



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

Acknowledgements	vii
Abstract	ix
Zusammenfassung	xi
1 Introduction	1
1.1 A	1
2 Theoretical Background	2
2.1 Molecular Dynamics	2
2.2 AutoPas	5
2.3 Autotuning in AutoPas	5
2.4 Fuzzy Logic	12
2.4.1 Fuzzy Sets	12
2.4.2 Fuzzy Logic Operations	12
2.4.3 Linguistic Variables	14
2.4.4 Fuzzy Logic Rules	15
2.4.5 Defuzzification	16
2.4.6 Structure of creating a Fuzzy Logic System	17
3 Implementation	20
3.1 Fuzzy Tuning Framework	20
3.2 Rule Parser	21
3.3 Tuning Strategy	23
3.3.1 Configuration Suitability Approach	23
3.3.2 Parameter Tuning Approach	24
4 Proof of Concept	25
4.1 Creating the Knowledge Base	25
4.2 Decision Trees	25
4.3 Fuzzy Decision Trees	26
4.4 Converting a Decision Tree into a Fuzzy Inference System	26
4.5 Creating a Fuzzy System for <code>md_flexible</code>	30
4.5.1 Data Collection	30
4.5.2 Creating the Fuzzy Rules	32
5 Comparison and Evaluation	35
5.1 A	35

6	Future Work	36
6.1	A	36
7	Conclusion	37
7.1	A	37
A	Appendix	38
A.1	Glossary	38
A.2	LiveInfoLogger Data Fields	39
A.3	TuninData Fields	40
A.4	ANTLR4 Rule Parser Grammar	41
A.5	Scenarios used for Data Generation	43
	Bibliography	49

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.

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 approach 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.

Add references to the methods

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 easily 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.

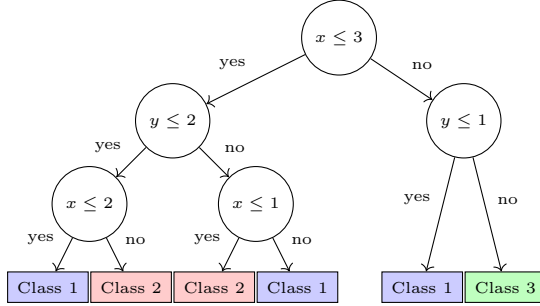


Figure 4.1: An example decision tree for a dataset with two features x and y . There are three distinct classes in the dataset

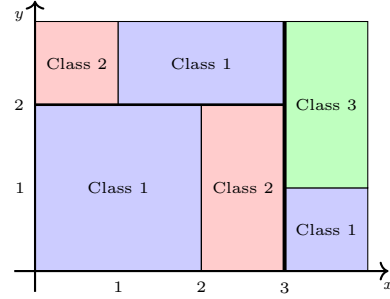


Figure 4.2: The decision surface of the decision tree from Figure 4.1 on $\mathcal{D} = [0, 4] \times [0, 3]$.

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 whether 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.

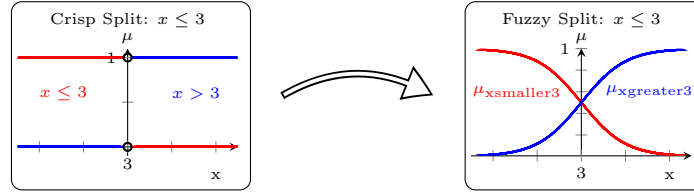


Figure 4.3: Conversion of a crisp decision surface to a fuzzy decision surface. The crisp decision surface $x \leq 3$ is replaced by two **sigmoid** membership functions $\mu_{xsmaller3}$ and $\mu_{xgreater3}$ that specify to which degree the comparison is true or false.

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.

