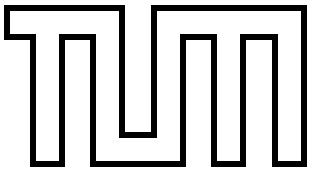


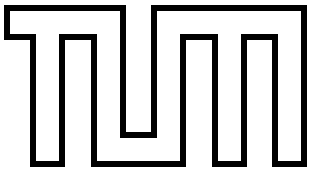
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

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

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Zusammenfassung

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

1. Introduction

Write some useful intro. Here are tips along the way:

1.1. A

2. Theoretical Background

2.1. Molecular Dynamics

Molecular Dynamics (MD) is a computational method used to simulate the behavior of atoms and molecules over time. In recent years, MD simulations have become essential in many scientific fields, including chemistry, physics, biology, and materials science. MD simulations are used to study various systems, ranging from simple gases and liquids to complex biological molecules and materials.

MD simulations' main goal is to understand complex systems' behavior at an atomic level and to predict their properties and behavior under different conditions. Using the power of computer simulations, where fundamental properties of the experiment can be changed by adjusting some formulas and parameters, allows researchers to study a wide variety of systems and conditions at a level of detail that is inaccessible to experimental methods alone [PS17].

Two examples of such simulations are shown in Figure 2.1 and Figure 2.2. The first image shows a simulation of the HIV-1 capsid, a protein shell that surrounds the genetic material of the human immunodeficiency virus (HIV). The second image shows an application of MD simulations in materials science, where researchers used computer simulations to study the crack formation in metallic glass.

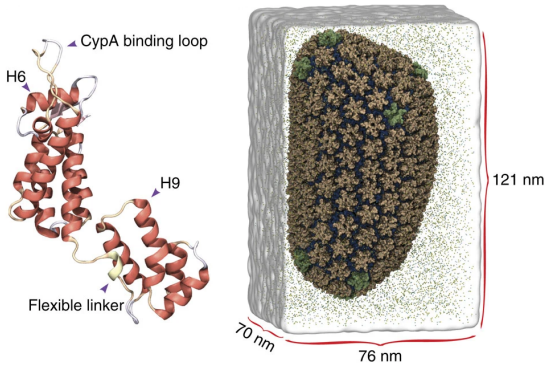


Figure 2.1.: Molecular dynamics simulations of the HIV-1 capsid. Perilla et al. [PS17] used a simulation containing 64,423,983 atoms to investigate different properties of the HIV-1 capsid at an atomic resolution.

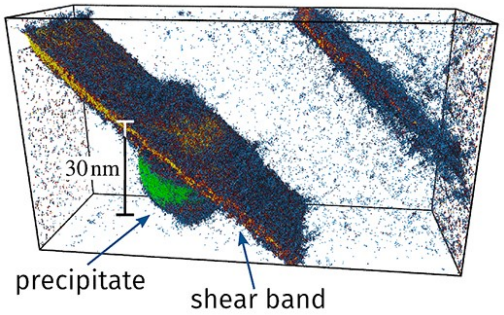


Figure 2.2.: Molecular dynamics simulations of shear band formation around a precipitate in metallic glass, as demonstrated by Brink et al. [BPR+16].

2.1.1. Quantum Mechanics and the Schrödinger Equation

Our current knowledge of physics suggests that the behavior of atoms and molecules is governed by the laws of quantum mechanics, where particles are described by wave functions and probabilities evolving. The physicist Erwin Schrödinger first formulated a mathematical model describing this phenomenon in 1926, which has gotten widespread acceptance and is now known widely as the Schrödinger equation. The Schrödinger equation is a partial differential equation describing how a physical system's wave function evolves over time. It is typically written as:

$$i\hbar\frac{\partial\Psi}{\partial t} = \hat{H}\Psi \tag{2.1}$$

Where Ψ is the system's wave function, $\hat{H}$ is the Hamiltonian operator describing the system's energy, t is the time, and $\hbar$ is the reduced Planck constant.

The Schrödinger equation provides a way to calculate the future states of a quantum system given the current state of the system. However, solving the Schrödinger equation for systems with many particles is computationally very expensive and quickly becomes infeasible for systems with more than a few particles [LM15]. Simulating a single water molecule consisting of three nuclei and 10 electrons would require solving a partial differential equation with 39 variables (The molecule consists of two hydrogen atoms, one oxygen atom, and 10 electrons. Each of those 13 objects is described by three spatial coordinates). Consequently, the resulting Shcrödinger equation also includes all 39 variables and looks like this:

$$i\hbar\frac{\partial\Psi}{\partial t} = -\hbar^2\sum_{i=1}^{13}\frac{1}{2m_i}\left(\frac{\partial^2\Psi}{\partial x_i^2}+\frac{\partial^2\Psi}{\partial y_i^2}+\frac{\partial^2\Psi}{\partial z_i^2}\right)+U_p(x_1,y_1,z_1,\ldots,x_{13},y_{13},z_{13})\Psi \tag{2.2}$$

Where m_i is the mass of the i-th object, x_i, y_i, and z_i are the spatial coordinates of the i-th object, and U_p is the potential energy function of the system.

As the Schrödinger equation is a partial differential equation, it is computationally expensive to solve for systems with many particles, as one quickly runs into the curse of dimensionality.

The Born-Oppenheimer approximation simplifies this problem by exploiting the significant mass difference between electrons and nuclei. From the perspective of fast-moving electrons, the much heavier nuclei appear nearly stationary, which justifies the separation of electronic and nuclear motions, treating the nuclei as classical particles. The electronic degrees of freedom can then be integrated out, deriving a new energy function U that depends solely on nuclear positions. This potential energy surface U is typically obtained through quantum mechanical calculations or fitted to empirical data.

However, this approach is still just an approximation, and depending on the chosen potential energy function U, the Born-Oppenheimer approximation may neglect certain quantum mechanical effects, causing inaccuracies in the simulation results.

Despite these limitations, the Born-Oppenheimer approximation is widely used in molecular dynamics simulations as it allows for the simulation of systems with many particles in the first place.

After applying the Born-Oppenheimer approximation and using Newton's second law of motion, the Schrödinger equation can be transformed into a system of ordinary differential equations. Each of those equations describes the motion of a single particle and is given by:

$$m_i \frac{d^2 x_i}{dt^2} = -\frac{\partial U}{\partial x_i} \tag{2.3}$$

$$m_i \frac{d^2 y_i}{dt^2} = -\frac{\partial U}{\partial y_i} \tag{2.4}$$

$$m_i \frac{d^2 z_i}{dt^2} = -\frac{\partial U}{\partial z_i} \tag{2.5}$$

Where m_i is the mass of the i-th particle, x_i, y_i, and z_i are the spatial coordinates of the i-th particle, and U is the newly derived potential energy function of the system.

2.1.2. Numerical Integration

Since the simulation domain potentially consists of a vast number of particles all interacting with each other, it is generally not possible to solve the equations of motion analytically. This problem is known under the N-body problem, and it can be shown that there are no general solutions for systems with more than two particles. We can, however, solve the equations of motion numerically using numerical integration methods. A standard method to solve the equations of motion numerically is the Verlet algorithm. This integration scheme is derived from the Taylor expansion of the position of the i-th object $\vec{r}_i$ at time $t - \Delta t$ and $t + \Delta t$ and is given by:

$$\vec{r}_i(t + \Delta t) = 2\vec{r}_i(t) - \vec{r}_i(t - \Delta t) + \vec{a}_i(t)\Delta t^2 \tag{2.6}$$

Where $\vec{a}_i(t)$ is the acceleration of the i-th object at time t and can be calculated from the particle mass and the acting forces using Newton's second law of motion $\vec{a}_i(t) = \frac{F_i}{m_i} = \frac{-\nabla_i U}{m_i}$.

2.1.3. Potential Energy Function

As stated above, the potential energy function U is a critical component of the molecular dynamics simulation as it fundamentally defines the properties of the system. MD simulations use many different potential energy functions, all of which are tailored to describe specific aspects of the system. Those potentials typically use a mixture of 2-body, 3-body, and 4-body interactions between the particles. The 2-body interactions express the effect of Pauli repulsion, atomic bonds, and coulomb interactions, while higher-order interactions allow for potentially asymmetric wave functions [LM15].

A common choice for the potential energy function is the Lennard-Jones potential. The simplicity of the Lennard-Jones potential makes it a popular choice for many systems, as it can be computed efficiently and still captures the attractive and repulsive forces between particles quite well. The Lennard-Jones potential is given by:

$$U_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{2.7}$$

Where r is the distance between the particles, ϵ is the depth of the potential well, and σ is the distance at which the potential is zero.

2.1.4. Simulation Loop

Using the methods described above, it is possible to simulate the behavior of a system of particles over time. The general simulation loop for a molecular dynamics simulation can be described as follows:

1. **Initialization**
   The simulation starts by initializing the positions and velocities of the particles. The initial positions and velocities can be chosen randomly or based on experimental data.
2. **Position Updates**
   In this step, the positions of the particles are updated using a numerical integration scheme such as the Verlet algorithm.
3. **Force Calculation**
   The forces acting on the particles are calculated based on the current positions of the particles and the chosen potential energy function U.
4. **Update Velocities and Accelerations**
   The acceleration of each particle is calculated using Newton's second law of motion. The velocities of the particles are then updated using the acceleration and the time step Δt.
5. **Apply External Forces or Constraints**
   In this step, the simulation can be modified by applying external forces or constraints to the particles. In this stage, it is possible to introduce boundary conditions, temperature control, or other outside effects to the simulation.
6. **Update Time and Repeat**
   The simulation time is updated by advancing it by the time step Δt. The simulation loop returns to step 2 and repeats until the desired number of integration steps is reached.

Many different software packages exist to perform such simulations. Some widely used examples of such systems are LAAMPS[1] and GROMACS[2]. Both efficiently solve the underlying N-body problem and provide the user with a high-level interface to specify the simulation parameters.

Many different approaches exist to efficiently solve the N-body problem, and no single best approach works well for all systems as the optimal implementation heavily depends on the simulation state and the hardware's capabilities to perform the simulation. However, LAAMPS and GROMACS use a single implementation and cannot adapt their algorithms to the current simulation state.

In the following section, we will introduce AutoPas, a library designed to efficiently deal with changing simulation states and capable of automatically adapting to the current simulation state to achieve optimal performance.

[1] `https://lammps.sandia.gov/`
[2] `https://www.gromacs.org/`

2.2. AutoPas

AutoPas is an open-source library designed to achieve optimal performance at the node level for short-range particle simulations. On a high level, AutoPas can be seen as a black box performing arbitrary N-body simulations with short-range particle interactions. The main goal of AutoPas is to provide a high-level interface for the user to perform simulations without worrying about the low-level details of the simulation. This is achieved by providing a high-level interface for the user to interact with the library while the library itself takes care of the low-level details of the force calculations

AutoPas provides many different algorithmic implementations for the problem of N-body simulations, each with different performance and memory usage trade-offs. No single implementation is optimal for all simulation scenarios , as the optimal implementation depends on the current simulation state. AutoPas is designed to be adaptive and can periodically switch between different implementations to achieve the best performance for the current simulation state. This is achieved by allowing AutoPas to automatically tune its internal parameters to find the best implementation for the current simulation state.

find reference

Since AutoPas provides a high-level interface for short-range N-body simulations, the user specifies the acting forces between the particles and controls the simulation loop. Fortunately, AutoPas also provides `md_flexible`, an example of a typical molecular dynamics simulation implementation.

2.2.1. Autotuning in AutoPas

AutoPas internally alternates between two phases of operation. The first phase is the *tuning phase*, where AutoPas tries to find the best configuration of parameters about a chosen performance metric. This optimal configuration is then used in the following *simulation phase*, assuming that the optimal configuration found in the tuning phase still performs reasonably well during the consequent simulation phase. As the simulation progresses and the characteristics of the system change, the previously chosen configuration can drift arbitrarily far from the actual optimal configuration. To counteract this, AutoPas periodically alternates between tuning and simulation phases to ensure that the used configuration is reasonably close to the optimal configuration. AutoPas acts as a black box during the simulation phase, solving the force calculations for the underlying N-body problem.

The power of AutoPas comes from its vast amount of tunable parameters and enormous search space. As mentioned, their design limits other software packages like LAAMPS and GROMACS to just one implementation. They can, therefore, operate outside of the theoretical optimal performance regime. In the following section, we will discuss the tunable parameters in AutoPas and the different tuning strategies available to find the best configuration of parameters for the current simulation state.

