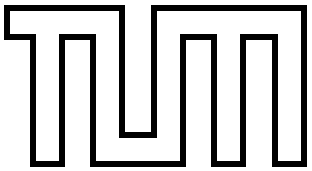


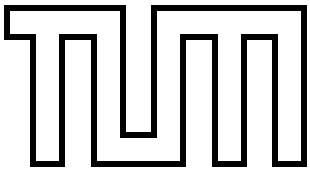
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploring Fuzzy Tuning Technique for Molecular Dynamics Simulations in AutoPas

Manuel Lerchner



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploring Fuzzy Tuning Technique for Molecular Dynamics Simulations in AutoPas

Remove all TODOS

Untersuchung von Fuzzy Tuning Verfahren für Molekulardynamik-Simulationen in AutoPas

Author:	Manuel Lerchner
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisors:	Manish Kumar Mishra, M.Sc. &
	Samuel Newcome, M.Sc.
Date:	10.08.2024

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 10.08.2024 Manuel Lerchner

Contents

4 Proof of Concept

This chapter presents a proof of concept for the fuzzy tuning technique. We will develop two different knowledge bases to predict the optimal configurations for `md_flexible` simulations.

As for all fuzzy systems, creating the knowledge base is one of the most complex parts of developing a fuzzy system, as it typically requires a profound understanding of the system to create meaningful rules. Luckily, methods such as data-driven approaches exist to semi-automate this process. This is extremely useful as those methods do not require prior expert knowledge about the system. For a problem as complex as tuning molecular dynamics simulations, it would be extremely hard to create a meaningful knowledge base from scratch. Such data-driven methods provide a good initial set of rules that experts can manually evaluate and adjust if desired. In this work, we will use a decision tree approach proposed by Crockett et al. [CBMO06]. This proposed method uses machine learning to first train decision trees on the dataset to create an initial crisp rule base. In the second step, the decision trees are converted into so-called fuzzy decision trees, which can then be used to extract the linguistic variables and fuzzy rules.

4.0.1 Decision Trees

Decision trees are prevalent machine learning algorithms used for classification and regression tasks. They work by recursively partitioning the input using axis-parallel splits so that the resulting subsets are as pure as possible. Concretely, they try to minimize a given impurity metric, such as the Gini impurity $I_G = \sum_{i=1}^{n} p_i(1-p_i)$ or the entropy $H = -\sum_{i=1}^{n} p_i \log_2(p_i)$ of the subsets [Mur12]. The probability p_i is the fraction of samples in the subset that belong to class i, and n is the total number of classes.

Since decision trees directly partition the input space into regions with different classes, they can also be depicted using their decision surface if the dimensionality allows it. The decision surface of a decision tree is a piecewise constant function that assigns the predicted class label to each point in the input space of the decision tree. An example decision tree and its decision surface are shown in Figure 4.1 and Figure 4.2.

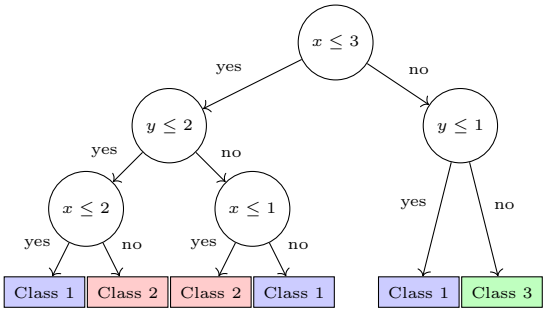


Figure 4.1: An example decision tree for a dataset with two features x and y. There are three distinct classes in the dataset

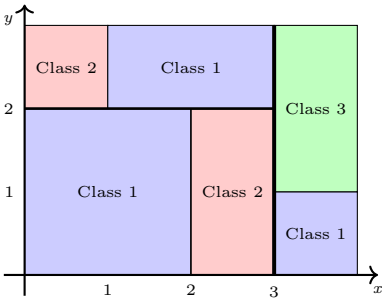


Figure 4.2: The decision surface of the decision tree from Figure 4.1 on $\mathcal{D} = [0, 4] \times [0, 3]$.

4.0.2 Conversion of Decision Trees to Fuzzy Control Systems

This section will demonstrate how to convert a classical decision tree into a fuzzy decision system using the fictional decision tree from Figure 4.1 as an example.

Fuzzy Decision Trees

As the conversion makes use of so-called fuzzy decision trees, we will briefly introduce them here. Fuzzy decision trees are a generalization of classical decision trees and allow for fuzzy logic to be used in the decision-making process. Instead of following the classical `if then else` logic to descend into the decision tree, it uses fuzzy sets at each node of the tree to fuzzily calculate the contribution of each branch to the final decision based on the degree of truth of both possible paths. Contrary to classical decision trees, which follow a single path from the root to a leaf node, fuzzy decision trees explore all possible paths simultaneously and make a final decision by aggregating the results of the paths using fuzzy logic operations.

Conversion

A classical decision tree is converted into a fuzzy decision tree by replacing the crisp decision (e.g., $x \leq 3$) at each internal node of the decision tree with a fuzzy set. Those fuzzy sets should maintain the same semantics as the crisp decision but should provide a continuous value in the range $[0, 1]$ specifying the degree of how much each branch should be considered. Classical decision trees can only make binary decisions $(0, 1)$, which causes them to only consider a single path of the decision tree. Allowing them to consider multiple paths simultaneously can drastically increase the decision-making capabilities of the decision tree, especially in boundary cases where the decision can be ambiguous.