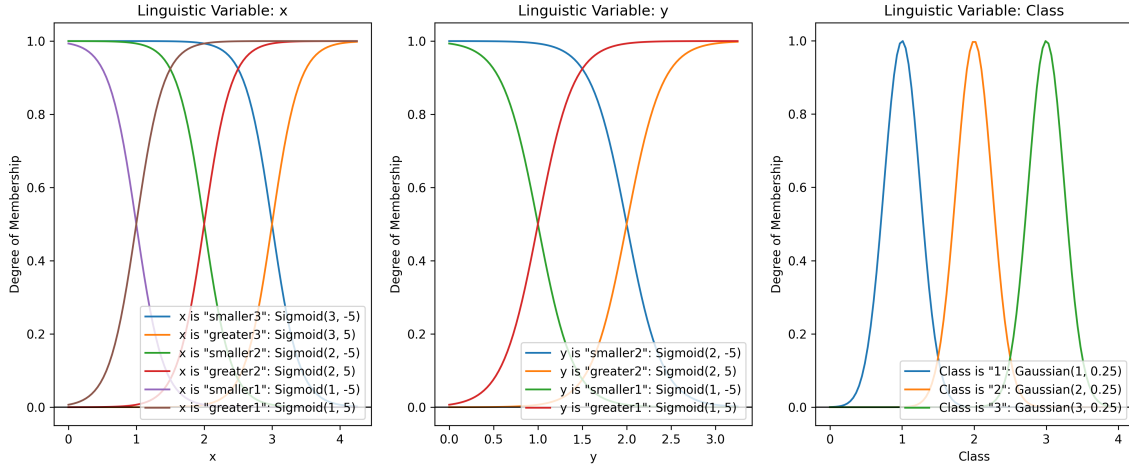


Figure 4.5: Linguistic variables used in the fuzzy decision tree in Figure 4.4.

