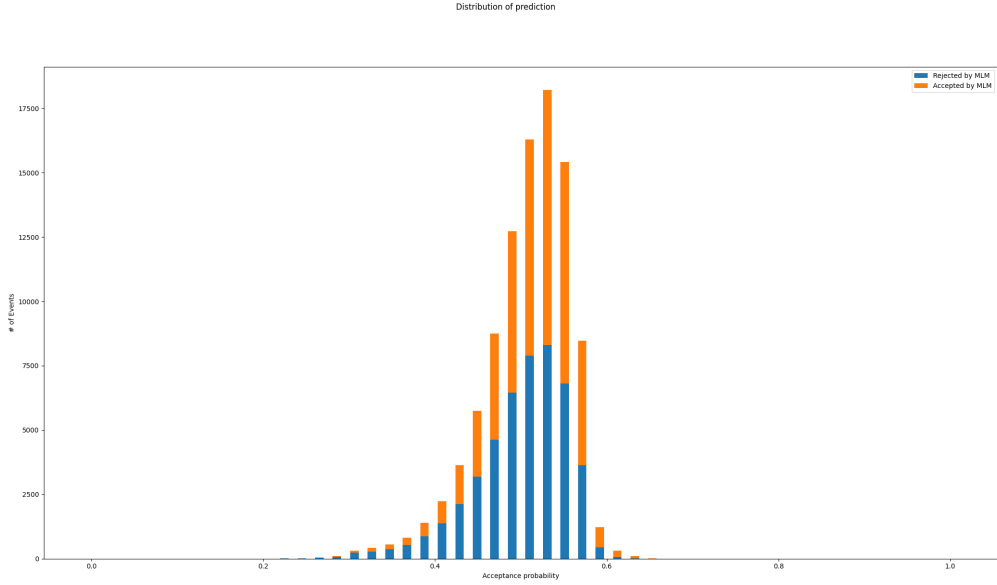


Figure 3.1: We have a distribution of acceptance probabilities of events. The output of the network is the sum of blue + orange. The two colors are meant to indicate the “truth” made by actually running MLM. So if our classifier were perfect, we would have all the blue on the 0.0 and all the orange on the 1.0.

3.3 Sample selection

We expect our classification for the $pp \rightarrow t\bar{t}$ sample to be difficult. Indeed, the veto probability for the MLM algorithm is mainly the Sudakov factor for the final state. This Sudakov factor will depend on the scale of each of the partons in the final state. If we only have $t\bar{t}$, both quarks will take around half the center of mass energy with very little variation. This means that the Sudakov for each event will be practically the same. We thus expect the veto probability for this sample to be nearly identical for every event.

To confirm this intuition, we can take a look at the results of a NN fit. What interest us is the acceptance probability distribution. This is based on the output of our network. For only the $t\bar{t}$ final state, we have 3.1. The more the distribution is “organised”, i.e., the blue is stacked near the 0.0 and the orange near the 1.0, the better our whole procedure will be. And if we look now at our other multiplicity sample, $pp \rightarrow t\bar{t} + j$, we have Figure 3.3.

We will thus only consider the 1 jet multiplicity for our analysis, and simply disregard the 0 jet multiplicity for the remainder of this chapter.

3.4 Neural Network structure

We use the pytorch package to build the NN in python. We choose a typical network with 3 intermediate linear layers with 20 width and one rectifier layer in the middle to give some non-linearity to our fit + 1 input and 1 output layers. We used mean squared loss entropy as the error function.

