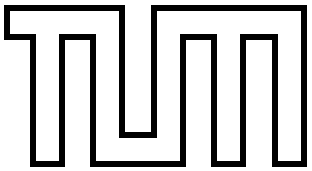


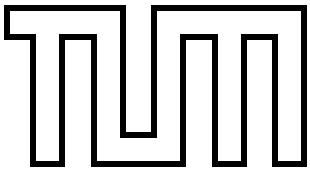
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

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

In this chapter, we present a proof of concept for the fuzzy tuning technique. We will develop a set of linguistic variables and fuzzy rules to predict the optimal configuration parameters for `md_flexible` simulations.

4.1. Creating the Knowledge Base

One of the hardest parts of developing a fuzzy system is creating the knowledge base, as it typically requires a very deep understanding of the system to be able to create meaningful rules. However, there also exist methods to use a data-driven approach to create the knowledge base automatically. This is especially useful as those methods don't require any prior expert knowledge about the system. But regardless of the way the knowledge base is created, it is still possible to manually evaluate and adjust the rules to add manual expert knowledge to the system. Using such data-driven methods can be a good starting point for creating a fuzzy system, as it can provide a good initial set of rules that can be further refined by experts.

Add references to the methods

There are several methods to automatically create and tune fuzzy systems based on data. Some of the most common methods include genetic algorithms , particle swarm optimization, and decision trees. In this work, we will use a decision tree aproach proposed by Crockett et al. [CBMO06] to create the knowledge base for the fuzzy system. This proposed method uses machine learning to first train a classical decision tree on the dataset and then converts the decision tree into a fuzzy decision tree which can then be used to extract the linguistic variables and fuzzy rules.

4.2. Decision Trees

Decision trees are very popular machine learning algorithms that are used for classification and regression tasks. They work by recursively partitioning the input using axis-parallel splits [Mur12], in such a way that the resulting subsets are as pure as possible. There are several algorithms to train decision trees, such as ID3, C4.5, CART, and many others, but they all work by the principle of minimizing the *impurity* of the resulting subsets. Decision trees are supervised learning algorithms, which means that they require labeled data to train.

A key feature of decision trees is their interpretability. This makes them a good choice for creating the initial knowledge base for a fuzzy system, as it is very easy for a human expert to understand and refine the rules created by the decision tree with additional knowledge.

Since decision trees directly partition the input space into regions with different classes, they can also be easyily represented by their decision surface (given that the dimensionality of the input space is low enough). The decision surface of a decision tree is a piecewise

constant function that assigns the predicted class to each region of the input space. An example decision tree and its decision surface are shown in Figure 4.1 and Figure 4.2.

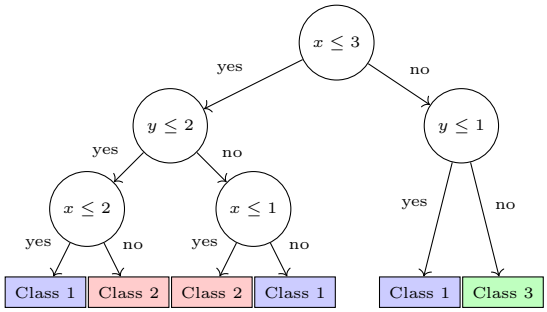


Figure 4.1.: An example decision tree for a dataset with two features x and y. There are three distinct classes in the dataset

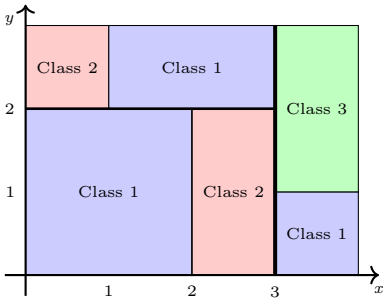


Figure 4.2.: The decision surface of the decision tree from Figure 4.1 on $\mathcal{D} = [0, 4] \times [0, 3]$.

4.3. Fuzzy Decision Trees

Fuzzy decision trees are a generalization of classical decision trees that allow for fuzzy logic to be used in the decision-making process. This extension allows to eliminate the crisp decision boundaries of classical decision trees and instead use fuzzy sets at each node of the tree to calculate the contribution of each branch to the final decision. This allows for a more flexible decision-making process that can take into account the uncertainty of the input data and the splits. Contrary to classical decision trees, which follow a single path from the root to a leaf node, fuzzy decision trees explore all possible paths at the same time and make a final decision by aggregating the results of all paths using fuzzy logic. This is possible, as each node in a fuzzy decision tree can fuzzily assign how much each of its children should contribute to the final decision.

