Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

David Korbany

born in Mühldorf am Inn (Germany)

2019

Symbolic Regression in Heavy-Ion Physics

This Bachelor Thesis has been carried out by David Korbany at the
Physikalisches Institut in Heidelberg
under the supervision of
Prof. Klaus Reygers

This page is intentionally left blank

Abstract

We introduce *Symbolic Regression* and implement it in Python featuring symbolic constants and support for data with uncertainty. We apply our program to a selection of artificial problems and data from heavy-ion physics. In particular, we are able to find functions describing p_T spectra from Pb-Pb collisions which show similarities to the physically motivated Hagedorn function.

Kurzzusammenfassung

Wir führen *Symbolische Regression* ein und implementieren dies in Python inklusive symbolische Konstanten und Kompabilität für Daten mit Fehlerbalken. Wir wenden das Programm auf eine Auswahl von künstlichen Problemen und Daten aus der Schwerionenphysik an. Wir sind insbesondere in der Lage Funktionen zu finden, die p_T Spektren aus Pb-Pb Kollisionen beschreiben und Ähnlichkeiten zu der physikalisch motivierten Hagedorn Funktionen zeigen.

Contents

Chapter 1

Motivation

Symbolic regression (SR) is broadly speaking a regression tool which makes no assumption about the underlying model. Instead, it performs evolutionary-inspired search in the space of mathematical expressions within a user-defined scope. Historically SR has been an application of genetic programming [1], which can be seen as an approach to automated programming and artificial intelligence, as depicted in figure 1.1.

There have been two papers sparking our interest. In [2] the authors use SR to find Lagrangians and Hamiltonians for simple mechanical systems from motion-tracked data.[1] More recently the authors in [4] use a deterministic SR technique to find governing differential equations of given systems from time series measurements. They are able to find the Navier–Stokes equation this way. These applications can be put in a category we dub "Towards Data-Driven Discovery of Natural Laws". Given these astonishing results, we decided to investigate possible applications of SR to particle physics. We didn't quite get to the point of discovering natural laws from data, but we will show how SR can be used as a powerful interpolation and modeling tool and in some cases is able to reverse engineer known physical laws.

In section 2 we introduce genetic algorithms using an interesting example from game theory, which shows the versatility of genetic algorithms and genetic programming techniques. We then give a more detailed introduction to SR in section 3 and discuss our implementation in section 4. In section 5 we present applications of SR to heavy-ion physics.

We will not give an overview of the many different approaches and ideas in SR. A recent short summary of state-of-the-art methods, addressed to non-experts, can be found in [5].

[1] We shall mention that there seems to be some need for clarification about their approach as pointed out in [3].

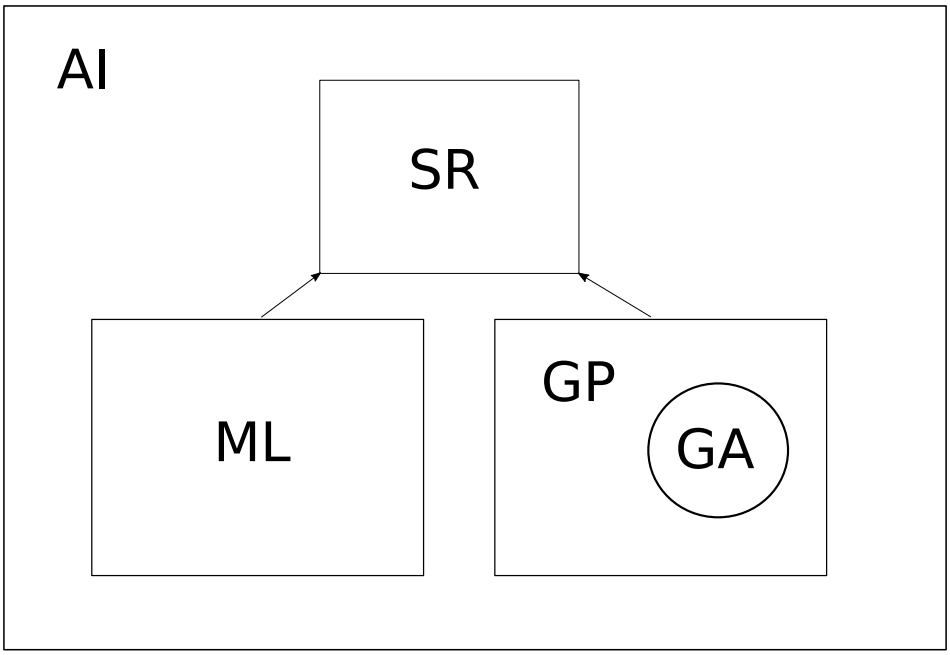


Figure 1.1. Relation (not absolute) of genetic programming (GP), genetic algorithms (GA), machine learning (ML) and SR to each other. They can all be seen as different approaches to artificial intelligence (AI). Apart from genetic programming-based SR there are more recent SR approaches which solely use machine learning techniques, e.g. *ffx* [6]. More sophisticated SR programs, like the commercial product *Eureqa*, which was originally created by the authors of [2], use both genetic programming and machine learning ideas.[2]

[2] https://www.nutonian.com/products/eureqa/

Chapter 2

A short Introduction to Genetic Algorithms

2.1 The Imitation Game

The idea of biologically inspired computing goes back to the very beginnings. To study the question "Can machines think?" Alan Turing [7] proposed the "imitation game": An interrogator asks two persons, Alice and Bob, questions. One of them is a machine, say Bob. The goal for the Interrogator is to determine who is the machine. Bob will pretend to be a man and Alice will tell the truth. The question is, can one build a machine which will win the game, i.e. fool the interrogator? Turing proposes:

> "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain."

The process of thus creating a computer program resembles evolution. Turing identifies:

- Structure of the child machine = Hereditary material
- Changes of the child machine = Mutation
- Natural selection = Judgment of the experimenter

The "Judgment of the experimenter" determines the *fitness* of the program. Furthermore, he remarks:

> "We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse."

Modern genetic algorithms (GAs) combine these ideas in a parallel way. Starting with an initial population of computer programs (child-machines) one evolves towards a new population using natural selection (survival of the fittest) and biologically-inspired operators like mutation and (chromosomal) crossover. The fitness, evaluated by the experimenter, determines the success rate for reproduction. Through this evolutionary process one hopes to create a better (adult-brain) program.

2.2 Example: The Prisoner's Dilemma

We will explain in a bit more detailed way how GAs work using the example of how Axelrod [8] used GAs to evolve strategies for the Prisoner's Dilemma. Axelrod describes the games

as follows: "Two individuals [A and B], can either cooperate or defect. No matter what the other does, the selfish choice of defection yields a higher payoff than cooperation. But if both defect, both do worse than if both had cooperated." This is summarized in table 2.1.

Table 2.1. Payoff matrix for the Prisoner's Dilemma. Here (a,b) represents A getting payoff a and B getting payoff b. Adapted from [8].

	Cooperate	Defect
Cooperate	3,3	0,5
Defect	5,0	1,1

There are 4 possible outcomes in a game:

- A and B cooperate: CC
- A cooperates and B defects: CD
- A defects and B cooperates: DC
- A and B defect: DD

A deterministic strategy, which determines the decision of a player depending on the result of one previous game, is a map

$$S : \{CC, CD, DC, DD\} \to \{C, D\}.$$

A strategy which depends on the outcome of n previous games is a map

$$S : \{CC, CD, DC, DD\}^n \to \{C, D\}.$$

This means that every possible outcome of the n previous games is mapped to an action: cooperation or defection.

Thus, a strategy depending on one previous game can be encoded in a string of length four. For example, "tit for tat" corresponds to CDCD: Given the outcome of a game one first looks up the position in the canonical ordered list of possible outcomes.[1] Then, one chooses the letter at this position in the string. For example, let's assume the outcome of the game was DC. This corresponds to the third outcome in the list. Thus, we look up the letter at position three in our strategy. In "tit for tat" (CDCD) this would be C.

In the iterated Prisoner's Dilemma studied by Axelrod the programs remembered the moves of three previous games. There are $4^3 = 64$ possible outcomes and thus 2^{64} different strategies encoded in strings of length 64. Axelrod added six extra letters encoding three hypothetical previous games. This determines the outcome of a strategy for the first three games, where no data from previous games is available. A program is thus a string of length 70. Note that this is a the same time a set of instructions [2] giving a strategy for the game, and a combination of genetic material subject to changes through an evolutionary process. A simple GA works as follows:

(i) *Create an inital population of candidate solutions.*

These are usually randomly generated. In our example we simply produce random strings of length 70 out of the letters C and D. The last six letters determine the first three moves.

[1] If C is zero and D is one every strategy is, after removing the leading zeros, a unique binary number and the canonical way to order is from the smallest number (CC=0) to the biggest (DD=3).

[2] Implemented in a computer a string could be compiled, so to say.

(ii) *Calculate the fitness of each individual.*

Axelrod let the programs play iterated games against eight human-designed strategies. (Not including "tit for tat".) The fitness was taken as the average score (using the payoff matrix).

(iii) *Evolution*

Based on the fitness select individuals for crossover (sexual reproduction) and mutation. Mutation is straightforward. Select a random position in the string and flip the letter at this position. An example of crossover works as follows: Given two programs, i.e. strings of length 70, choose a random position and exchange the sub strings up-to and after this position. For example, CCCC and DDDD could be crossed over at the second letter to produce the offspring DDCC and CCDD.

(iv) *Replace the population with the offspring.*

(v) *With the new population go to step 2 or stop*

Axelrod evolved 40 randomly generated populations of size 20 over 50 generations. He observed that the strategies evolved similarities to "tit for tat". In all runs the final population scored on average better than the human-designed strategies. In eleven of the forty runs the final population even scored on average better than "tit for tat".[3] This is a striking result, given that each run roughly only tested 1000 out of 2^{70} possible programs

[3] In this particular environment consisting of the eight human designed strategies.

Chapter 3

The next step: Genetic Programming and Symbolic Regression

Genetic programming (GP) generalizes GAs in a way such that the size, i.e. complexity, of the programs can evolve as well [1]. It has been successfully applied to real world problems. In [9, table 5.2] the authors list "Thirty-six instances of human-competitive results produced by genetic programming". One application of GP, on which we will focus in the following, is Symbolic Regression (SR). In SR the programs are mathematical functions and the problem is to find a function describing a set of data. The goal is to find a good expression w.r.t. a fitness function, e.g. the χ^2 sum.

3.1 Representation

We represent functions as trees. The leaves (non-internal nodes) of trees are called *terminals*. They represent arguments of the basis functions, a.k.a. *primitives*, which are the internal nodes in the tree.

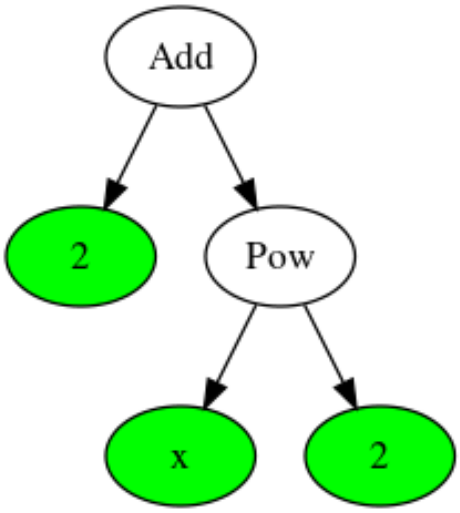


Figure 3.1. Example of a tree representing a function in SR. The primitives are the internal nodes Add and Pow. The terminals are the leaves (green) $2, x$ and 2. The nodes are ordered from left to right. Thus, the tree evaluates to $\mathrm{Add}(2, \mathrm{Pow}(x, 2))$, which gives $2 + x^2$ with our convention for Pow.

The tree in figure 3.1 represents the function x^2+2, where we read the arguments from left to right. The *size* of the trees is the number of nodes. Unlike in GAs the size of the individuals is variable.[1] The *depth* of a node is the number of edges that need to be traversed to reach the node starting from the root [10]. The *depth* of a tree is the maximum depth of all nodes. For example the tree in figure 3.1 has size five and depth two.

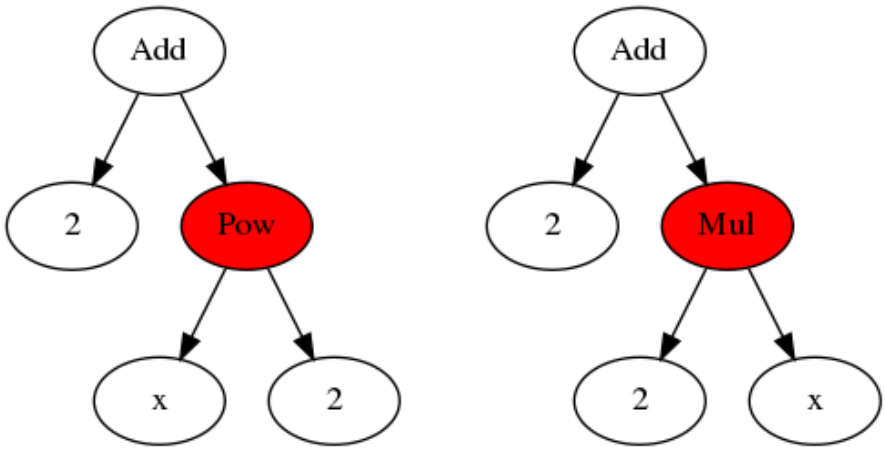
3.2 Basis Functions

Depending on the problem at hand, one has to choose a basis set of functions and terminals, called the *primitive set*. These are the building blocks, i.e. genes, of individuals. The function set can include anything from basic arithmetic operators like addition or multiplication to special functions like the exponential or Gamma function. The terminals consist of the arguments of primitives, i.e. variables and (numerical) constants. In a three dimensional problem, there would be three variables and so on.

3.3 Genetic Operators

3.3.1 Mutation

We can mutate an individual by changing nodes or subtrees in its representing tree.