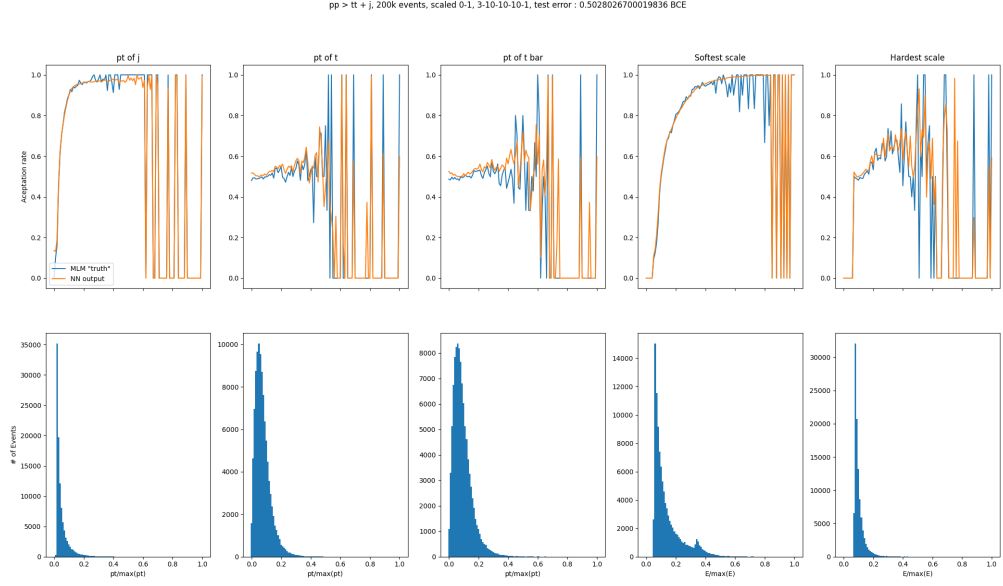


Figure 3.2: We can see in the top row the comparison between the acceptance rate of MLM and our NN. On the bottom row, the distribution of events for each parameter of interest is shown. The x axis is scaled between 0 and 1 by simply dividing the individual value by the maximum for the set.

3.5 Results

We can now show the results of the previous considerations. We generated three runs containing 200K events of $pp \rightarrow t\bar{t}$ and $pp \rightarrow t\bar{t} + j$. We use one run to train the network, one to compute its error, and finally we used the third one to construct Figure 3.2 that show the resulting fit. Having three different runs is, here, preferable to avoid over fitting. We can see that our network is capable of fitting the selected variables reasonably well.

If we actually want to apply a veto at this stage, we need to choose an acceptance probability under which we will veto all events. The problem is that, as mentioned earlier, we do not want to veto events that MLM would have kept if our NN had not been in place. If we look at Figure 3.3, it is not possible to choose an acceptance probability that both removes a meaningful number of events that should be rejected while rejecting no or very few events that we should keep. A moderate gain of performance cannot justify such an alteration of the final physical result, this approach is thus not acceptable.

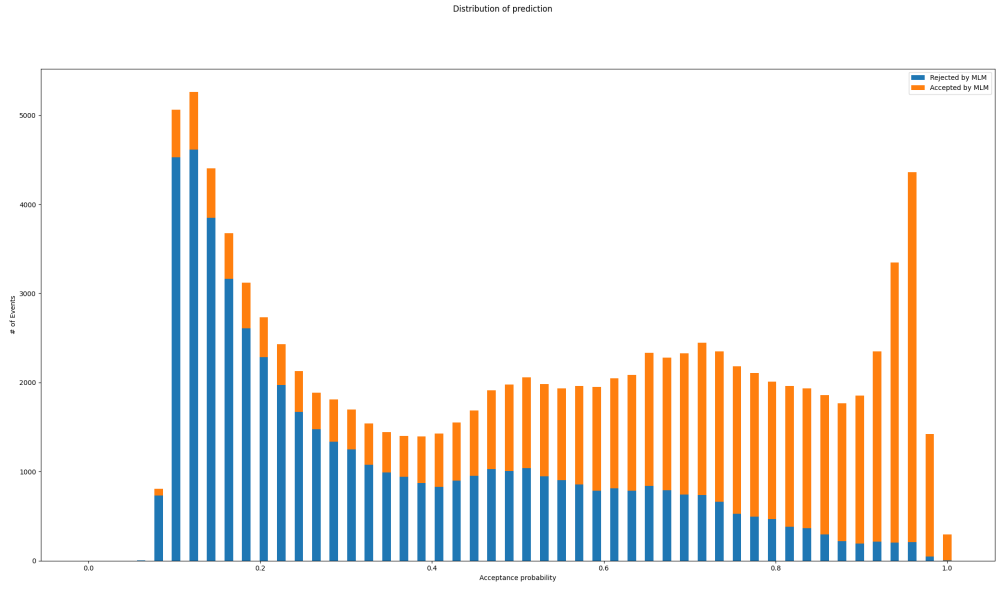


Figure 3.3: Same plot construction as in Figure 3.1 but for the $pp \rightarrow t\bar{t} + j$ sample.