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

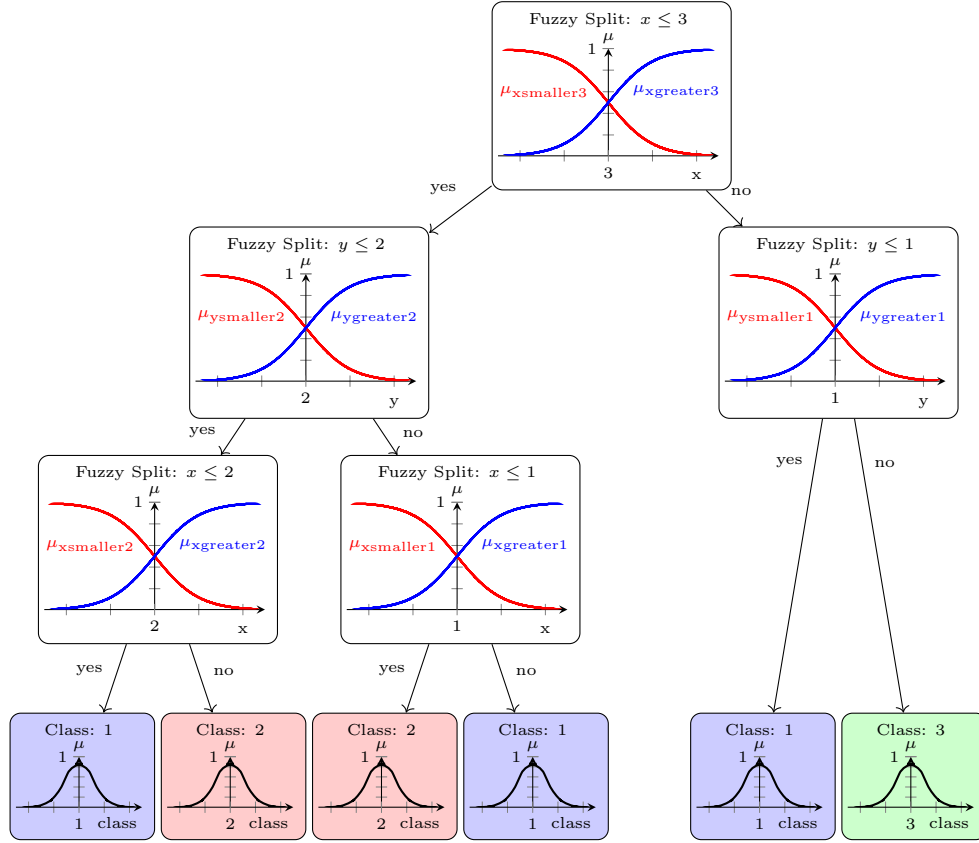


Figure 4.4: The fuzzy decision tree corresponding to the decision tree in Figure 4.1. Each internal node in the fuzzy decision tree uses two **sigmoid** membership functions (μ_{smaller} and μ_{greater}) to specify to which degree the comparison is true or false. The leaf nodes use different **gaussian** 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 essentially 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	$x \text{ is smaller3} \wedge y \text{ is smaller2} \wedge x \text{ is smaller2}$	$class \text{ is } 1$
2	$x \text{ is smaller3} \wedge y \text{ is smaller2} \wedge x \text{ is greater2}$	$class \text{ is } 2$
3	$x \text{ is smaller3} \wedge y \text{ is greater2} \wedge x \text{ is smaller1}$	$class \text{ is } 2$
4	$x \text{ is smaller3} \wedge y \text{ is greater2} \wedge x \text{ is greater1}$	$class \text{ is } 1$
5	$x \text{ is greater3} \wedge y \text{ is smaller1}$	$class \text{ is } 1$
6	$x \text{ is greater3} \wedge y \text{ is greater1}$	$class \text{ is } 3$

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.

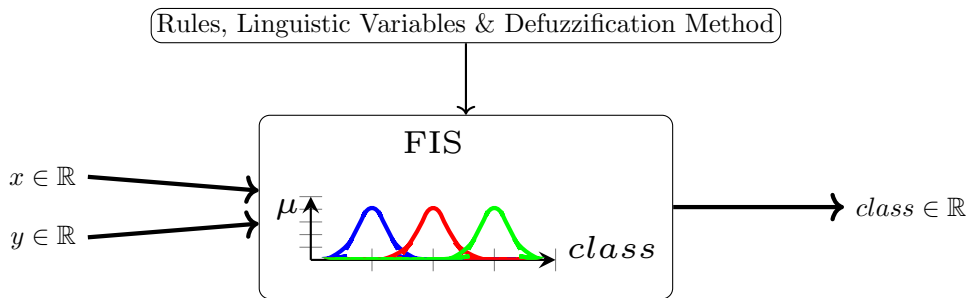


Figure 4.6: The fuzzy inference system created from the fuzzy decision tree in Figure 4.4 can be seen as a black box that maps continuous input features to continuous output classes.

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 methods 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

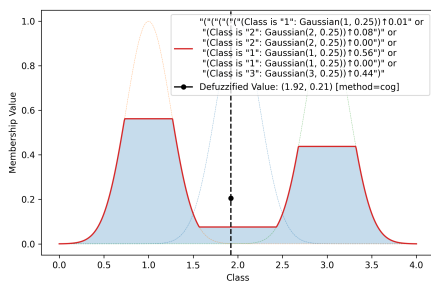


Figure 4.7: Resulting fuzzy set after applying the rules from Table 4.1 on the data point $(x = 2.95, y = 2.5)$. There are clear peaks at the class values 1 and 3. The COG method however returns Class 2, as it lies right in between the two peaks, turing the two good predictions

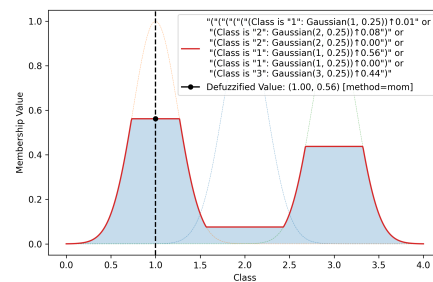


Figure 4.8: The MOM method returns the class value 1, as it is the mean of the two peaks at class values 1 and 3. This is a much better prediction than the one made by the COG method.

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.

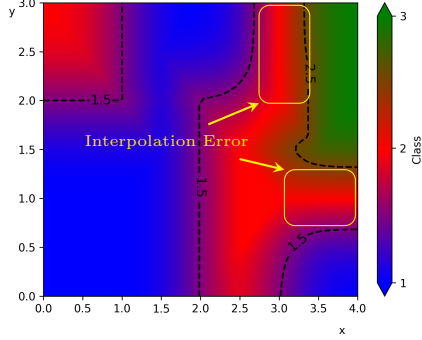


Figure 4.9: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the COG defuzzification method. The highlighted area shows the interpolation error of the COG method described in Figure 4.7.

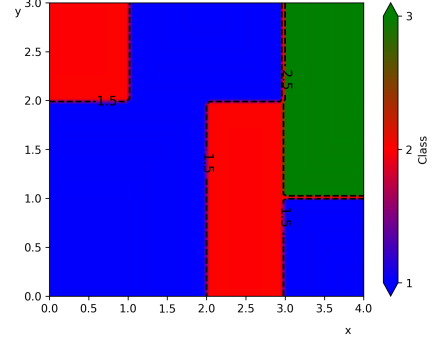


Figure 4.10: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the MOM defuzzification method.