(a) x^2+2 **(b)** $2x+2$

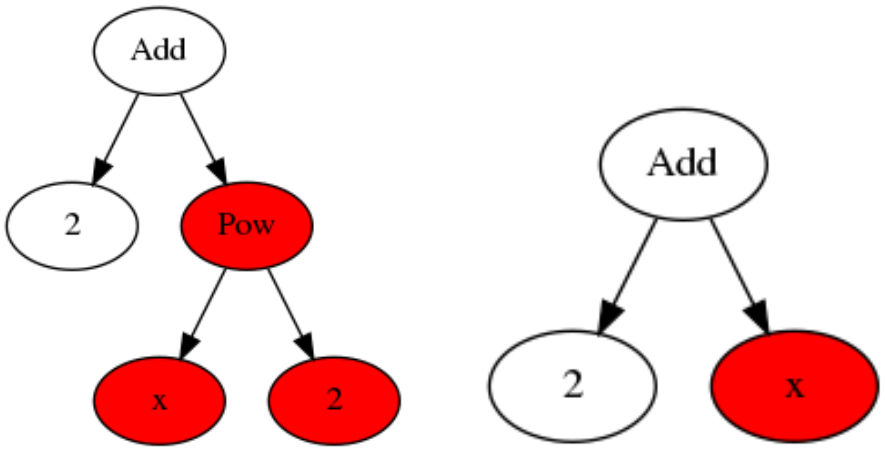
Figure 3.2. Point mutation. Here we replaced the function Pow with Mul. This only works if the functions have the same arity.

Two examples are visualized in the figures 3.2 and 3.3.

3.3.2 Crossover

The idea is to imitate biological sexual reproduction, whereby the offspring takes genetic material from both parents. There is a huge variety of crossover operators in the GP literature, see e.g [10]. In *one-point crossover* we select a common node of the parents and swap the subtrees at this node, producing two children. This is illustrated in figure 3.4.

[1] Trees of fixed size can be written as strings of letters. Then, we have the same situation as in GAs.



(a) $x^2 + 2$ **(b)** $x + 2$

Figure 3.3. Shrinking mutation. We replaced the subtree $\text{Pow}(x, 2)$ with x. This is an example in which the size of the tree changes.

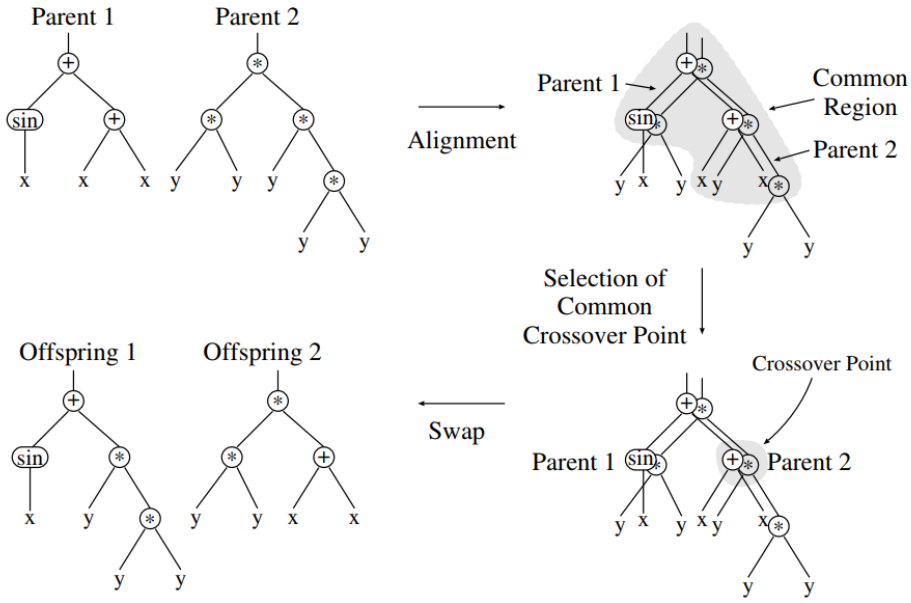


Figure 3.4. One-point crossover. Taken from [10].

3.4 Evolution

Equipped with crossover and mutation we can evolve a population. Given a data set we wish to model we proceed as follows:

(i) *Create an initial population.*

We generate random trees, limiting the size and depth.

(ii) *Determine the fitness of individuals.*

Calculate the χ^2 sum or some other measure chosen.

(iii) *Evolution.*

Based on the fitness, select parents for crossover and mutation and thus create new offspring. This is a evolutionary inspired heuristic search in the landscape of all possible functions one can build from the primitive set.

(iv) *Repeat (ii) and (iii) for a specified number of generations or stop if the fittest individual has the desired fitness.*

3.5 Symbolic Constants

From the description of SR so far there is one obvious shortcoming. Say we have a primitive set with operators add and mul and one terminal x. This is sufficient to build a polynomial with integer coefficients $\sum_i c_i x^i$, $c_i \in \mathbb{Z}$. If we wish to express a polynomial with non-integer coefficients, e.g. $x^2 + 3.14x$, the coefficients need to be terminals.

There have been different approaches how to implement numerical constants. For an overview see [11]. One approach which has emerged is what we will refer to as *symbolic constants*, which basically behave like variables. Let's go back to the polynomial mentioned above: If we add a symbolic constant c to the terminal set, we could build the function $x^2 + cx$. Using ordinary regression we can determine the best value of c. Using SR with symbolic constants thus adds an extra step to determine the optimal values for the symbolic constants.

That is, the evolutionary approach described will produce candidate models $f(x_i, c_i)$. One might call this step global optimization. Then, we determine the best values c_i to describe the given set of data. This step one might call local optimization.

3.6 Fundamental Limitations: No free lunch

We should think of GP as a highly parallel search strategy in the set of all possible programs one can build from the primitive set.[2] Each program has a fitness determined by one or more objectives. Thus, we refer to this set as the fitness landscape. See [12, Chapter 2] for examples and a discussion. The no free lunch theorem [13] states that averaged over all problems any two (search) algorithms perform equally well. It makes no assumption about the nature of the algorithm and thus covers all general-purpose optimization techniques such as GP and GAs. Thus, unless we put knowledge of the problem into the algorithm we cannot expect to perform better than random search. In other words, it is important to match the problem and algorithm. The parallel nature of GP and GAs may help avoiding getting stuck in a local minima and may thus make these strategies better than random search for a large class of problems, though it is not clear how to identify the "right" problems. One Ansatz is analysis of the fitness landscape [14]. The problem here is the dependence of the topology of the fitness landscape on the representation [12, section 2.7].[3]

[2] Parallel in the sense that every individual of a population is more or less independently searching for an optimum.

[3] This issue has not been properly addressed to my knowledge.

Chapter 4

SR setup

The requirements for a SR program for applications in physics are:

(i) It needs to be able to handle data with uncertainty. This is crucial in physics but surprisingly absent in the field of SR.

(ii) Symbolic constants: Physics is an exact science.

(iii) Open source.[1]

As we did not find a satisfying program fullfilling these requirements we decided to write our own SR program in the end.[2] In the rest of this section we describe the implementation and show some examples. In addition the (unfinished) documentation may be helpful.

4.1 Implementation

All code is written in Python and uses only open source libraries. Some of the ideas are inspired by *DEAP* [15], although the implementation is actually quite different. We heavily use the computer algebra system *sympy* [16].

4.1.1 The primitive set

Recall that the primitive set contains the building blocks: functions, a.k.a. primitives, and terminals.

- *Terminal Set*

  The terminal set can contain numerical constants, variables and symbolic constants. The variables can be functions of the "pure" variables. For example, if we know that the problem has a spherical symmetry we may use x^2 as variable.

- *Function set*

  Any function which is available in sympy, or user-defined functions build of sympy functions, can be used. Unlike many other approaches we do not define discontinuities as zero; solutions which are not well-defined are discarded in the end. Furthermore, the functions can have fixed arguments which are either numerical constants or symbolic constants. For example, one can use the function $x \mapsto x^k$, where k is a symbolic constant subject to regression.

[1] Research should be "in principle" reproducible, thus open source is strictly speaking not a requirement but certainly an advantage.

[2] Hopefully available on GitHub soon. For the moment feel free to ask the author for a copy. It's roughly 1300 lines of code.

4.1.2 Initializing a population

We generate random trees to create an initial population. These are always limited in depth and can also be limited in size. The depth of every leaf follows a binomial distribution: When generating a random (sub) tree at some point the internal nodes, i.e. functions, are connected to a number of new nodes which is equal to the arity of this particular function.[3] For each of these new nodes we pick, with equal probability, either a terminal or a function of the primitive set. If we choose to pick a terminal the probability of picking a variable is always 50%, whereas when picking a function each function of the primitive set has equal probability, although one could in principle use any probability distribution. There are three possible modes for initializing a population of a specified maximum depth d:

- *Random*

  The depth of all leafs of every tree follows a binomial distribution, as described above, cut off at d.

- *Full*

  The depth of all leafs of every tree is forced to equal d, i.e. all nodes are filled with functions until we reach the depth d.

- *Ramped half and half*

  Half of the trees are generated in the random mode, the other half in the full mode. This method was introduced in [1].

If we choose to limit the size we randomly create trees in one of the modes above and pick the ones which satisfy the size bound. The way we produce random trees always has a bias towards certain shapes and thus functions. For example the probability of picking a function as argument of a function scales with the arity. It is thus less likely to produce nested functions of the form $f_1(f_2(...), ...)$ if the outermost function has arity one than if it has arity two. To counter this we can change the probability distribution for functions of arity one, such that picking a function as argument is 75% and the probability of picking a terminal is 25%. This is the same probability distribution as for a function of arity two. We use this feature in the example in section 4.2.2.

4.1.3 Evaluation

An individual of a population is a model $f(x, c_i)$ with variable $x \in \mathbb{R}^n$ and parameters c_i, i.e. symbolic constants. In principle one can provide any method to calculate the fitness. We use the χ^2 sum. Given a candidate model f and a data set $\{x_i, y_i, y\_err_i\}$, where $x_i \in \mathbb{R}^n$ are input data and $y_i \in \mathbb{R}$ measured values with uncertainty $y\_err_i \in \mathbb{R}$, we first determine the optimal values $\{c*_i\}$ for the parameters using the Levenberg–Marquardt algorithm, see e.g. [17]. This is done using the curve_fit function in the python module scipy [18]. Then the fitness [4] $\|f\|_{\chi^2}$ is given by

$$\|f\|_{\chi^2} = \chi^2(\{c*_i\}) = \sum_i \left(\frac{f(x_i, \{c*_i\}) - y_i}{y\_err_i} \right)^2 .$$

[3] If we use functions with variable arity we randomly pick a number of arguments from a specified range. For example the sympy function *add* can take any number of arguments greater than 2.

[4] Note that the actual goodness of the fit is given by the reduced χ^2 sum which is given by χ^2/ndf with $ndf = \#datapoints - \#parameters$.

If $\|f\|_{\chi^2}$ turns out complex or NaN (not a number), e.g. because of poles, we discard the solution. The fitness is used to select individuals for mutation and crossover as described in the next section. One can still use another objective to choose the best solution.

Depending on the problem we can also do a validation step. Here the power of the computer algebra system sympy is useful. For example, we can ensure that our solutions satisfy given boundary conditions. An example is shown in section 4.2.3. There are two problems in this step:

(i) Checking algebraic properties can be very slow. Thus, they should be carefully chosen.

(ii) It can happen that most of the individuals are invalidated which results in a smaller gene pool available for later crossover. To counter this one can increase the mutation rate.

4.1.4 Genetic Operations

Based on the fitness we select individuals for one-point crossover and mutation, as described in section 3.3, to produce offspring. We use two methods:

(i) *Tournaments*

We randomly pick a number n of individuals of the population. The two best individuals are then used for crossover or mutation with probability *cross_pb* and $1 - $ *cross_pb*, respectively. In case of a mutation event only the best individual is mutated and added to the offspring.

(ii) *Roulette*

First, we remap the fitness such that one is the best possible value and zero the worst. Then, we use a probability distribution proportionate to the new fitness, simply by dividing by the size of the population, to pick two individuals subject to mutation or crossover as in the tournament.

Note that crossover and mutations can increase the number of symbolic constants. This can be limited, e.g. when using one-point crossover we can try different common nodes until we find offspring within the set bounds. If this is not possible, the offspring is discarded and we keep the parents. Using only one-point crossover reduces the bloat problem, see e.g. [12, 11.2], as trees cannot increase in size. Before we describe the algorithm we use, let us first introduce the concept of a hall of fame.

4.1.5 Hall of Fame

The evolutionary process will by means of natural selection return a final population with a high mean fitness. The best individuals may, however, have lived in earlier populations. Thus we use a hall of fame which keeps track of the best individuals throughout all generations. One can use the fitness or any other objective chosen to decide which individuals are kept in the hall of fame. It may be the case that there is more than one fitness objective. Say we use the χ^2 sum and the complexity or size, i.e. the number of nodes, as objectives. We can assign a fitness to an individual f as

$$\lambda_1\|f\|_{\chi^2} + \lambda_2\|f\|_{size}$$

for weights λ. Choosing useful weights is difficult and problem dependent. This Ansatz may also discard non-linear trade offs. An effective way of handling multi-objective fitness is *Pareto efficiency*. If we have several objectives, we say an individual dominates another one if it is

better w.r.t. all objectives. The Pareto front is the set of non-dominated functions. The concept is visualized in figure 4.1.

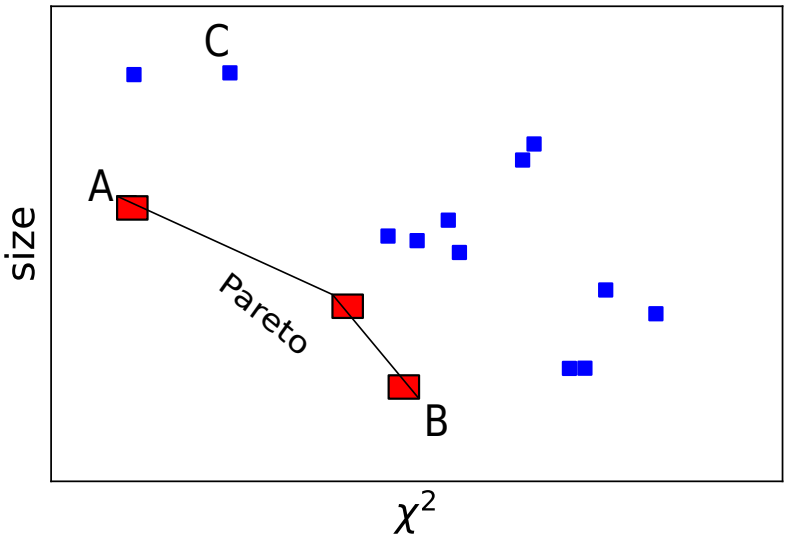


Figure 4.1. Visualization of Pareto front. The blue points are dominated by the red ones, e.g. A has a smaller size and χ^2-value than C and thus dominates C.The members of the Pareto front cannot be compared. B has a worse χ^2-value than A, but also a smaller size. We assume that both objectives are better for smaller values. The concept also works in the obvious way for objectives which are subject to maximization. In this case, the geometric picture is flipped.