The shape of the membership functions of the fuzzy sets can be chosen arbitrarily, but since trees work with one-sided boundaries, typical choices include complementary `sigmoid`-shaped functions that are centered around the crisp decision boundary (See Figure 4.3), as those function shapes maintain the semantics of the branching idea. Crockett et al. [CBMO06] suggested creating those sigmoid shapes with a *width* proportional to the standard deviation of the attribute. In Particular, the authors suggested an interval $[t - n \cdot \sigma, t + n \cdot \sigma]$ for the membership function, where t is the value of the decision boundary, σ is the standard deviation of the attribute and n is a parameter that can be adjusted to control the *width* of the membership function. This interval specifies the region of the membership function where most of the change in membership occurs. The value of n is typically chosen from the interval $n \in [0, 5]$ as the membership function can become too broad otherwise and could even weaken the decision-making process [CBMO06]. In this work, we will use $n = 2$ as a default value. Using this method, it is possible to fully automate the conversion of a decision tree into a fuzzy decision tree, which we will utilize in this work.

Once the internal nodes of the decision tree have been converted, the next step is to convert the leaf nodes of the decision tree to fuzzy leaf nodes. As the outputs of decision trees are specific class labels, we can define a single linguistic variable consisting of all possible class labels together with a corresponding fuzzy set. The shapes of the membership functions for the fuzzy sets can again be chosen mostly arbitrarily, but we will use `gaussian` functions with a different mean as they are a good choice for representing class labels in a continuous domain. This way of placing the fuzzy sets is not directly obvious, and the placement can

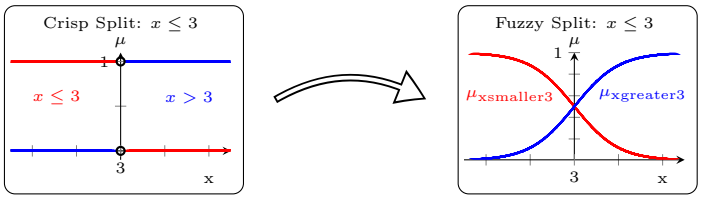


Figure 4.3: Conversion of crisp set membership functions to fuzzy set membership functions. The classical membership functions $x \leq 3$ and $x > 3$ of a decision tree node are replaced by two `sigmoid`-shaped membership functions $\mu_{\text{xsmaller3}}$ and $\mu_{\text{xgreater3}}$ that specify to which degree one should traverse left or right. The *width* of the membership functions is data dependent and is determined by $n \cdot \sigma = 2 \cdot \sigma$.

significantly influence the defuzzification process depending on the chosen defuzzification function, as we will see later. However, we can minimize this issue by carefully choosing a suitable defuzzification method. The resulting conversion of the decision tree in Figure 4.1 into a fuzzy decision tree is shown in Figure 4.4.

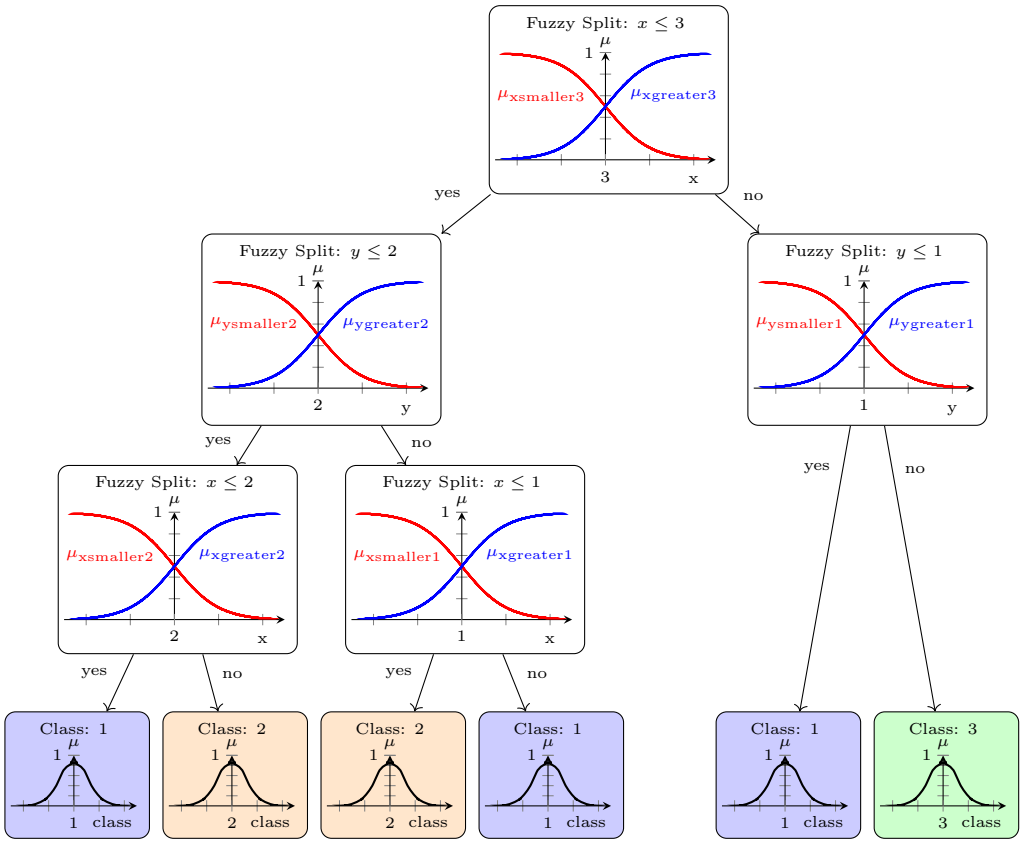


Figure 4.4: The fuzzy decision tree corresponding to the decision tree in Figure 4.1. Internal nodes use two `sigmoid` membership functions (μ_{smaller} and μ_{greater}) instead of a crisp decision. The leaf nodes use different `gaussian` membership functions centered around a unique mean.