Tunable Parameters

AutoPas currently provides six tunable parameters, which can mostly[3] be combined freely. A collection of parameters is called a *Configuration*, and the set of all possible configurations is called the *Search Space*. A configuration consists of the following parameters:

[3]There are some exceptions as some choices of parameters are incompatible with each other.

1. **Container Options:**
   The container options are related to the data structure used to store the particles. The most important categories of data structures in this section are:

   a) **DirectSum**
   DirectSum does not use any additional data structures to store the particles. Instead, it simply holds a list of all particles and performs a brute-force calculation of the forces between all pairs of particles. This results in a complexity of $O(N^2)$ distance checks in each iteration. This method is simple and does not require additional data structures, but it has an inferior complexity, making it unsuitable for larger simulations. *Generally should not be used except for tiny systems or demonstration purposes. [VBC08]*

   b) **LinkedCells**
   LinkedCells segments the domain into a regular cell grid and only considers interactions between particles from neighboring cells. This results in the trade-off that particles further away than the cutoff radius are not considered for the force calculation. This is not a big issue in practice, as all short-range forces drop off quickly with distance anyway. Additionally, LinkedCells provides a high cache hit rate as particles inside the same cell can be stored contiguously in memory. Typically, the cell size is chosen to be equal to the cutoff radius r_c, meaning that each particle only needs to check the forces with particles inside the $3 \times 3 \times 3$ cell grid around it as all other particles are guaranteed to be further away than the cutoff radius. This reduction in possible interactions can result in a complexity of just $O(N)$ distance checks in each iteration. However, there is still room for improvement as constant factors can be pretty high. This is especially obvious, as most distance checks performed by LinkedCells still do not contribute to the force calculation [GST+19]. This trend can be explained due to the uneven scaling of sphere and cube volumes, especially for higher dimensions. For example, in 3D, the ratio of the volume of a sphere with radius r_c to the volume of a cube with side length $3r_c$ is given by:

$$\frac{\text{Interaction Volume}}{\text{Search Volume}} = \frac{V_{sphere}(r_c)}{V_{cube}(3r_c)} = \frac{\frac{4}{3}\pi r_c^3}{(3r_c)^3} = \frac{4}{81}\pi \approx 0.155 \tag{2.8}$$

   This means that only about 15.5% of all particles present in the $3 \times 3 \times 3$ cell grid around a particle are within the cutoff radius.

   However, still generally good for large, homogeneous[4] systems.

   c) **VerletLists**
   VerletLists is another approach to creating neighbor lists for the particles. Contrary to LinkedCells, VerletLists does not rely on a regular grid but instead uses a spherical region around each particle to determine its relevant neighbors. The algorithm creates and maintains a list of all particles present in a sphere within radius $r_c \cdot s$ around each particle, where r_c is the cutoff radius and $s > 1$ is the skin factor allowing for a buffer zone around the cutoff radius. By choosing a

[4] Homogeneous in this context, the particles are distributed evenly across the domain.

suitable buffer zone, such that no fast-moving particle can enter the cutoff radius unnoticed, it is possible to only recalculate the neighbor list every few iterations. This approach can benefit systems with high particle density and frequent interactions, as the neighbor list only needs to be updated every n iteration. This results in a complexity of $O(N)$ distance checks in each iteration. We can repeate the calculation from above to determine the ratio of the interaction volume to the search volume for VerletLists:

$$\frac{\text{Interaction Volume}}{\text{Search Volume}} = \frac{V_{sphere}(r_c)}{V_{sphere}(r_c \cdot s)} = \frac{\frac{4}{3}\pi r_c^3}{\frac{4}{3}\pi (r_c \cdot s)^3} = \frac{1}{s^3} \tag{2.9}$$

The ratio can be adjusted by changing the skin factor s this time. Ideally, the skin factor should be chosen so the ratio is close to 1. However, this reduces the buffer zone around the cutoff radius, meaning that the neighbor list needs to be updated more frequently. We conclude that choosing a skin factor that is too small can result in particles entering the cutoff radius unnoticed, leading to incorrect results, while choosing a skin factor that is too large can result in unnecessary distance checks.

Compared to LinkedCells, VerletLists can be constructed so that there are few unnecessary distance checks. However, constructing the neighbor list is quite memory intensive and can result in a high memory overhead. Additionally, the neighbor list needs to be updated every few iterations, which can result in a performance overhead.

Generally good for large systems with high particle density.

d) **VerletClusterLists**
VerletClusterLists differ from regular VerletLists in the way the neighbor lists are stored. Instead of storing the neighbor list for each particle separately, $n_{cluster}$ particles are grouped into a so-called *cluster*, and a single neighbor list is created for each cluster. This reduces memory overhead as the neighbor list only needs to be stored once for each cluster. Whenever two clusters are close, all interactions between the particles in the two clusters are calculated. This also results in a complexity of $O(N)$ distance checks in each iteration.

The main advantage of VerletClusterLists is the reduced memory overhead compared to regular VerletLists. However, constructing the neighbor list is still quite memory intensive and can result in a high memory overhead.

Generally suitable for large systems with high particle density

2. **Load Estimator Options:**
The Load Estimator Options are related to the way the simulation is parallelized. The load estimator is responsible for estimating the computational load of each MPI rank and how the work is distributed among the ranks. In this thesis, however, we will not further detail the Load Estimator Options as we primarily focus on the tuning aspect of single-node simulations.

3. **Traversal Options:**
These options are related to the traversal algorithm used to calculate the forces between

the particles. The traversal determines the order in which the particles are visited and how the forces are calculated. The different traversal options efficiently prevent race conditions when using multiple threads and allow load balancing at the node level [SGH+21].

Many different traversal algorithms are available in AutoPas, each with different performance and optimization potential trade-offs. In the following, we will discuss the most interesting traversal categories:

a) **Colored Traversal**
Since both LinkedCells and VerletLists only consider interactions with particles from neighboring cells/particles, it is possible to parallelize the force calculation by calculating forces for particles in different cells in parallel. This is, however, only possible if all simultaneously calculated particles do not share familiar neighbors, as this would introduce data races when updating the forces of the particles. This is where the concept of coloring comes into play. Coloring is a way to assign a color to each cell so that cells with the same color do not share familiar neighbors. This allows for the force calculation of particles in cells with the same color to be parallelized trivially, as data races are impossible.

There are many different ways to color the domain. Some of the most interesting coloring options are:

- **C01**
The C01 traversal is the simplest way of coloring the domain, as all cells get colored the same way. This means all cells can be perfectly parallelized (embarrassingly parallel) as long as Newton 3 is disabled. This, however, also means that all forces are calculated twice, once for each of the two particles involved.
- **C18**
The C18 traversal is a more sophisticated way of coloring the domain. The domain is divided into 18 colors, so no two neighboring cells share the same color. This method also utilizes the Newton 3 law to reduce the number of force calculations. This is achieved by only computing the forces with forward neighbors (neighbors with greater index.) [GSBN21]

b) **Sliced Traversal**
Sliced Traversal is a way to parallelize the force calculation by dividing the domain into different slices and calculating the forces for particles in different slices in parallel. It uses locks to prevent data races [GSBN21].

4. **Data Layout Options:**
The Data Layout Options determine how the particles are stored in memory. The two possible data layouts are:

a) **SoA**
The SoA (Structure of Arrays) data layout stores the particles' properties in separate arrays. For example, all particles' x-,y- and z-coordinates are stored in separate arrays. This data layout is beneficial for vectorization as the properties of the particles are stored contiguously in memory. This allows for efficient

vectorization of the force calculations as the properties of the particles can be loaded into vector registers in a single instruction.

find reference

b) **AoS**
The AoS (Array of Structures) data layout stores all particle properties in many structures. This allows for efficient cache utilization as the properties of the same particle are close to each other in memory. However, this data layout is not beneficial for vectorization as the properties of the particles are not stored contiguously in memory. This means that the properties of the particles need to be loaded into vector registers one by one, which can result in inefficient vectorization of the force calculations.

find reference

5. **Newton 3 Options:**
The Newton 3 Options relate to how the forces between the particles are calculated. The Newton 3 law states that for every action, there is an equal and opposite reaction

cite

. This means that the force between two particles is the same, regardless of which particle is the source and which is the target. In Molecular Dynamics simulations, this rule can be exploited to reduce the distance checks needed to calculate the forces between all pairs of particles by a factor of 2. The two possible Newton 3 options are:

a) **Newton3 Off**
If Newton 3 is turned off, the forces between all pairs of particles are calculated twice, once for each particle. This results in a constant overhead of factor 2.

b) **Newton3 On**
If Newton 3 is turned on, the forces between all pairs of particles are calculated only once. There is no more overhead due to recalculating the forces twice, but turning on Newton 3 requires additional bookkeeping, especially in multi-threaded environments. This results in more complicated traversal algorithms and can result in a performance overhead. *Generally should be turned on whenever available.*

6. **Cell Size Factor:**
The Cell Size Factor is a parameter that is used to determine the size of the cells in the LinkedCells-Container[5]. The cell size factor is typically chosen to be equal to the cutoff radius r_c, meaning that each particle only needs to check the forces with particles inside the $3 \times 3 \times 3$ cell grid around it as all other particles are guaranteed to be further away than the cutoff radius. We saw in the previous section that this could result in a high number of unnecessary distance checks as only about 15.5% of all particles present in the $3 \times 3 \times 3$ cell grid around a particle are within the cutoff radius. By choosing smaller cell sizes, this ratio can be increased, reducing the number of unnecessary distance checks. However, the increased overhead of managing more cells quickly offset the performance gain.

[5]The option is also relevant for other containers such as VerletLists those configurations internally also build their neighbor lists using a Cell Grid

Tuning Strategies

Tuning strategies are the heart of AutoPas. They implement the critical functionality of AutoPas' dynamic tuning aspect. The main goal of the tuning strategies is to find the best parameters for the current simulation state, which can then be used for the following simulation phase.

The default tuning strategy in AutoPas uses a brute-force approach to find the best parameters for the current simulation state by trying out all possible combinations of parameters and choosing the one that optimizes the chosen performance metric (e.g., time, energy usage). This approach is called *Full Search* and is guaranteed to find the best parameters for the current simulation state. However, it is typically very costly in terms of time and resources as it has to spend a lot of time measuring bad parameter combinations. This is a big issue as the number of possible parameter combinations can grow exponentially with the number of parameters. This makes the full search approach infeasible, especially if more tunable parameters are added to AutoPas.

To overcome this issue, AutoPas provides a couple of different tuning strategies that can be used to reduce the number of parameter combinations that need to be tested. This is generally achieved by allowing the tuning strategy to modify the set of parameters that need to be tested, thereby reducing the total number of configurations that need to be tested. It is, therefore, of great interest to develop tuning strategies that can effectively prune the search space, as this can result in a significant speedup of the tuning process.

In the following, we will briefly discuss the basic tuning strategies available in AutoPas:

1. **Full Search**
   The Full Search strategy is the default tuning strategy in AutoPas. It tries out all possible combinations of parameters and chooses the one that optimizes the chosen performance metric. It is guaranteed to find the best parameters for the current simulation state, but as mentioned before, it is very costly.
2. **Random Search**
   The Random Search strategy is a simple tuning strategy that randomly samples a given number of configurations from the search space and chooses the one that optimizes the chosen performance metric. This approach is faster than the Full Search strategy as it does not need to test all possible combinations of parameters. However, it is not guaranteed that the best parameters for the current simulation state will be found, which could suggest a wrong configuration.
3. **Predictive Tuning**
   The Predictive Tuning strategy attempts to extrapolate previous measurements to predict how the configuration would perform in the current simulation state. It prunes the search space only, keeping configurations predicted to perform reasonably well. The extrapolations are accomplished using methods such as linear regression or constructing polynomial functions through the data points. The method generally works well but is very sensitive to timing fluctuations.
4. **Bayesian Search**
   Two implementations of Bayesian tuning exist in AutoPas. Those methods apply Bayesian optimization techniques to predict suitable configurations using performance evidence from previous measurements.