For example one can keep track of a model which has a worse χ^2 value than the current best solution but has fewer parameters. Furthermore, it is possible to keep the hall of fame unique. Given a candidate for the hall of fame we can test if we already found the same solution before. This is done using the sympy module. This is, again, computationally expensive and it may be easier to use the χ^2 value to decide if two solutions are identical; the probability of having different models with exactly the same χ^2 value seems rather small.

It is of course also possible to collect many results from different runs with different initial population. Then, we can choose the best solutions as the Pareto front w.r.t. to chosen objectives or select results which satisfy given algebraic properties. If the fitness is already restrictive, in the sense that mostly functions with e.g. the right asymptotics have a good fitness, it is computationally more efficient to do these checks at this stage rather than during the evolution.

4.1.6 Age-fitness Pareto optimization

We use the Age-fitness Pareto optimization algorithm as described in [19]. The algorithm follows the general structure laid out in section 3.4.

Individuals have two fundamental properties: age and fitness. Each individual starts with age one. If it is passed to the next generation the age increases by one. During crossover or mutation events the offspring inherits the maximum age of its parents. The key idea is to pass non-Pareto dominated individuals to next generations, i.e. we discard solutions which are dominated w.r.t. age and fitness. This allows younger individuals to survive even if they have a worse fitness compared to older ones. In the following we describe our implementation:

(i) *Create an initial population*

This is done as described in section 4.1.2.

(ii) *Determine the fitness of individuals*

We use the χ^2 sum normalized by the number of data points.[5] In general this includes fitting parameters, i.e. symbolic constants, first. Depending on the problem we may, as described in section 4.1.3, add a validation step. Furthermore, the hall of fame is updated.

(iii) *Evolution*

There are two steps: First we add offspring to the population: $breed\_rate \times size(population)$ times we use tournament or roulette to produce offspring by means of crossover or mutation. By default we use tournaments with probability for crossover $cross\_pb = 0.9$ and set $breed\_rate = 0.5$. Additionaly, $0.05 \times size(population)$ new random individuals are added. In the second step, we randomly choose 50 individuals and remove the Pareto-dominated (w.r.t. age and fitness) individuals. This is repeated until the size of the population is smaller or equal to the size of the initial population.

The second and third step are repeated a specified number (*generations*) of times.

4.2 Artificial Problems

In this section we show some simple artificial examples in which we know the original equation. This is not supposed to be a benchmark, rather it demonstrates simple usage of our implementation. Knowing the equation, we pick our function sets accordingly, which of course increases our success rate immensely compared to real problems, in which we don't know the answer. As SR is not deterministic, we restart the algorithm for each problem ten times to demonstrate that the results are not just the product of pure luck, but are found with a very high probability. A single run takes $\sim \mathcal{O}(1\text{min})$ on a quad-core notebook.[6] Increasing the population size and number of generations does often, but not always, improve the results, at the price of a longer execution time.

Table 4.1. Python code used to produce data with 10% error.

```
import numpy as np
def f(x):
        return 3.14*x**4 + 0.5*x**3 + x**2 + 2*x
x = np.linspace(-1,1,100)
y = f(x)
#Measurement
y_i = [np.random.normal(y,np.abs(0.1*y)) for i in range(10)]
y = np.mean(y_i, axis = 0)
y_err = np.std(y_i, axis = 0)
```

[5] We merely do this for convenience, as we would have to use the p-value to compare fitness when using χ^2/ndf. Using χ^2 rather than χ^2/ndf may result in over fitted models. Thus, for the final results one should always calculate χ^2/ndf.

[6] We do not use parallel computing, although this is supported in our implementation.

4.2.1 Quartic Polynomial

Very often the quartic polynomial

$$x^4 + x^3 + x^2 + x$$

is used as demonstration for SR. Using symbolic constants we are able to find such polynomials with arbitrary coefficients. We generate data for the polynomial

$$3.14x^4 + 0.5x^3 + x^2 + 2x$$

using the code in table 4.1. We run the Age-fitness Pareto optimization ten times over ten generations with an inital population of size 300 with the settings given in table 4.2.

Table 4.2. Parameters for SR runs.

Function Set:	add, mul
Terminal Set:	x_0, c_0
Initial Population:	ramped half-and-half, depth = 4, sizelimit = 40
Algorithm:	age_fitness with default settings

Within small variation of the coefficients, we find the correct solution in every run.

4.2.2 Gaussian

A somewhat more complex and interesting problem is a Gaussian function. With the code as in table 4.1 we produce data for

$$\exp\left(-\frac{(x-1)^2}{2}\right),$$

i.e. a non-normalized Gaussian with mean = sigma = 1. With the changed probability distribution for the initial population as described in the end of section 4.1.2, we used the Age-fitness Pareto optimization ten times over ten generations with an initial population of size 1000. We used the same settings (table 4.2) as before, except we added the exponential function to the function set. In nine out of the ten runs we found the exact solution.

4.2.3 Line with boundary conditions

We produce data for the linear function $x \mapsto x$ in the interval $[-1, 1]$ and use SR to look for solutions which do not diverge as $x \to \infty$. Using the function set add, mul, and $x \mapsto x^{-1}$ we find in each run functions such as

$$\frac{x}{0.01x^2 + 1},$$

which goes to zero as $x \to \infty$. Expanding this yields

$$x - 0.01x^3 + O\left(x^4\right),$$

which shows that for small x the linear term is dominant.

4.2.4 Fermi Distribution

As a physical example we take the Fermi distribution

$$n(\epsilon) = \frac{1}{\exp((\epsilon - \mu)/T) + 1},$$

where ϵ is the energy, μ the chemical potential and T the temperature (in natural units with $k_B = 1$). We set $\mu = 10^{-4}$eV and $T = 3.44 \times 10^{-4}$eV, which corresponds to 4K. The value of μ is typical for He^3 at low pressure as $T \to 0$ [20][table 4.1]. We produce sample data as before. Using the function set add, mul, exp and $x \mapsto x^{-1}$ and the same settings as in the example in section 4.2.2 we find solutions of the form

$$\frac{1}{\exp(c_0\epsilon) + c_1},$$

in each of the ten runs. This is the Fermi distribution for $\mu = 0$. To account for the non-vanishing chemical potential the optimal value of c_1 differs from one. The optimal value of c_0 is 0.27, which is close to 0.29K^{-1}, the inverse temperature of our model. So even tough we don't find the exact equation with the stated settings, we are still able to extract the temperature with a relatively high precision.

4.3 Further ideas

There is a plethora of ideas how to improve SR. Below we list some examples of different difficulty (in terms of implementation).

- *Simplification operator*

  It seems straightforward to use sympy to create a simplification operator.

- *Multiple Regression GP*

  In [21] the authors introduce a systematic way of improving GP results by decoupling the programs (functions in our case) into genes and determining the best linear combination of sub-expressions (combination of genes).

- *Use Machine Learning*

  It should be possible to implement something similar to "Estimation of Distribution Algorithms" as described in [10]. We already use some specified probability distributions to generate random trees. The idea now is to learn correlations of the building blocks from the solutions produced in each generation. Then we use this knowledge to generate the (in this case less-) random trees we inject in every generation.

Chapter 5

Applying SR to Heavy-Ion Physics

We analyzed several transverse-momentum (p_T) spectra for p-p and Pb-Pb collisions and two R_{AA}-spectra. We use natural units throughout this section. The p_T spectra describe the distribution of produced particles as function of p_T in nucleus-nucleus collisions. It is given by the invariant yield[1] defined as

$$E\frac{d^3N}{d\vec{p}^3} = \frac{d^2N}{2\pi p_\mathrm{T} dy dp_\mathrm{T}},$$

where $\vec{p}$ is the momentum of the produced particles (with p_T the momentum transverse to the beam), E the energy, and y the rapidity co-linear to the collision. There are physical models to fit p_T-spectra in p-p collisions. For small $p_\mathrm{T} < 1-2$ GeV the spectra are described by an exponential with inverse slope giving an effective temperature [22, chapter 2]. The QCD-inspired Hagedorn function describing the invariant yield is [23, eq. (B.15)]

$$A\left(\frac{p_0}{p_\mathrm{T}+p_0}\right)^n = \begin{cases} A(1-\frac{n}{p_0}p_\mathrm{T}) \approx A\exp\left[-\frac{n}{p_0}p_\mathrm{T}\right] & \text{for } p_\mathrm{T}\to 0 \\ A\left(\frac{p_0}{p_\mathrm{T}}\right)^n & \text{for } p_\mathrm{T}\to\infty \end{cases} \tag{5.1}$$

We will refer to these kind of power laws as Hagedorn-type functions. For Pb-Pb collisions, however, no parametrizations are known. Interpolated p_T distributions are e.g. used to determine the number of direct photons in cocktail simulations, see e.g. [24].

The R_{AA} is the ratio of the p_T spectra in A-A and p-p collisions scaled by $\langle T_\mathrm{AA}\rangle = \langle N_{\text{ coll}}\rangle / \sigma^\mathrm{NN}_\mathrm{inel}$. For Pb-Pb it is observed to be smaller than one and this indicates a suppression of particle production in the Quark-Gluon Plasma, see e.g. [25, Lecture 8] and references therein.

We filter the SR results for p_T spectra for solutions which go to zero as $p_\mathrm{T}\to\infty$, which energy-momentum conservation demands. This is done using sympy. As computing limits can be quite slow we generally only do this for the Pareto fronts, i.e. the best results. Most results could be further simplified by introducing new constants, e.g. we could replace things like $\frac{e^{c_1}}{c_2}$ with a new constant $\frac{e^{c_1}}{c_2} = c_1'$. Generally we do, however, write the functions as found by the computer. The argument is generally always named x_0.[2]

[1] Sometimes the yield $= \frac{d^3N}{d\vec{p}^3}$ is used. This is not Lorentz invariant.

[2] Sometimes the units would be wrong. Then, one should think of x_0 as the physical variable divided by its unit, i.e. p_T/GeV most of the time.

5.1 J/Ψ p_{T}-spectrum

The p_{T} spectra of J/Ψ mesons are typically (e.g. in [26]) parameterized with a modified Hagedorn function:

$$f(p_{\mathrm{T}}) = C_0 \frac{p_{\mathrm{T}}}{(1 + (p_{\mathrm{T}}/p_0)^2)^n}, \tag{5.2}$$

for parameters C_0, p_0, and n. Note that here we describe the yield and not the invariant yield. We analyzed one J/ψ spectra from LHCb (pp, 8 TeV)[27]. The spectra fitted with eq. (5.2) is plotted in figure 5.1. The optimal parameters found are listed in table A.1. The goodness of the fit is $\chi^2/ndf = 2.59$.

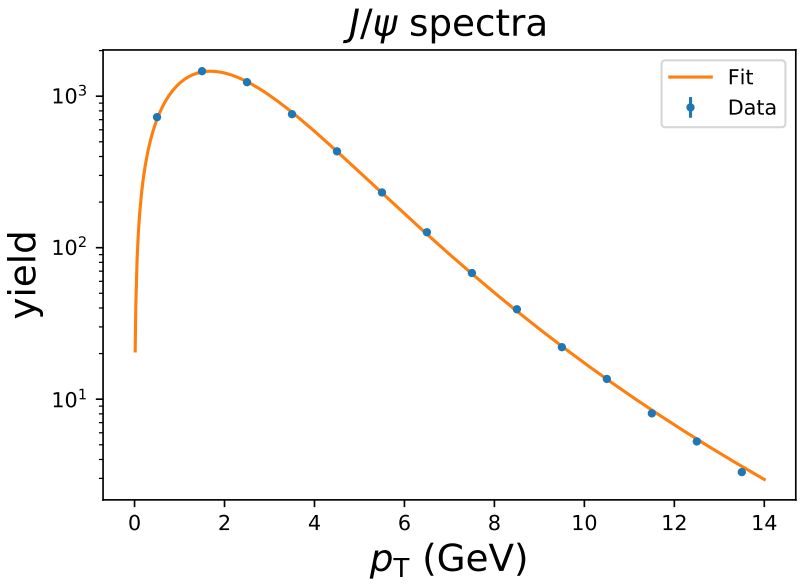


Figure 5.1. J/Ψ spectra. Data from [27].

Without knowing (5.2), we run the Age-fitness Pareto optimization 100 times over 40 generations with an initial population of size 1000 with the settings given in table 5.1. The best results we found are plotted with χ^2/ndf vs. size and with χ^2/ndf vs. number of parameters in figure 5.2.

Table 5.1. Parameters for SR run.