It is now possible to fully extract all linguistic variables from the fuzzy decision tree. Every fuzzy set defined over the same variable can be collected into a single linguistic variable. This results in linguistic variables consisting of a bunch of different `sigmoid` membership functions for input variables (internal nodes) and a single linguistic variable with different `gaussian` membership functions for the output variable (leaf nodes). The resulting linguistic variables are shown in Figure 4.5.

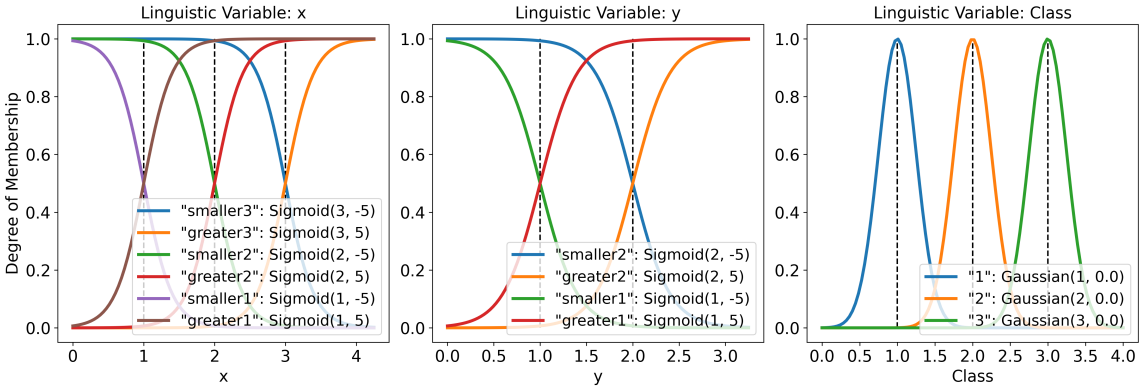


Figure 4.5: Linguistic variables used in the fuzzy decision tree of Figure 4.4. The standard deviation of the attributes is assumed to be $\sigma \approx 0.5$ such that the *width* of the sigmoid membership functions is $n \cdot \sigma \approx 1$. The standard deviation of the class values is chosen so that they do not overlap too much.

Rule Extraction

The final step is to extract the fuzzy rules from the tree. This can be done by traversing the tree in a depth-first manner and collecting the correct membership functions for each path ending in a leaf node along the way. As all conditions of this path have to hold simultaneously, all fuzzy sets are connected using the `AND` operation. This newly created fuzzy set, which consists of the intersection of all fuzzy sets along the way, is the premise of selecting the fuzzy set of the leaf node. Consequently, both fuzzy sets are connected using the Mamdani `IMPLICATION` operator. This implication then forms a rule for the fuzzy system. Therefore, each unique path traversing the tree results in a unique rule. This process essentially mimics the decision surface seen in Figure 4.2, as we create precisely one rule for each region of the decision surface. The rules extracted from the fuzzy decision tree in Figure 4.4 using this method are shown in Table 4.1.

Rule	Antecedent	Consequent
1	x *is* smaller3 $\wedge$ y *is* smaller2 $\wedge$ x *is* smaller2	*class is* 1
2	x *is* smaller3 $\wedge$ y *is* smaller2 $\wedge$ x *is* greater2	*class is* 2
3	x *is* smaller3 $\wedge$ y *is* greater2 $\wedge$ x *is* smaller1	*class is* 2
4	x *is* smaller3 $\wedge$ y *is* greater2 $\wedge$ x *is* greater1	*class is* 1
5	x *is* greater3 $\wedge$ y *is* smaller1	*class is* 1
6	x *is* greater3 $\wedge$ y *is* greater1	*class is* 3

Table 4.1: Extracted fuzzy rules from the fuzzy decision tree in Figure 4.4 in the format: **IF** Antecedent **THEN** Consequent

Fuzzy Control System

With the linguistic variables and fuzzy rules extracted from the decision tree, we can finally use them to create a fuzzy system that can predict the class of a new data point based on its features. Since the fuzzy system can be seen as a black box mapping continuous input features to continuous output classes, we represent them as shown in Figure 4.6 from now on. It is also possible to visualize the fuzzy system's decision surface by evaluating the system for each point in the input space, as shown in **??** for two different defuzzification methods. Such visualizations can help understand the decision-making process of the fuzzy system but, unfortunately, are not possible for higher-dimensional input spaces.

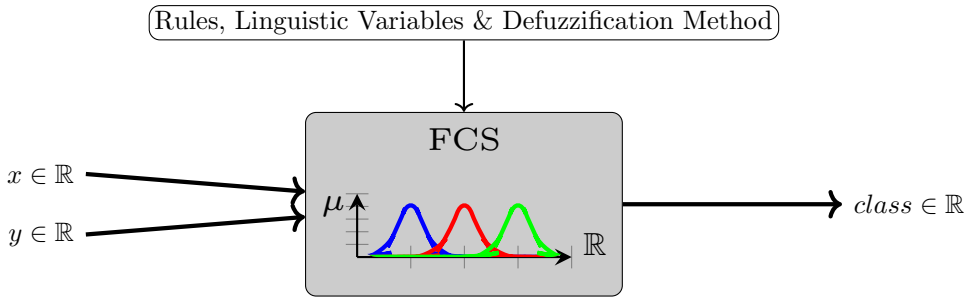


Figure 4.6: The fuzzy control system created from the fuzzy decision tree in Figure 4.4 can be seen as a black box that maps continuous input features to continuous output classes.