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 `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, while the `TuningDataLogger` logs the current configuration and the time it took to execute the last tuning step. The full 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 don't 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. The data was collected on the CoolMUC-2 and primarily stems from the exam- add specs
 ple 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.5. All simulations were run on the Serial-Partition of the the cluster were repeated twice, to account for fluctuations in performance. Furthermore every simulation was run with 1, 4, 12, 24 and 28 threads to also gather data on how parallelization affects the ideal configuration.

To verify the sanity of the collected data, we can make plots about the distribution of the data and the nominal values of the collected data. The boxplot in Figure 4.11 shows the distribution of the collected data, while the pie charts in Figure 4.12 shows the relative proportions of the collected parameters. We can see that the data is quite balanced and that the nominal values are spread out quite evenly, which is a good sign for the quality of the collected data.

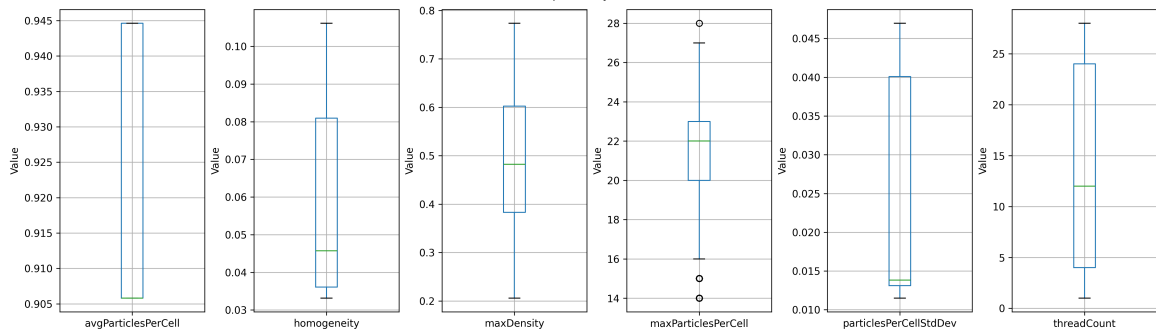


Figure 4.11: The boxplot shows the distribution of the collected data. The boxplot shows the median, the first and third quartile, and the whiskers show the range of the data.

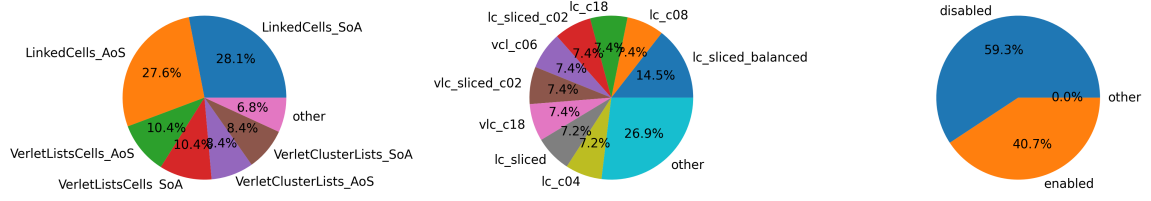


Figure 4.12: We can see that LinkedCells regardless of the Datalayout dominates the data. The Traversals and Newton3 options are spread out quite evenly

We can also do a more detailed analysis of the average performance of the different configuration options. As we froze the simulation during the tuning process, we can safely use the runtime of each iteration as a performance metric to compare all the tested configurations. For each tuning phase there is a unique ranking of all the configurations based on their runtime which we can use to calculate the relative speedup of each configuration compared to the best configuration. The formula for the relative speedup is given by:

$$\text{Relative Speedup} = \frac{\text{Best Configuration Runtime}}{\text{Configuration Runtime}} \quad (4.1)$$

This means that all relative speedup values are going to be in the range $[0, 1]$, with 1 being the best possible value. 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 4.13, Figure 4.14, Figure 4.15 and Figure 4.16 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 option, while the configuration VerletListCells_AoS_vlc_spliced_balanced_enabled is the best configuration in most cases on the Dataset we collected.