5. **Rule Based Tuning**
   The Rule Based Tuning strategy uses a set of predefined rules to automatically filter out configurations that will perform poorly. The rules are built up using the configuration order of the form:

```
if numParticles < lowNumParticlesThreshold:
      [dataLayout="AoS"] >= [dataLayout="SoA"] with same
            container, newton3, traversal, loadEstimator;
endif
```

   The data layout "AoS" is generally better than "SoA" if the number of particles is below a certain threshold. The rule-based method can be very effective as expert knowledge can be encoded into the rules. Unfortunately, it is also its biggest drawback, as creating a good set of rules is not trivial.

This thesis aims to extend these tuning strategies with a new approach based on Fuzzy Logic. Conceptually, this new fuzzy logic-based tuning strategy is very similar to the rule-based tuning strategy as it uses expert knowledge of fuzzy rules to prune the search space. However, contrary to classical rules, fuzzy logic can deal with imprecise and uncertain information, which allows it to only partially activate rules and assign a degree of activation to each rule. All the rules can then be combined based on their degree of activation rather than just following the binary true/false logic. This allows for a more nuanced approach and allows the tuning strategy to interpolate the effect of many different rules to choose the best possible configuration, even if there is no direct rule for this specific case.

cite

Such fuzzy logic-based approaches can be especially beneficial when dealing with complex systems as they allow for a more nuanced approach to the problem. In the following section, we will introduce the basic concepts of fuzzy logic and how it is used to create a new tuning strategy for AutoPas.

2.3. Fuzzy Logic

Fuzzy Logic is a mathematical framework that allows for reasoning under uncertainty. It is an extension of classical logic and extends the concept of binary truth values (false/0 and true/ 1) to a continuous range of truth values in the interval $[0, 1]$. This allows for a more nuanced representation of the truth values of statements, which can be beneficial when dealing with imprecise or uncertain information. Instead of just having true or false statements, it is now possible for statements to be, for example, 40% true.

This concept is beneficial when modeling human language, as the words tend to be imprecise. For example, "hot" can mean different things to different people. For some people, a temperature of 30 degrees Celsius might be considered hot, while for others, a temperature of 40 degrees Celsius might be considered hot. There is no clear boundary between what is considered hot and what is not, but rather a gradual transition between the two. Fuzzy Logic allows for modeling such gradual transitions by assigning a degree of truth to each statement.

make image

2.3.1. Fuzzy Sets

Mathematically, the concept of Fuzzy Logic is based on Fuzzy Sets. A Fuzzy Set is a generalization of a classical set where an element can lie somewhere between being a set member and not being a member. Instead of having a binary membership function that assigns a value of 1 to elements that are members of the set and 0 to elements that are not, elements in a fuzzy set have a certain degree of membership in the set. This degree of membership is a value in the interval $[0, 1]$ that represents the degree to which the element is a member of the set, with 0 meaning that the element is not a member of the set at all and one meaning that the element is a full member of the set.

Formally a fuzzy set $\tilde{A}$ over a crisp/standard set X is defined by a membership function

$$\mu_{\tilde{A}} : X \to [0, 1] \tag{2.10}$$

that assigns each element $x \in X$ a value in the interval $[0, 1]$ that represents the degree to which x is a member of the set $\tilde{A}$. The classical counterpart is classically written using the element operator $\in_A$: $X \to \{true, false\}$.

The shape of the function can be chosen freely and depend on the specific application, but typical choices involve triangular, gaussian, or sigmoid-shaped functions.

insert image

2.3.2. Fuzzy Logic Operations

Fuzzy Sets are a generalization of classical sets, and as such, they also support the classical set operations of union, intersection, and complement. However, how these operations are defined differs from the classical case, as the membership functions are continuous and can take on any value in the interval $[0, 1]$.

The extension of classical sets makes use of the so-called De Morgan Triplets. Such a triplet $(\top, \bot, \neg)$ consists of a t-norm $\top : [0, 1] \times [0, 1] \to [0, 1]$, a t-conorm $\bot : [0, 1] \times [0, 1] \to [0, 1]$ and a strong complement operator $\neg : [0, 1] \to [0, 1]$. Those operators generalize the classical logical operators, which are only defined on the binary truth values $\{0, 1\}$ to values from the continuous interval $[0, 1]$. $\top$ can generalize the logical AND operator to fuzzy sets, while $\bot$ and $\neg$ can generalize the logical OR and NOT operators. The binary operators $\top$ and $\bot$ are often written in infix notation as $a \top b$ and $a \bot b$, similar to how classical logical operators are written.

For the t-norm $\top$ to be valid, it needs to satisfy the following properties:

$$\begin{aligned} a \top b &= b \top a && \text{//Commutativity} \\ a \bot b &\leq c \top d \quad \text{if } a \leq b \wedge b \leq d && \text{//Monotonicity} \\ a \top (b \top c) &= (a \top b) \top c && \text{//Associativity} \\ a \top 1 &= a && \text{//Identity Element} \end{aligned}$$

fix the layout

The complement operator $\neg$ needs to satisfy the following properties:

$$\begin{aligned} \neg 0 &= 1 && \text{//Boundary Conditions} \\ \neg 1 &= 0 && \text{//Boundary Conditions} \\ \neg y &\leq \neg x \quad \text{if } x \leq y && \text{//Monotonicity} \end{aligned}$$

Additionally, it is called a strong complement operator if it satisfies the following property:

$$\neg\neg x = x \quad \text{//Involution} \tag{2.11}$$

$$\neg y < \neg x \quad \text{if } x < y \qquad \text{//Strong Monotonicity} \tag{2.12}$$

The standard negation operator $\neg x = 1 - x$ is a strong complement operator as it satisfies all the properties above and is the most common choice for the negation operator in practice.

Some common choices for t-norms and t-conorms used in practice are shown in Table 2.1.

T-Norm Name	T-Norm $a \top b$	Corresponding T-Conorm $a \bot b$
Minimum	$\min(a, b)$	$\max(a, b)$
Product	$a \cdot b$	$a + b - a \cdot b$
Lukasiewicz	$\max(0, a + b - 1)$	$\min(1, a + b)$
Drastic	$\begin{cases} b & \text{if } a = 1 \\ a & \text{if } b = 1 \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} b & \text{if } a = 0 \\ a & \text{if } b = 0 \\ 1 & \text{otherwise} \end{cases}$
Einstein	$\frac{a \cdot b}{2-(a+b-a \cdot b)}$	$\frac{a+b}{1+a \cdot b}$

Table 2.1.: Common T-Norms and corresponding T-Conorms concerning the standard negation operator $\neg x = 1 - x$ for $a, b \in [0, 1]$

In the following sections, we will only consider the standard negation operator, the minimum t-norm, and the maximum t-conorm, as they are the most common choices in practice in fuzzy logic systems. However, if needed, the exact choices can be easily exchanged for other t-norms and t-conorms.

With these choices of t-norms, t-conorms, and negation operators, it is possible to define the classical set operations of union, intersection, and complement for fuzzy sets. The operations are defined as follows:

- **Intersection**
  The intersection $\tilde{C} = \tilde{A} \cap \tilde{B}$ of two fuzzy sets $\tilde{A}$ and $\tilde{B}$ both defined over the same crisp set X is defined by the new membership function

  $$\mu_{\tilde{C}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) \tag{2.13}$$

  This means that the degree of membership of an element x in the intersection set $\tilde{C}$ is just the minimum of the degrees of membership of x in the sets $\tilde{A}$ and $\tilde{B}$.

- **Union**
  The union $\tilde{C} = \tilde{A} \cup \tilde{B}$ of two fuzzy sets $\tilde{A}$ and $\tilde{B}$ both defined over the same crisp set X is defined by the new membership function

  $$\mu_{\tilde{C}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) \tag{2.14}$$

  This means that the degree of membership of an element x in the union set $\tilde{C}$ is just the maximum of the degrees of membership of x in the sets $\tilde{A}$ and $\tilde{B}$.

- **Complement**
  The complement $\tilde{C} = \neg\tilde{A}$ of a fuzzy set $\tilde{A}$ defined over the crisp set X is defined by the standard negation operator

$$\mu_{\tilde{C}}(x) = 1 - \mu_{\tilde{A}}(x) \tag{2.15}$$

  Again, this means that the degree of membership of an element x in the complement set $\tilde{C}$ is just 1 minus the degree of membership of x in the set $\tilde{A}$.

add grafical representation of the operations

2.3.3. Linguistic Variables

Linguistic variables collect multiple fuzzy sets defined over the same crisp set X into a single object. This variable then allows us to reason about the possible states of the variable more naturally.

For example, the linguistic variable "temperature" might have the linguistic terms *cold* *warm*, and *hot*, each of which is defined by a fuzzy set. This representation is very natural as it abstracts away the specific temperature values and allows us to reason about the temperature in a more human-like way.

All the underlying fuzzy sets can be chosen arbitrarily and can also overlap with each other.

add grafical representation of the linguistic variable

2.3.4. Fuzzy Logic Rules

Fuzzy Logic Rules are a way to encode expert knowledge into a Fuzzy Logic system. The rules specify the relationship between input and output variables of the system and, therefore, are the backbone of fuzzy logic systems. The rules are typically encoded in a human-readable way and often have the form "IF antecedent THEN consequent" where both the antecedent and the consequent are fuzzy sets. The antecedent is a condition that must be satisfied for the rule to be applied, while the consequent is the action taken if the rule is applied. Since we are not dealing with binary truth values, the antecedent can only be partially satisfied, and consequently, the rule is only partially applied.

The antecedent can be arbitrarily complicated and involve multiple fuzzy sets and logical operators. The consequent is typically a single fuzzy set modified by the rule but could theoretically be arbitrarily complicated. In this work, we allow rules following the grammar defined below:

$$\begin{aligned}
\langle\text{rule}\rangle ::= \;& \text{IF } \langle\text{fuzzy set}\rangle \text{ THEN } \langle\text{fuzzy set}\rangle && \text{//Rule} \\
\langle\text{fuzzy set}\rangle ::= \;& (\ \langle\text{fuzzy set}\rangle\ ) && \text{//Parentheses} \\
\mid\; & \langle\text{fuzzy set}\rangle \text{ AND } \langle\text{fuzzy set}\rangle && \text{//Conjunction} \\
\mid\; & \langle\text{fuzzy set}\rangle \text{ OR } \langle\text{fuzzy set}\rangle && \text{//Disjunction} \\
\mid\; & \text{NOT } \langle\text{fuzzy set}\rangle && \text{//Negation} \\
\mid\; & \tilde{A} \;=\; a && \text{//Selection}
\end{aligned}$$

The boolean operators AND, OR, and NOT represent the set operations of intersection, union, and complement, respectively.

Using this grammar, a typical rule might look like this:

$$\text{IF } (\tilde{A} = a \text{ AND } \tilde{B} = b) \text{ THEN } \tilde{C} = c \tag{2.16}$$

This rule states that if the state of the linguistic variable A is a and the state of the linguistic variable B is b, then the state of the linguistic variable C should be c. However, contrary to classical logic, the rule does not have to activate fully but can have a degree of activation in the interval $[0, 1]$. Should the antecedent be only partially true (for example, if $\mu_{\tilde{A}}(a) = 0.8$ and $\mu_{\tilde{B}}(b) = 0.6$), the rule is only partially applied and the effect of adapting the consequent is reduced accordingly? In the example above, the degree of activation of the rule would be $\min(0.8, 0.6) = 0.6$.

The inference step can be seen as an extension of the boolean implication operator

$$\text{IF antecedent THEN consequent} \iff \text{antecedent} \implies \text{consequent}$$

Moreover, it could be deduced from the choice of the t-norm and t-conorm operators and would lead to $(a \implies b) = \neg a \vee b = \max(1 - a, b)$. However, the Mamdani implication is typically used as the AND operation $\min(a, b)$. From a logical point of view, this choice is counterintuitive as it violates the standard definition. However, in the context of fuzzy systems, it is a perfect choice for computing the degree of validity for a rule [BMK96]. In practice, we are not interested in the resulting fuzzy set of the implication but rather in the extent to which the consequent should be adapted. Therefore, we introduce a slightly different inference algorithm:

Consider the rule IF $\tilde{A} = a$ THEN $\tilde{C} = c$

1. Obtain the input values $(x_1, x_2, \ldots, x_n) \in X_A$ occuring in the crisp set of the antecedent.
2. Evaluate the degree of membership μ those input values have in the antecedent. This is the degree to which the antecedent is satisfied, and the rule is activated.
3. Define a new fuzzy set $R = \tilde{C} \uparrow \mu$ where $\uparrow$ is the cut operator. This operator is defined as $\mu_R(x) = \min(\mu_{\tilde{C}}(x), \mu)$, which means that it cuts off all membership values of the fuzzy set $\tilde{C}$ that are above the degree of activation μ. This resulting set R contains the information to which extent the consequent should be adapted. We see that in the extreme cases where $\mu = 0$, the set R is also empty and does not affect further computations. If $\mu = 1$, the rule is activated fully, and the set R is equal to $\tilde{C}$. In all other cases, the set R is trimmed to the extent of activation.