Choice of Defuzzification Method

The exact shape of the decision surface depends on the specific defuzzification method used. The most common choice in literature is the COG method, which calculates the x-position of the center of gravity of the membership function of the final fuzzy set. However, using the COG method can lead to undesired results when using nominal values for the output classes, which we will see later. The main problem is that the values have no concept of order. Without such an ordering, the interpolation between the different classes performed by methods such as COG is not meaningful and leads to wrong predictions. Other methods, such as the MOM method, can be used instead. This method calculates the mean value of the maximum membership functions. In most cases, this method will return precisely

the center of the membership function with the highest value and is a good choice for such types of knowledge bases. A direct comparison of the two methods on a critical datapoint is shown in Figure 4.7 and Figure 4.8. Using the COG method results in an opposite class than originally predicted from the fuzzy system, while the MOM method correctly predicts the class.

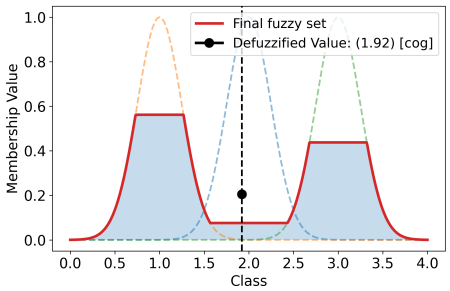


Figure 4.7: Defuzzification on the fuzzy obtained by applying the rules from Table 4.1 on the data point ($x = 2.95, y = 2.5$). There are clear peaks at the class values 1 and 3. However, the COG method incorrectly suggests values close to class 2, thus turning the two good predictions into a bad one.

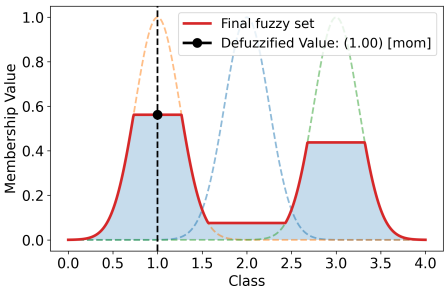


Figure 4.8: Defuzzification on the fuzzy obtained by applying the rules from Table 4.1 on the data point ($x = 2.95, y = 2.5$). The MOM method correctly suggests the class value 1, as it is the class with the highest membership value.

As previously mentioned, we can recreate the process depicted in Figure 4.6 for every possible input data point to visualize the fuzzy systems' decision surface. Both resulting decision surfaces are shown in Figure 4.7 and Figure 4.8, respectively. The decision surface using the COG tries to smoothly interpolate between the different classes, which causes interpolation errors if there are other classes in between. The decision surface using the MOM method is valid and closely resembles the decision surface of the crisp decision tree in Figure 4.2.

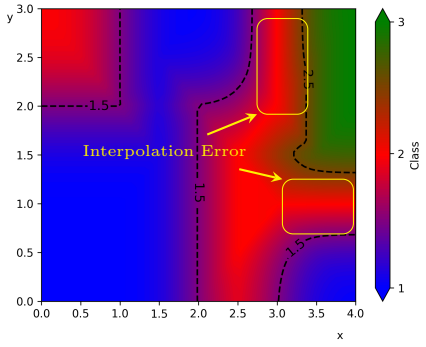


Figure 4.9: Decision surface of the developed fuzzy-system using the COG defuzzification method. The highlighted areas show interpolation errors caused by the defuzzification.

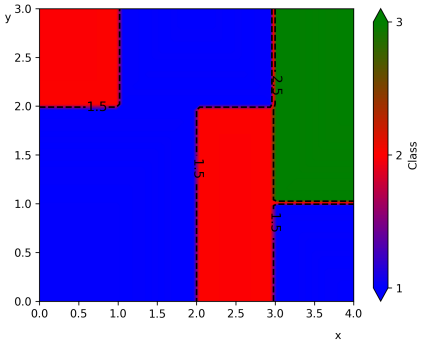


Figure 4.10: Decision surface of the developed fuzzy-system using the MOM defuzzification method. There are no invalid regions in the decision surface.

4.1 Fuzzy Control Systems for md_flexible

Following the fuzzy decision tree approach from the previous sections, we can create a fuzzy system to predict optimal configuration parameters for md_flexible simulations. Contrary to the previous example, we must first collect a dataset of simulation runs with different configuration parameters and their corresponding performance metrics, which can then be used to train the crisp decision tree. After converting the crisp decision tree to a FCS, a human expert can evaluate and adjust the rules and membership functions if necessary.

The resulting fuzzy system can then be used to predict the optimal configuration parameters for new simulation runs based on the current state of the simulation.

4.1.1 Data Collection

Using the LiveInfoLogger and TuningDataLogger classes of the AutoPas framework, it is possible to collect all the necessary data needed to train the decision tree. Both loggers create a .csv file containing each tuning step's simulation parameters and current runtime results. The LiveInfoLogger logs summary statistics about the simulation state, such as the average number of particles per cell or the current homogeneity-estimation of the simulation. In contrast, the TuningDataLogger logs the current configuration and the time it took to execute the last tuning step. The complete list of currently available parameters and their descriptions can be found in Section A.2 and Section A.3, respectively.