4.5.2 Creating the Fuzzy Rules

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

1. Collect the data from the simulation runs using the `LiveInfoLogger` and `TuningDataLogger`.
2. Train a decision tree on the collected data using a machine learning algorithm.
3. Convert the decision tree into a fuzzy decision tree using the method described in the previous sections.
4. Extract the fuzzy rules from the fuzzy decision tree.
5. Create the fuzzy system using the extracted rules and membership functions.

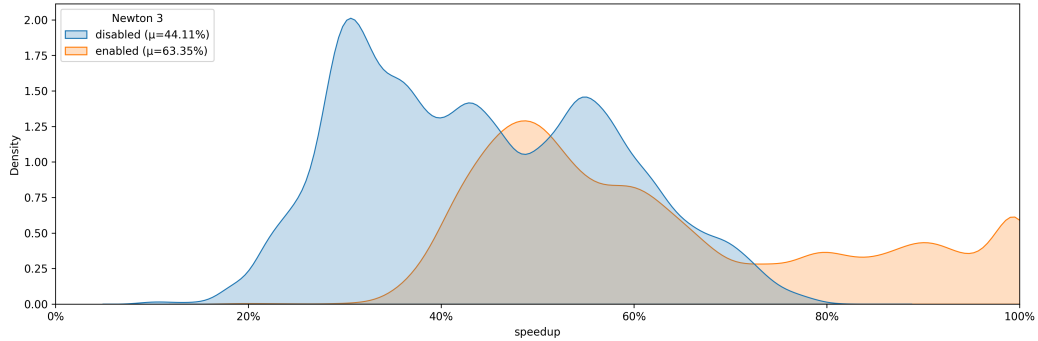


Figure 4.13: Density plot showing the distribution of the Newton3 options with respect to their relative speedup compared to the best configuration during each tuning phase. We can clearly see the two peaks whether Newton3 is enabled or disabled. It is also very obvious that Newton3=enabled is the better option as it generally allows for a higher relative speedup. It is interesting to note that all performances having relative speedups of at least 80% make use of the Newton3 optimization. Therefore we can confirm that Newton3 is generally a good option to enable.

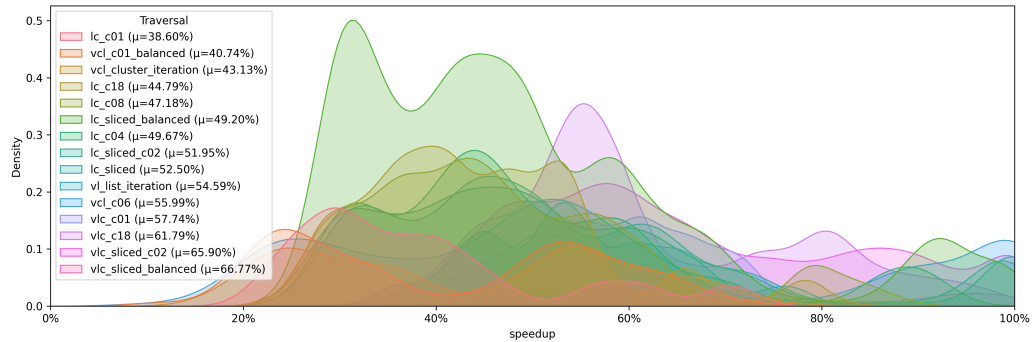


Figure 4.14: The density plot shows the distribution of the Traversal option with respect to the relative speedup compared to the best configuration during each tuning phase. There are no clear peaks in the data, but we can see that the vlc.sliced.balanced option generally performed better than the other options with an expected relative speedup of 66% compared to the best configuration.