4.4. Converting a Decision Tree into a Fuzzy Inference System

In this section, we will demonstrate how to convert a classical decision tree into a fuzzy inference system using the fictional decision tree from Figure 4.1 as an example.

The conversion of a classical decision tree to a fuzzy decision tree is done by replacing the crisp decision boundaries (e.g., $x \leq 3$) at each internal node with fuzzy membership functions. Those membership functions should have the same semantics as the crisp decision boundaries, but instead of returning a binary value of wheter to continue down the left or right branch, they return a value in the range $[0, 1]$ that specifies to which degree each branch should be taken. The shape of the membership functions can be chosen arbitrarily, but since the decision should be one-sided, typical choices include complementary `sigmoid`-shaped functions. This conversion process is shown in Figure 4.3.

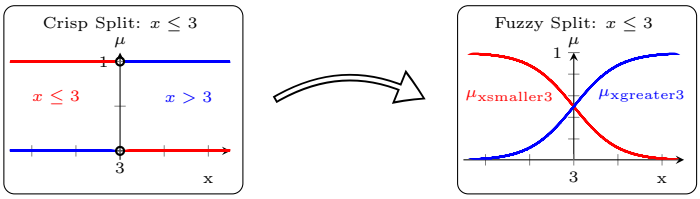


Figure 4.3.: Conversion of a crisp decision surface to a fuzzy decision surface. The crisp decision surface $x \leq 3$ is replaced by two `sigmoid` membership functions $\mu_{\text{xsmaller3}}$ and $\mu_{\text{xgreater3}}$ that specify to which degree the comparison is true or false.

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, representing the class values. This is also done by replacing each crisp class value with a fuzzy membership function that assigns a degree of membership to the class. The shape of the membership functions can again be chosen arbitrarily, but typical choices include `gaussian` functions with a specific mean and variance. The ordering of the different class-specific membership functions is very important, as it can heavily influence the defuzzification process for some defuzzification methods. The conversion of the complete decision tree to a fuzzy decision tree is shown in Figure 4.4.

After the translation of the decision tree, all membership functions operating on the same variable can be combined into a single linguistic variable. Note that the specific shapes of the membership functions has been picked arbitrarily and can be adjusted to better fit the data. The resulting linguistic variables are shown in Figure 4.5.

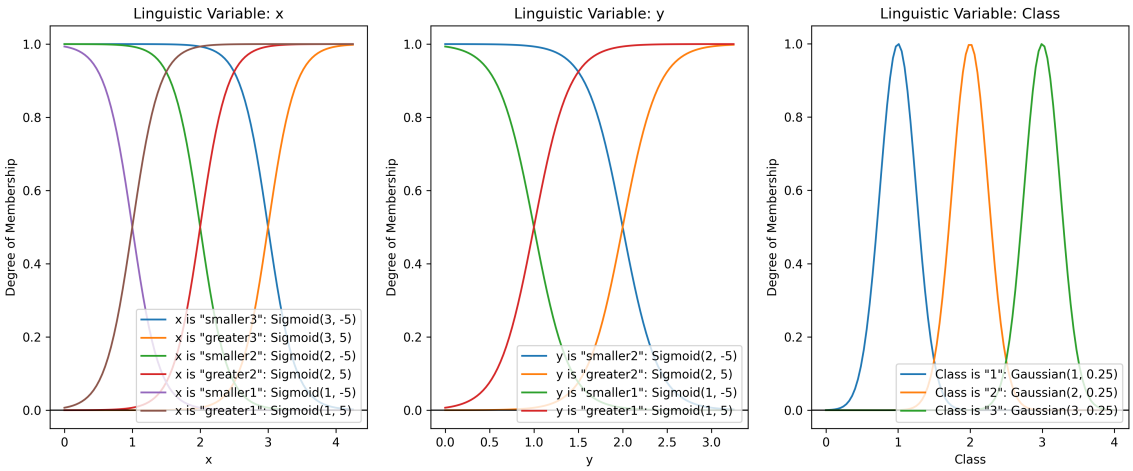


Figure 4.5.: Linguistic variables used in the fuzzy decision tree in Figure 4.4.