We will only make use of a subset, however, as we are only interested in *relative* values, that does not change when the simulation is scaled up or down and are therefore only include: avgParticlesPerCell, maxParticlesPerCell, homogeneity, textttmaxDensity, particlesPerCellStdDev and threadCount.

All the values were collected with the PAUSE_SIMULATION_DURING_TUNING cmake option enabled to ensure that the simulation state does not change during the tuning process. This ensures a fair comparison of the different configurations, as all of them are evaluated under the same conditions.

The data was collected on the CoolMUC-2 and primarily stems from the example scenarios provided by md_flexible such as explodingLiquid.yaml, fallingDrop.yaml, SpinodalDecomposition.yaml and some simulations of uniform cubes with different particle counts and densities. The exact scenarios files used for the simulations can be found in Section A.4. All simulations were run on the serial partition of the cluster and were repeated twice to account for fluctuations in performance. Furthermore, every simulation was run with 1, 4, 12, 24, and 28 threads to gather data on how parallelization affects the ideal configuration.

add specs

4.1.2 Speedup Analysis

In order to make predictions about the performance of different configurations, we first need to define the metric used to compare them. As we froze the simulation during the tuning process, we can safely use the runtime of configuration during a tuning phase to find an ordering. To make these timings comparable between tuning phases, we introduce the term *relative speedup*. The relative speedup describes the performance of a configuration relative

to the best configuration in the tuning phase. The formula for the relative speedup is given by:

$$\text{speedup}^{(i)}_{\text{config}} = \frac{t^{(i)}_{\text{best}}}{t^{(i)}_{\text{config}}} \tag{4.1}$$

Where $t^{(i)}_{\text{best}}$ is the runtime of the best configuration during the i-th tuning phase and $t^{(i)}_{\text{config}}$ is the runtime of the configuration we are interested in.

This means that all relative speedup values are going to be in the range $[0, 1]$, with 1 being the best possible value only achieved by configurations performing optimally and 0 being the worst possible value only achieved by configurations performing *infinitely* worse than the best configuration.

We can then make plots of the distribution of the relative speedup values for each configuration option to see how they affect the performance of the simulation. The density plots in Figure A.3, Figure A.4, Figure A.5 and **??** show the distribution of the relative speedup values for the Newton3, Traversal, Container-Datalayout and some complete configurations respectively. We can see that the Newton3 option generally leads to a higher relative speedup, while the Traversal option does not show a clear trend. The Datalayout option shows that the VerletListCells_AoS option is generally the best, while the configuration VerletListCells_AoS_vlc_spliced_balanced_enabled is the best configuration in most cases on the Dataset we collected.

The ordering of the different parameters in the dataset matches our intuition about the different implementations, and we can confidently proceed with creating the fuzzy system.

4.1.3 Creating the Fuzzy Rules

Using the decision tree approach described in the previous sections, we can create a fully automated system to transform the collected data into rule files. The process is as follows:

1. Preprocess the data according to the approach used (see next section).
2. Train the decision trees and prune the tree to avoid overfitting.
3. Select the best-performing decision trees and convert them into fuzzy decision trees.
4. Extract the fuzzy rules from the fuzzy decision trees.
5. Generate the linguistic variables and terms for the fuzzy system.
6. Create the OutputMapping Export and export everything to a rulefile.

As described previously, we will create two different kinds of knowledge bases, one for the Suitability Approach and one for the Individual Tuning Approach. In the following sections, we will describe both approaches in detail.

4.1.4 Individual Tuning Approach

This approach tries to create different fuzzy systems for each of the tunable parameters of the simulation. We decided to combine the `container` and `dataLayout` parameters into a single parameter as they are closely related, and the performance of one is heavily dependent on

the other. Consequently, we want to create systems for `ContainerDataLayout`, `Traversal`, and `Newton3`.

As we only want to create a fuzzy system predicting suitable configurations, we first remove all configurations performing worse than a certain threshold as depicted in Figure 4.11. We chose to only include configurations with a relative speedup of at least 70%

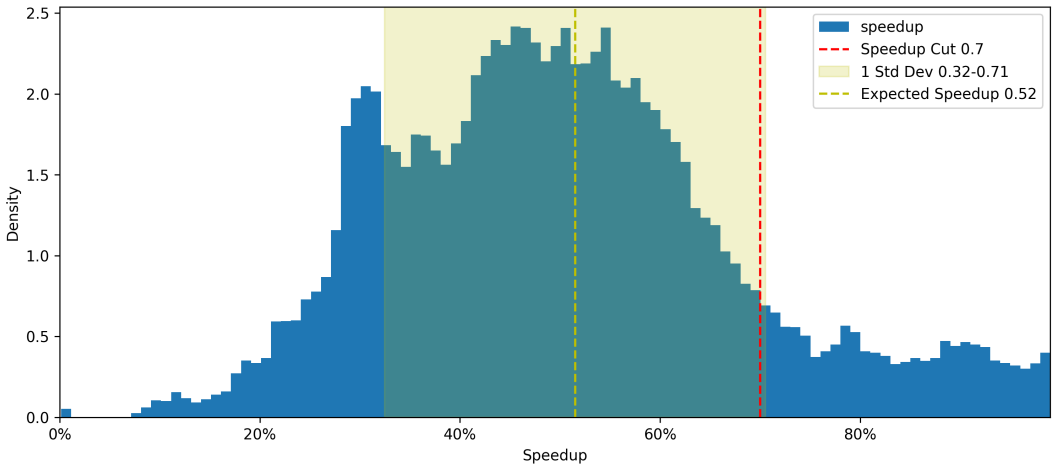


Figure 4.11: The density plot shows the distribution of all the collected configurations concerning the relative speedup compared to the best configuration during each tuning phase. This distribution shows that the average tested configuration performs just 50% as well as the best. With some configurations being ten times slower than the best configuration. Efficiently finding good configurations is key, as testing all possible configurations causes a huge performance loss.