2.3.5. Defuzzification

The final step in a Fuzzy Logic system is the defuzzification step. In this step, the fuzzy output of the system is converted back into a crisp value that can be used to control natural-world systems. The first step in defuzzification is to collect all the rules operating on the same output variable and combine their trimmed consequents into a single fuzzy set. This is done by just taking the fuzzy union of all those consequences, which results in a new

fuzzy set that represents the combined effect of all the rules on the output variable. This fuzzy set can be converted back into a single crisp value representing some aspect of the fuzzy set using a defuzzification method.

There are many different ways to defuzzify a fuzzy set. Some of the most common methods are:

- **Centroid**
  The Centroid method calculates the center of mass of the fuzzy set and returns this value as the crisp output. This method is intuitive as it tries to find a weighted interpolation of all the activated fuzzy sets. It is defined as:

$$\text{Centroid} = \frac{\int_X x \cdot \mu_{\tilde{C}}(x)\, dx}{\int_X \mu_{\tilde{C}}(x)\, dx} \tag{2.17}$$

- **Mean of Maximum**
  The Mean of Maximum method calculates all the input values that result in the maximum membership value of the fuzzy set and returns the average of these values as the crisp output. When there is just one maximum value, this method can be thought of as just returning the x-position of the most likely value.

  It is defined as follows:

$$\text{Mean of Maximum} = \frac{\int_{X'} x\, dx}{\int_{X'} dx} \tag{2.18}$$

  where $X' = \{x \in X \mid \mu_{\tilde{C}}(x) = \max(\mu_{\tilde{C}}(x))\}$ is the set of all input values that result in the maximum membership value of the fuzzy set.

- **Weighted Average**
  The Weighted Average method calculates the average of all the input values weighted by their membership values. Contrary to the Centroid method, which integrates over the whole domain, the Weighted Average method only considers a singular point from each membership function where the membership value is maximal. This can be seen as a simplification of the Centroid method and is defined as:

$$\text{Weighted Average} = \frac{\sum_{x \in X'} x \cdot \mu_{\tilde{C}}(x)}{\sum_{x \in X'} \mu_{\tilde{C}}(x)} \tag{2.19}$$

  where X' is the set of all input values resulting in their fuzzy set's maximum membership value.

Also, many other methods exist to defuzzify a fuzzy set, but the ones mentioned above are the most common choices in practice. This thesis focuses on the Centroid and the Mean of Maximum methods.

2.3.6. Structure of creating a Fuzzy Logic System

All the building blocks of a Fuzzy Logic System have been introduced in the previous sections and can now be combined to create a complete Fuzzy Logic System. The general structure of a Fuzzy Logic System is as follows:

1. **Fuzzification**
   The first step in a Fuzzy Logic System is to convert the crisp input values into fuzzy sets. This is done by evaluating the membership functions of the fuzzy sets at the crisp input values. This results in a bunch of membership values, which can be used to calculate the boolean operations of the antecedents of the rules.
2. **Inference**
   The next step is to apply the fuzzy logic rules to the fuzzy input values. This is done by using the degree of membership of the input values in the rules' antecedents to calculate each rule's activation degree. The consequent of each rule is then cut by the degree of activation of the rule to determine the effect of the rule on the output variable. This results in many fuzzy sets representing all the active effects on the output variable.
3. **Aggregation**
   The fuzzy sets resulting from the inference step are combined into a single fuzzy set containing the combined effect of all the rules on the output variable. This is done by taking the fuzzy union of all the fuzzy sets.
4. **Defuzzification**
   The final step is to convert the fuzzy output value into a crisp value that can be used to control natural-world systems. This is done by applying a defuzzification method to the fuzzy set representing the combined effect of all the rules on the output variable.

If the system is finished, it can be seen as a black box that takes crisp input values and returns crisp output values similar to a function $f : X \to \mathbb{R}$.

Using such a system, however, requires much expert knowledge as the rules and the membership functions of the fuzzy sets need to be defined by hand. This can be time-consuming and sometimes even impossible if the system is too complex. Luckily, other methods exist that attempt to automate defining the parameters of the fuzzy logic system. Some standard methods are:

- **Genetic Algorithms**
  Genetic Algorithms are a class of optimization algorithms that are inspired by the process of natural selection. They maintain a population of candidate solutions to a problem and iteratively improve them by applying genetic operators such as mutation and crossover. Genetic Algorithms can optimize the parameters of a fuzzy logic system by treating the parameters as an individual's genes and the system's performance as the individual's fitness. By iteratively evolving the population of individuals, it is possible to find a set of parameters that optimizes the performance of the fuzzy logic system.
- **Data Driven Methods**
  Data Driven Methods are a class of optimization algorithms that work by using data to optimize the parameters of a fuzzy logic system. Those methods are often based on machine learning algorithms such as decision trees. They work by trying to find some interpretable representation of the data that can be used to define concrete rules for the fuzzy logic system.

- **Fuzzy Clustering**
  Fuzzy Clustering is a class of clustering algorithms that work by assigning each data point to a cluster with a certain degree of membership. It can be used to optimize the parameters of a fuzzy logic system by treating the data points as the input values of the system and the clusters as the fuzzy sets of the system. By iteratively updating the clusters to better fit the data points, it is possible to find a set of parameters that optimizes the performance of the fuzzy logic system.

find sources for all of them

3. Implementation

This chapter describes the implementation of the Fuzzy Tuning technique in AutoPas. The implementation is divided into two main parts: the Fuzzy Tuning framework and the Tuning Strategy. The Fuzzy Tuning framework is the core of this implementation and implements the mathematical foundation of this technique. The tuning strategy acts as the interface between the fuzzy tuning framework and the AutoPas simulation and is therefore responsible for updating AutoPas' configuration queue.

3.1. Fuzzy Tuning Framework

The Fuzzy Tuning framework implements the mathematical foundation of the Fuzzy Tuning technique. It consists of several components that work together to apply the Fuzzy Rules to the input variables and generate the output variables. The components of the Fuzzy Tuning framework are as follows:

- **Crisp Set:** The Crisp Set is used to model k-cells used as the underlying sets over which the Fuzzy Sets are defined. A k-cell is a hyperrectangle in the k-dimensional space constructed from the Cartesian product of k intervals $I = I_1 \times I_2 \times \ldots \times I_k$ where $I_i = [x_{low}, x_{high}] \subset \mathbb{R}$ is an interval in the real numbers. They are used to define the underlying sets over which the corresponding fuzzy set is defined and can be thought of as the parameter space of the input variable.
- **Fuzzy Set:** A fuzzy set consists of a membership function that assigns a degree of membership to each element of the Crisp Set. There are two types of membership functions: The `BaseMembershipFunction` and the `CompositeMembershipFunction`. The `BaseMembershipFunction` implements a function $f : \mathbb{R} \to [0, 1]$ that directly maps the crisp value to the degree of membership. The `CompositeMembershipFunction` is used to create new fuzzy sets by recursively combining existing fuzzy sets and their membership functions with generic functions. This helps to split up the complex fuzzy sets of the rule base into smaller, more manageable parts. Let us say we have a fuzzy set $\tilde{A}$ defined over the Crisp Set X and a fuzzy set $\tilde{B}$ defined over the Crisp Set Y. Both membership functions are functions mapping the respective Crisp Set to the degree of membership ($\mu_{\tilde{A}} : X \to [0, 1]$ and $\mu_{\tilde{B}} : Y \to [0, 1]$). Since both membership functions directly map the Crisp Set to the degree of membership, they are considered `BaseMembershipFunctions`. When we want to create a new fuzzy set $\tilde{C} = \tilde{A} \cap \tilde{B}$ this new fuzzy set is defined over the Crisp Set $X \times Y$ and thus needs to provide a membership function $\mu_{\tilde{C}} : X \times Y \to [0, 1]$. As described in this membership function is defined as $\mu_{\tilde{C}}(x, y) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y))$, which can be thought of as recursively combining the membership functions of $\tilde{A}$ and $\tilde{B}$ with the minimum function.

  add chcpter

  This way of combining fuzzy sets builds a tree structure where the leaves calculate a direct membership value, and the inner nodes combine the membership values of their

children and pass them up to their parents. Figure 3.2 shows how a complex fuzzy set can be constructed from more straightforward fuzzy sets.

- **Linguistic Variable:** A linguistic variable is a variable whose values are terms in a natural language. Each term is associated with a fuzzy set that defines its concept using the language of fuzzy logic. The linguistic variables can then be used to express rules in a human-readable way.
- **Fuzzy Rule:** A fuzzy rule is a conditional statement that describes the relationship between input- and output variables. It consists of an antecedent and a consequent, both of which are fuzzy sets. During the evaluation of the rule, the antecedent is evaluated to determine the degree to which the rule is satisfied; thus, the rule's effect can be reduced accordingly.
- **Fuzzy Control System:** The Fuzzy Control System combines all the concepts described above to create a system that can evaluate a set of fuzzy rules and generate an output based on the input variables. Such a control system acts like a black box $f : \mathbb{R}^n \to \mathbb{R}$ that takes a set of crisp input variables and returns the predicted value.

A simplified class diagram of the Fuzzy Tuning strategy can be seen in Figure 3.1.

3.2. Tuning Strategy

The Tuning Strategy implements the interface between the Fuzzy Tuning framework and the AutoPas simulation. It is responsible for interacting with AutoPas and updating the configuration queue.

It does this by evaluating all fuzzy systems present in the rule file with the current information on the simulation. The LiveInformation is a snapshot of the current state of the simulation. It contains summary statistics about various aspects of the simulation such as the number of particles, the average density, or the total number of particles. Each evaluation of a Fuzzy Control System yields a single numeric value, which is then passed on to the `OutputMapper`, which assigns the associated configurations to this predicted value.

This method of assigning concrete configurations to the continuous output space of the Fuzzy Control Systems is inspired by Mohammed et al. [MKEC22]'s work on scheduling algorithms. Internally, the `OutputMapper` stores an ideal location for each available configuration and determines the configuration closest to the predicted value.

The resulting list of configurations predicted by the Fuzzy Tuning Strategy is then used to update AutoPas's configuration queue. In the following iterations, all of those configurations are used to run a few steps of the simulation, and the configuration with the best performance is then used for the following simulation phase.

Currently, two different modes of interpreting the rules are implemented:

3.2.1. Individual Tuning Approach

A more lightweight approach is the Parameter Tuning Approach. This approach creates a single Fuzzy Control System for each tunable parameter. All Fuzzy Control Systems then attempt to independently predict the optimal value for their respective parameter. Using this approach, the rule file only needs to define $\#Parameters$ Fuzzy Control Systems,

which is much more manageable. The drawback of this approach is that the continuous output space of the Fuzzy Control Systems needs to be directly translated to discrete values, which is a non-trivial task. The main problem is that the tunable parameters are nominal values and thus have no natural order. This means that the interpolation between different linguistic terms, naturally performed by the Fuzzy Control Systems, is not meaningful in this context. To avoid this problem, we can use a defuzzification method that does not rely on performing some interpolation. One such method is the SOM method, which selects the smallest x-value for which the membership function is maximal. Using this method, only a discrete set of possible outputs can be directly mapped to the nominal values of the tunable parameters using the OutputMapping construct of the Rule Parser. Since there is no interpolation between the linguistic terms, the order of the terms in the OutputMapping can be chosen arbitrarily.

find citations for fuzzy rules for nominal values

link to later section

3.2.2. Suitability Tuning Approach

The Suitability Approach is designed to tackle rule bases that try to predict a suitability value directly for *each* configurations. The rule file needs to define $\#Containers \cdot \#Traversals \cdot \#DataLayouts \cdot \#Newton3\_options$ different Fuzzy Control Systems—one for each possible configuration of those parameters. As a result, we receive a direct ranking (maybe from 0 to 100% applicability) for each configuration. The Tuning Strategy then selects all configurations within a certain threshold of the highest-ranking configuration and rewrites the queue of configurations with these selected configurations.