Function Set:	add, mul, exp, $x \mapsto x^{-1}$, $x \mapsto x^k$
Terminal Set:	x_0, c_0
Initial Population:	ramped half-and-half, depth = random(3,4,5), sizelimit = 40
Algorithm:	age_fitness with default settings[3]

The best functions found are listed in the tables A.3 and A.4. We did not find exactly the same expression as eq. (5.2) but several similar functions. The most similar function we found is

$$c_0 \left(c_1 + p_{\mathrm{T}}^2\right)^{k_1} \left(\frac{1}{p_{\mathrm{T}}}\right)^{k_0} \tag{5.3}$$

with optimal parameters listed in table A.2. Pulling out c_1 we get

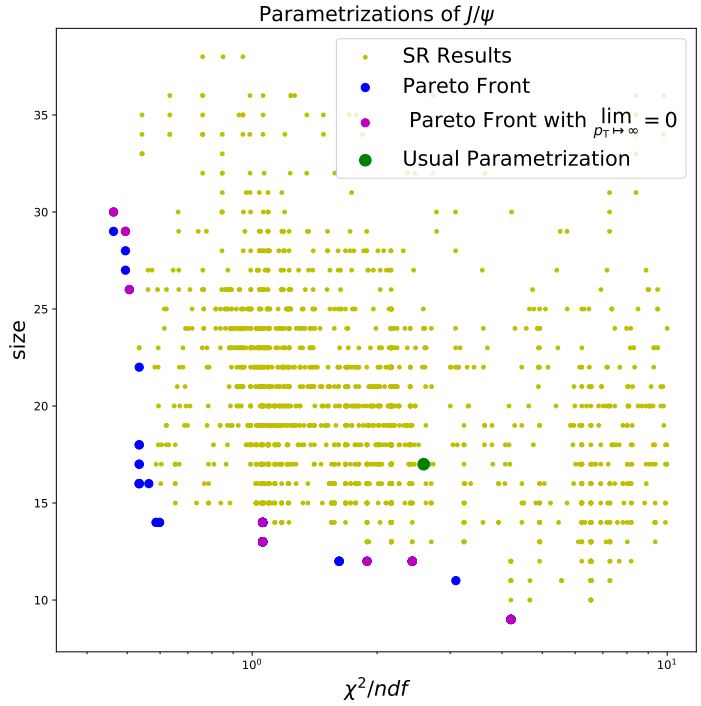
$$c_0 c_1^{k_1} p_{\mathrm{T}}^{-k_0} \left(1 + \frac{p_{\mathrm{T}}^2}{c_1}\right)^{k_1}. \tag{5.4}$$

[3] breed_rate = 0.5, cross_Pb = 0.9, depth_limit = 5, size_limit = 40

This is almost the same as eq. (5.2). The parameters p_0 and C_0 correspond to c_0 and c_1 but in eq. (5.4) c_0 and c_1 are correlated. Furthermore, the linear factor p_T in eq. (5.2) gets replaced by $p_{\mathrm{T}}^{-k_0}$, for a new parameter k_0. The optimal value is close to -1. Looking at eq. (5.4) and knowing it was found by a computer one might argue that it is canonical to alter it slightly to get

$$c_0 p_{\mathrm{T}}^{-k_0} \left(1 + \frac{p_{\mathrm{T}}^2}{c_1}\right)^{k_1} . \tag{5.5}$$

Notice that we did not use a square function in the function set. The term p_{T}^2 was realized as $mul(p_{\mathrm{T}}, p_{\mathrm{T}})$.



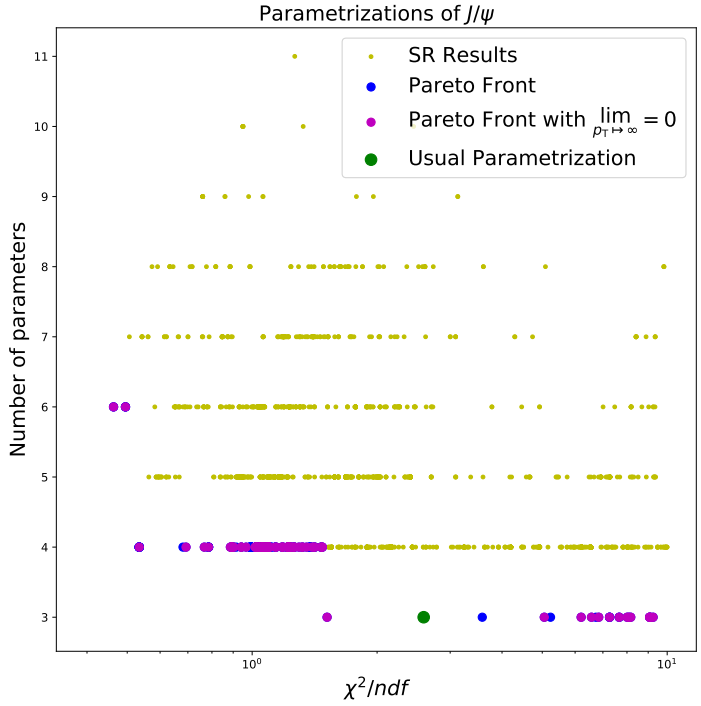


Figure 5.2. Visualization of results of SR for J/Ψ p_{T}-spectra. The size is the number of nodes in the trees, which we use as a measure for the complexity of the functions. In the left figure it seems we found many parametrizations better than eq. (5.2). But when looking at the number of parameters per model, eq. (5.2) is part of the Pareto front.

5.2 p_T-spectra for p-p collisions

We analyzed pion, kaon and proton p_T spectra (pp, 2.76 TeV) with the data reported in [28]. They are plotted in figure 5.3. The goal was to find a function describing all three spectra, i.e. a

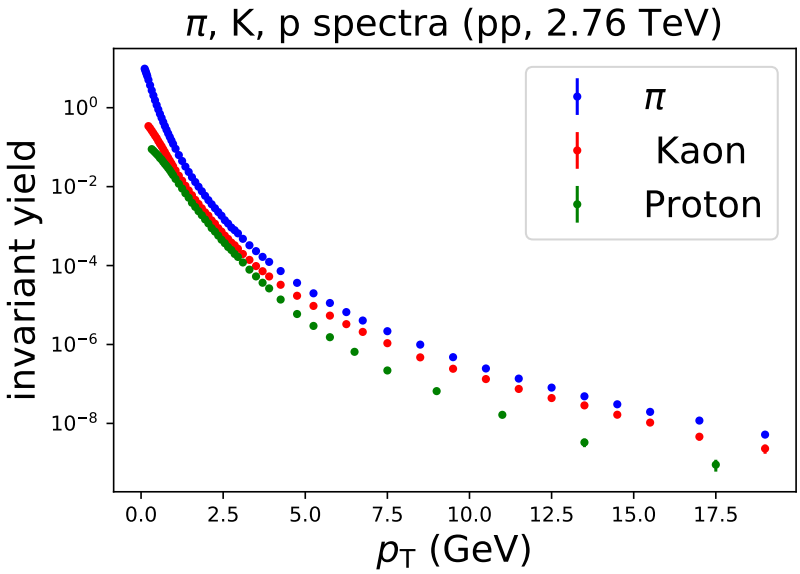


Figure 5.3. π, K, p spectra (pp, 2.76 TeV

function with different optimal values for the parameters for the different species.[4] We run the Age-fitness Pareto optimization 100 times over 30 generations with an initial population of size 1000 with the settings given in table 5.2.

Table 5.2. Parameters for SR run.

Function Set:	add, mul, exp, $x \mapsto x^{-1}$, $x \mapsto x^k$
Terminal Set:	x_0, c_0
Initial Population:	ramped half-and-half, depth = random(3,4), sizelimit = 40
Algorithm:	age_fitness with default settings.

The best function found by the computer is $\exp(\exp(c_0))\left(\exp(c_1)+p_T^{k_0}\right)^{k_2} p_T^{-k_1}$. We can simplify this to

$$c_0 p_T^{k_0}(c_1 + p_T^{k_1})^{k_2}. \tag{5.6}$$

This is a modified version of the Hagedorn function (eq. (5.1)).[5] The equation is plotted in figure 5.4 with the optimal values as listed in the tables B.1, B.2, and B.3. The fits were done with the MINUIT algorithm[29] via the iminuit[30] Python interface. For high p_T values the fit is slightly off, especially for the pion and proton spectra.

One approach used in particle physics is the Tsallis model (TM) [31]:

$$\frac{d^2N}{2\pi p_T dp_T dy} \propto \left(1 + \frac{q-1}{T} m_T\right)^{-1/(q-1)} \tag{5.7}$$

[4] It is of course possible to find better functions for the individual species, which we indeed did: There were solutions for the kaon and proton spectra with $\chi^2/ndf < 2$ and $\chi^2/ndf < 4$ for pions.

[5] Note that with $k_0 = 1$ eq. (5.6) would describe the yield rather then the invariant yield. So $k_0 = 1$ would be wrong. Indeed, the optimal value is not 1.

Here $m_{\mathrm{T}} = \sqrt{m^2 + p_{\mathrm{T}}^2}$, where m is the mass of the particle, is the transverse mass. The TM fit was worse for kaons and protons, but performed better for the pion. Like eq. (5.6) high p_{T} values are not described well.[6] The χ^2/ndf-values are listed in table 5.2.

Table 5.3. Goodness of fits. For the kaon and proton spectra we can fix the parameters $c_1 = 2 \times mass$ and $k_1 = 3/2$.

	SR		TM
species	χ^2/ndf	χ^2/ndf with fixed para	χ^2/ndf
K	2.72	3.59	5.78
p	3.55	3.60	6.01
π	13.28	-	4.61

[6]The TM model assumes thermal equilibrium, which may not be given for high p_{T} as pointed out in [31]. In practice it is still used for an arbitrary range. One approach is to use a two component fit; the spectra is split into low (soft) and high (hard) p_{T} components which are fitted separately.

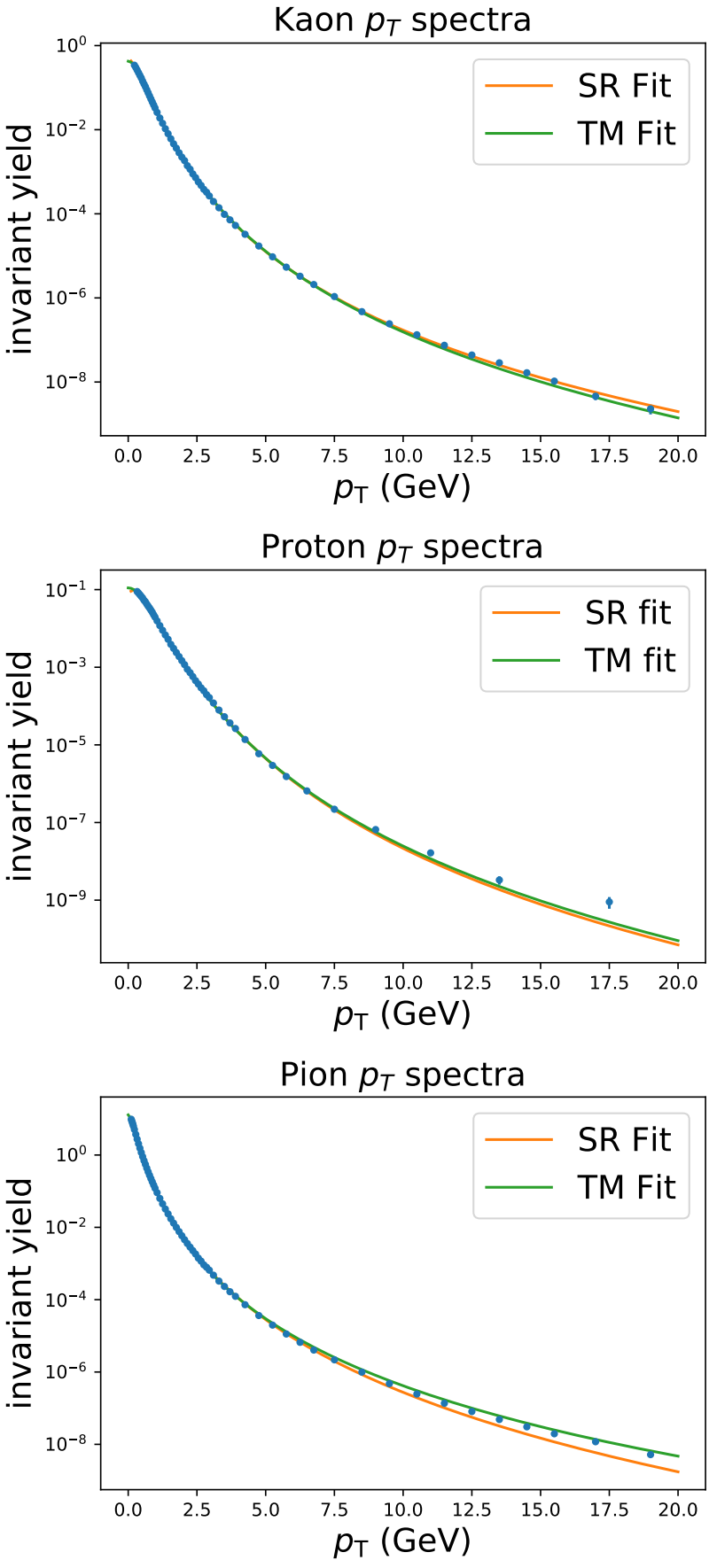


Figure 5.4. π, K, p spectra fitted with eq. (5.6) (SR) and the Tsallis Model (TM).

5.3 p_T-spectra for Pb-Pb collisions

We took data for pion, kaon and proton p_T spectra (Pb-Pb, 2.76 TeV, 0-5%) from [32]. They are plotted in figure 5.5. We tried finding a solution describing all three spectra as before;

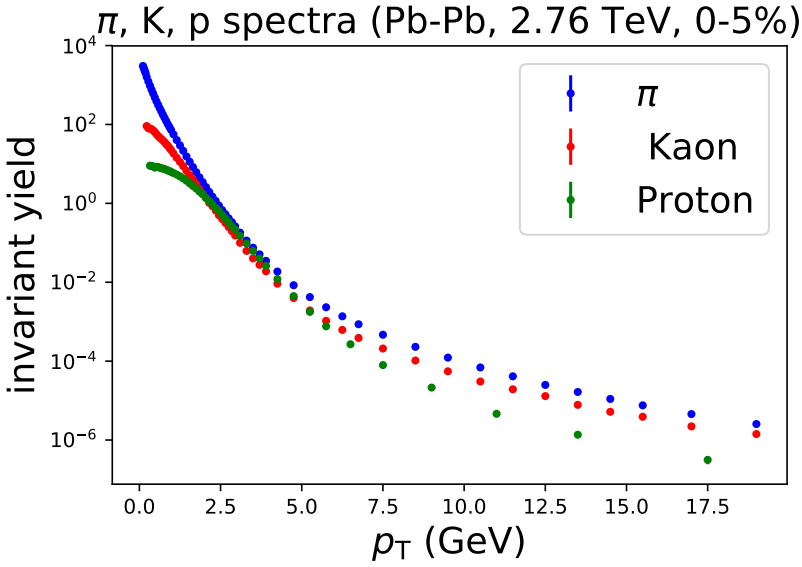


Figure 5.5. π, K, p spectra (Pb-Pb, 2.76 TeV)

the ones we found were good for one species and off for high p_T for the other two species. Nevertheless, the search for each of the species produced results which turned out to describe all spectra.

