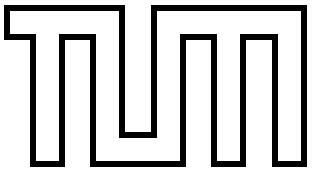


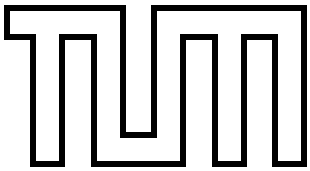
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

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 an important tool in many scientific fields, including chemistry, physics, biology, and materials science as they allow getting insights into the behavior of complex systems which may be difficult or impossible to study experimentally. Using the power of computer simulations, where properties of the model can be changed by just adjusting some formulas and thermodynamic parameters allows reasearches to study a vast variety of systems and conditions wich would be completly infeasible to reproduce in a laboratory setting.

add images

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 that evolve over time. The physisicst Erwin Schrödinger first formulated a mathematical model describing this phenomenon back in 1926 which has gotten wide spread acceptance and is now known widely as the Schrödinger equation. The Schrödinger equation is a partial differential equation that describes how the wave function of a physical system evolves over time. Its typically written as:

cite

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

where Ψ is the wave function of the system, $\hat{H}$ is the Hamiltonian operator describing the energy of the system. t is the time, and $\hbar$ is the reduced Planck constant.

Using the Schrödinger Equation it is possible to describe the future behavior of systems of molecules. Benedict Leimkuhler et al. [LM15] gave an example describing the behavior of a single water molecule consisting of three nuclei (two hydrogen atoms and one oxygen atom) and 10 electrons (2 from the hydrogen atoms and 8 from the oxygen atom) in the introductary chapter of their book. Since all of those 13 objects are characterized by their position in 3D space, the wave function of the system is a function of 39 variables and can be written as:

$$\Psi = \Psi(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_{13}, y_{13}, z_{13}, t) \tag{2.2}$$

by plugging this wave function into the Schrödinger equation we get the following partial differential equation:

$$i\hbar\frac{\partial\Psi}{\partial t} = -\hbar^2\sum_{i=1}^{13}\frac{\nabla_i^2\Psi}{2m_i} + U_p\Psi \tag{2.3}$$

where m_i is the mass of the i-th object, ∇_i^2 is the Laplace operator describing the kinetic energy of the i-th object and U_p is the atomic potential energy fuinction describing the interaction forces between the objects.

As the Schrödinger equation is a partial differential equation, it is very costly to solve it numerically for systems with many particles and we are quickly limited by the curse of dimensionality. Using the Born-Oppenheimer approximatio, the problem can be simplified to a classical mechanics problem by exploiting the fact that out of the perspective of the fast moving electrons the nuclei appear to be stationary which drastically limits their kinetic energy impact on the system. Nuclei on the other hand are much slower and can be treated as classical particles. This allows us to integrate out the electronic degrees of freedom and derive a new energy function U that only depends on the positions of the nuclei. The new energy function U is typically derived from quantum mechanical calculations or empirical data and can be quite complex depending on the system being studied. Yet this model is still just a very crude approximation of the real system and can lacks many quantum mechanical effects and therfore chemical behaviors relying on them. However, it is still a very powerful tool to study the behavior of complex systems as it allows for simulating them in the first place.

The resulting classical equations of motion for the nuclei are given by:

ask kunal to check this

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = -\nabla_i U \tag{2.4}$$

where $\vec{r}_i$ is the position of the i-th object and m_i is its mass. The force acting on the i-th object is given by the gradient of the energy function U.

This change turned the Schrödinger equation (which is a partial differential equation) into a nonelinear system of ordinary differential equations which can surprisingly be easier delt with numerically. Even for systems with many particles.

In practice there exist many potential energy functions U each designed to solve a specific problem. Many Potentials rely on a mixture of 2-body, 3-body and 4-body interactions between the particles to describe the behavior of the system. The 2-body interactions describe the effect of Pauli repulsion, atomic bonds and coulomb interactions while higher order interactions model the potentially assymetric wave function [LM15]. Partical simulation often use the Lennard-Jones potential to describe the 2-body interactions between the particles. The Lennard-Jones potential is a simple and lightweight model that can describe the behavior of many systems quite well . It is given by:

cite tgus

$$U_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{2.5}$$

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.

Hovever the Lennard-Jones potential is not suitable for all systems and many other potentials exist to tackle those problems. In this work we will however only work with types of simulations that can be accurately described by the Lennard-Jones potential and therfore will not go into further detail about other potentials.

Since the simulation domain potentially consists of a huge 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 name N-body problem and it can be shown that there

cite this

are no general solutions for systems with more than 2 particles . We can however solve the equations of motion numerically using numerical integration methods. The most common method to solve the equations of motion numerically is the Verlet algorithm which is a symplectic integrator that is energy conserving and has good long-term stability properties

cite this

. 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 $\vec{a}_i(t) = \frac{F_i}{m_i} = \frac{-\nabla_i U}{m_i}$.

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. Additionally, the simulation parameters such as the time step Δt, the number of integration steps and the potential energy function U need to be set.
2. **Update Positions**
   In this step, the positions of the particles are updated using the Verlet algorithm. The new positions are calculated based on the current positions, velocities and accelerations of the particles.
3. **Calculate Forces**
   The forces acting on the particles are calculated based on the current positions of the particles. The forces are typically calculated using the gradient of the potential energy function U. The forces are then used to calculate the accelerations of the particles.
4. **Update Velocities and Accelerations**
   The velocities and accelerations of the particles are updated based on the forces acting on the particles. The new velocities are calculated based on the current velocities, accelerations and forces of the particles.
5. **Apply Outside Effects** 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 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 then returns to step 2 and repeats until the desired number of integration steps is reached.

check this

This simulation loop can be repeated many times to simulate behavior of the system over time. The evolution of the system can then be analyzed using various statistical methods to extract information about the system and deduce properties not directly observable.