Rule Extraction

Once the fuzzy decision tree has been created, the next 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 along the way. Each connection between two internal nodes in the tree corresponds to a `AND` operation, while each final connection between

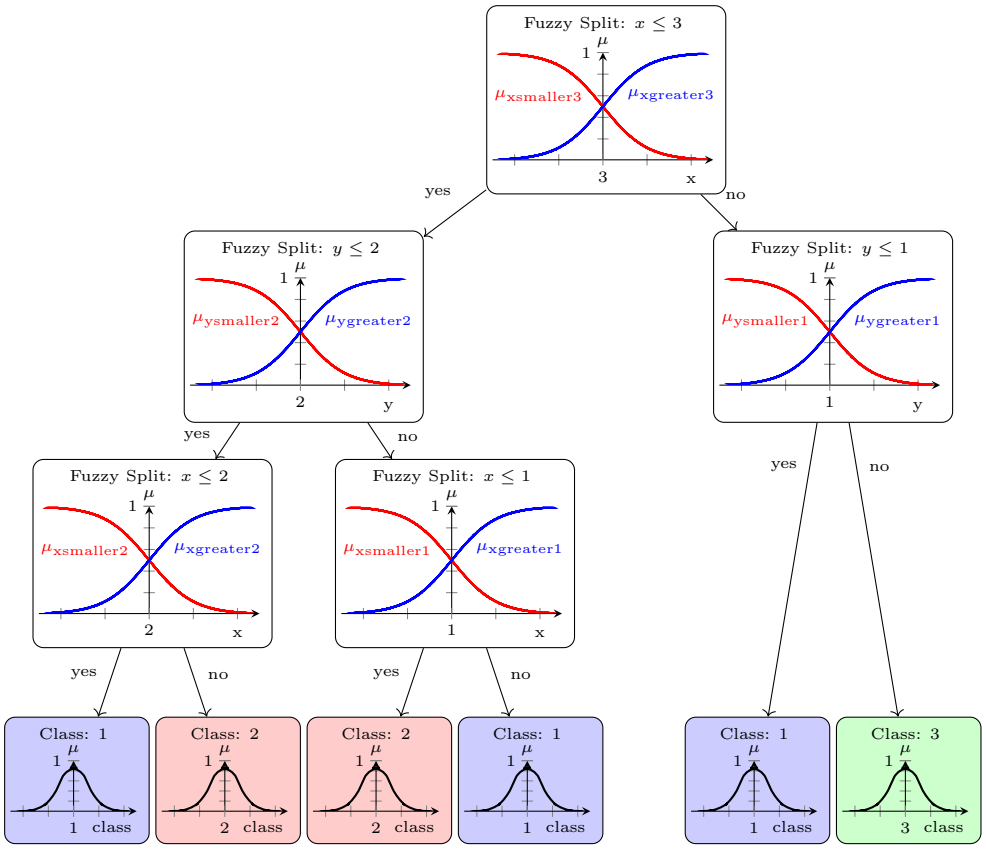


Figure 4.4.: The fuzzy decision tree corresponding to the decision tree in Figure 4.1. Each internal node in the fuzzy decision tree uses two `sigmoid` membership functions (μ_{smaller} and μ_{greater}) to specify to which degree the comparison is true or false. The leaf nodes use different `gaussian` membership functions centered around their class value.

an internal node and a leaf node corresponds to an `IMPLIES` operation. This implication then forms a rule for the fuzzy system. This process esentially mimics the decision surface seen in Figure 4.2, as we create exactly one rule for each region of the decision surface. The rules extracted from the fuzzy decision tree in Figure 4.4 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 Inference System

With the linguistic variables and fuzzy rules extracted from the decision tree, we can now 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 (see Figure 4.6), it is possible to visualize the decision surface of the fuzzy system by evaluating the membership functions of the rules for each point in the input space. This decision surface can then be used to understand the decision-making process of the fuzzy system and to identify possible errors in the rules or membership functions.