For each species we run the Age-fitness Pareto optimization 100 times over 100 generations with an inital population of size 1000 with the settings given in table 5.4.

Table 5.4. Parameters for SR run.

Function Set:	add, mul, exp, log, $x \mapsto x^{-1}$, $x \mapsto x^k$
Terminal Set:	x_0, c_0
Initial Population:	random if depth> 4, else ramped half-and-half,
	depth = random(3,4,5,6,7), sizelimit = None
Algorithm:	age_fitness with
	size_limit = None, depth_limit = 15, n_const_limit=None.

The solutions with $lim_{p_T \to \infty} = 0$ for the pions and kaons are listed in the tables C.1 and C.2. The Pareto Fronts for the kaons and pions are visualized in figure 5.6 and for the protons in figure 5.7, respectively. For the pion spectra the best model we found diverges for $p_T \to 0$. The second best solution is

$$\left(p_T^{k_2} + \left(\frac{1}{p_T}\right)^{k_1}\right)\left(c_0 p_T + \left(p_T^3\right)^{k_0}\right)\left(\frac{1}{c_2 + c_3 p_T + p_T^{k_3}}\right)^{k_4} e^{-c_1}. \tag{5.8}$$

with the optimal parameters, as in table C.1, eq. (5.8) goes to zero as $p_T \to 0$. Let us rewrite it as [7]

$$c_1 p_T^{k_2} f^\pi(p_T) + c_1 p_T^{k_1} f^\pi(p_T), \tag{5.9}$$

[7] The regression actually fails when using c_1 instead of e^{-c_1}.

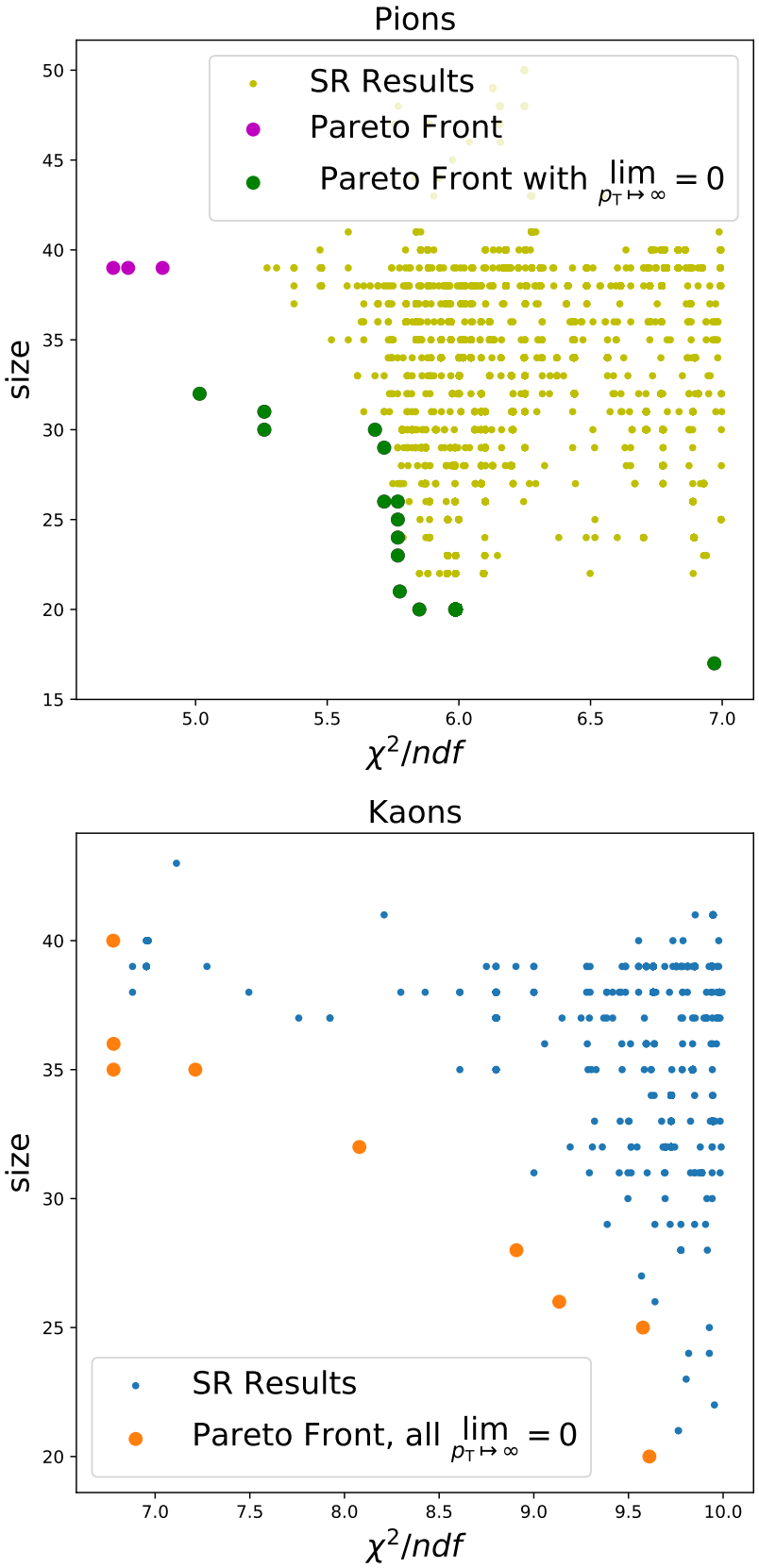


Figure 5.6. Best SR fits for pion and kaon p_T spectra for Pb-Pb. There are less good solutions for the kaon than for the pion.

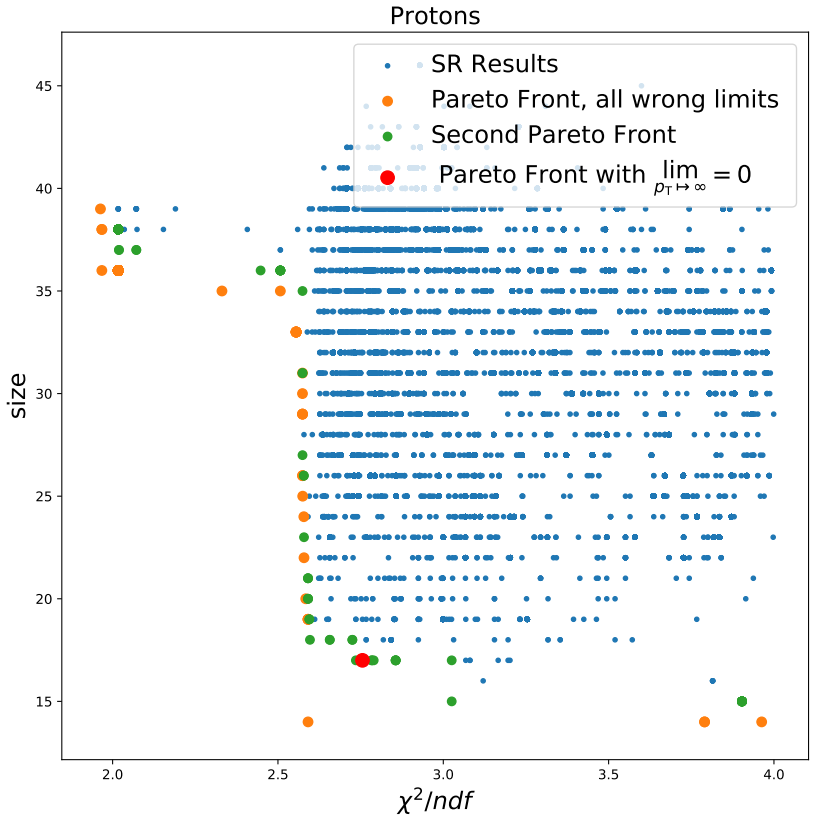


Figure 5.7. Best SR fits for proton p_T spectra for Pb-Pb. The solutions in the first Pareto front all diverge for $p_T \to \infty$. In the second Pareto front (i.e. the Pareto front of the set without the Pareto front) there is one solution satisfying energy momentum conservation.

where $k_1 \mapsto -k_1$, $e^{-c_1} \mapsto c_1$, and

$$f^{\pi}(p_T) = \left(c_0 p_T + \left(p_T^3\right)^{k_0}\right) \left(\frac{1}{c_2 + c_3 p_T + p_T^{k_3}}\right)^{k_4}.$$

Eq. (5.9) could be described as a sum of two modified Hagedorn-type functions with a shared amplitude c_1.

Another interesting function we found is

$$\left(c_1 + (p_T/c_0)^{k_0}\right) \left(\left(\frac{1}{p_T + p_T^{k_1}}\right)^{k_2} + e^{-p_T^{k_3}}\right)^{k_4}. \tag{5.10}$$

The factor $\left(\frac{1}{p_T + p_T^{k_1}}\right)^{k_2}$ is close to the Hagedorn function itself [8], and

$$\left(\left(\frac{1}{p_T + p_T^{k_1}}\right)^{k_2} + e^{-p_T^{k_3}}\right)^{k_4}$$

is then a kind of recursive, modified Hagedorn-type function. A similar exponential term in a Hagedorn function was also used e.g. in [33, eq. (30)]. Both solutions can also be fitted to

[8] We also found one solution in which the linear factor p_T was replaced by a constant. This resulted in a slightly worse fit.

the kaon and pion spectra. All fits are plotted in figure 5.8. There is no visible offset, except for the protons with eq. (5.8) fitted. The χ^2/ndf values are given in table 5.5, the optimal values for the parameters can be found in the tables C.3, C.4, and C.5.

Table 5.5. Goodness of fits.

	SR Fit 1 eq. (5.8)	SR Fit 2 eq. (5.10)
species	χ^2/ndf	χ^2/ndf
K	11.25	11.02
π	5.26	5.85
p	2.86	9.8

We also tried looking for solutions for the pions which take the transverse mass $m_\mathrm{T} = \sqrt{p_T^2 + m^2}$ as argument.[9] The best function we found is

$$\left(c_0 + m_\mathrm{T}^{0.5k_0}\right)\left(c_2 m_\mathrm{T}^{0.5} + m_\mathrm{T}^{0.5k_2}\right)^{k_3}\left(c_1 + m_\mathrm{T}^{0.5} + m_\mathrm{T}^{0.5k_1}\right), \tag{5.11}$$

with optimal values for the parameters given in table C.6. The $\chi^2/ndf = 6.80$. The fit is plotted in figure 5.9. The best solutions for the kaon (table C.2) is plotted in figure 5.10.

The best solution for the proton is

$$\log\left(c_0 + \left(c_1 p_\mathrm{T}\left(p_\mathrm{T} p_\mathrm{T}^{k_0} + 1/c_2\right)\right)^{k_1}\right)^{-k_2}. \tag{5.12}$$

The inner part

$$\left(c_1 p_\mathrm{T}\left(p_\mathrm{T} p_\mathrm{T}^{k_0} + 1/c_2\right)\right)^{k_1}$$

is almost the same as the Hagedorn function (5.1) describing the yield, except the linear factor has the exponent k_1. It is plotted in figure 5.11. The fit is quite good with $\chi^2/ndf = 2.76$. The optimal parameters are listed in table C.7.

[9] If this is optimal, SR should be able to find, given enough time, a functional form in terms of m_T. As mentioned in section 3.6, giving more information always improves the algorithm; by specifying the argument we drastically shrink the search space.

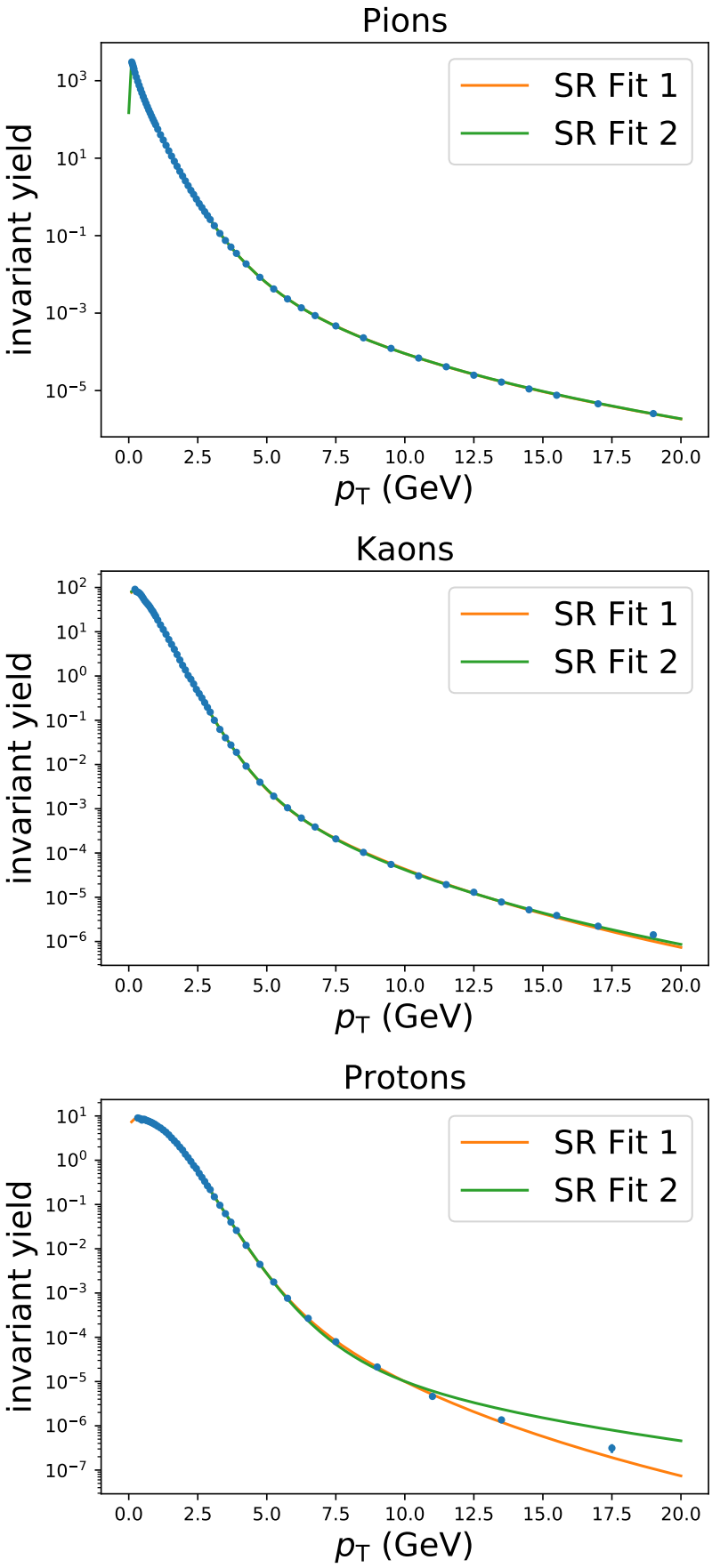


Figure 5.8. π, K, p p_T spectra (Pb-Pb, 2.76 TeV) fitted with eq. (5.8) (SR Fit 1) and eq. (5.10) (SR Fit 2).