Many different sofware packages exist to perform such types of simulations. Some widely used examples of such systems are LAAMPS[1] and GROMACS[2]. Both of them implement a efficient way to solve the underlying N-body problem and provide the user with a high-level interface to specify the simulation parameters. There exist many different approaches to efficiently solve the N-body problem, and there is no single best approach that works well for all systems as the optimal implementation heaviliy depends on the simulation state and the capabilities of the hardware used to perform the simulation. Both LAAMPS and GROMACS however use a single implementations and are therefore not capable of adapting to the current simulation state.

check this

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

2.2. AutoPas

AutoPas is an open-source library designed to achive 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 having to worry 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 trade-offs in terms of performance and memory usage. There is no single implementation that is optimal for all simulation scenarios , as the optimal implementation depends on the current simulation state. AutoPas is designed to be adaptive and is capable of periodically switching 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 just provides a high-level interface to allow for short-range N-body simulations, the user is responsible for specifying the acting forces between the particles and has full control over the simulation loop. Fortunately AutoPas also provides `md_flexible` which is an example implementation of a typical molecular dynamics simulation.

2.3. Autotuning in AutoPas

AutoPas internaly alternates between two phases of operation. The first phase is the *tuning phase* where AutoPas tries to find the best configuration of parameters with regard to a chosen performance metric. This optimal configuration is then used in the following *simulation phase* with the assumption that the optimal configuration found in the tuning phase still performs reasonabley well during the consequent simulation phase. Obviously as the simulation progresses, and the characteristics of the system change, the previously

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

chosen configuration can drift arbitrarily far from the actual optimal configuration. To counteract this, AutoPas periodically alternate between tuning and simulation phases to ensure that the used configuration is reasonably close to the optimal configuration. During the simulation phase, AutoPas just acts as a black-box solving the force calculations for the underlying N-body problem.

The power of AutoPas comes from its vast amount of tunable parameters and therefore enourmous search space. As mentioned previously, other software packages like LAAMPS and GROMACS are limited by their design to just one particular implementation and can therefore operate outside of the theoretical optimal performance regime. In the following 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 6 tunable parameters which can mostly[3] be combined freely with each other. 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:

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 any additional data structures but has a very poor complexity, making it completly unsuitable for larger simulations. *Generally shouldn't be used except for very small 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 of that particles further away than the cutoff radius are not considered for the force calculation. In practice this is not a big issue 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 quite high. This is especially obvious, as most of the remaining distance checks performed by LinkedCells still do not

[3]There are some exceptions as some choices of parameters are not compatible with each other.

contribute to the force calculation [GST+19]. This trend can be explained due to the uneven scaling 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:

$$\frac{\text{Interaction Volume}}{\text{Search Volume}} = \frac{V_{sphere}(r_c)}{V_{cube}(3r_c)} = \frac{\frac{4}{3}\pi r_c^3}{(3r_c)^3} = \frac{4}{81}\pi \approx 0.155 \tag{2.7}$$

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

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

c) **VerletLists**
VerletLists is another approach to create 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 chosing a 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 be beneficial for systems with high particle density and frequent interactions, as the neighbor list only needs to be updated every n iterations. 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:

$$\frac{\text{Interaction Volume}}{\text{Search Volume}} = \frac{V_{sphere}(r_c)}{V_{sphere}(r_c \cdot s)} = \frac{\frac{4}{3}\pi r_c^3}{\frac{4}{3}\pi (r_c \cdot s)^3} = \frac{1}{s^3} \tag{2.8}$$

This time the ratio can be adjusted by changing the skin factor s. Ideally, the skin factor should be chosen such that the ratio is close to 1. This however reduces the buffer zone around the cutoff radius which means 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, which can lead 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 such that there are very few unnecessary distance checks. However, the construction of 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.

[4] Homogeneous in this context means that the particles are distributed evenly across the domain. If many particles are concentrated in a small area, the behavior of LinkedCells can quickly resemble that of DirectSum.

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 together into a so called *cluster* and a single neighbor list is created for each cluster. This results in a reduced memory overhead as the neighbor list only needs to be stored once for each cluster. Whenever two clusters are close to each other, 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, the construction of the neighbor list is still quite memory intensive and can result in a high memory overhead.

Generally good 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 affects how the work is distributed among the ranks. In this thesis we will however not go into further detail about the Load Estimator Options as we primarily focus on the tuning aspect of AutoPas.

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 the way the forces are calculated. The different traversal options provide a efficient way to prevent race conditions when using multiple threads and allow for load balancing at the node level [SGH+21].

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

see https://mediatum.ub.tum.de/doc/1735578/1735578.pdf

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 don't share common 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 such that cells with the same color do not share common 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 just colored the same way. This means that 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 different colors such that 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 grater 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 makes use of locks to prevent data-races [GSBN21].

4. **Data Layout Options:**
The Data Layout Options are related to the way the particles are stored in memory. The two possible data layouts are:

a) **SoA**
The SoA (Structure of Arrays) data layout stores the properties of all the particles separate arrays. For example, the x- ,y- and z-coordinates of all particles 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 a big array consisting of 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 are related to the way the forces between the particles are calculated. The Newton 3 law states that for every action there is an equal and opposite reaction . This means that the force between two particles is the same, regardless of cite which particle is considered the source and which particle is considered the target. In Molecular Dynamics simulations, this rule can be exploited to reduce the number of 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 turing on Newton 3 requires additional bookkeeping especially in multi-threaded environments. This results in more complicated traversal algorithms and can therefore also 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 can 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 actually within the cutoff radius. By choosing smaller cell sizes, this ratio can be increased, reducing the number of unnecessary distance checks. However, the performance gain is quickly offset by the increased overhead of managing more cells.

find reference

Tuning Strategies

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

The default tuning strategy in AutoPas is to use 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, but 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 potentially grow exponentially with the number of parameters. This makes the full search approach infeasible, especially if more tunable parameters are added to AutoPas.

find refernece as to how bad those are

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 hereby reducing the total number of configurations that need to be tested. It is therfore 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.

find refenrence to how bad this is

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

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

performance metric. It is guaranteed to find the best parameters for the current simulation state, but as mentioned before, 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 to find the best parameters for the current simulation state and could result in suggesting a bad 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. The it prunes the search space only keeping configurations that are predicted to perform reasonably well. The extrapolations is accomplished using methods such as linear regression or constructing polynomial functions through the data points. The method generally works well, it is however very sensitive to timing fluctuations.
4. **Bayesian Search**
   There exist two implementations of bayesian tuning in AutoPas. Those methods apply bayesian optimization techniques to predict good 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 configuration order of the form:

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

   To state that the dataLayout "AoS" is generally better than "SoA" if the number of particles is below a certain threshold. The rule based method can be generally very effective as expert knowledge can be encoded into the rules, which is unfortunately also its biggest drawback as it is not trivial to create a good set of rules.