This method is a natural choice for rule bases as its style closely relates to the idea of Fuzzy Control Systems. It is possible to fully use the continuous output space of the Fuzzy Control Systems, and the method provides a straightforward mapping to the discrete AutoPas configuration space.

The huge downside of this approach is the need to define a massive amount of Fuzzy Control Systems and rules in the rule file. This leads to a considerable rule file, quickly becoming infeasible to maintain by hand.

link to later section

3.3. Rule Parser

The Rule Parser is responsible for parsing the rule base supplied by the user and converting it into the internal representation used by the Fuzzy Tuning framework. It uses the ANTLR4[1] parser generator to create a parser for a domain-specific language tailored to the needs of the Fuzzy Tuning. The language is inspired by common standards such as the Fuzzy Control Language (FCL)[2] but is designed to be more lightweight. The transformation of the parsed rules into the internal representation is done by a visitor pattern that traverses the parse tree generated by ANTLR4 and internally builds the corresponding objects of the Fuzzy Tuning framework.

[1] https://www.antlr.org/
[2] https://www.fuzzylite.com/

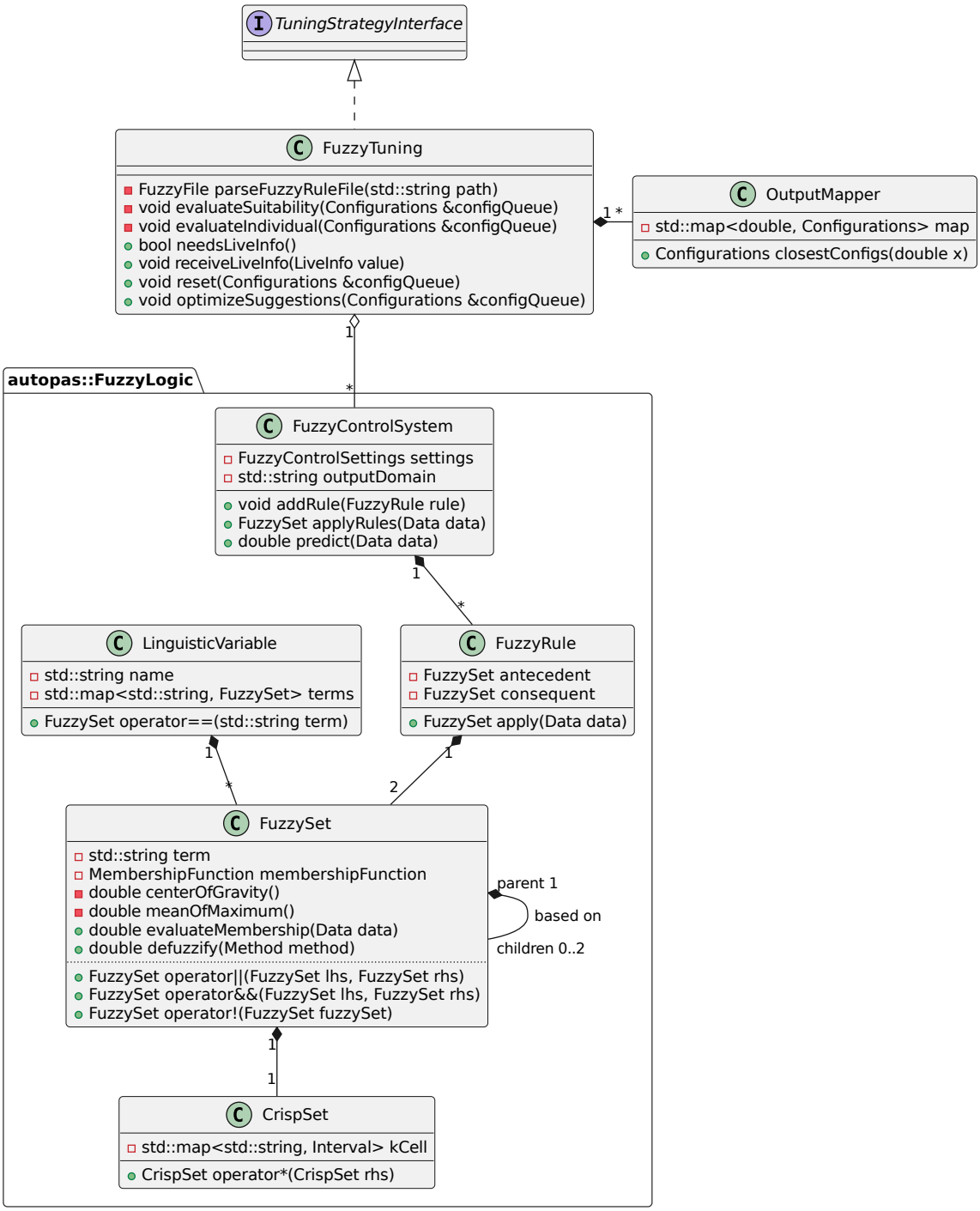


Figure 3.1.: Simplified class diagram of the Fuzzy Tuning strategy. There is a clear separation between implementing the Fuzzy Logic framework and the tuning strategy. This allows for an easy reuse of the Fuzzy Logic framework in other parts of AutoPas if desired.

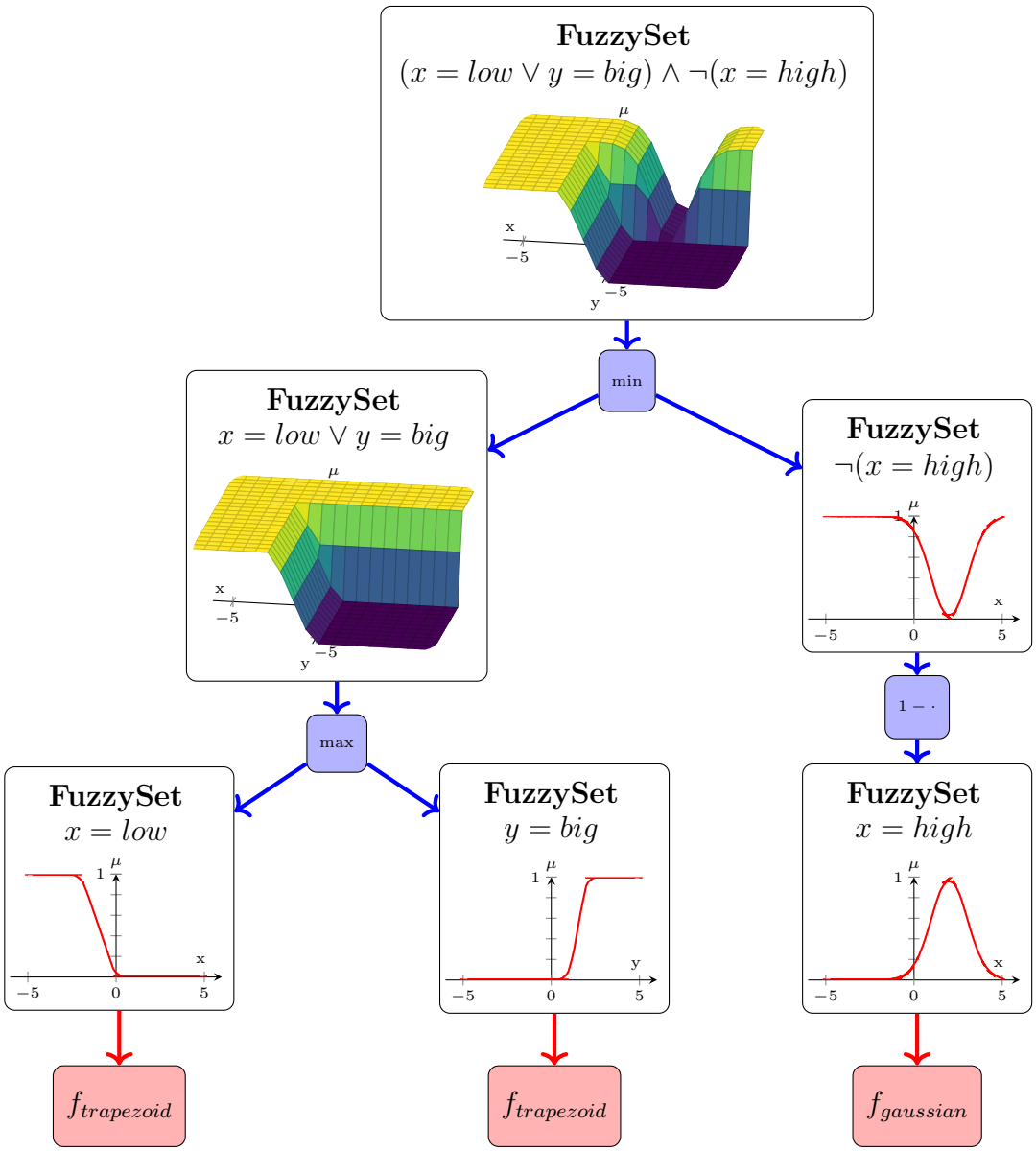


Figure 3.2.: Recursive construction of a complex fuzzy set from simpler fuzzy sets. Using the linguistic variables x with the terms $\{low, high\}$ and y with the terms $\{big, small\}$ we can construct the fuzzy set $(x = low \vee y = big) \wedge \neg(x = high)$ by combining the fuzzy sets $x = low \vee y = big$ and $\neg(x = high)$. Those fuzzy sets are again constructed from the simpler fuzzy sets $x = low$, $y = big$ and $x = high$.
The fuzzy sets at the leaf level can be directly constructed using predefined `BaseMembershipFunctions` (e.g., trapezoid, sigmoid, gaussian ...) and provide the foundation for the more complex fuzzy sets All other fuzzy sets are created by combining other fuzzy sets using `CompositeMembershipFunctions`. The logical operators min, max, and $1 - \cdot$ are implemented this way, as they directly act on top of other fuzzy sets.

4. Proof of Concept

This chapter presents 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

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. However, there are also methods to use a data-driven approach to create the knowledge base automatically. This is especially useful as those methods do not require prior expert knowledge about the system. However, regardless of how the knowledge base is created, it is still possible to manually evaluate and adjust the rules to add manual expert knowledge to the system. Such data-driven methods can be a good starting point for creating a fuzzy system, providing a good initial set of rules that experts can further refine.

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 train a classical decision tree on the dataset. Then, it converts the decision tree into a fuzzy decision tree, which can be used to extract the linguistic variables and fuzzy rules.

Add references to the methods

4.2. 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 [Mur12] so 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 straightforward 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.

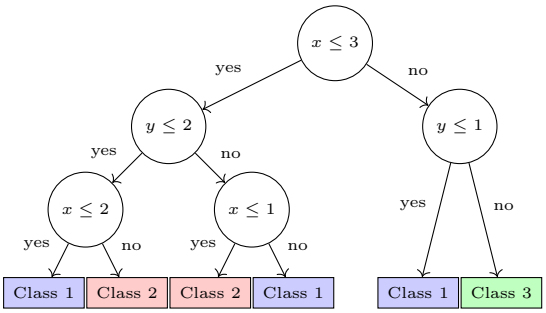


Figure 4.1.: An example decision tree for a dataset with two features x and y. There are three distinct classes in the dataset

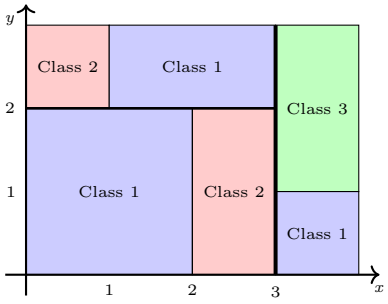


Figure 4.2.: The decision surface of the decision tree from Figure 4.1 on $\mathcal{D} = [0, 4] \times [0, 3]$.

4.3. Fuzzy Decision Trees

Fuzzy decision trees are a generalization of classical decision trees that allow for fuzzy logic to be used in decision-making. This extension eliminates the crisp decision boundaries of classical decision trees. Instead, it uses 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 to consider the input data's uncertainty 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 simultaneously 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

This section 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.

A classical decision tree is converted to a fuzzy decision tree 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. However, instead of returning a binary value of whether to continue down the left or right branch, they return a value in the $[0, 1]$ range 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. An obvious choice for the membership functions is to use `sigmoid` functions with a center at the crisp decision boundary, as this maintains the semantics of the branch. Crockett et al. [CBMO06] suggested defining symmetric sigmoid functions on the interval $[t - n \cdot \sigma, t + n \cdot \sigma]$ where t is the crisp 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. The *width* of the membership function describes the interval where there is a significant change in the membership value. Outside of this interval, the membership value is close to 0 or 1, depending on the side of the interval.