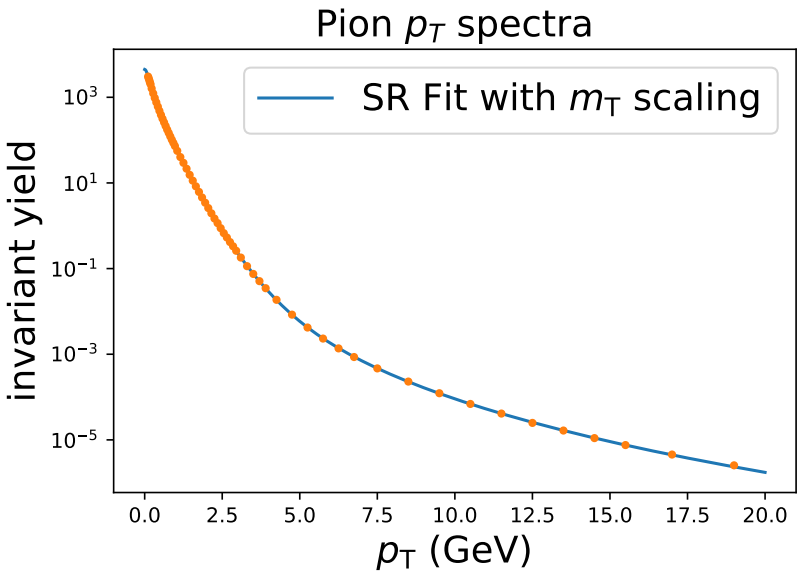


Figure 5.9. Pion p_T spectra (Pb-Pb, 2.76 TeV) fitted with eq. (5.11).

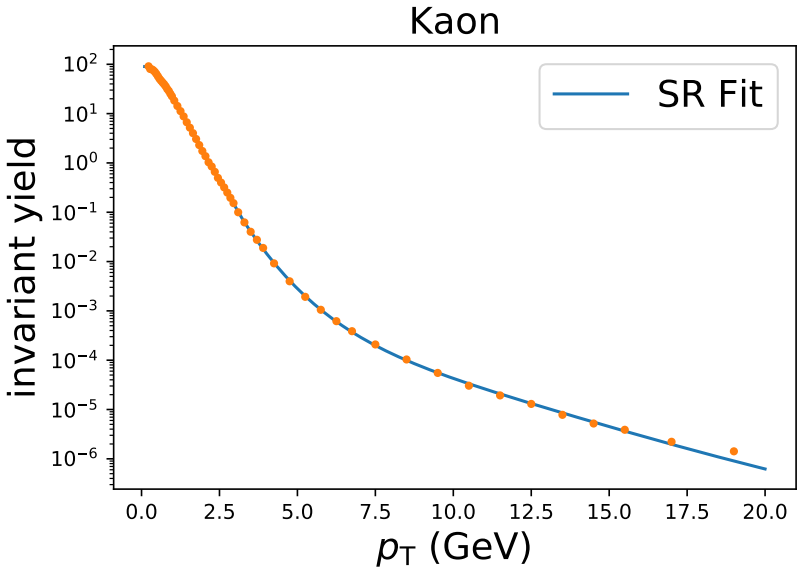


Figure 5.10. Kaon p_T spectra (Pb-Pb, 2.76 TeV) fitted with the first function in table C.2.

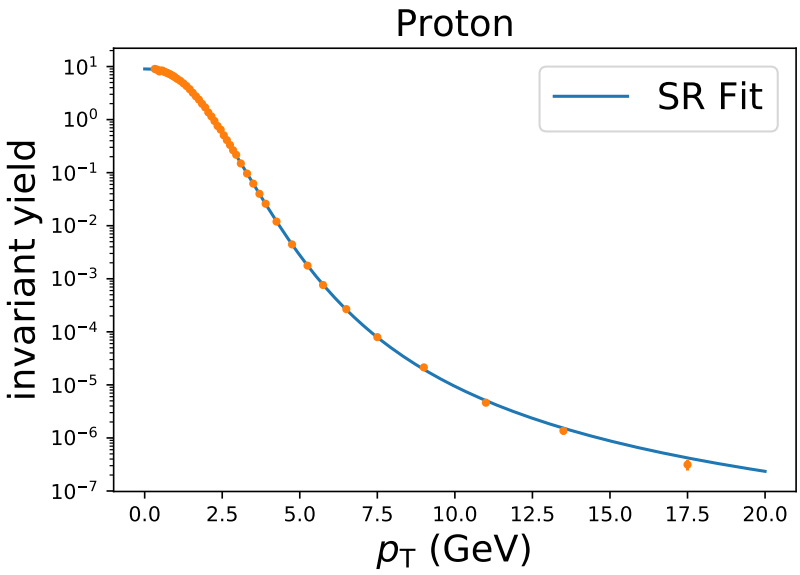


Figure 5.11. Proton p_{T} spectra (Pb-Pb, 2.76 TeV) fitted with eq. (5.12).

5.4 Charged-hadron R_{AA}

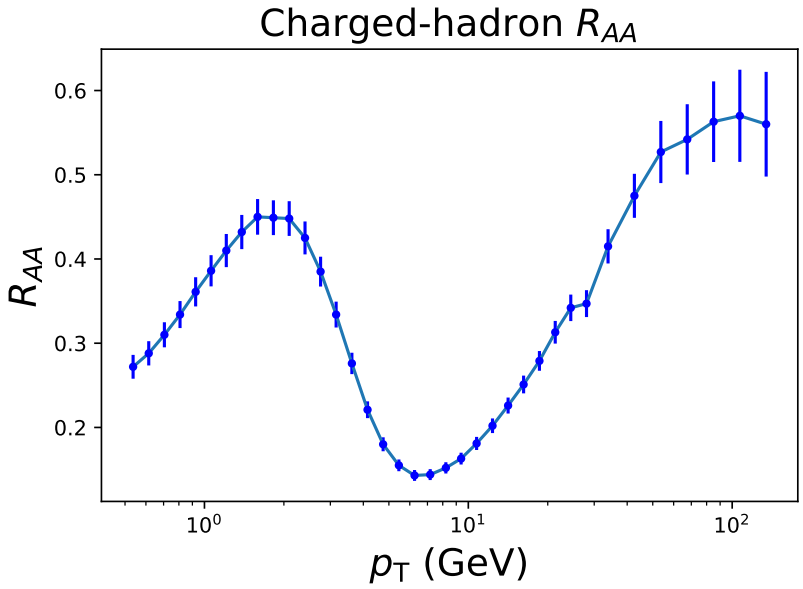


Figure 5.12. Charged-hadron R_{AA} (Pb-Pb, 2.76 TeV, 0-5%, ATLAS).

We analyzed a charged-hadron R_{AA}. (Pb-Pb, 2.76 TeV, 0-5%, ATLAS). The data was taken from [34]. We run the Age-fitness Pareto optimization 100 times over 30 generations with an initial population of size 1000 with the settings given in table 5.6.

Table 5.6. Parameters for SR run.

Function Set:	add, mul, exp, log, sin, cos, $x \mapsto x^{-1}$, $x \mapsto x^k$
Terminal Set:	x_0, c_0
Initial Population:	random if depth>4, else ramped half-and-half, depth = random(3,4,5,6), sizelimit = 40
Algorithm:	age_fitness with default settings.

We found several good fits with quite different asymptotic behavior. The solutions are plotted with χ^2/ndf vs. size in figure 5.13.
Demanding finite limits invalidates most solutions as can be seen in figure 5.13. The best functions in the Pareto front (w.r.t. χ^2/ndf and size) of solutions with constant limits are plotted in figure 5.14.
We also tried to fit another data set (Pb-Pb, 5.02 TeV, 0-5%, CMS) taken from [35]. The data is plotted in figure 5.15. The search with the same settings as before was not successful. We only found solutions which were completely wrong after the first peak. This may be because of the last three data points which seem a bit disconnected to the ones before. On the other hand the uncertainty is large for these points.

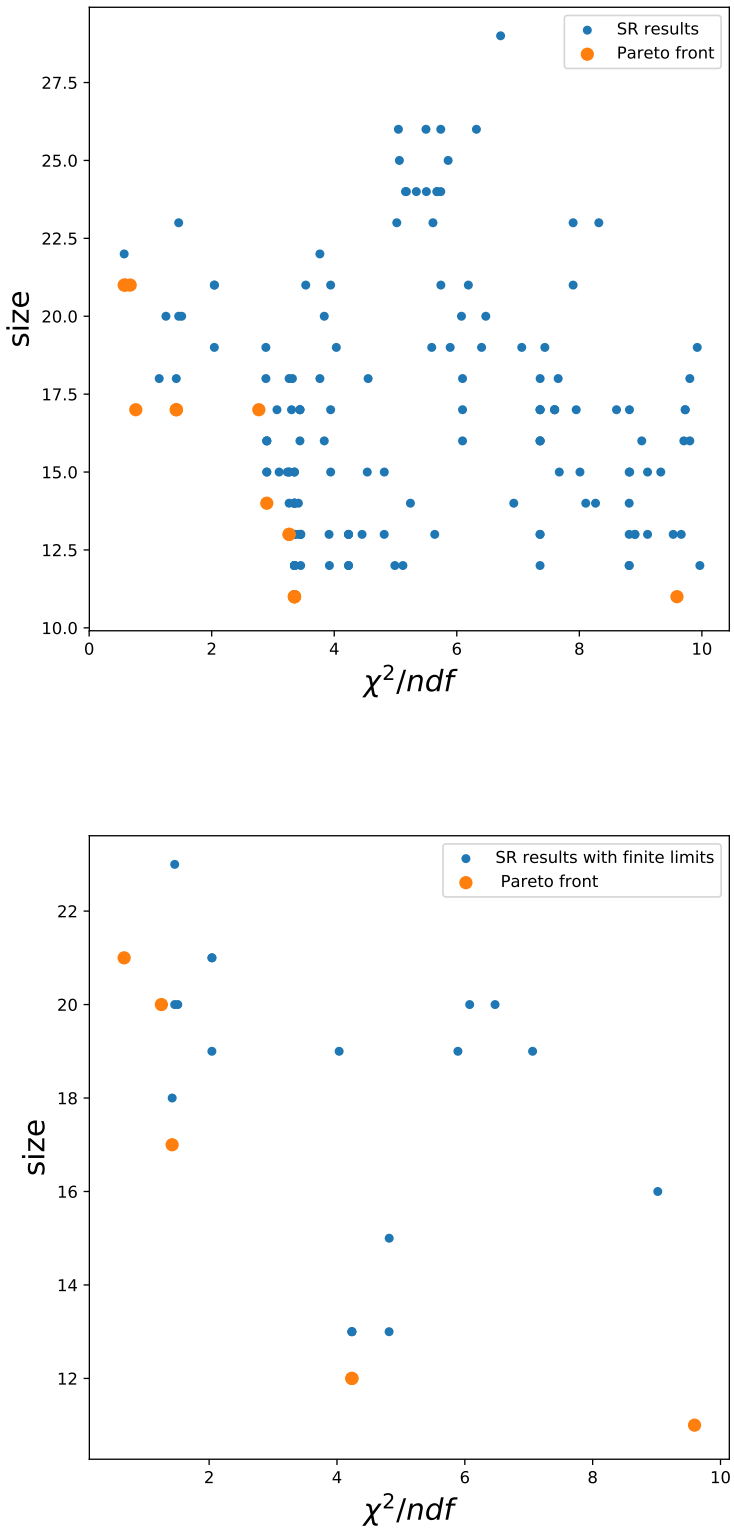


Figure 5.13. Best SR fits for charged-hadron R_{AA} (Pb-Pb, 2.76 TeV, 0-5%, ATLAS). Demanding finite limits invalidates most solutions.