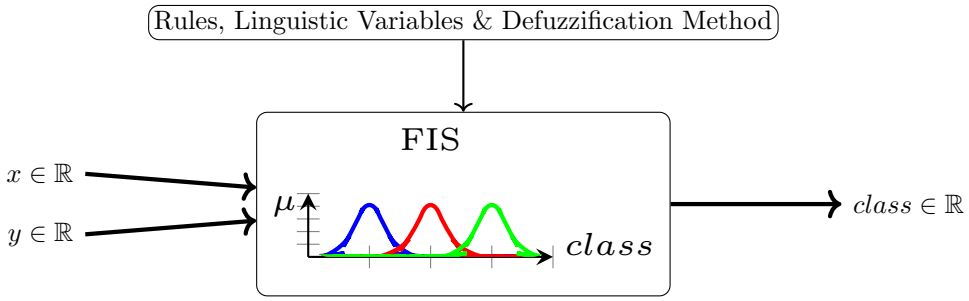


Figure 4.6.: The fuzzy inference system created from the fuzzy decision tree in Figure 4.4 can be seen as a black box that maps continuous input features to continuous output classes.

Choice of Defuzzification Method

The exact shape of the decision surface depends on the used defuzzification method. The most common choice is the COG method, which calculates the x-position of the center of gravity of the resulting membership function. However, using the COG method can lead to undesired results when using nominal values for the output classes, as there is no concept of ordering among the values. Without such an ordering, the interpolation between the different classes performed by methuds such as COG is not meaningful and leads to bad predictions. In such cases, 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 exactly the center of the membership function with the highest value and is therefore a good choice for nominal values. A direct comparison of the two methods on a critical datapoint is shown in Figure 4.7 and Figure 4.8

check

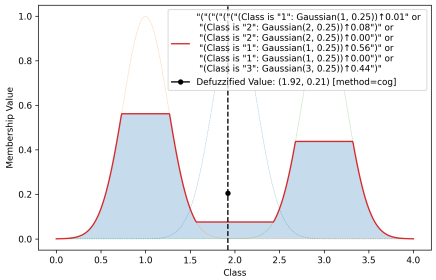


Figure 4.7.: Resulting fuzzy set after applying the rules from Table 4.1 on the data point ($x = 2.95, y = 2.5$). There are clear peaks at the class values 1 and 3. The COG method however returns Class 2, as it lies right in between the two peaks, turing the two good pre-

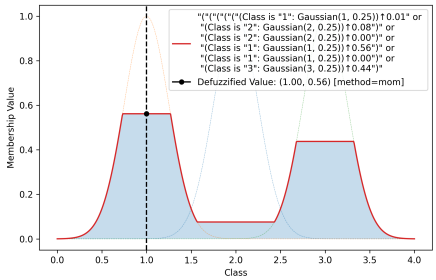


Figure 4.8.: The MOM method returns the class value 1, as it is the mean of the two peaks at class values 1 and 3. This is a much better prediction than the one made by the COG method.

It is also possible to calculate the whole decision surface of the fuzzy system by evaluating the membership functions of the rules for each point in the input space. Both the decision surface using the COG and MOM defuzzification methods are shown in Figure 4.9 and Figure 4.10 respectively.

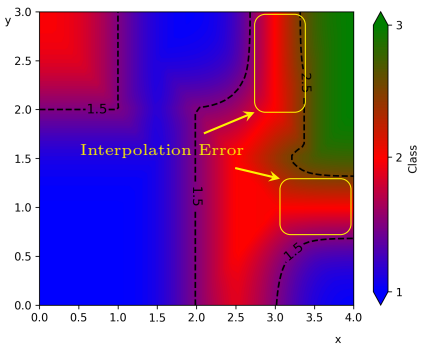


Figure 4.9.: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the COG defuzzification method. The highlighted area shows the interpolation error of the COG method described in Figure 4.7.

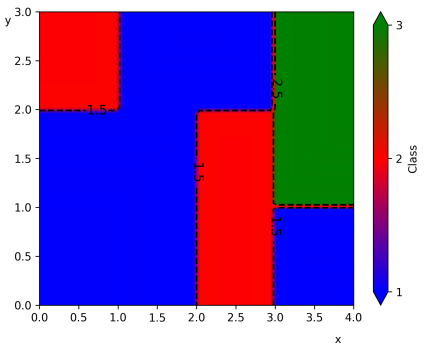


Figure 4.10.: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the MOM defuzzification method.