The value of n is typically chosen from the interval $n \in [0, 5]$ as otherwise, the membership function can become too broad, weakening the decision-making process [CBMO06]. In this work, we will use $n = 2$ as a default value.

By using a data-dependent *width* of the membership functions, it is possible to fully automate the conversion of a decision tree into a fuzzy decision tree, which we will utilize in this work.

The conversion of a crisp decision boundary into a fuzzy decision boundary is shown in Figure 4.3 using the example split $x \leq 3$.

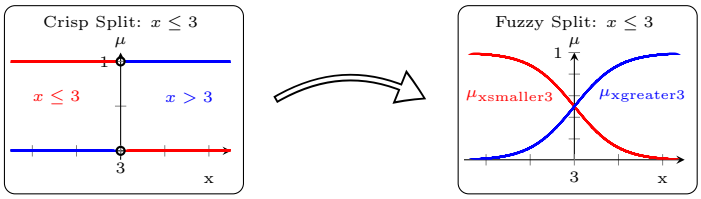


Figure 4.3.: Conversion of a crisp decision surface to a fuzzy decision surface. The crisp decision surface $x \leq 3$ is replaced by two `sigmoid` membership functions $\mu_{\text{xsmaller3}}$ and $\mu_{\text{xgreater3}}$ that specify to which degree the comparison is true or false. The *width* of the membership function is data dependent and is determined by $n \cdot \sigma = 2 \cdot \sigma$.

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 we will use `gaussian` functions with a specific mean and variance. Placing the different class-specific membership functions is very important, as it can heavily influence the defuzzification process for some defuzzification methods. Fuzzy systems typically use an interpolation between the different class values to determine the final output. The resulting conversion of the 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 membership functions' specific shapes have been picked arbitrarily and can be adjusted to better fit the data. The resulting linguistic variables are shown in Figure 4.5.

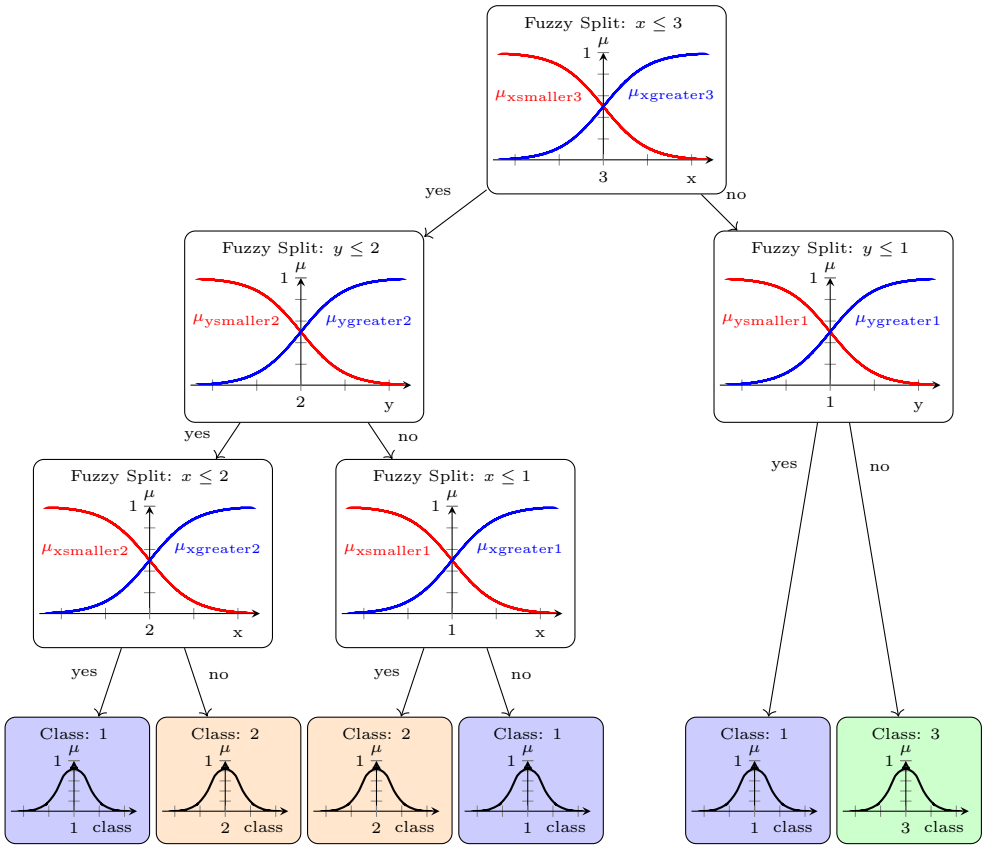


Figure 4.4.: The fuzzy decision tree corresponding to the decision tree in Figure 4.1. Each internal node in the fuzzy decision tree uses two `sigmoid` membership functions (μ_{smaller} and μ_{greater}) to specify to which degree the comparison is true or false. The leaf nodes use different `gaussian`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

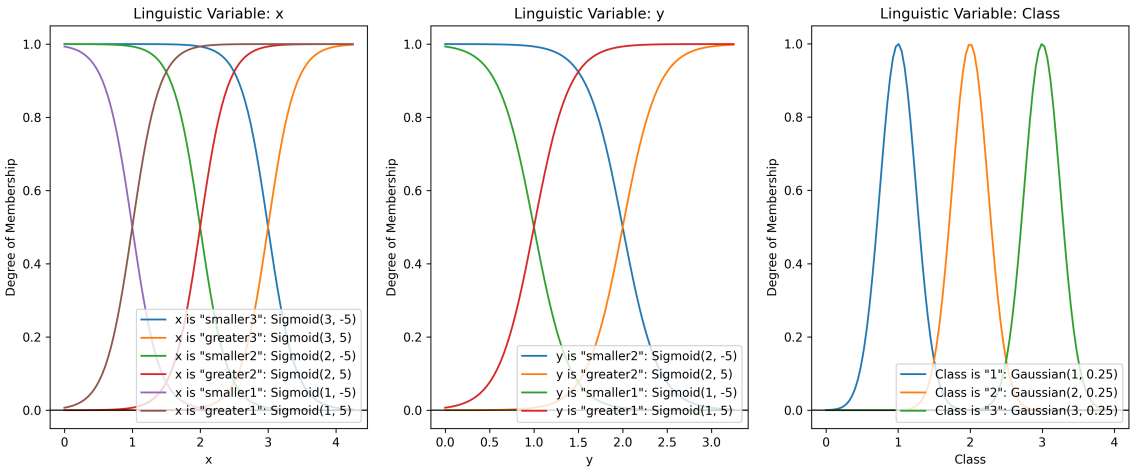


Figure 4.5.: Linguistic variables used in the fuzzy decision tree in Figure 4.4. The standard deviation of the attributes is assumed to be $\sigma \approx 0.5$ such that the *width* of the sigmoid membership functions is $n \cdot \sigma \approx 1$. The class values are assumed to be normally distributed and are places where they don't overlap much.

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. In contrast, each final connection between 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 precisely 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 fuzzy system's decision surface by evaluating the rules' membership functions for each point in the input space. This decision surface can then be used to understand the fuzzy system's decision-making process and identify possible errors in the rules or membership functions.

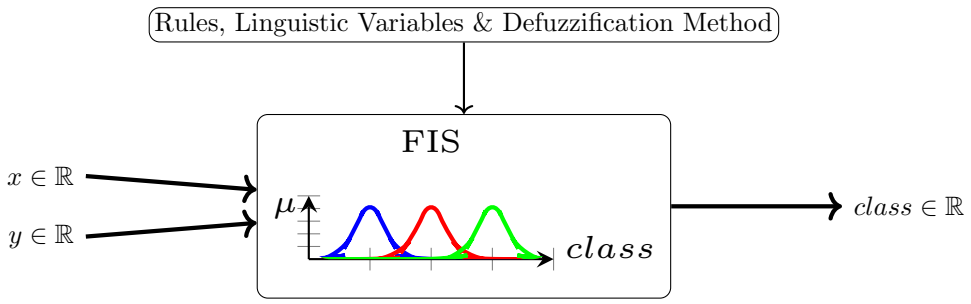


Figure 4.6.: The fuzzy inference system created from the fuzzy decision tree in Figure 4.4 can be seen as a black box that maps continuous input features to continuous output classes.

Choice of Defuzzification Method

The exact shape of the decision surface depends on the defuzzification method used. 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 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, 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

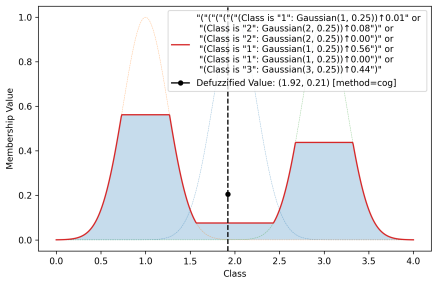


Figure 4.7.: Resulting fuzzy set after applying the rules from Table 4.1 on the data point ($x = 2.95, y = 2.5$). There are clear peaks at the class values 1 and 3. However, the COG method returns Class 2, as it lies right between the two peaks, turning the good predictions into bad ones.

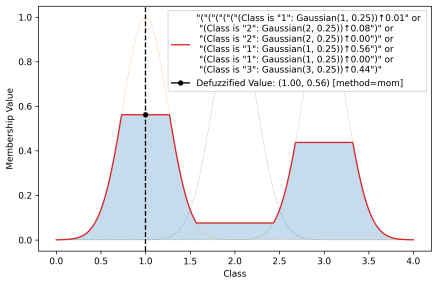


Figure 4.8.: The MOM method returns the class value 1, as it is the mean of the two peaks at class values 1 and 3. This is a much better prediction than the one made by the COG method.

Calculating the fuzzy system's whole decision surface is also possible by evaluating the rules' membership functions 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.

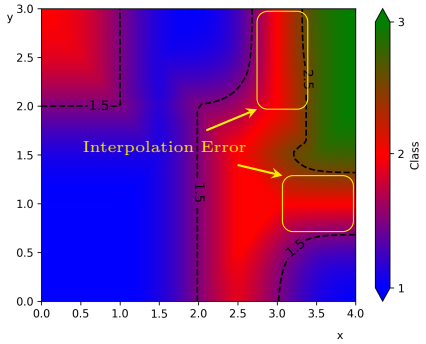


Figure 4.9.: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the COG defuzzification method. The highlighted area shows the interpolation error of the COG method

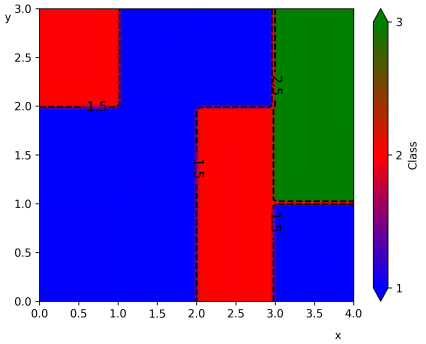


Figure 4.10.: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the MOM defuzzification method.

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. The following section uses this approach to create a fuzzy system to predict optimal configuration parameters for `md_flexible` simulations.

4.5. Creating Fuzzy Rules 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 FIS, 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.

4.5.1. 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 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

4.5.2. Data Analysis

To verify the sanity of the collected data, we can make plots about the data's distribution and the collected data's nominal values. The boxplot in Figure A.1 shows the distribution of the collected data, while the pie charts in **??** show the relative proportions of the collected parameters. We can see that the data is quite balanced and that the nominal values are spread out relatively evenly, which is a good sign for the quality of the collected data.

4.5.3. Speedup Analysis

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. Each tuning phase has 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:

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

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.

4.5.4. 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 rule bases, one for the Suitability Approach and one for the Individual Tuning Approach. In the following sections, we will describe both approaches in detail.