The topic of this thesis is 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 also uses expert knowledge in the form 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 chose the best possible configuration, even if there is no direct rule for this specific case.

Such a 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 cite

we will introduce the basic concepts of fuzzy logic and how it is used to create a new tuning strategy for AutoPas.

2.4. 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 especially useful when modeling human language as the words tend to be unprecise. For example, the word "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 the modeling of such gradual transitions by assigning a degree of truth to each statement.

make image

2.4.1. Fuzzy Sets

Mathematically the concept of Fuzzy Logic is based on Fuzzy Sets. A Fuzzy Set is a generalization of a classical set where a element can lie somewhere between being a member of the set and not being a member of the set. 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 members of the set, 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 1 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

$$\mu_{\tilde{A}} : X \to [0, 1] \tag{2.9}$$

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

2.4.2. 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, the way these operations are defined is different 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 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 be thought of as a generalization of the logical AND operator to fuzzy sets, while $\bot$ and $\neg$ can generalize the logical OR and NOT operators respectively. The binary operators $\top$ and $\bot$ are often written in infix notation as $a \top b$ and $a \bot b$ similar to the way 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:

$$\neg\neg x = x \quad \text{//Involution} \tag{2.10}$$

$$\neg y < \neg x \quad \text{if } x < y \qquad \text{//Strong Monotonicity} \tag{2.11}$$

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 with respect to 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 maximum t-conorm as they are the most common choices in practice in fuzzy

logic systems. However the exact choices can be easily exchanged for other t-norms and t-conorms if needed.

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

  $$\mu_{\tilde{C}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) \tag{2.12}$$

  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

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

  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

  $$\mu_{\tilde{C}}(x) = 1 - \mu_{\tilde{A}}(x) \tag{2.14}$$

  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

2.4.3. Linguistic Variables

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

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 values of the temperature and allows to reason about the temperature in a more human-like way.

add grafical representation of the linguistic variable

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

2.4.4. 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 needs to be satisfied for the rule to be applied, while the consequent is the action that is taken if the rule is applied. Since we are not dealing with binary truth values, the antecedent can be only partially satisfied and as consequence the rule is only partially applied.

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

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

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:

$$\text{IF } (\tilde{A} = a \text{ AND } \tilde{B} = b) \text{ THEN } \tilde{C} = c \tag{2.15}$$

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. But 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

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

and could be deduced from the choice of the t-norm and t-conorm operators and would lead to $(a \implies b) = \neg a \vee b = \max(1 - a, b)$. In practice however the Mamdani implication is typcally used, which is just defined as the AND operation $\min(a, b)$. In a logiccal point of view this choice is counter intuitive as it clearly violates the standart defintion, but in the context of fuzzy systems its a very good choice for computing the degree of validity for a rule. [BMK96]. In practice wer are not interested in the resulting fuzzy set of the implication, but rather to which extend the consequent should be adapted therefore we introduce a slightly different infernence 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 extend the consequent should be adapted. We see that in the extreme cases where $\mu = 0$ the set R is also empty and has therefore no effect on further computations. If $\mu = 1$ the rule activated fully and the set R is equal to $\tilde{C}$. In all other cases the set R is trimmed down to the extend of the activation.

2.4.5. 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 real world systems. The first step in the defuzzification process is to collect all the rules operating on the same output variable and combine their trimed consequents into a single fuzzy set. This is done by just taking the fuzzy union of all those consequence which results in a new fuzzy set that represents the combined effect of all the rules on the output variable. Using a defuzzification method, this fuzzy set can then be converted back into a single crisp value that represents some aspect of the fuzzy set.

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 very intuitive as it tries to find a weighted interpolation of all the activated fuzzy sets. It is defined as:

$$\text{Centroid} = \frac{\int_X x \cdot \mu_{\tilde{C}}(x)\, dx}{\int_X \mu_{\tilde{C}}(x)\, dx} \tag{2.16}$$

- **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. In the case where there is just one maximum value, this meathod can be thought of as just returning the x-poisition of the most likely value.

  It is defined as follows:

$$\text{Mean of Maximum} = \frac{\int_{X'} x\, dx}{\int_{X'} dx} \tag{2.17}$$

  where $X' = \{x \in X \,|\, \mu_{\tilde{C}}(x) = \max(\mu_{\tilde{C}}(x))\}$ 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 the 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:

$$\text{Weighted Average} = \frac{\sum_{x \in X'} x \cdot \mu_{\tilde{C}}(x)}{\sum_{x \in X'} \mu_{\tilde{C}}(x)} \tag{2.18}$$

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

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

2.4.6. 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 the 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 antecedents of the rules to calculate the degree of activation of each rule. 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 a bunch of fuzzy sets that represent all the active effects on the output variable.
3. **Aggregation**
   The fuzzy sets resulting from the inference step are then combined into a single fuzzy set that represents conatins 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 real world systems. This is done by applying a defuzzification method to the fuzzy set that represents 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 a lot of expert knowledge as the rules and the membership functions of the fuzzy sets need to be defined by hand. This can be quite time consuming and in some cases even impossible if the system is too complex. Luckiliy there exist other methods which attempt to automate the process of defining the parameters of the fuzzy logic system. Some common methods are:

- **Genetic Algorithms**
  Genetic Algorithms are a class of optimization algorithms that are inspired by the process of natural selection. They work by maintaining a population of candidate solutions to a problem and iteratively improving the solutions by applying genetic operators such as mutation and crossover. Genetic Algorithms can be used to optimize the parameters of a fuzzy logic system by treating the parameters as the genes of an individual and the performance of the system as the fitness of the individual. 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 three main parts: the generic Fuzzy Tuning framework, the Tuning Strategy, and the Rule Parser. The Fuzzy Tuning framework is the core of this implementation and implements the mathematical foundation of this technique. The Tuning Strategy is the interface between the Fuzzy Tuning framework and the AutoPas simulation. It is responsible for interacting with AutoPas and updating the queue of configurations. 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. The implementation of the Fuzzy Tuning technique in AutoPas is designed to be as generic as possible to allow for easy integration of new types of rule bases.