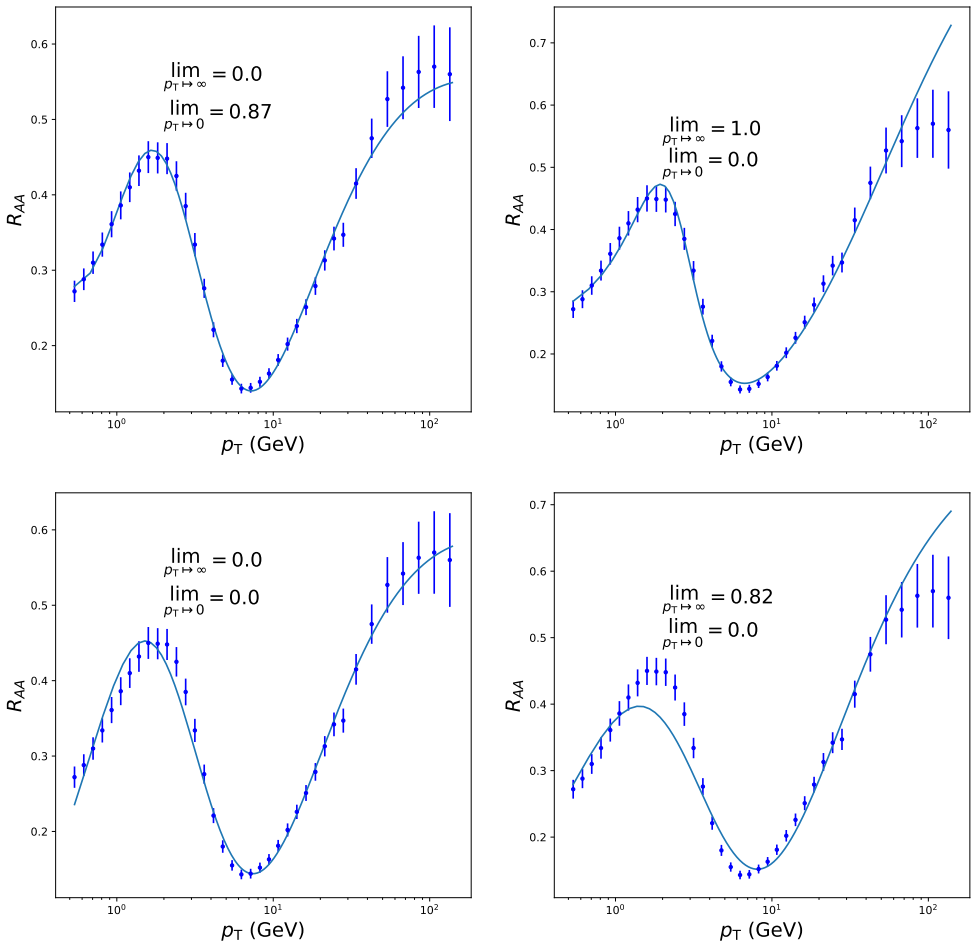


Figure 5.14. SR Fits for charged-hadron R_{AA} (Pb-Pb, 2.76 TeV, 0-5%, ATLAS) with constant limit as $p_T \mapsto 0, \infty$. They correspond to the four best functions listed in table D.1.

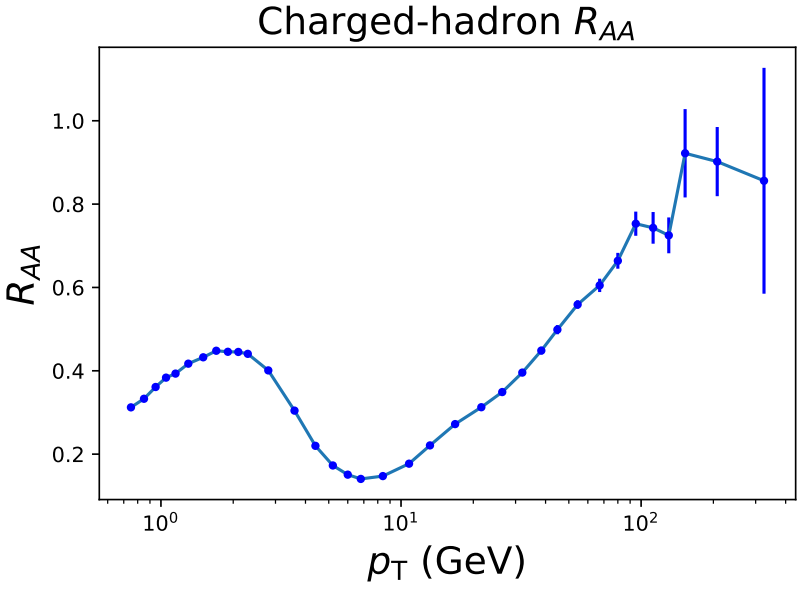


Figure 5.15. Charged-hadron R_{AA} (Pb-Pb, 5.02 TeV, 0-5%, CMS).

Chapter 6

Conclusion and Outlook

We investigated SR and its applications to heavy-ion physics. We wrote a Python module to perform SR supporting, in particular, symbolic constants and data with uncertainty. While these are not novel ideas, there is a lack of SR programs combining these, or similar, features. On one hand, there are general-purpose GP libraries with abstraction layers such as *DEAP* [15], which allow for user-defined fitness. In particular, the χ^2 sum can be easily implemented as fitness. At the same time, however, the abstraction-layer makes it difficult to implement symbolic constants and increases the effort for usage, which makes it less attractive for domain experts. On the other hand, there are ready-to-use SR programs like *ffx* [6] and the *FindForumala* function in Mathematica.[1] These can be easily integrated into the workflow of domain experts. The simple API comes at the price of functionality and both programs, unfortunately, do not support data with uncertainty. In this sense, we took a step towards filling the gap and make SR principally usable for domain experts in physics. Our case studies show that SR can be a useful tool: For eight out of nine data sets our SR approach found functions describing the given data sets. We discovered generalizations of Hagedorn-type functions which describe p_{T} spectra for p-p and Pb-Pb collisions. The use of the exponential function in the function set and the lack of solutions using the exponential suggest that Hagedorn-type functions are indeed the simplest models describing p_{T} spectra.[2] Furthermore, in one case we were able to model the R_{AA} for charged hadrons.

It is common practice in particle physics to "guess" functions for interpolation. SR could be a time-saving alternative. Given that there is much room for improvement it seems possible that SR will become a standard tool for modeling data with specified asymptotics and other algebraic properties, once a certain level of speed, quality, and user-friendliness is reached. The *white-box* nature of SR solutions allows for deep insight. This may inspire phenomenological models, provided the data is good enough to restrict the models to give unique and physical results, e.g. when parameters can be interpreted physically.

> "We can only see a short distance ahead, but we can see plenty there that needs to be done."
>
> *Alan Turing [7]*

[1] https://reference.wolfram.com/language/ref/FindFormula.html

[2] As SR is a heuristic technique one should be careful drawing strong conclusions.

Bibliography

[1] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[2] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[3] Christopher Hillar and Friedrich Sommer. Comment on the article" distilling free-form natural laws from experimental data". *arXiv preprint arXiv:1210.7273*, 2012.

[4] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[5] Yiqun Wang, Nicholas Wagner, and James M Rondinelli. Symbolic regression in materials science. *arXiv preprint arXiv:1901.04136*, 2019.

[6] Trent McConaghy. Ffx: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*, pages 235–260. Springer, 2011.

[7] Alan Turing. Computing machinery and intelligence. *Mind*, 59(236):433, 1950.

[8] Robert Axelrod et al. The evolution of strategies in the iterated prisoner's dilemma. *The dynamics of norms*, pages 1–16, 1987.

[9] John R Koza and Riccardo Poli. Genetic programming. In *Search Methodologies*, pages 127–164. Springer, 2005.

[10] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. *A field guide to genetic programming*. Lulu. com, 2008.

[11] Michael Kommenda. *Local Optimization and Complexity Control for Symbolic Regression*. PhD thesis, Johannes Keppler Universität Linz, 2018.

[12] William B Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer Science & Business Media, 2002.

[13] David H Wolpert, William G Macready, et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[14] Erik Pitzer. *Applied Fitness Landscape Analysis*. PhD thesis, Johannes Keppler Universität Linz, 2013.

[15] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(Jul):2171–2175, 2012.

[16] Aaron Meurer et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[17] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[18] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 04.09.2019].

[19] Michael Schmidt and Hod Lipson. Age-fitness pareto optimization. In *Genetic Programming Theory and Practice VIII*, pages 129–146. Springer, 2011.

[20] Franz Schwabl. *Statistische Mechanik*. 01 2006.

[21] Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 879–886. ACM, 2014.

[22] Wojciech Florkowski. *Phenomenology of ultra-relativistic heavy-ion collisions*. World Scientific Publishing Company, 2010.

[23] R. Hagedorn. Multiplicities,ptdistributions and the expected hadron quark-gluon phase transition. *La Rivista del Nuovo Cimento (1978-1999)*, 6(10):1–50, Oct 1983.

[24] Klaus Reygers. Direct Photon and High-pT Particle Productionwith ALICE, 2014. URL: https://www.physi.uni-heidelberg.de/~reygers/talks/2014/kruger/reygers_kruger_2014.pdf. Last visited on 31.03.2019.

[25] Klaus Reygers and Johanna Stachel. Lecture on Quark-Gluon Plasma Physics , 2015. URL: https://www.physi.uni-heidelberg.de/~reygers/lectures/2015/qgp/qgp_lecture_ss2015.html. Last visited on 31.03.2019.

[26] Shreyasi Acharya et al. Energy dependence of forward-rapidity J/ψ and ψ(2S) production in pp collisions at the LHC. *Eur. Phys. J.*, C77(6):392, 2017.

[27] R Aaij et al. Production of J/psi and Upsilon mesons in pp collisions at sqrt(s) = 8 TeV. *JHEP*, 06:064, 2013.

[28] Betty Bezverkhny Abelev et al. Production of charged pions, kaons and protons at large transverse momenta in pp and Pb–Pb collisions at $\sqrt{s_{\rm NN}}$ =2.76 TeV. *Phys. Lett.*, B736:196–207, 2014.

[29] F. James and M. Roos. Minuit – a system for function minimization and analysis of the parameter errors and correlations. *Computer Physics Communications*, 10:343–367, December 1975.

[30] iminuit team. iminuit – a python interface to minuit. https://github.com/iminuit/iminuit. Accessed: 09.04.2019.

[31] Zebo Tang, Yichun Xu, Lijuan Ruan, Gene van Buren, Fuqiang Wang, and Zhangbu Xu. Spectra and radial flow at RHIC with Tsallis statistics in a Blast-Wave description. *Phys. Rev.*, C79:051901, 2009.

[32] Jaroslav Adam et al. Centrality dependence of the nuclear modification factor of charged pions, kaons, and protons in Pb-Pb collisions at $\sqrt{s_{\rm NN}}$ = 2.76 TeV. *Phys. Rev.*, C93(3):034913, 2016.

[33] A. Adare et al. Detailed measurement of the e^+e^- pair continuum in $p+p$ and Au+Au collisions at $\sqrt{s_{NN}}$ = 200 GeV and implications for direct photon production. *Phys. Rev.*, C81:034911, 2010.

[34] Georges Aad et al. Measurement of charged-particle spectra in Pb+Pb collisions at $\sqrt{s_{\mathrm{NN}}} = 2.76$ TeV with the ATLAS detector at the LHC. *JHEP*, 09:050, 2015.

[35] Vardan Khachatryan et al. Charged-particle nuclear modification factors in PbPb and pPb collisions at $\sqrt{s_{\mathrm{N\,N}}} = 5.02$ TeV. *JHEP*, 04:039, 2017.

Appendix A

SR and fitting results for J/Ψ p_{T} spectra

Table A.1. Optimal Parameters for J/Ψ spectra fitted with eq. (5.2).

Para	popt
C_0	1485.32 ± 15.81
p_0	4.15 ± 0.03
n	3.52 ± 0.02

Table A.2. Optimal parameters for J/Ψ spectra fitted with eq. (5.4).

Para	popt
c_0	45515015.73
c_1	18.44
k_0	-0.93
k_1	-3.55

Table A.3. Result with age_fitness for J/Ψ spectra. Pareto Front w.r.t. χ^2/ndf and size. Only solutions with $\lim_{x_0\to\infty} = 0$ andχ^2/ndf >0.8.

Function	χ^2/ndf	$\lim_{x_0\to 0}$
$\left(c_0 x_0 + x_0^{k_0}\right)^{k_1} e^{-c_1} e^{-x_0}$ popt: $c_0 : 19546.39, k_0 : 5.3, k_1 : 1.57, c_1 : 7.3$	1.06	0
$c_1 \left(c_0 x_0 + x_0^{k_0}\right)^{k_1} e^{-x_0}$ popt: $k_0 : 5.3, c_0 : 19546.39, c_1 : 0.0, k_1 : 1.57$	1.06	0
$\frac{e^{-x_0}}{c_1} \left(c_0 x_0 + x_0^{k_0}\right)^{k_1}$ popt: $c_0 : 19546.38, c_1 : 1476.05, k_0 : 5.3, k_1 : 1.57$	1.06	0
$c_0 \left(c_1 + x_0^2\right)^{k_1} \left(\frac{1}{x_0}\right)^{k_0}$ popt: $c_1 : 18.44, k_0 : -0.93, k_1 : -3.55, c_0 : 45515015.73$	1.89	0
$\frac{\left(c_0 x_0 + x_0^{k_0}\right)^{k_1}}{c_1 x_0}$ popt: $c_1 : 0.0, k_0 : -0.32, c_0 : 0.09, k_1 : -7.73$	2.43	0
$\frac{\left(c_0 x_0 + x_0^{k_0}\right)^{k_1}}{c_1 x_0}$	2.43	0

Table A.3. (continued)

Function	χ^2/ndf	$\lim_{x_0\to 0}$
popt: $k_0 : -0.32, c_1 : 0.0, c_0 : 0.09, k_1 : -7.73$		

Table A.4. Result with age_fitness for J/Ψ spectra. Best solution for each class of models with fixed number of parameters. Only solutions with $\lim_{x_0\to\infty} = 0$ andχ^2/ndf >0.8.

Function	χ^2/ndf	$\lim_{x_0\to 0}$
$2x_0^2 \left(c_1 + x_0\right)^{k_0} \left(3x_0 + \frac{1}{x_0}\right) e^{c_0}$ popt: $c_1 : 6.25, k_0 : -13.52, c_0 : 31.84$	1.52	0
$x_0^{-k_2} \left(c_0 x_0 + x_0^{k_0} + 3e^{x_0}\right)^{k_1} e^{-x_0}$ popt: $c_0 : 16982.23, k_2 : -0.72, k_0 : 5.58, k_1 : 0.84$	0.89	0

Appendix B

SR and fitting results for p-p p_{T} spectra.

B.1 SR Results

Table B.1. Result of fit with eq. (5.6) for Kaons.

	Name	Value	Hesse Error	Minos Error-	Minos Error+	Fixed?
0	c0	0.710397	0.0580933	-0.0478419	0.054347	No
1	c1	1.05613	0.00903426	-0.0091	0.00907156	No
2	k0	0.0592364	0.0385329	-0.0333489	0.0351326	No
3	k1	1.51993	0.024017	-0.021151	0.0207659	No
4	k2	-4.35468	0.115589	-0.103882	0.0978785	No

Table B.2. Result of fit with eq. (5.6) for Protons.

	Name	Value	Hesse Error	Minos Error-	Minos Error+	Fixed?
0	c0	13.8835	0.307369	-3.76357	0.307369	No
1	c1	1.89261	0.00920488	-0.00920488	0.0301729	No
2	k0	0.399425	0.0114989	-0.0788647	0.0862406	No
3	k1	1.44282	0.00328392	-0.0432317	0.0417626	No
4	k2	-6.25775	0.019888	-0.019888	0.019888	No

Table B.3. Result of fit with eq. (5.6) for Pions.