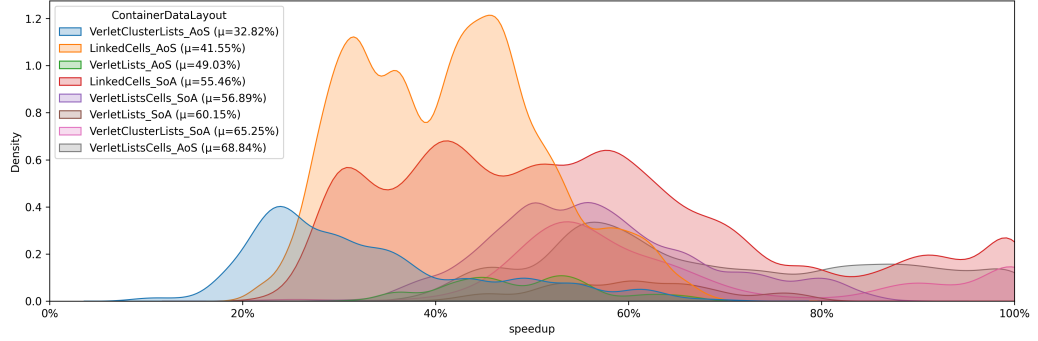


Figure 4.15: Density plot showing the distribution of the container-datalayout option with respect to the relative speedup compared to the best configuration during each tuning phase. The VerletListsCells_AoS container-data-layout is the best configuration in most cases with 68% of the runtime as the best configuration on average.

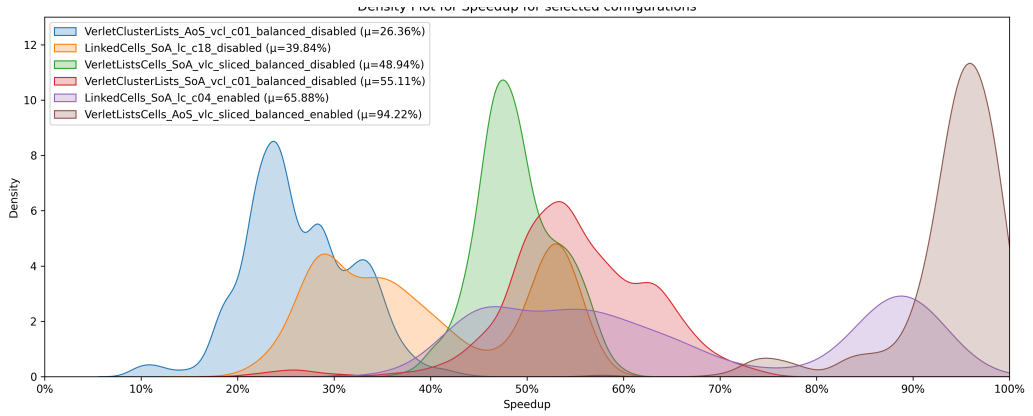


Figure 4.16: The density plot shows the distribution of the collected configurations with respect to the relative speedup compared to the best configuration during each tuning phase. The VerletListsCells_AoS_vlc_spliced_balanced_enabled configuration is the best configuration in most cases with 94% of the runtime as the best configuration on average.

List of Figures

3.1	Example of modular fuzzy set construction	22
4.1	Decision tree used for the example	26
4.2	Decision surface of the example decision tree	26
4.3	Conversion of crisp tree node into fuzzy tree node	27
4.5	Linguistic variables for the converted fuzzy decision tree	27
4.4	Fuzzy decision tree created from the regular decision tree	28
4.6	Fuzzy inference system created from the fuzzy decision tree seen as a black box	29
4.7	Resulting Fuzzy Set after applying the Rules on specific Data, COG Method	29
4.8	Resulting Fuzzy Set after applying the Rules on specific Data, MOM Method	29
4.9	Decision surface of the fuzzy rules using COG method	30
4.10	Decision surface of the fuzzy rules using MOM method	30
4.11	Boxplot of the collected Dataset	31
4.12	Pi Charts of nominal values of the collected Dataset	32
4.13	Speedup density plot of Newton 3 option	33
4.14	Speedup density plot of Traversal option	33
4.15	Speedup density plot of Configuration-Datalayout option	34
4.16	Speedup density plot of configurations	34

List of Tables

2.1	Common T-Norms and corresponding T-Conorms with respect to the standard negation operator $\neg x = 1 - x$ for $a, b \in [0, 1]$	13
2.2	Similarities between classical and fuzzy set operations	15
4.1	Extracted fuzzy rules from the fuzzy decision tree	28

Listings

3.1	Rule snippet depicting the Suitability Approach	23
3.2	Rule snippet depicting the Parameter Tuning Approach	24
A.1	ANTLR4 Rule Parser Grammar	41
A.2	explodingLiquid.yaml	43
A.3	spinodalDecompositionEquilibration.yaml	43
A.4	spinodalDecomposition.yaml	44
A.5	fallingDrop.yaml	45

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