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` implement 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. Lets 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 `BaseMembershipFunction`s. 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 add chcpt 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.

This way of combining fuzzy sets builds a tree structure where the leafs calculate a direct membership value and the inner nodes combine the membership values of their children and pass them up to their parent. Figure 3.1 shows an example of how a complex fuzzy set can be constructed from simpler fuzzy sets using this method.

- **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 and thus the effect of the rule 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.

Add UML Diagram

3.2. 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 makes use of 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 more lightweight and also allows for the connection of the Fuzzy Tuning framework to the AutoPas simulation. 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.

Appart from the obvious grammar rules for the fuzzy sets and rules, the language encompasses a few additional constructs to allow for the connection to the AutoPas simulation. This is a crucial part of the tuning framework as the continuous outputs of the Fuzzy Control Systems need to be mapped to the discrete configuration space of the AutoPas simulation. We chose a approach inspired by Mohammed et al. [MKEC22]'s work on scheduling algorithms and used a ranking system that embeds all available configurations in the continuous output space of the Fuzzy Control System. The Tuning Strategy then selects the configuration closest to the predicted value.

Add example image

An example of a rule file making use of this construct is described in Listing 3.2.

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

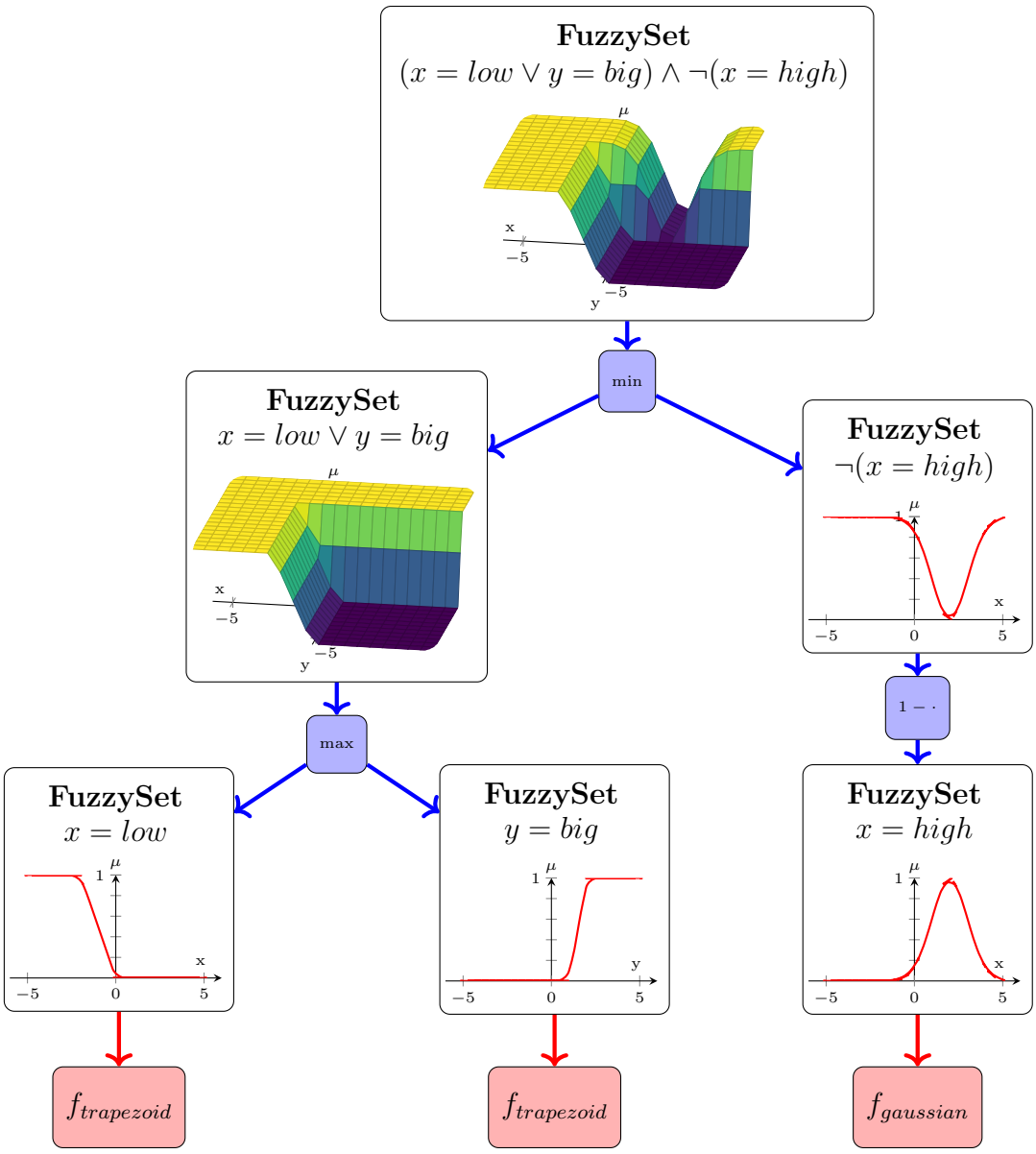


Figure 3.1.: Using the linguistic variables x with the terms $\{low, high\}$ and y with the terms $\{big, small\}$ we can construct the fuzzy set $(x = low \vee y = big) \wedge \neg(x = high)$ by combining the fuzzy sets $x = low \vee y = big$ and $\neg(x = high)$. Those fuzzy sets are again constructed 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.

The full grammar specification of the language can be found in Listing A.1.

3.3. 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 queue of configurations.