Afterward, we group all configurations evaluated in the same tuning phase and aggregate all the present values of tunable parameters into a single term. This term represents all *good* values for the parameters to be chosen under the conditions of the tuning phase. The resulting training data is shown in Table 4.2 and can be used to train the decision trees. After applying the transformation process described in the previous sections, we end up with rules shown in Table 4.3. This concludes the creation of the fuzzy rules for the Individual Tuning Approach.

LiveInfo Data	Container_DataLayout	Traversal	Newton3
(0.905797, 0.055112, 0.297891, 15, 0.015171, 4)	"LinkedCells_SoA, VerletClusterLists_SoA, VerletListsCells_AoS"	"lc_sliced, lc_sliced_balanced, lc_sliced_c02, lc_c04"	"enabled"
(0.944637, 0.084061, 0.673320, 25, 0.039916, 24)	"LinkedCells_SoA, VerletClusterLists_SoA, VerletListsCells_AoS"	"lc_c04, lc_c08, lc_sliced, lc_sliced_balanced"	"disabled, enabled"
(0.905797, 0.041394, 0.336900, 20, 0.013546, 24)	"VerletClusterLists_SoA, VerletListsCells_AoS"	"vcl_c06, vlc_c01, vlc_c18, vlc_sliced_c02"	"disabled, enabled"
⋮	⋮	⋮	⋮

Table 4.2: Selection of the prepared training data for the Individual Tuning Approach. The table shows the live info data and the corresponding prediction of top-performing values for each tunable parameter. Each row represents a different tuning phase.

Antecedent				Consequent
avgParticlesPC	**homogeneity**	**particlesPCStdDev**	**threadCount**	**ContainerDataLayout**
`lower than 3.454`	`lower than 0.05`		`lower than 18.0`	"VerletClusterLists_SoA, VerletListsCells_AoS"
`lower than 3.454`	`higher than 0.05`	`lower than 0.024`	`higher than 18.0`	"LinkedCells_SoA, VerletClusterLists_SoA, VerletListsCells_AoS"
⋮	⋮	⋮	⋮	⋮

Antecedent				Consequent
avgParticlesPC	**homogeneity**	**particlesPCStdDev**	**threadCount**	**Traversal**
`lower than 1.553`	`higher than 0.047`	`lower than 0.023`	`higher than 2.5`	"lc_sliced, vlc_c18, lc_sliced_c02"
	`lower than 0.037`	`lower than 0.023`	`lower than 26.0`	"vcl_c06, vlc_c18, vlc_sliced_c02"
⋮	⋮	⋮	⋮	⋮

Antecedent				Consequent
avgParticlesPC	**homogeneity**	**particlesPCStdDev**	**threadCount**	**Newton 3**
		`higher than 0.03`	`higher than 18.0`	"disabled, enabled"
		`higher than 0.023` ∧ `lower than 0.037`	`lower than 18.0` ∧ `higher than 8.0`	"enabled"
⋮	⋮	⋮	⋮	⋮

Table 4.3: Extracted fuzzy rules for the Individual Tuning Approach. The table shows a selection of the rules extracted from the decision trees trained on the training data in Table 4.2. The columns of the antecedent represent the different fuzzy sets taking part in the rule.

We can also visualize the training data in a scatterplot as shown in Figure 4.12.

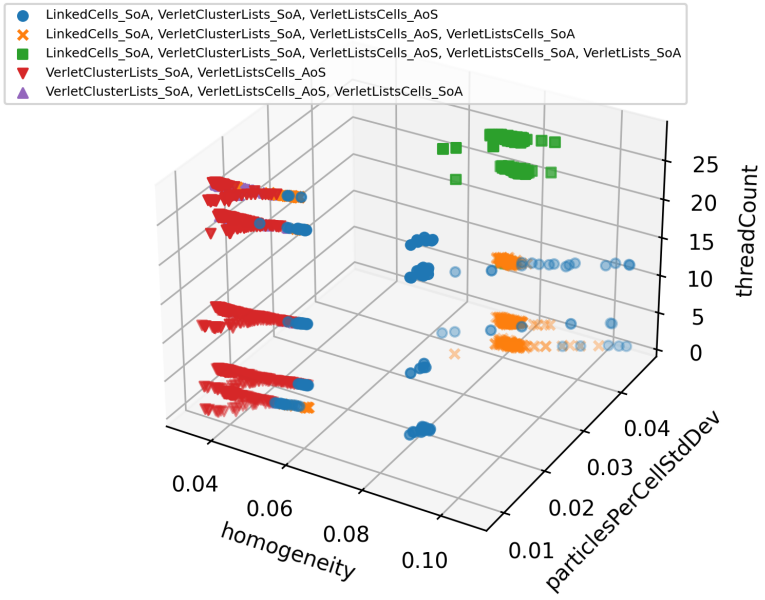


Figure 4.12: The scatterplot shows the influence of the variables `homogeneity`, `particlesPerCellStdDev` and `threadCount` on the optimal `ContainerDataLayout` parameter. We can see regions that will be learned by the decision tree.