Chapter 4

Parton Shower implementation

In order to use the CKKW algorithm as we wished, we have to implement our own parton shower. This PS is based on a tutorial by the MCnet collaboration at the CTEQ / MCnet School 2021. Much of the consideration that we will discuss in this chapter comes from the lecture material used during the school.

This parton shower is limited to final state radiations, we thus must select a process that can't emit initial state radiation. We choose $e^+e^- \rightarrow jj$. We of course need to add a jjj final state to be able to perform merging.

As said in the introduction, three parts are necessary for an implementation :

- The splitting functions
- The kinematic
- The veto algorithm

Splitting functions

We used the splitting functions from [7] equation 3.22. These splitting functions will require two parameters instead of just z as was the case in their description in Chapter 2. It is necessary to use a modified version because of the need not only of the two children of the splitting but also of a recoiler particle. This requirement will be discussed in the implementation of the kinematics itself. In short, we must transfer some momentum with the recoiler and, for this, we require an extra degree of freedom, which is encoded in the y variable.

As made necessary by the veto algorithm, we also associate a simplified splitting function ($g(z)$) which respect $g(z) > f(z) \forall z \in [0, 1]$. We have :

$$\begin{aligned}g_{qq}(z) &= C_F \frac{2}{1-z} \\g_{gg}(z) &= C_A \frac{1}{1-z} \\g_{gq}(z) &= T_R \frac{1}{2}\end{aligned}$$

Kinematics

We need to appropriately distribute the momentum between the two splitted particle. To accomplish this, we will need a so-called spectator or recoiler particle which

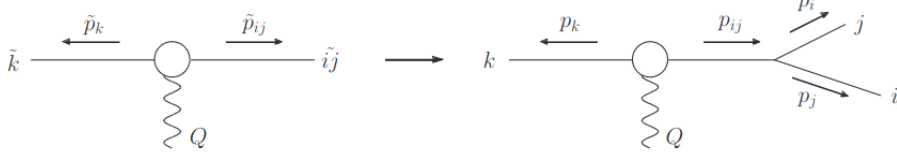


Figure 4.1: Distribution of momenta in a splitting

allows us to give an appropriate momentum to the splitter while conserving global momentum for the event. A representation of this process is given in Figure 4.1.

We can rescale the p_{ij} momentum using a y fraction of momentum of p_k . We define

$$y := \frac{p_{ij}^2}{2\tilde{p}_{ij}\tilde{p}_k}$$

And we get (off-shell):

$$p_{ij} = \tilde{p}_{ij} + y\tilde{p}_k \quad p_k = (1 - y)\tilde{p}_k$$

We can then establish three on-shell equation for the three momenta: p_i and p_j the momentas of the children and p_k the momentum of the spectator :

$$\begin{aligned} p_i &= z\tilde{p}_{ij} + (1 - z)y\tilde{p}_k + k_\perp \\ p_j &= (1 - z)\tilde{p}_{ij} + zy\tilde{p}_k - k_\perp \\ p_k &= (1 - y)\tilde{p}_k \end{aligned}$$

with $\tilde{p}_{ij}$ the initial particle momentum and $\tilde{p}_k$ the momentum of the spectator before the splitting. We also have that $k_\perp^2 p_{ij}^2 z(1 - z) = k_\perp^2 = 2\tilde{p}_{ij}\tilde{p}_k yz(1 - z)$. For our equation to be on-shell we must have that $z \in [z_-, z_+]$ and

$$z_\pm = \frac{1}{2}(1 \pm \sqrt{1 - 4k_\perp^2/p_{ij}^2})$$

Veto Algorithm

The implementation of the veto algorithm is straightforward from the description that was given previously in Chapter 2. We use $p_\perp$ as ordering variable, so we have that $t = 2\tilde{p}_{ij}\tilde{p}_k yz(1 - z)$ and $z_\pm = (1 \pm \sqrt{1 - 2t_0/(\tilde{p}_{ij}\tilde{p})})$.

4.1 Limitations

Our PS model is not perfect, and some inaccuracies are expected. We do not here intend to build a replacement for mainstream tools but we still want to have a reasonable agreement with a more advanced implementation. We will thus compare the results of our shower with Pythia.