It does this by evaluating all Fuzzy Systems present in the rule file with the current LiveInformation of the simulation. The LiveInformation is a snapshot of the current state of the simulation and contains summary statistics about various aspects of the simulation. This evaulation yields a list of all configurations selected by the Fuzzy Control Systems and those replace the current queue of configurations which need to be tested.

Currently two different modes of interpreting the rules are implementet:

3.3.1. Configuration Suitability Approach

The Suitability Approach is designed to tackle rule bases that try to directly predict a suitability value for *each* configurations. This means that 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 (lets say 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 make full use of the continuous output space of the Fuzzy Control Systems and the method provides a straight forward mapping to the discrete AutoPas configuration space.

The huge downside of this approach is the need to define massive amount of Fuzzy Control Systems and rules in the rule file. This leads to a very big rule file which quickly becomes infeasible to maintain by hand. The code snippet in Listing 3.1 shows a small excerpt of a rule file using the Suitability Approach.

```
FuzzySystemSettings:
    defuzzificationMethod: "CoG"
    interpretOutputAs: "Suitability"

// Input Variables
FuzzyVariable: domain: "threadCount" range: (-20.34, 49.34)
    "higher than 18.0":   SigmoidFinite(39.38, 18.0, -3.23)
    ...

FuzzyVariable: domain: "particlesPerCellStdDev" range: (-0.017, 0.07)
    "lower than 0.029":   SigmoidFinite(0.055, 0.029, 0.004)
    ...

//Define Suitability Variable for a specific configuration
FuzzyVariable: domain: "Suitability LinkedCells_AoS_lc_c01_disabled" range: (0, 1)
    "terrible":           Gaussian(0.125, 0.018)
    "rubbish":            Gaussian(0.25,  0.018)
    "bad":                Gaussian(0.375, 0.018)
    "medium":             Gaussian(0.5,   0.018)
    "ok":                 Gaussian(0.625, 0.018)
    "good":               Gaussian(0.75,  0.018)
    "excellent":          Gaussian(0.875, 0.018)

...

// A rule describing the suitability of a specific configuration under certain conditions
if ("threadCount" == "higher than 18.0") && ("particlesPerCellStdDev" == "lower than 0.029")
  then ("Suitability_LinkedCells_AoS_lc_c01_disabled" == "bad")
```

```
...
```

Listing 3.1: Rule snippet depicting the Suitability Approach

3.3.2. Parameter 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 lies in the fact that the tunable parameters are nominal values and thus have no natural order. This means that the interpolation between different linguistic terms, which is naturally performed by the Fuzzy Control Systems, is not meaningful in this context. To avoid this problem, we can however use a defuzzification method that does not rely on performing some kind of interpolation. One such method is the SOM method which selects the smallest x-value for which the membership function is maximal. By using this method there is only a discrete set of possible outputs which can be directly mapped to the nominal values of the tunable parameters by making use of 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

The code snippet in Listing 3.2 shows a small excerpt of a rule file using the Parameter Tuning Approach. In this rule file both the *container* and the *dataLayout* parameter are predicted together as they are very related anyway.

maybe rename individual systems

```
FuzzySystemSettings:
    defuzzificationMethod: "SoM"
    interpretOutputAs: "IndividualSystems"

// Input Variables
FuzzyVariable: domain: "threadCount" range: (-20.34, 49.34)
    "lower than 18.0":    SigmoidFinite(39.34, 18.0, -3.34)
    ...

FuzzyVariable: domain: "homogeneity" range: (-0.024, 0.156)
    "lower than 0.049":   SigmoidFinite(0.093, 0.049, 0.005)
    ...

//Define a variable with nominal values
FuzzyVariable: domain: "Container_DataLayout" range: (0, 4)
  "LinkedCells_SoA, VerletClusterLists_SoA, VerletListsCells_AoS":                    Gaussian(0.666, 0.133)
  "LinkedCells_SoA, VerletClusterLists_SoA, VerletListsCells_AoS, VerletListsCells_SoA": Gaussian(1.333, 0.133)

...

// Define how the defuzzified output should be mapped to AutoPas configurations
OutputMapping:
  "Container_DataLayout":
    0.666 =>  [container="LinkedCells", dataLayout="SoA"],
              [container="VerletClusterLists", dataLayout="SoA"],
              [container="VerletListsCells", dataLayout="AoS"]
    1.333 =>  [container="LinkedCells", dataLayout="SoA"],
              [container="VerletClusterLists", dataLayout="SoA"],

    ...

// A rule describing the optimal configuration under certain conditions
if ("homogeneity" == "lower than 0.049") && ("threadCount" == "lower than 18.0")
  then ("Container_DataLayout" == "VerletClusterLists_SoA, VerletListsCells_AoS")

...
```

Listing 3.2: Rule snippet depicting the Parameter Tuning Approach

4. Proof of Concept

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

4.1. Creating the Knowledge Base

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

Add references to the methods

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

4.2. Decision Trees

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

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

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

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

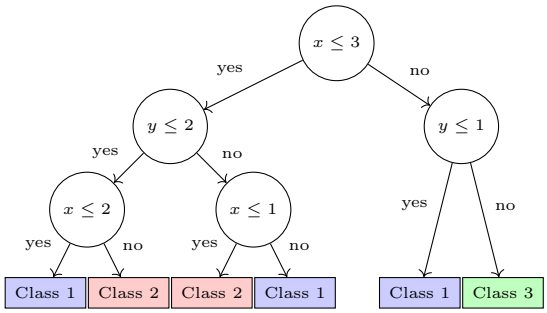


Figure 4.1.: An example decision tree for a dataset with two features x and y. There are three distinct classes in the dataset

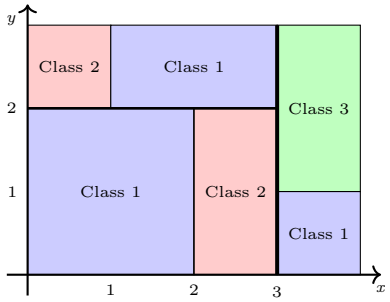


Figure 4.2.: The decision surface of the decision tree from Figure 4.1 on $\mathcal{D} = [0, 4] \times [0, 3]$.