	Name	Value	Hesse Error	Minos Error-	Minos Error+	Fixed?
0	c0	985.357	2.87082	-2.87082	2.87082	No
1	c1	0.885999	0.000271082	-0.000271082	0.000271082	No
2	k0	1.46942	0.00134736	-0.00134736	0.00134736	No
3	k1	0.696996	0.000206761	-0.000206761	0.000206761	No
4	k2	-14.3532	0.0048495	-0.0048495	0.0048495	No

B.2 Tsallis Model

Table B.4. Result of TM fit for Kaons.

	Name	Value	Hesse Error	Minos Error-	Minos Error+	Limit-	Limit+	Fixed?
0	q	1.13812	0.000279213	-0.000280285	0.000276455	1.0		No
1	m0	0.497648	1					Yes
2	T	0.125992	0.000457937	-0.000453577	0.000459233	0.0		No
3	a	9.92505	0.0966614	-0.0962177	0.0964568			No

Table B.5. Result of TM fit for protons.

	Name	Value	Hesse Error	Minos Error-	Minos Error+	Limit-	Limit+	Fixed?
0	q	1.11513	0.000489567	-0.000480458	0.000478438	1.0		No
1	m0	0.93827	1					Yes
2	T	0.108928	0.00097937	-0.000956075	0.00095892	0.0		No
3	a	43.9823	1.52207	-1.45742	1.52021			No

Table B.6. Result of TM fit for pions.

	Name	Value	Hesse Error	Minos Error-	Minos Error+	Limit-	Limit+	Fixed?
0	q	1.14725	7.71739e-05	-7.71197e-05	7.71235e-05	1.0		No
1	m0	0.13957	1					Yes
2	T	0.103786	6.19246e-05	-6.18769e-05	6.18607e-05	0.0		No
3	a	43.9428	0.0506216	-0.0505365	0.0506201			No

Appendix C

SR results for Pb-Pb p_{T} spectra.

Table C.1. Result with age_fitness for Pion p_{T} spectra for Pb-Pb. Pareto Front w.r.t. χ^2/ndf and number of size. Only solutions with $\lim_{x_0\to\infty} = 0$.

Function	χ^2/ndf	$\lim_{x_0\to 0}$
$x_0\left(c_1+x_0\right)\left(x_0^{k_1}+x_0^{k_2}\right)\left(c_0x_0+\left(x_0^3\right)^{k_0}\right)\left(\frac{1}{c_2+c_3x_0+x_0^{k_3}}\right)^{k_4}$ popt: $k_0:1.8, c_3:-0.66, k_4:8.1, c_1:-0.05, k_1:-4.37, c_0:1501.91, k_2:-1.66, c_2:1.25, k_3:1.33$	5.02	$-\infty$
$\left(x_0^{k_2}+(1/x_0)^{k_1}\right)\left(c_0x_0+\left(x_0^3\right)^{k_0}\right)\left(\frac{1}{c_2+c_3x_0+x_0^{k_3}}\right)^{k_4}e^{-c_1}$ popt: $k_0:1.81, c_3:2.35, k_4:6.79, c_1:-4.4, k_1:-4.37, c_0:1637.51, k_2:6.96, c_2:0.04, k_3:2.69$	5.26	0
$\left(x_0^{k_1}+x_0^{k_2}\right)\left(c_0x_0+\left(x_0^3\right)^{k_0}\right)\left(\frac{1}{c_2+c_3x_0+x_0^{k_3}}\right)^{k_4}e^{-c_1}$ popt: $k_0:1.81, c_3:2.35, k_4:6.79, c_1:-4.4, k_1:4.37, c_0:1637.51, k_2:6.96, c_2:0.04, k_3:2.69$	5.26	0
$x_0^{k_3}\left(c_0+\left(x_0+x_0^{k_0}+1/x_0\right)^{k_1}\right)^{k_2}\left(c_1+c_3x_0^2\left(c_2+2x_0+1/x_0\right)+x_0\right)$ popt: $c_2:-9.58, k_0:2.26, k_3:-2.5, c_1:27772.73, k_1:1.05, c_0:5.22, c_3:139.85, k_2:-2.82$	5.68	0
$\left(x_0+x_0^{k_1}\right)\left(c_0x_0+\left(x_0^3\right)^{k_0}\right)\left(\frac{1}{c_2+c_3x_0+x_0^{k_2}}\right)^{k_3}e^{-c_1}$ popt: $k_0:1.82, c_3:3.79, c_1:-4.29, k_1:-0.75, c_0:1688.79, k_2:2.78, c_2:1.62, k_3:4.42$	5.72	0
$x_0^{k_3}\left(x_0\left(c_0x_0^{k_0}+x_0\right)\left(c_1+c_2x_0+x_0\right)\right)^{-k_1}\left(c_3+x_0+x_0^{k_2}\right)$ popt: $k_2:4.38, c_0:0.19, c_1:0.43, k_3:7.21, k_1:3.4, c_2:0.63, k_0:3.09, c_3:1405.8$	5.77	0
$x_0^{k_3}\left(\left(c_2+x_0^{k_1}\right)\left(c_3+x_0\right)\right)^{-k_2}\left(c_0+2x_0+x_0^{k_0}\right)/c_1$ popt: $c_1:0.02, c_2:5.37, c_3:0.26, k_2:3.4, k_0:4.38, k_1:2.09, c_0:1408.95, k_3:0.4$	5.77	0
$c_1x_0^{k_1}\left(c_0+\left(x_0^2\right)^{k_0}\right)\left(\frac{1}{c_2+c_3x_0+x_0^{k_2}}\right)^{k_3}$ popt: $k_0:2.19, c_3:4.94, c_1:49.31, k_1:0.39, c_0:1396.59, k_2:2.99, c_2:1.35, k_3:3.5$	5.78	0

Table C.1. (continued)

Function	χ^2/ndf	$\lim_{x_0\to 0}$
$\left(c_1+(x_0/c_0)^{k_0}\right)\left(\left(\frac{1}{x_0+x_0^{k_1}}\right)^{k_2}+e^{-x_0^{k_3}}\right)^{k_4}$ popt: $c_0:0.07, k_0:2.11, k_3:0.96, k_2:1.49, c_1:149.45, k_1:-0.01, k_4:5.52$	5.85	149.45
$\left(c_1x_0+(c_0x_0)^{k_0}\right)\left(\left(\frac{1}{c_2+x_0^{k_1}}\right)^{k_2}+e^{-x_0}\right)^{k_3}$ popt: $c_0:6.63, k_3:4.93, k_0:2.33, k_2:1.97, c_1:3.36, c_2:0.31, k_1:0.83$	5.99	0
$\left(c_1+\left(c_0^{-k_0}x_0\right)^{k_1}\right)\left(\left(\frac{1}{c_2+x_0^{k_2}}\right)^{k_3}+e^{-x_0}\right)^{k_4}$ popt: $c_0:1.99, k_3:1.53, k_0:-3.66, k_2:1.07, c_1:89.34, c_2:0.81, k_1:1.98, k_4:4.88$	5.99	6217.59
$(c_0x_0+x_0)\left(\left(\frac{1}{c_1+x_0^{k_0}}\right)^{k_1}+e^{-x_0}\right)^{k_2}$ popt: $c_0:195.38, k_2:4.11, c_1:0.48, k_1:2.34, k_0:0.73$	6.97	0

Table C.2. Result with age_fitness for Kaon p_{T} spectra for Pb-Pb. Pareto Front w.r.t. χ^2/ndf and number of size. Only solutions with $\lim_{x_0\to\infty}=0$.

Function	χ^2/ndf	$\lim_{x_0\to 0}$
$\left(\frac{c_0+\left(\left(e^{1/x_0}\right)^{-k_0}/c_1x_0\right)^{k_1}}{c_2\log(2x_0)+c_3+x_0^2+x_0+\left(x_0e^{c_4}\left(e^{c_5+x_0}\right)^{k_2}\right)^{k_3}}\right)^{k_4}$ popt: $c_3:2.94, c_0:10.65, c_4:-0.41, k_1:105.6, k_0:2.7, c_2:-0.18, k_3:5.19, c_1:0.14, k_4:3.88, c_5:0.83, k_2:-0.11$	6.78	0
$\left(\frac{c_0+\left(\left(e^{1/x_0}\right)^{-k_0}/c_1x_0\right)^{k_1}}{c_2+x_0^2+x_0+x_0^{k_2}+\left(x_0e^{c_3}\left(e^{c_4+x_0}\right)^{k_3}\right)^{k_4}}\right)^{k_5}$ popt: $c_3:-0.01, c_0:10.81, c_4:4.34, k_1:104.59, k_5:3.89, c_2:1.91, k_3:-0.11, c_1:0.14, k_4:5.14, k_0:2.7, k_2:-0.15$	6.78	0
$\left(\frac{c_0+\left(\left(e^{1/x_0}\right)^{-k_0}/c_1x_0\right)^{k_1}}{c_2+x_0^2+x_0+x_0^{k_2}+\left(c_3x_0\left(e^{c_4+x_0}\right)^{k_3}\right)^{k_4}}\right)^{k_5}$ popt: $c_3:0.88, c_0:10.81, c_4:3.26, k_1:104.56, k_5:3.89, c_2:1.91, k_3:-0.11, c_1:0.14, k_4:5.14, k_0:2.7, k_2:-0.15$	6.78	0

Table C.3. Optimal values for Pion fit Pb-Pb

	SR Fit 1 eq. (5.8)	SR Fit 2 eq. (5.10)
para	popt	popt
k_0	2.11 ± 0.02	1.81 ± 0.02
k_1	-0.01 ± 0.0	4.37 ± 0.21
k_2	1.49 ± 0.02	6.96 ± 0.18
k_3	0.96 ± 0.0	2.69 ± 0.03
k_4	5.52 ± 0.08	6.79 ± 0.19
c_0	0.07 ± 0.0	1637.52 ± 165.69
c_1	149.45 ± 4.51	-4.4 ± 0.22
c_2	-	0.04 ± 0.0
c_3	-	2.35 ± 0.05

Table C.4. Optimal values for Kaon fit Pb-Pb

	SR Fit 1 eq. (5.8)	SR Fit 2 eq. (5.10)
para	popt	popt
k_0	1.34 ± 0.14	2.33 ± 0.05
k_1	0.05 ± 0.04	6.73 ± 2.24
k_2	2.23 ± 0.06	7.08 ± 13.7
k_3	1.07 ± 0.01	2.58 ± 0.37
k_4	3.36 ± 0.15	7.99 ± 6.43
c_0	0.03 ± 0.01	27121.04 ± 6065.49
c_1	2.75 ± 2.37	-5.48 ± 8.95
c_2	-	0.04 ± 0.0
c_3	-	2.35 ± 0.05

Table C.5. Optimal values for Proton fit Pb-Pb

	SR Fit 1 eq. (5.8)	SR Fit 2 eq. (5.10)
para	popt	popt
k_0	6.25 ± 0.31	2.27 ± 0.05
k_1	0.68 ± 0.03	4.14 ± 5286.44
k_2	1.97 ± 0.06	4.14 ± 5289.93
k_3	0.92 ± 0.01	3.48 ± 0.33
k_4	5.8 ± 0.28	5.23 ± 1.96
c_0	0.48 ± 0.02	47743.91 ± 13174.19
c_1	-0.01 ± 0.0	-4.68 ± 4.61
c_2	-	0.04 ± 0.0
c_3	-	2.35 ± 0.05

Table C.6. Optimal values for eq. (5.11).

para	popt
c_0	0.00
k_1	2.67
k_3	-9.05
k_2	-0.48
k_0	-4.92
c_1	371.39
c_2	0.21

Table C.7. Optimal values for eq. (5.12).

para	popt
c_1	0.0
c_2	0.01
k_0	1.86
k_2	5.94
k_1	2.45
c_0	1.99

Appendix D

SR results for charged hadron R_{AA}

Table D.1. Pareto front w.r.t. χ^2/ndf and size for SR results of R_{AA} with finite limits. The limits were computed with sympy.

function	χ^2/ndf	limits $x \to \infty, 0$
$(c_2 + c_3 + x_0)^{-k_2} \left(c_1 + x_0^2 x_0^{k_1} + x_0 + x_0^{k_0} e^{c_0}\right)$ popt: $c_1 : 932.69, c_2 : -29.51, c_3 : 31.81, k_1 : 6.25, k_0 : 3.58, c_0 : 8.89, k_2 : 8.35$	0.67	0,0.87
$\frac{x_0}{c_0+c_1+x_0} + \frac{1}{c_2 x_0 + c_4 + x_0^{k_0} + \log(x_0) + \frac{1}{c_3}}$ popt: $k_0 : 2.02, c_1 : 26.06, c_3 : 0.26, c_0 : 26.06, c_4 : 2.44, c_2 : -4.4$	1.25	1,0.0
$x_0^{-k_0} \left(x_0^{k_1} + e^{c_0} + \frac{1}{x_0}\right) \left(\frac{1}{c_1+x_0}\right)^{k_2}$ popt: $k_2 : 6.53, k_0 : -2.09, c_1 : 3.2, k_1 : 4.36, c_0 : 8.47$	1.42	0,0.0
$\left(c_0 + \frac{1}{x_0}\right)^{k_0} \left(\left(\frac{1}{x_0}\right)^{k_1} + e^{c_1}\right)$ popt: $k_1 : 3.91, c_1 : -8.09, c_0 : 0.21, k_0 : -5.08$	4.23	0.82,0.0
$\left(c_0 + \frac{1}{x_0}\right)^{k_0} \left(\left(\frac{1}{x_0}\right)^{k_1} + \cos(c_1)\right)$ popt: $k_1 : 3.91, c_1 : 4.71, c_0 : 0.21, k_0 : -5.08$	4.23	0.82,0.0
$\left(c_0 + x_0^{k_0}\right) \left(c_1 + x_0 + \frac{1}{x_0}\right)^{k_1}$ popt: $c_0 : 1033.55, c_1 : 7.69, k_1 : -3.46, k_0 : 3.41$	9.59	0,0.0

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 17.April 2019,