Pythia has some more functionality than our implementation, we can give as an example the matrix element correction to improve the radiation pattern of the hardest parton which is not done in our implementation. Pythia is also back tested on experimental data and has benefited from a lot more work compared to our

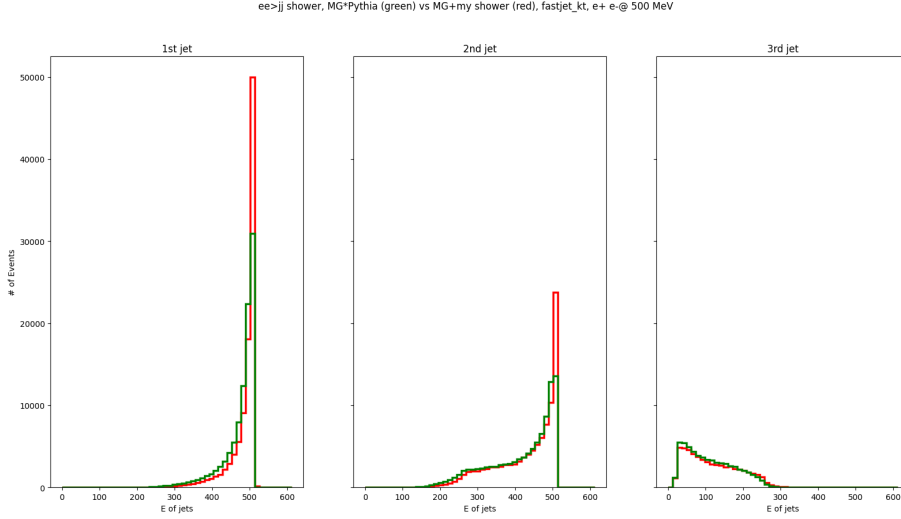


Figure 4.2: This plot is generated from the result of a shower on a $e^+e^- \rightarrow jjj$ process. We show here a comparison of the 3 hardest jets reclustered with a $k_\perp$ -algorithm. We have in green the results from the Pythia shower and in red, the data from our PS implementation.

implementation. All of this to say that we expect to see variation between our shower and a Pythia shower.

We will thus directly compare showering from our implementation with a shower generated using Pythia. The result of such a comparison for a showering of $e^+e^- \rightarrow jjj$ is given in Figure 4.2. We chose to compare the 3 first jets as their distribution are one of the most sensitive parameters to merging. Differences are clearly visible. Our shower implementation has more concentrated emissions toward the pic for the 1st and 2nd jets. For the 3rd jet, there is no such concentrated variation.

Chapter 5

CKKW using Sudakovs from the MLM fit

The idea of this second approach is to use the veto rate of the MLM algorithm as an approximation of the analytical Sudakov necessary in the CKKW algorithm. This approach allows us to keep every events while retaining the Sudakov from MLM. We will use a similar architecture to Chapter 3. So we will conserve a NN that takes as input the 4-momenta of tree-level partons, we will in this case not use it as a classifier but simply as a fit of the acceptance rate. We can then take the prediction for each event from this network and use it as a reweighting on the event after running the CKKW PS. This new approach allows us to keep some infrastructure we previously established but allow us to get rid of the strict condition (no rejection of events that should be accepted) imposed in the approach of Chapter 3.

As mentioned in the introduction, this second approach was also unsuccessful for reasons discussed in section 5.3 at the end of this chapter. But we will still give some consideration to this approach to keep the connection with what was discussed in the pre-defence of this work.

Because of the restrictions on our PS implementation, we will have to select a new process with no initial state radiation. The obvious choice will be $e^+e^- \rightarrow jj$ as the 0 jet multiplicity ($n=0$) merged with $e^+e^- \rightarrow jjj$ as ($n=1$).

5.1 Limitations

This approach to merging will be limited mainly by two factors. Firstly, as explained earlier, the way in which we compute the Sudakov is only an approximation an analytical Sudakov traditionally used in CKKW algorithm. Secondly, the fit of the Sudakov itself with the neural network will inevitably introduce some errors. Thus, we don't expect to have a perfect correspondance between this implementation and a standard MLM implementation. Part of this additional errors can be recouped by an decrease in statistical uncertainty, given that we keep more events than the MLM algorithm.