As described previously, we use `gaussian` membership functions for each linguistic term of the consequent linguistic variables. The placement of the values is irrelevant, and we place them in a way that does not overlap. Figure 4.13 and Figure 4.14 show the resulting linguistic variables for an antecedent linguistic variable (homogeneity) and a consequent linguistic variable (Newton3), respectively. The visualization of the other variables follows a similar pattern but is more complex due to the higher number of terms, which are therefore not shown here.

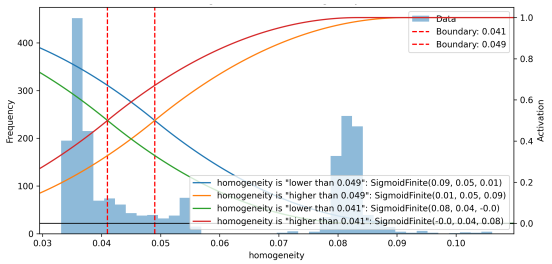


Figure 4.13: This figure shows the linguistic variable for the homogeneity attribute. We see the different fuzzy sets created from the decision trees. The background shows the histogram of all homogeneity values present in the dataset.

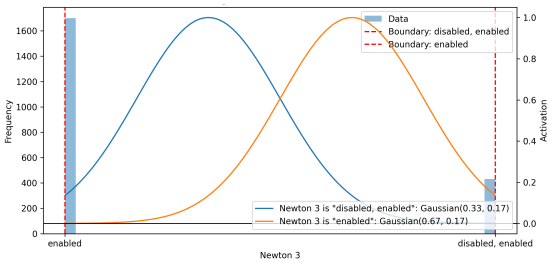


Figure 4.14: This figure shows the linguistic variable for the Newton3 attribute. We see the two `gaussian` membership functions representing the two possible values for the Newton3 attribute.

At the end of the process, we have different fuzzy systems for each of the tunable parameters of the simulation of the form shown in **??**. The fuzzy system can be seen as a black box that maps the LiveInfo data into a sort of *index* representing the predicted values of the parameter. Using the MOM method, the output will always correspond to the fuzzy set with the highest membership value and be free of interpolation errors.

4.1.5 Suitability Approach

The suitability approach differs from the individual tuning approach in that it tries to predict a configuration's numerical *suitability* value under the current conditions. Therefore, each possible configuration is assigned a unique fuzzy system tailored to just evaluating this configuration—the suitability value of a configuration defined as the relative speedup already described in the previous section.

The process is similar to the one described in the previous section. However, we do not group the configurations together and directly use the calculated relative speedup as the suitability value.

To train the decision trees, we again use a classification-based approach with `terrible`, `bad`, `average`, `good`, and `excellent` as the possible linguistic terms (see Figure 4.15). The calculated suitability values of each configuration during a specific tuning phase are mapped to the corresponding class.

The final training data is shown in Table 4.4, and it is again possible to train the decision trees and extract the fuzzy rules from them. It is important to note that each configuration has its own fuzzy system and rules tailored to it. This makes the suitability approach more flexible but also causes more computational overhead during the inference process.

The resulting rules are shown in Table 4.5, and the linguistic variables for the suitability attribute are shown in Figure 4.15. The output mapping for the suitability approach is shown in **??**.

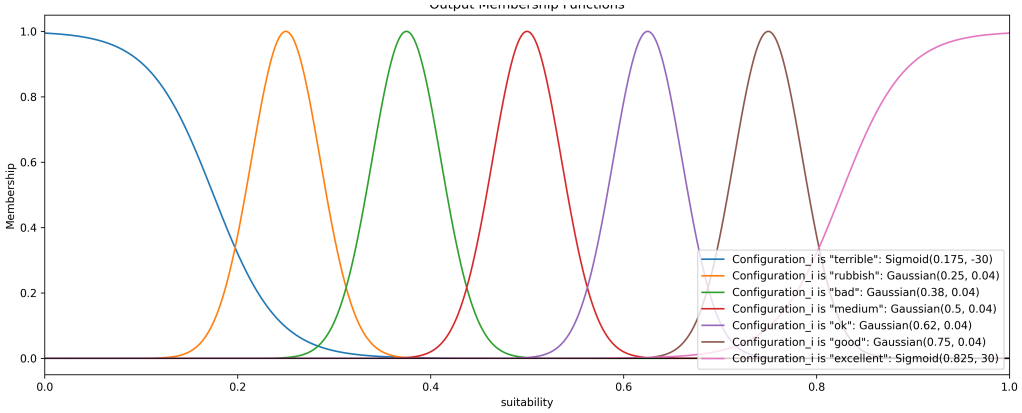


Figure 4.15: Linguistic terms for the suitability variables. The fuzzy sets consist of `sigmoid` membership functions at the borders and `gaussian` membership functions in the middle. The values are placed coherently. This time, we will use the COG defuzzification method and will make use of the interpolation between the values.

LiveInfo Data [a]	**Configuration** [b]	**Speedup**	**Suitability**
(0.905797, 0.035496, 0.531948, 0.012989, 1)	LinkedCells, AoS, lc_sliced, enabled	0.450641	"bad"
(0.944637, 0.083797, 0.691920 , 0.012989, 28)	VerletClusterLists, AoS,vcl_c06, disabled	0.319094	"rubbish"
(0.944637, 0.079441, 0.040082 , 0.012989, 12)	LinkedCells, SoA,lc_sliced, c02,enabled	0.989101	"excellent"
⋮	⋮	⋮	⋮

Table 4.4: Selection of the prepared training data for the Individual Tuning Approach. The table shows the live info data and the corresponding prediction of top-performing values for each tunable parameter. Each row represents a different tuning phase.