4.3. Fuzzy Decision Trees

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

4.4. Converting a Decision Tree into a Fuzzy Inference System

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

The conversion of a classical decision tree to a fuzzy decision tree is done by replacing the crisp decision boundaries (e.g., $x \leq 3$) at each internal node with fuzzy membership functions. Those membership functions should have the same semantics as the crisp decision boundaries, but instead of returning a binary value of wheter to continue down the left or right branch, they return a value in the range $[0, 1]$ that specifies to which degree each branch should be taken. The shape of the membership functions can be chosen arbitrarily, but since the decision should be one-sided, typical choices include complementary `sigmoid`-shaped functions. 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 symetric 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 typical chosen from the interval $n \in [0, 5]$ as otherwise the membership function can become too wide, 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 $x \leq 3$.

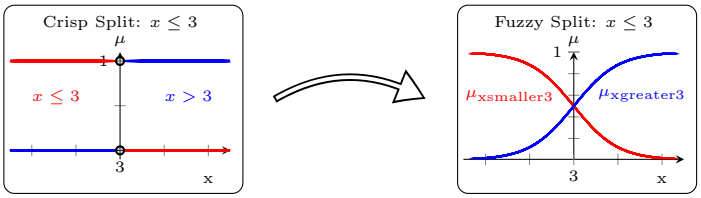


Figure 4.3.: Conversion of a crisp decision surface to a fuzzy decision surface. The crisp decision surface $x \leq 3$ is replaced by two `sigmoid` membership functions $\mu_{\text{xsmaller3}}$ and $\mu_{\text{xgreater3}}$ that specify to which degree the comparison is true or false. The *width* of the membership function is data dependent and is determinend 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. The placement of the different class-specific membership functions is very important, as it can heavily influence the defuzzification process for some defuzzification methods as fuzzy systems typically use a 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 specific shapes of the membership functions has been picked arbitrarily and can be adjusted to better fit the data. The resulting linguistic variables are shown in Figure 4.5.

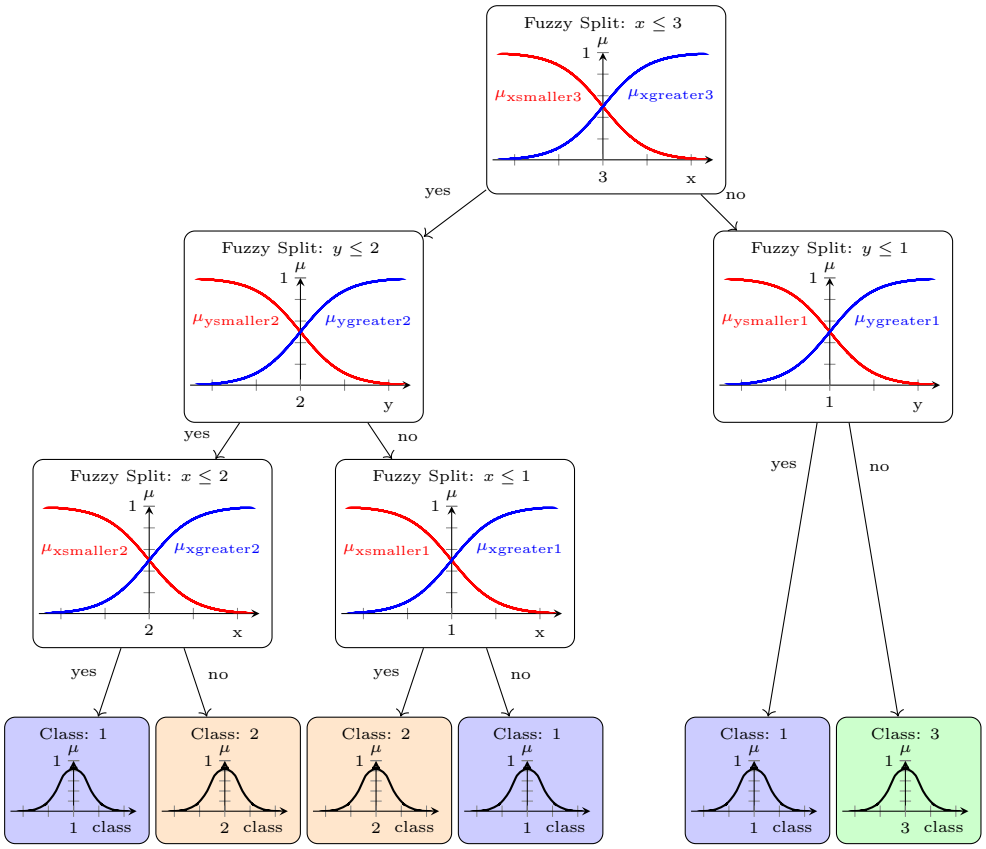


Figure 4.4.: The fuzzy decision tree corresponding to the decision tree in Figure 4.1. Each internal node in the fuzzy decision tree uses two `sigmoid` membership functions (μ_{smaller} and μ_{greater}) to specify to which degree the comparison is true or false. The leaf nodes use different `gaussian`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