We also need to keep in mind that our PS implementations, used both as a standard shower for generating the MLM selection and modified to run CKKW are also limited, as described in Chapter 4. So, any conclusion we could put forward later will always have to be treated with extra caution.

5.2 Overview

The hard events are generated using madgraph[5]. The parton shower, the MLM and CKKW algorithms are the implementations described previously. For the multiple reclustering, the Fastjet library[8] was used.

The $e^+e^- \rightarrow jj$ and $e^+e^- \rightarrow jjj$ events are first generated using Madgraph. We limit ourselves to events with $N = 1$ so we have a mix of events with $n = 0, 1$. The event simulates electron-positron collisions, with each electron bringing 500Gev of energy.

We implemented a neural network, taking as input the p_x, p_y and E of the two or three partons in the final state. The output of this network is a single real number $0 \leq p \leq 1$ which is our fit of the probability for MLM to reject a given event. We use a mean squared error to perform the training and testing.

Our fit will be performed on the two different jet multiplicity samples separately. We have the same concern as in Chapter 3, where we discussed that the 0 jet sample acted as noise in the fit for the 1 jet sample. So we create two copies of our network, each trained on one of the jet multiplicities and then reassemble the prediction data for all events.

5.3 Results

We could not produce meaningful results with this approach.

This does in no way prove that this approach is not possible, as our CKKW implementation could be at fault. Indeed, a CKKW implementation is quite tricky to get right. In particular, the veto shower is difficult to implement in a $p_\perp$ ordered PS with a cut not in $p_\perp$ because the original CKKW algorithm [9] uses a $k_\perp$ -like measure for the cut. We could not test our CKKW implementation alone because the source for our Sudakov is the MLM fit. We thus can't, from this results alone, discredit the use of this fit in this way.

We still have some intuition as to why this approach was doomed from the start. The MLM veto rate is a replacement for a Sudakov reweighting. But it is more than that, it also replaces the suppression of hard radiation from the PS. These two steps are done explicitly in CKKW. So if we use this veto rate to reweight the events as if it was the Sudakov in CKKW, then we double suppress the hard radiation. Both the MLM veto rate and the CKKW modified shower will impact these radiations. Some more supporting arguments are given in Chapter 6.

These considerations led us to a third and arguably simpler approach, use our fit of the acceptance rate of the MLM algorithm simply as a weight on events to entirely replace the MLM merging. We will also need to reweight with the α_s 's.

Chapter 6

Replacement of the MLM veto by a reweighting

The previous chapter revealed that using the fit of the MLM algorithm acceptance rate as a reweighting for unmerged events was perhaps sufficient to build the merged jet distribution (we also have to reweight with new α_S 's). As we treat again only with MLM, we can go back to the simulation tools of Chapter 3. So we, again, will use MadGraph + Pythia to generate and merge the processes $pp \rightarrow t\bar{t}$ and $pp \rightarrow t\bar{t} + j$. And will reuse (but modify) the neural network built in Chapter 3. To be certain to extract all the useful information present, we will use as input p_x, p_y and E of every parton in the final state at tree-level. We disregard the internal cuts used previously, as they are extremely correlated with the other variables. And, because we do not fully understand their origin, they could spoil any physical conclusion we could draw from our results.

Here, however, we will simply use the prediction of the network, which is a number $\in [0, 1]$ to reweight each event. This also means that we can't ignore the 0 jet multiplicity as we had done in the classifier because we need a reweighting for every event. Thus, we used two different NN trained separately, each on a particular jet multiplicity. The result of the training is given in Figure 6.1.

A comparison of this methodology and a classic Pythia MLM treatment is given in Figure 6.2. As can be seen on the figure, the two curves for every jet are very similar. This shows that this approach works well. The same kind of distribution, but in $p_\perp$ is given in Figure 6.3. This type of transverse jet distribution is what is commonly used as validation for merging. This shows that, at least for the merging between 0 and 1 jet multiplicities, the reweighting with the acceptance rate of MLM is a workable concept to build such distributions.

This shows that the approach used in Chapter 5 was not possible at all. We recall that we used this reweighting only as a Sudakov and then CKKW would impose an extra cut on the radiation emitted by the PS. This would necessarily modify the jet spectrum and so we could never have recovered the same distributions.