The choice of the defuzzification method also marks the end of the conversion process, since we now have a complete fuzzy system that can be used to predict new data points based on their features. In the next section, we use this approach to create a fuzzy system to predict optimal configuration parameters for `md_flexible` simulations.

4.5. Creating a Fuzzy System for `md_flexible`

By 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 the conversion of the crisp decision tree to a FIS, a human expert can evaluate the rules and membership functions and adjust them 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.

4.5.1. 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 the simulation parameters and current runtime results for each tuning step. The full list of currently available parameters can be found in Section A.2 and Section A.3.

A. Appendix

A.1. Glossary

AutoPas Node-level auto-tuned particle simulation library written in C++. See `https://github.com/AutoPas/AutoPas`. 9, 17–19

COG Center of Gravity Defuzzification Method. 8, 9

FIS Fuzzy Inference System. 8, 9

md_flexible A flexible molecular dynamics simulation framework built on top of AutoPas. xiii, 4, 9

MOM Mean of Maximum Defuzzification Method. 8, 9

A.2. LiveInfoLogger Data Fields

The following fields are currently available in the LiveInfo data file, which contains information about the simulation state at each iteration. The data is collected and logged by the `LiveInfoLogger` class of the AutoPas library.

Iteration	The current iteration number of the simulation.
avgParticlesPerCell	The average number of particles per cell in the simulation domain.
cutoff	The cutoff radius for the interaction of particles, beyond which particles do not interact.
domainSizeX	The size of the simulation domain in the X dimension.
domainSizeY	The size of the simulation domain in the Y dimension.
domainSizeZ	The size of the simulation domain in the Z dimension.
estimatedNumNeighborInteractions	The estimated number of neighbor interactions between particles.
homogeneity	A measure of the distribution uniformity of particles across the cells.
maxDensity	The maximum density of particles in any cell.
maxParticlesPerCell	The maximum number of particles found in any single cell.
minParticlesPerCell	The minimum number of particles found in any single cell.
numCells	The total number of cells in the simulation domain.
numEmptyCells	The number of cells that contain no particles.
numHaloParticles	The number of particles in the halo region (boundary region) of the simulation domain.
numParticles	The total number of particles in the simulation domain.
particleSize	The size of each particle.
particleSizeNeededByFunctor	The particle size required by the functor (the function used for calculating interactions).
particlesPerBlurredCellStdDev	The standard deviation of the number of particles per blurred cell, providing a measure of particle distribution variability.
particlesPerCellStdDev	The standard deviation of the number of particles per cell, indicating the variability in particle distribution.
rebuildFrequency	The frequency at which the neighbor list is rebuilt.
skin	The skin width added to the cutoff radius to create a buffer zone for neighbor lists, ensuring efficient interaction calculations.
threadCount	The number of threads used for parallel processing in the simulation.

Define the metric

Which unit?

A.3. TuninData Fields

The following fields are currently available in the TuningResults data file, which contains the current performance data for a given configuration at a particular iteration. The data is collected and logged by the `TuningDataLogger` class of the AutoPas library.

Date The date and time when the data was collected.

Iteration The current iteration number of the simulation.

Container The type of container used to store the particles in the simulation (e.g., LinkedCells, VerletLists).

CellSizeFactor A factor that determines the size of the cells relative to the cutoff radius.

check https://med

Traversal The method used to traverse the cells and calculate interactions between particles.

Load Estimator The strategy used to estimate and balance the computational load across different parts of the simulation domain.

Data Layout The arrangement of particle data in memory (e.g., AoS for Array of Structures, SoA for Structure of Arrays).

Newton 3 Indicates whether the Newton's third law optimization is used to reduce computation by only calculating forces once per particle pair (Yes/No).

sample$i$ The performance data for the configuration at the i-th sample point. The number of total sample points per iteration can be configured via the `.yaml` configuration file.

Reduced The reduced performance data for all sample points, calculated by aggregating the data across all sample points. The specific aggregation method can be configured via the `.yaml` configuration file.

Smoothed A smoothed version of the reduced performance data.

List of Figures

List of Tables

Bibliography

[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.

[Gra17] Fabio Alexander Gratl. Task based parallelization of the fast multipole method implementation of ls1-mardyn via quicksched. Master's thesis, Institut für Informatik 5, Technische Universität München, Garching, November 2017.

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