[a] The format of the LiveInfo tuple is: (avgParticlesPC, homogeneity, maxDensity,particlesPerCellStdDev, threadCount)

[b] The format of the Configuration is: Container, DataLayout, Traversal, Newton3

Antecedent				**Consequent**
avgParticlesPC	**homogeneity**	**particlesPCStdDev**	**threadCount**	**LinkedCells_AoS_lc_c01_disabled**
	`lower than 0.084`	`higher than 0.029`	`higher than 26.0`	"medium"
	`higher than 0.084`	`higher than 0.029`	`higher than 26.0`	"bad"
		`higher than 0.02`	`lower than 2.5`	"rubbish"
⋮	⋮	⋮	⋮	⋮

Antecedent				**Consequent**
maxParticlesPerCell	**homogeneity**	**particlesPCStdDev**	**threadCount**	**LinkedCells_AoS_lc_c04_disabled**
`higher than 18.5`	`lower than 0.082`		`higher than 18.0` ∧ `lower than 26.0`	"medium"
`higher than 18.5`	`higher than 0.082`		`higher than 18.0` ∧ `lower than 26.0`	"bad"
⋮	⋮	⋮	⋮	⋮

Table 4.5: Extracted fuzzy rules for the Suitability Approach. The table shows a selection of the rules extracted from the decision trees trained on the training data in Table 4.4. The columns of the antecedent represent the different fuzzy sets taking part in the rule.

List of Figures

List of Tables

Listings

Bibliography

[BMK96] Bernadette Bouchon-Meunier and Vladik Kreinovich. Axiomatic description of implication leads to a classical formula with logical modifiers: (in particular, mamdani's choice of "and" as implication is not so weird after all). 1996.

[BPR+16] Tobias Brink, Martin Peterlechner, Harald Rösner, Karsten Albe, and Gerhard Wilde. Influence of crystalline nanoprecipitates on shear-band propagation in cu-zr-based metallic glasses. *Phys. Rev. Appl.*, 5:054005, May 2016.

[CBMO06] Keeley Crockett, Zuhair Bandar, David Mclean, and James O'Shea. On constructing a fuzzy inference framework using crisp decision trees. *Fuzzy Sets and Systems*, 157(21):2809–2832, 2006.

[Che] Chemie.de. Lennard-jones-potential. `https://www.chemie.de/lexikon/Lennard-Jones-Potential.html`. Accessed: 2024-07-07.

[GSBN21] Fabio Alexander Gratl, Steffen Seckler, Hans-Joachim Bungartz, and Philipp Neumann. N ways to simulate short-range particle systems: Automated algorithm selection with the node-level library autopas. *Computer Physics Communications*, 273:108262, 2021.

[GST+19] Fabio Alexander Gratl, Steffen Seckler, Nikola Tchipev, Hans-Joachim Bungartz, and Philipp Neumann. Autopas: Auto-tuning for particle simulations. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 748–757, 2019.

[Iri21] Karl Irikura. Physics-guided curve fitting for potential-energy functions of diatomic molecules. *Authorea*, April 2021.

[JPRS06] E. Jodoin, C.A. Pena Reyes, and E. Sanchez. A method for the fuzzification of categorical variables. In *2006 IEEE International Conference on Fuzzy Systems*, pages 831–838, 2006.

[LM15] Benedict Leimkuhler and Charles Matthews. *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*. Interdisciplinary Applied Mathematics. Springer, May 2015.

[MAMM20] C Y Maghfiroh, A Arkundato, Misto, and W Maulina. Parameters (sigma, epsilon) of lennard-jones for fe, ni, pb for potential and cr based on melting point values using the molecular dynamics method of the lammps program. *Journal of Physics: Conference Series*, 1491(1):012022, October 2020.

[MKEC22] Ali Mohammed, Jonas H. Müller Korndörfer, Ahmed Eleliemy, and Florina M. Ciorba. Automated scheduling algorithm selection and chunk parameter calculation in openmp. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4383–4394, 2022.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[NGNB23] Samuel James Newcome, Fabio Alexander Gratl, Philipp Neumann, and Hans-Joachim Bungartz. Towards auto-tuning multi-site molecular dynamics simulations with autopas. *Journal of Computational and Applied Mathematics*, 433:115278, 2023.

[Phy] Nexus Physics. The lennard-jones potential. `https://www.compadre.org/nexusph/course/The_Lennard-Jones_Potential`. Accessed: 2024-07-07.

[PS17] Juan R. Perilla and Klaus Schulten. Physical properties of the hiv-1 capsid from all-atom molecular dynamics simulations. *Nature Communications*, 8(1):15959, 2017.

[RdCC12] Pilar Rey-del Castillo and Jesús Cardeñosa. Fuzzy min-max neural networks for categorical data: application to missing data imputation. *Neural Computing and Applications*, 21(7):1349–1362, 2012.

[SGH+21] Steffen Seckler, Fabio Gratl, Matthias Heinen, Jadran Vrabec, Hans-Joachim Bungartz, and Philipp Neumann. Autopas in ls1 mardyn: Massively parallel particle simulations with node-level auto-tuning. *Journal of Computational Science*, 50:101296, 2021.

[VBC08] G. Viccione, V. Bovolin, and E. Pugliese Carratelli. Defining and optimizing algorithms for neighbouring particle identification in sph fluid simulations. *International Journal for Numerical Methods in Fluids*, 58(6):625–638, 2008.

[ZBB+13] Franck Zielinski, Pierre Baudin, Gérard Baudin, Pierre Baudin, and Gérard Baudin. Quantum states of atoms and molecules. *Chemical Education Digital Library*, 1(1):1–10, 2013.