Another interesting point is that the NN only takes for parameters the momenta of the final state partons at tree-level. This means that the reweighting factors that perform the merging do not depend directly on the result of the shower for each event. Of course, the internal parameters of the NN after training will depend implicitly on the shower and MLM implementation.

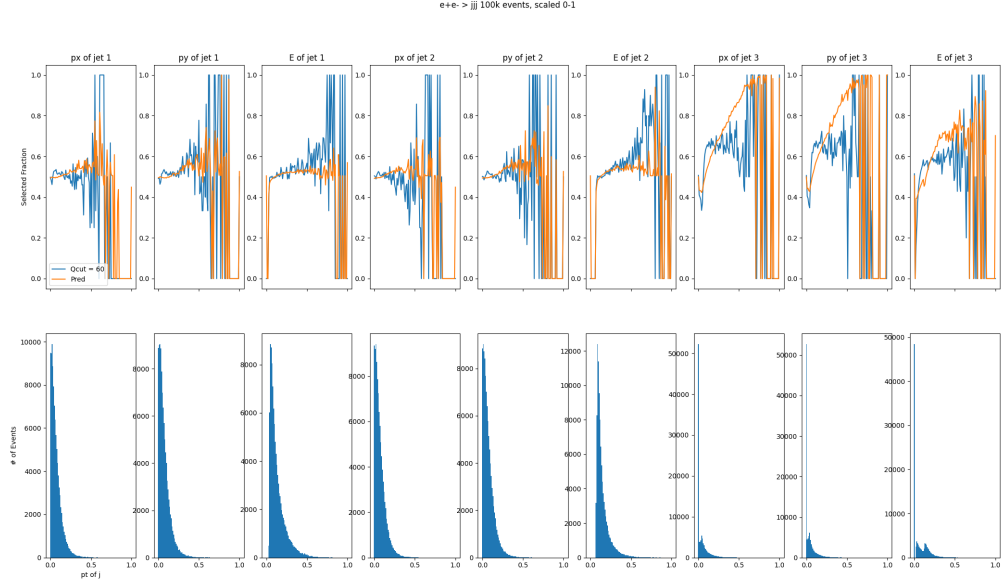


Figure 6.1: Similar plot as in Figure 3.2 but with different variables, namely, p_x, p_y and E for every partons from the ME.

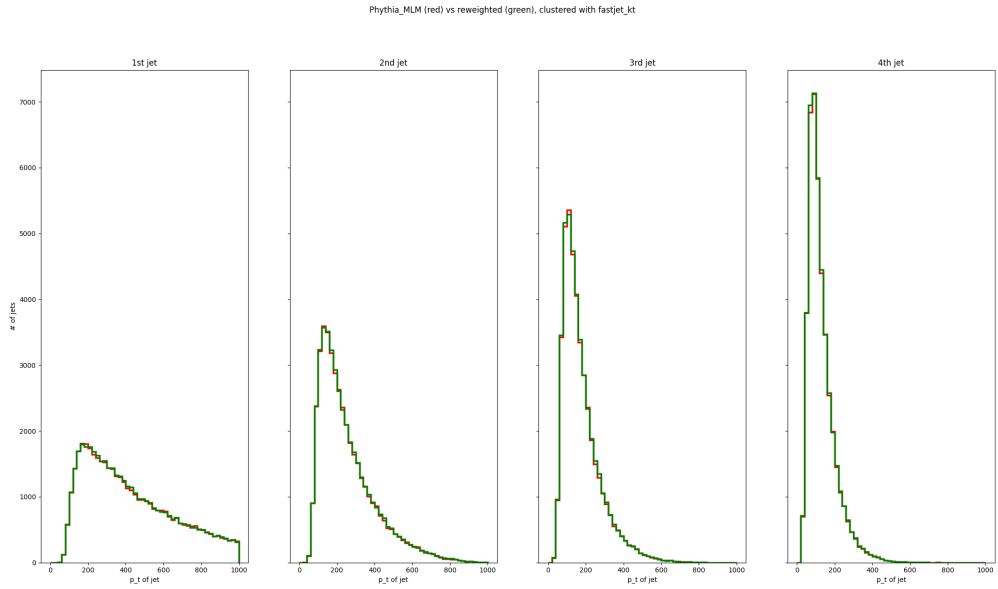


Figure 6.2: We plotted the distributions of the 3 hardest jets in energy of 100k event of $pp \rightarrow t\bar{t}$ and $pp \rightarrow t\bar{t} + j$. In red, the distribution is build using Pythia MLM, in green, we have our reweighting by our fit of the same events.