4.5.5. 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%

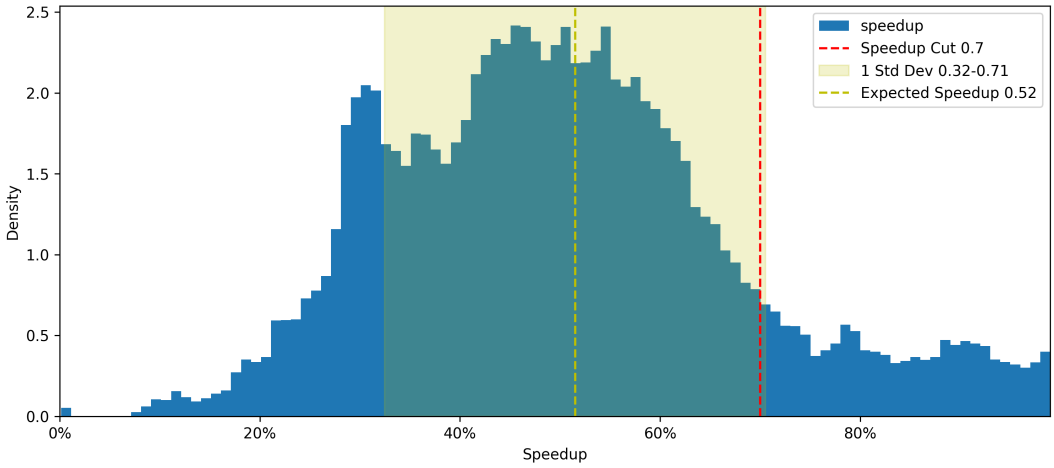


Figure 4.11.: The density plot shows the distribution of all the collected configurations concerning the relative speedup compared to the best configuration during each tuning phase. This distribution shows that the average tested configuration performs just 50% as well as the best. With some configurations 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.

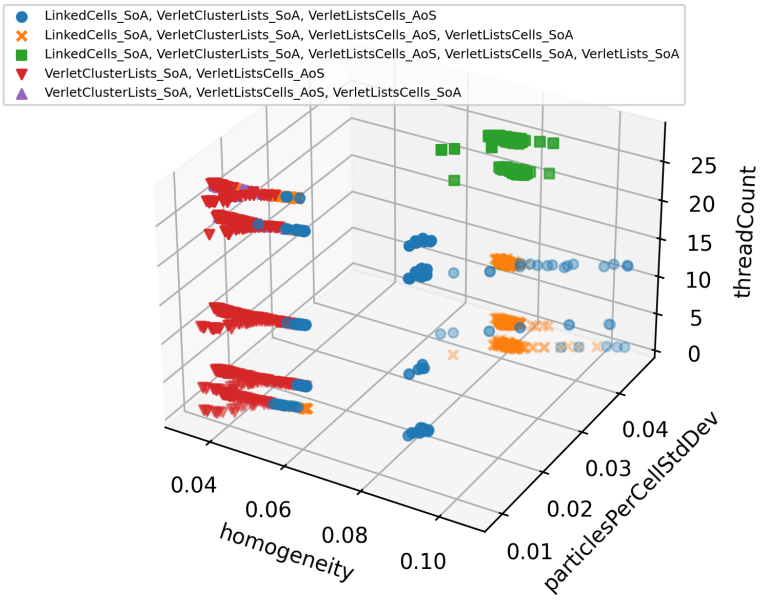


Figure 4.12.: The scatterplot shows the influence of the variables `homogeneity`, `particlesPerCellStdDev` and `threadCount` on the optimal `ContainerDataLayout` parameter. We can see regions that will be learned by the decision tree.

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.

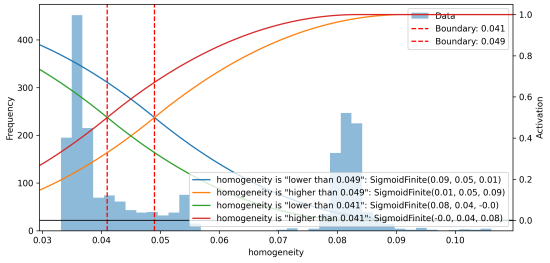


Figure 4.13.: This figure shows the linguistic variable for the homogeneity attribute. We see the different fuzzy sets created from the decision trees. The background shows the histogram of all homogeneity values present in the dataset.

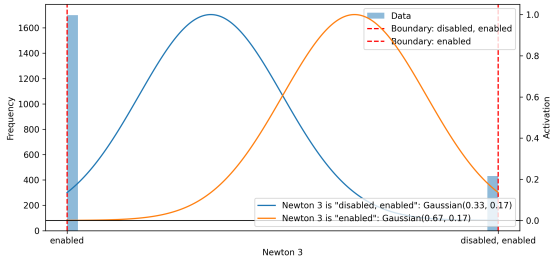


Figure 4.14.: This figure shows the linguistic variable for the Newton3 attribute. We see the two `gaussian` membership functions representing the two possible values for the Newton3 attribute.

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.

4.5.6. 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 **??**.

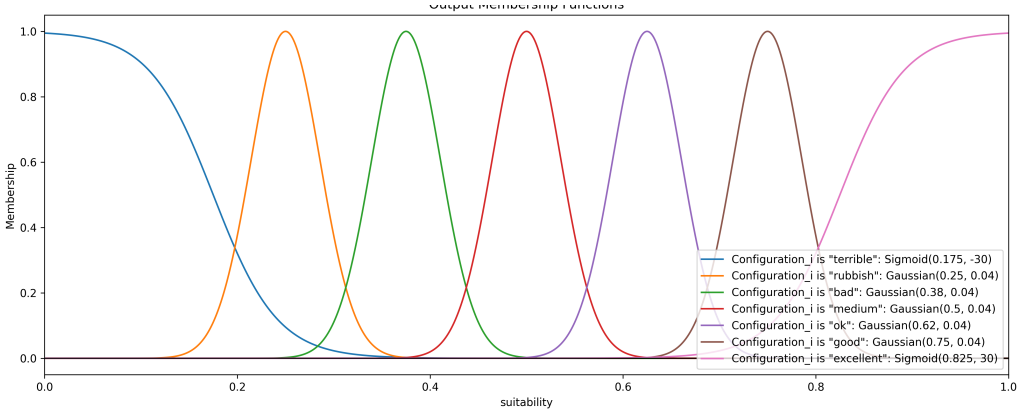


Figure 4.15.: Linguistic terms for the suitability variables. The fuzzy sets consist of `sigmoid` membership functions at the borders and `gaussian` membership functions in the middle. The values are placed coherently. This time, we will use the COG defuzzification method and will make use of the interpolation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

5. Comparison and Evaluation

In this section, we compare the fuzzy tuning technique with other tuning techniques present in AutoPas and evaluate its performance.

To measure the performance of the fuzzy tuning strategy, we also use the scenarios present in md_flexible and compare the results with the other tuning strategies present in AutoPas. The benchmarks are run on the CoolMUC-2[1] cluster and are repeated with 1, 12, 24, and 28 threads. We use the `timeSpentCalculatingForces` metric to evaluate the performance of the tuning strategies as it gives a good indication of the overall performance of the simulation.

5.0.1. Exploding Liquid Benchmark (Close to Training Data)

The exploding liquid benchmark simulates a high-density liquid that expands outwards as the simulation progresses. As the data of this scenario was included in the training data, we expect the fuzzy tuning technique to perform well. We only include the benchmark results with one thread for brevity, as the results for the other thread counts are very similar.

The plot in Figure 5.1a shows the time spent calculating the forces for each tuning strategy throughout the simulation. The fuzzy tuning strategies typically perform close to optimal and are very stable. All other tuning strategies show a much higher variance caused by testing many configurations during the tuning phases.

The low tuning overhead is the most significant contributor to the performance of the fuzzy tuning strategies. As the tuning phases of the fuzzy tuning strategies are very short and mainly consist of evaluating already known suitable configurations, there is no overhead caused by the tuning phases. This contrasts with the classical tuning strategies, which spend significant time in the tuning phases.

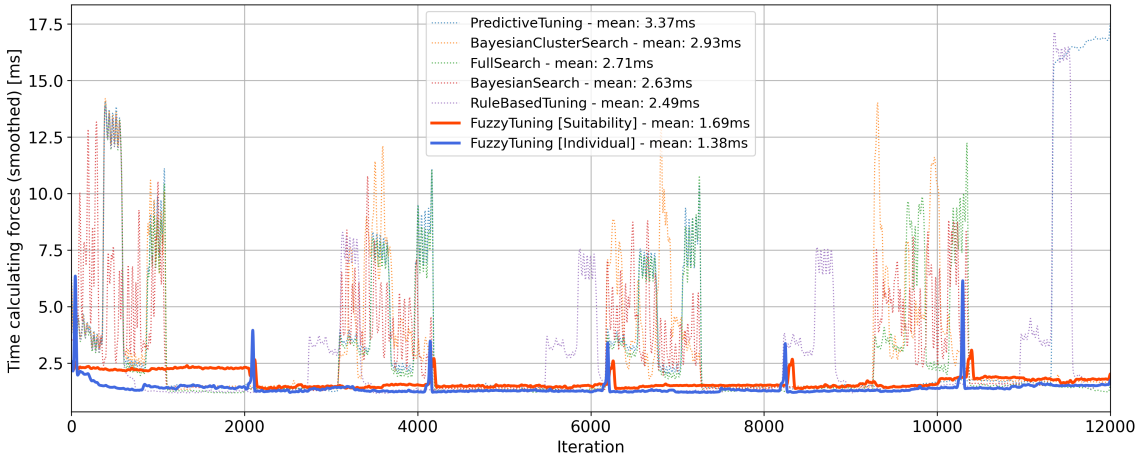
To show this in more detail, we also include a boxplot of the time spent calculating the forces for each tuning strategy based on the current phase in Figure 5.1b. All tuning strategies show similar timings during the simulation phases, as they eventually found a perfect configuration during the tuning phases but differ drastically in the tuning phases. The fuzzy tuning strategies have a much lower median time spent during tuning phases, with the individual tuning approach performing best. We see that the suitability approach performs worse than the other strategies during simulation phases because the suitability approach chose a suboptimal configuration for the first simulation phase, which was then corrected from the second tuning phase onwards. All other strategies eventually found a perfect configuration during the tuning phases, which caused them to perform better during the simulation phases.

[1] CoolMUC-2 is a supercomputer located at the Leibniz Supercomputing Centre in Garching, Germany. It consists of 812 Haswell-based nodes with 14 cores each. As a result of hyperthreading, each node supports up to 28 threads. More information can be found at `https://doku.lrz.de/coolmuc-2-11484376.html`

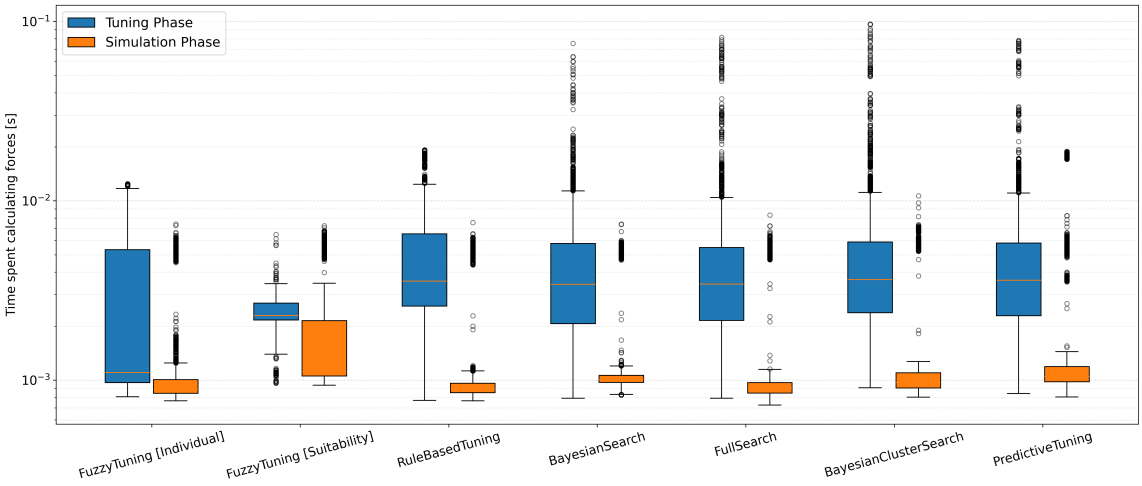
This plot also shows that the interquartile range of the classical tuning strategies is very similar, with all having nearly identical means. However, all of them are plagued by massive outliers, sometimes taking ten times longer than the median configuration and up to 100 times longer than the optimal configuration. Those extremely bad configurations are the main reason for the poor performance of the classical tuning strategies.