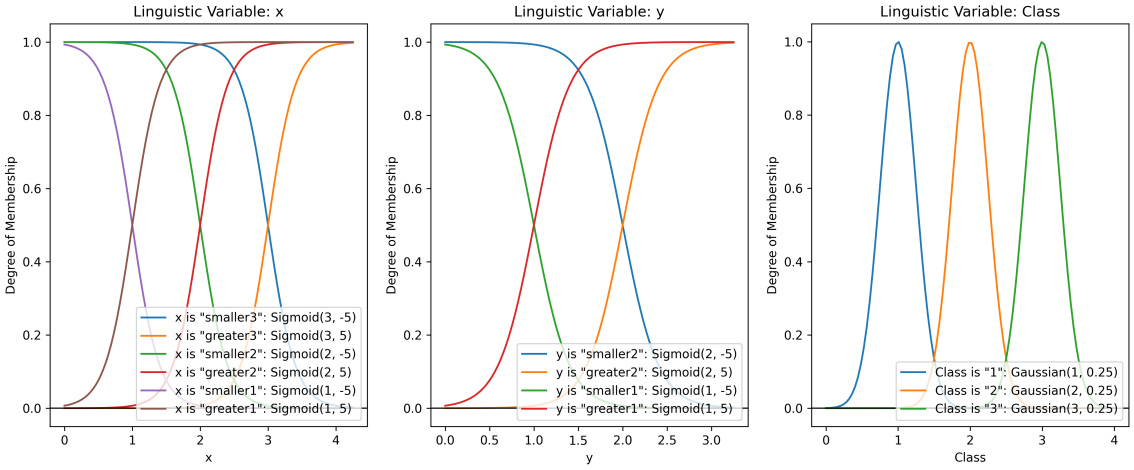


Figure 4.5.: Linguistic variables used in the fuzzy decision tree in Figure 4.4. The standard deviation of the attributes is assumed to be $\sigma \approx 0.5$ such that the *width* of the sigmoid membership functions is $n \cdot \sigma \approx 1$. The class values are assumed to be normally distributed and are places such that they dont 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, while 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 esentially mimics the decision surface seen in Figure 4.2, as we create exactly one rule for each region of the decision surface. The rules extracted from the fuzzy decision tree in Figure 4.4 are shown in Table 4.1.

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

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

Fuzzy Inference System

With the linguistic variables and fuzzy rules extracted from the decision tree, we can now use them to create a fuzzy system that can predict the class of a new data point based on its features. Since the fuzzy system can be seen as a black box mapping continuous input features to continuous output classes (see Figure 4.6), it is possible to visualize the decision surface of the fuzzy system by evaluating the membership functions of the rules for each point in the input space. This decision surface can then be used to understand the decision-making process of the fuzzy system and to identify possible errors in the rules or membership functions.

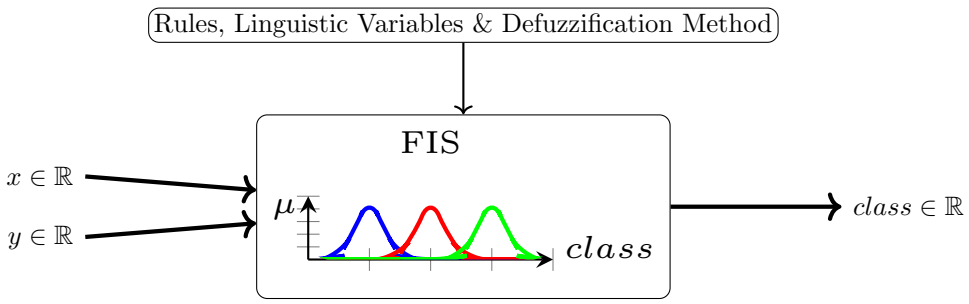


Figure 4.6.: The fuzzy inference system created from the fuzzy decision tree in Figure 4.4 can be seen as a black box that maps continuous input features to continuous output classes.

Choice of Defuzzification Method

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

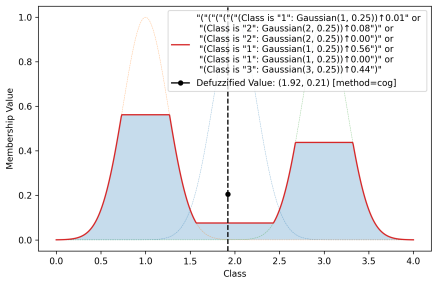


Figure 4.7.: Resulting fuzzy set after applying the rules from Table 4.1 on the data point $(x = 2.95, y = 2.5)$. There are clear peaks at the class values 1 and 3. The COG method however returns Class 2, as it lies right in between the two peaks, turing the two good predictions into a bad one.

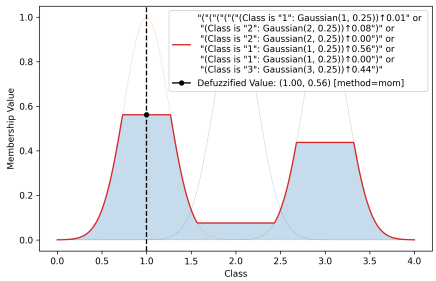


Figure 4.8.: The MOM method returns the class value 1, as it is the mean of the two peaks at class values 1 and 3. This is a much better prediction than the one made by the COG method.

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

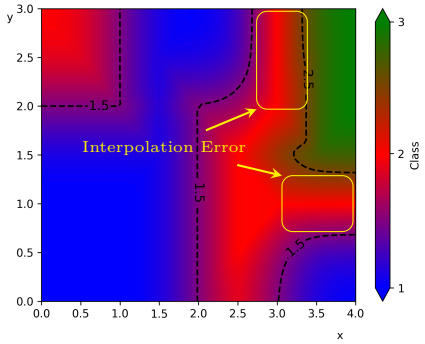


Figure 4.9.: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the COG defuzzification method. The highlighted area shows the interpolation error of the COG method

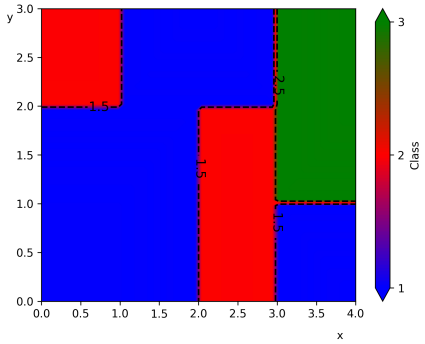


Figure 4.10.: The decision surface of the FIS created from the fuzzy decision tree in Figure 4.4 over $\mathcal{D} = [0, 4] \times [0, 3]$ using the MOM defuzzification method.

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

4.5. Creating a Fuzzy System for `md_flexible`

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

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

4.5.1. Data Collection

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

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

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

add specs

The data was collected on the CoolMUC-2 and primarly 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.5. All simulations were run on the Serial-Partition of the the cluster were repeated twice, to account for fluctuations in performance. Furthermore every simulation was run with 1, 4, 12, 24 and 28 threads to also gather data on how parallelization affects the ideal configuration.