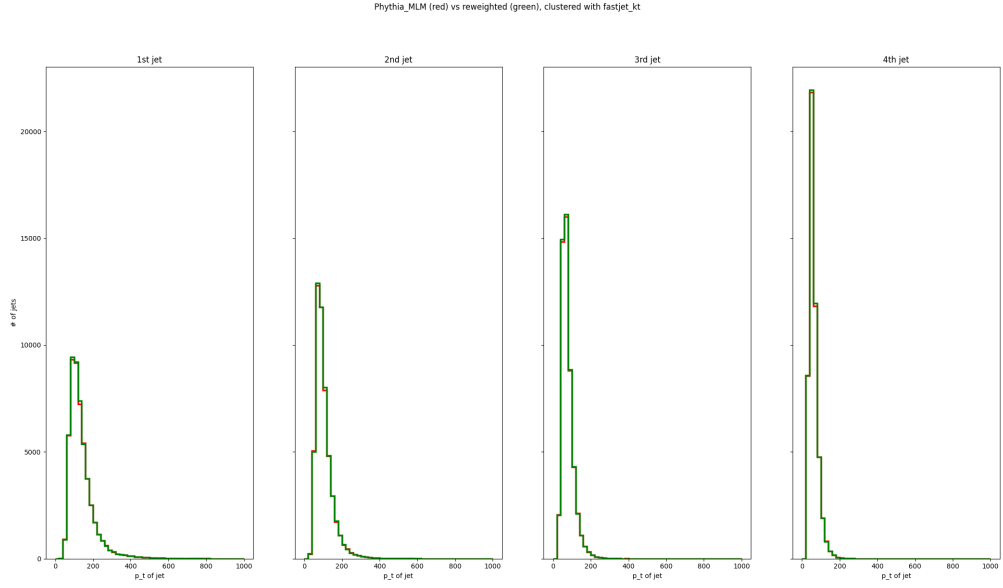


Figure 6.3: The graph is the same as 6.2 except that the distributions are made in $p_{\perp}$.

6.1 Limitations

The quality of our fit will be of the utmost importance. The fit we show in Figure 6.2 is far from perfect, but the zones where the prediction deviates from the truth do not contain a huge amount of events. This is sufficient to come to some conclusion, but the fit could be improved nonetheless.

We did not do any quantitative analysis, and so it is not possible to know to what extent our results will deteriorate compared to a standard MLM. We can still note that part of the error introduced by the fit can be compensated by the increase in statistical significance as we keep all events, which could be a big factor if we generate a sample with few events.

Chapter 7

Conclusion

Chapter 3 showed us that a classifier that used only tree-level information could not predict which events the MLM algorithm would reject with enough accuracy to be a useful tool. This led us to use this classifier as a simple NN fit. We identified the result of this fit, i.e., the veto probability of MLM as the Sudakov factor needed in the CKKW algorithm. This identification was erroneous, the veto probability also incorporates the restriction on hard radiation from the PS that is needed for merging. So we can in fact use this probability as a reweighting directly on each event in order to build jet distributions, at least for a $pp \rightarrow t\bar{t}$ in the 0 + 1 jet multiplicities. The reweighting by the α_S 's factors is also necessary. This result could possibly be extended to higher jet multiplicity and other processes. It is to expect that our trained NN depend on the process, the jet multiplicities involved, the energy of the collision, the value of the merging scale, and, possibly, some other variables

To deploy this as a merging tool, some training would probably be necessary for each specific use case, which makes this approach not very flexible.

This last result also shows that most, if not all, of the information needed to perform MLM on a set of events is present at tree-level. On an event by event basis, however, this is not true, as the classifier in Chapter 3 could not veto event as MLM could. This is not totally surprising, as the PS is a fundamentally probabilistic process.