The last plot in Figure 5.1c shows the total time spent calculating the forces for each tuning strategy, again divided into simulation and tuning time. The fuzzy tuning strategies have the lowest total time, with practically no time spent in the tuning phases. Both fuzzy tuning approaches perform similarly and are by far the best-performing strategies. All other strategies typically spend more than 50% of their time in tuning phases where they potentially encounter very bad configurations, which causes them to perform much worse than the fuzzy tuning strategies.

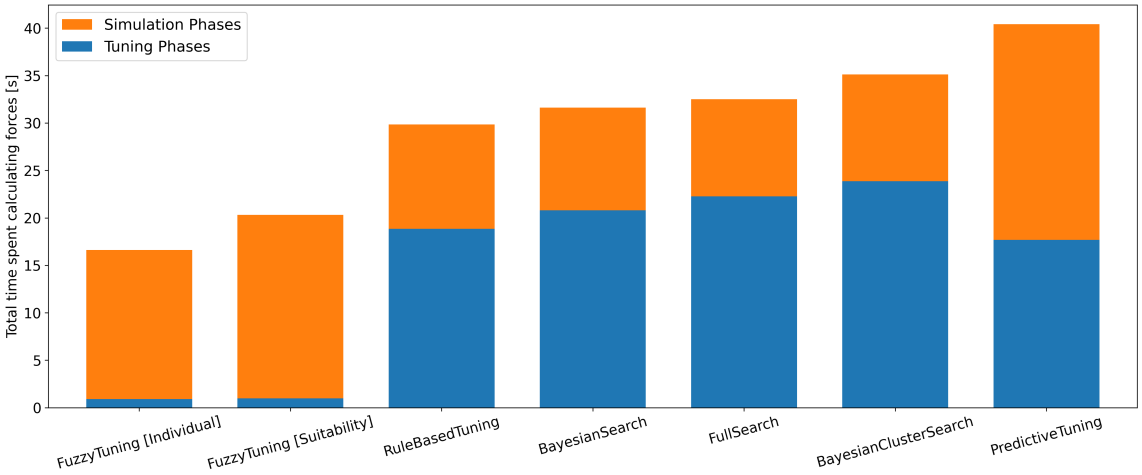
To summarize, the fuzzy tuning strategies perform very well in this scenario, as they can quickly select suitable configurations, which causes them to spend very little time in the tuning phases. All selected configurations are close to optimal, which causes the fuzzy tuning strategies to perform very well in the following simulation phases. On the other hand, classical tuning strategies spend significant time in the tuning phases, where they encounter many incorrect configurations, drastically slowing down the simulation.



(a)



(b)



(c)

Figure 5.1.: Exploding liquid benchmark with 1 thread. (a) Time spent calculating forces for each tuning strategy. (b) Boxplot of time spent calculating forces for each tuning strategy divided into simulation and tuning phases. (c) The time spent calculating forces for each tuning strategy is divided into simulation and tuning phases.

6. Future Work

We demonstrated that the expert knowledge extracted from the collected dataset can lead to a significant reduction of tuning time and can provide an overall improvement in the performance of the simulation. However, the expert knowledge is imperfect and can be improved in several ways. In this chapter, we discuss some of the possible improvements that can be made to the expert knowledge and the data collection process.

6.1. Better Data Collection

The data collection process can be improved in several ways. One of the main limitations of the current data collection process is that it is challenging and possibly impossible to collect a dataset representative of all possible scenarios. The current dataset is limited to a few scenarios, and the expert knowledge extracted from this dataset may not apply to other scenarios.

We showed that using the fuzzy-tuning strategy leads to a near-optimal prediction of configurations when the current scenario is represented in the dataset, as it is probably impossible to account for all possible scenarios. One could look into adaptively updating the expert knowledge as new scenarios are encountered. This could be done by spending extra time during the simulation to evaluate the performance data of recently executed configurations and update the expert knowledge accordingly.

6.2. Verification of Expert Knowledge

The currently used expert knowledge is directly extracted from the collected dataset without further validation. It would be interesting to investigate ways of validating the expert knowledge to rule out implausible rules and insert concepts not currently covered by the expert knowledge.

6.3. Future Work on Tuning Strategies

Another follow-up tuning strategy that could be investigated is using adaptive neuro-fuzzy inference systems (ANFIS). Those systems combine neural networks and fuzzy logic and the learning capabilities of neural networks with the uncertainty-handling capabilities of fuzzy logic. ANFIS systems could be used to predict the suitability values of the configurations with way more flexibility than currently allowed by expert knowledge.

7. Conclusion

7.1. A

A. Appendix

A.1. Glossary

AutoPas Node-level auto-tuned particle simulation library written in C++. See `https://github.com/AutoPas/AutoPas`. 31, 44–46

COG Center of Gravity Defuzzification Method. 30, 37

FIS Fuzzy Inference System. 29–31

md_flexible A flexible molecular dynamics simulation framework built on top of AutoPas. 6, 25, 31, 39

MOM Mean of Maximum Defuzzification Method. 30, 37

SOM Start of Maximum Defuzzification Method. 22

A.2. LiveInfoLogger Data Fields

The following fields are currently available in the LiveInfoData file. The LiveInfoData file is a CSV file containing summary statistics 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 all particles in the simulation domain.
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 number of bytes used to store a single particle in memory.
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 provides 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 is 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.

A.3. TuningData Fields

The following fields are currently available in the TuningData file. The TuningData file is a CSV file containing the performance information for all tested configurations. 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.

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 Newton's third law optimization is used to reduce computation by only calculating forces once per particle pair (enabled/disabled).

Reduced The reduced performance data for all sample points is 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.

A.4. Scenarios used for Data Generation

The scenarios used for data generation stem directly from the examples provided in the `AutoPas` library. Their state at the time of the data generation can be looked up at `https://github.com/AutoPas/AutoPas/tree/563528b4ef55d29b8ec148f6387f0be1adb400ed/examples/md-flexible/input`

The specific scenarios used are: `explodingLiquid`, `spinodalDecompositionEquilibration`, `spinodalDecomposition`, and `fallingDrop`.

A.5. Data Analysis

We performed some exploratory data analysis to gain insights into the collected dataset. The results are presented in the following figures.

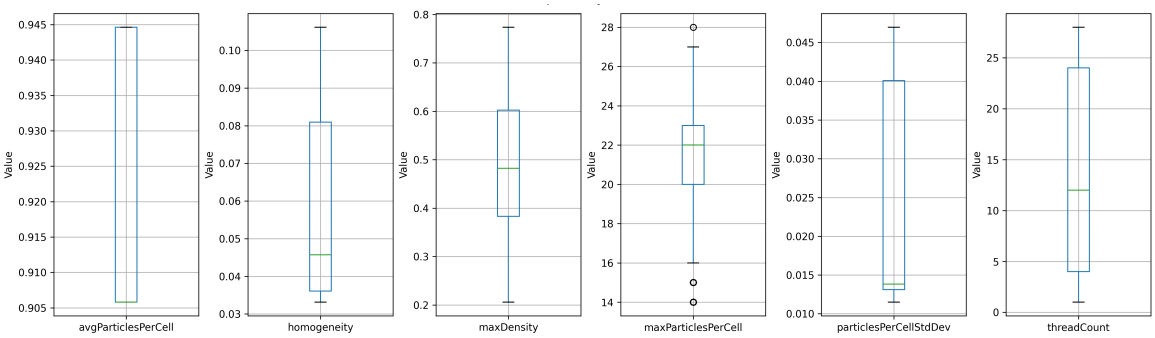


Figure A.1.: Boxplot showing the distribution of the collected data of the LiveInfoData files

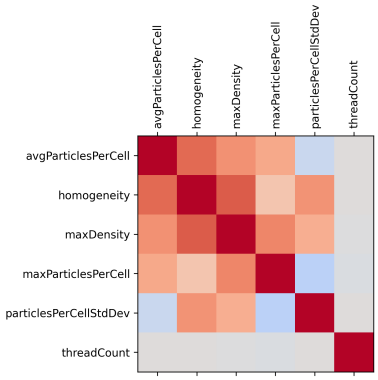


Figure A.2.: Correlation matrix showing the correlation between the collected parameters of the LiveInfoData files. We can see that many of the collected parameters are slightly positively correlated with each other.

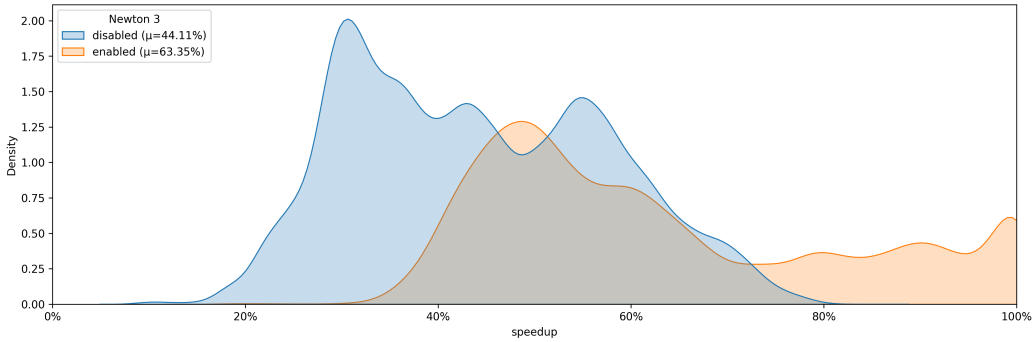


Figure A.3.: Density plot showing the distribution of the relative speedup based on the Newton 3 option. We can see that Newton3=enabled is generally the better option as it generally allows for a higher relative speedup. All performances with relative speedups of at least 80% use the Newton3 optimization. Therefore, we can confirm that Newton 3 is generally a good option.

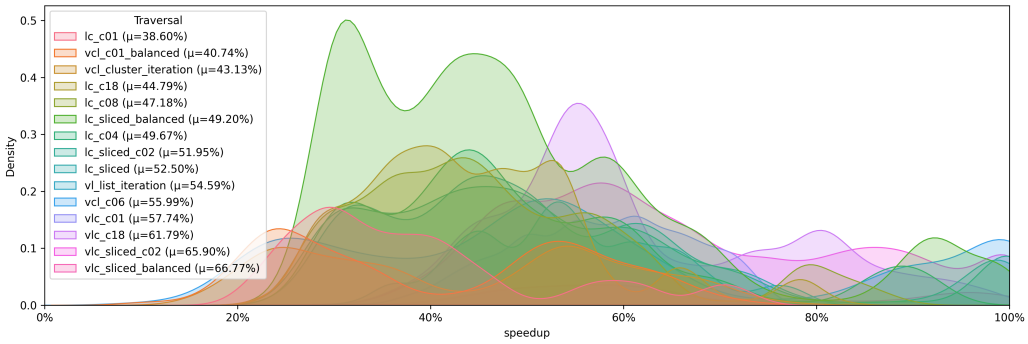


Figure A.4.: Density plot showing the distribution of the relative speedup based on the Traversal option. We can see that the vlc_sliced_balanced option generally performed better than the other options on this dataset with an expected relative speedup of 66%.

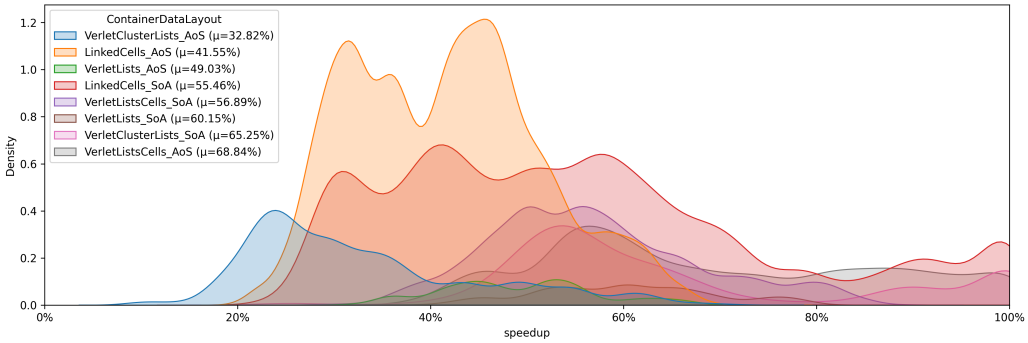


Figure A.5.: Density plot showing the distribution of the relative speedup based on the ContainerDatalayout option. The VerletListCells_AoS ContainerDatalayout performed best on this dataset with an expected relative speedup of 68.8%.

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.

[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.

[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.

[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.