Chapter 8

Annex

8.1 Reproduce the plots

We will describe how to reproduce each plot shown in this work from the code uploaded to GitHub at github.com/guiwi/thesis_mlm.

First, some general considerations. The parsing of hepmc files is quite slow, especially since we use pyHEPMC which extracts a lot of information. To avoid parsing the event files multiple times, we use the *csv_generator.py* script. It will generate csv files with the relevant information for our NN, i.e., the momenta of tree-level partons and if the event has been vetoed by MLM at 3 different merging scales. Some other informations are stored, such as some internal scales from MadGraph.

The NN *neural_fit.py* can then use these csv files as input to build the training, testing, and prediction datasets. This NN will output a text file containing the predictions for each event.

After this, the *clustering_fit.py* script can be used to compare Pythia merging with our reweighting.

To generate events using MadGraph and Pythia, we imposed some parameters in the Pythia card of MadGraph. We have to disable the hadronisation to compare only the partonic distributions, this is done using *hadronlevel : all = off*. By default Pythia will not save MLM vetoed events, we have to keep them to perform the fit, *SysCalc : qWeed = 0.0000000000e + 00* will ensure this.

Figure 3.2 and Figure 6.1

This is a plot always generated when making predictions with the *neural_fit.py*. The current state of *neural_fit.py* will generate Figure 6.1. It is possible to modify the script to produce Figure 3.2. The input layer of the NN needs to be modified as well as the input data, but all the necessary variables are present in the csv file used as input.

For Figure 6.1, we used 2 sets of 1M events for training and testing and a set of 100k events for the plots and the prediction. We limited ourselves to 100k events, as the fastjet clustering in *clustering_fit.py* uses a lot of RAM.

Figure 3.3 and 3.1

The two figures are also generated using *neural_fit.py*, but some modifications have been made to the plotting part of the script. An *if* statement was added each

time to keep only one or the other sample in the plotting. Similarly to Figure 3.2, the variables currently used as input by the NN are different, but the resulting plot should be very similar.

Figure 6.2 and 6.3

After making the weights with *neural_fit.py*, we can use *clustering_fit.py*. By default, this script will generate both figures.

Figure 4.2

It is first needed to run the *shower_from_mdg.py* script which will shower the final state of a MadGraph event. Then the result of this shower will be stored in a file. We can then run *clustering_myshower.py*. This last script will recluster both the shower from pythia and ours.

Bibliography

- [1] Sherpa Team, 2024.
- [2] Michael E. Peskin and Daniel V. Schroeder. *An Introduction to quantum field theory*. Addison-Wesley, Reading, USA, 1995.
- [3] Torbjörn Sjöstrand, Stephen Mrenna, and Peter Skands. Pythia 6.4 physics and manual. *Journal of High Energy Physics*, 2006(05):026–026, May 2006.
- [4] J. Alwall, S. Höche, F. Krauss, N. Lavesson, L. Lönnblad, F. Maltoni, M.L. Mangano, M. Moretti, C.G. Papadopoulos, F. Piccinini, S. Schumann, M. Trecani, J. Winter, and M. Worek. Comparative study of various algorithms for the merging of parton showers and matrix elements in hadronic collisions. *The European Physical Journal C*, 53(3):473–500, December 2007.
- [5] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.-S. Shao, T. Stelzer, P. Torrielli, and M. Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *Journal of High Energy Physics*, 2014(7), July 2014.
- [6] Christian Bierlich, Smita Chakraborty, Nishita Desai, Leif Gellersen, Ilkka Helenius, Philip Ilten, Leif Lönnblad, Stephen Mrenna, Stefan Prestel, Christian T. Preuss, Torbjörn Sjöstrand, Peter Skands, Marius Utheim, and Rob Verheyen. A comprehensive guide to the physics and usage of pythia 8.3, 2022.
- [7] Stefan Höche. Introduction to parton-shower event generators, 2015.
- [8] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. Fastjet user manual: (for version 3.0.2). *The European Physical Journal C*, 72(3), March 2012.
- [9] S. Catani, Yuri L. Dokshitzer, M. Olsson, G. Turnock, and B. R. Webber. New clustering algorithm for multi - jet cross-sections in $e^+ e^-$ annihilation. *Phys. Lett. B*, 269:432–438, 1991.