4.5.2. Data Analysis

To verify the sanity of the collected data, we can make plots about the distribution of the data and the nominal values of the collected data. The boxplot in Figure 4.11 shows the distribution of the collected data, while the pie charts in Figure 4.13 shows the relative proportions of the collected parameters. We can see that the data is quite balanced and that the nominal values are spread out quite evenly, which is a good sign for the quality of the collected data.

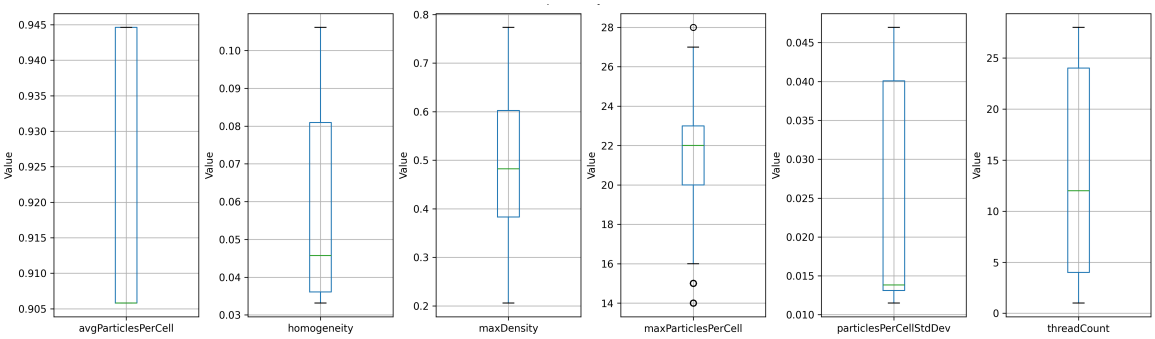


Figure 4.11.: The boxplot shows the distribution of the collected data. The boxplot shows the median, the first and third quartile, and the whiskers show the range of the data.

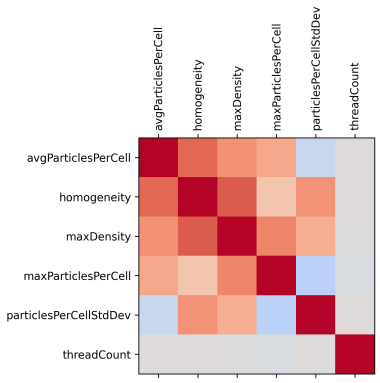


Figure 4.12.: The correlation matrix shows the correlation between the different parameters of the collected dataset. We can see many of the collected parameters are slighly positively correlated with each other. `particlesPerCellStdDev` is negatively correlated with `avgParticlesPerCell` and `maxParticlesPerCell` which is also expected, as a higher standard deviation of the particles per cell should lead to unbalanced cells.

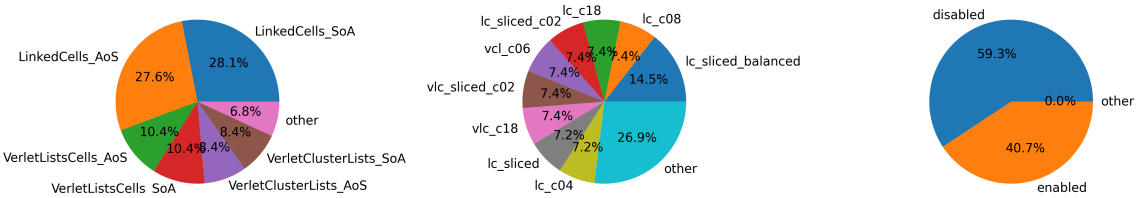


Figure 4.13.: This figures show the relative proportions of the collected parameters in the dataset. All parameters are spread out quite evenly, which means that the dataset is quite balanced. The data however stems from every iteration, not just the tuning phases.

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. For each tuning phase there is a unique ranking of all the configurations based on their runtime which we can use to calculate the relative speedup of each configuration compared to the best configuration. The formula for the relative speedup is given by:

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

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 4.14, Figure 4.15, Figure 4.16 and Figure 4.17 show the distribution of the relative speedup values for the Newton3, Traversal, Container-Datalayout and some complete configurations respectively. We can see that the Newton3 option generally leads to a higher relative speedup, while the Traversal option does not show a clear trend. The Datalayout option shows that the VerletListCells_AoS option is generally the best option, while the configuration VerletListCells_AoS_vlc_spliced_balanced_enabled is the best configuration in most cases on the Dataset we collected.

The ordering of the different parameters in the dataset matches the intuition we have about the different implementations and we can be confidently proceed with the creation of the fuzzy system.

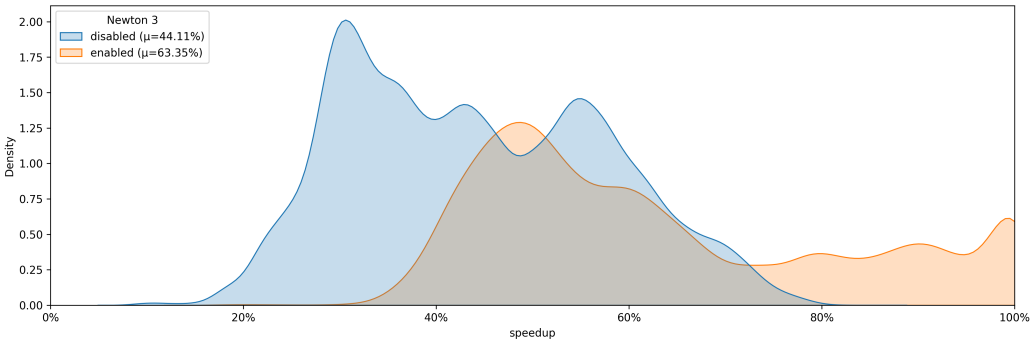


Figure 4.14.: Density plot showing the distribution of the Newton3 options with respect to their relative speedup compared to the best configuration during each tuning phase. We can clearly see the two peaks wheter Newton3 is enabled or disabled. It is also very obvious that Newton3=enabled is the better option as it generally allows for a higher relative speedup. It is interesting to note that all performances having relative speedups of at least 80% make use of the Newton3 optimization. Therefore we can confirm that Newton3 is generally a good option to enable.

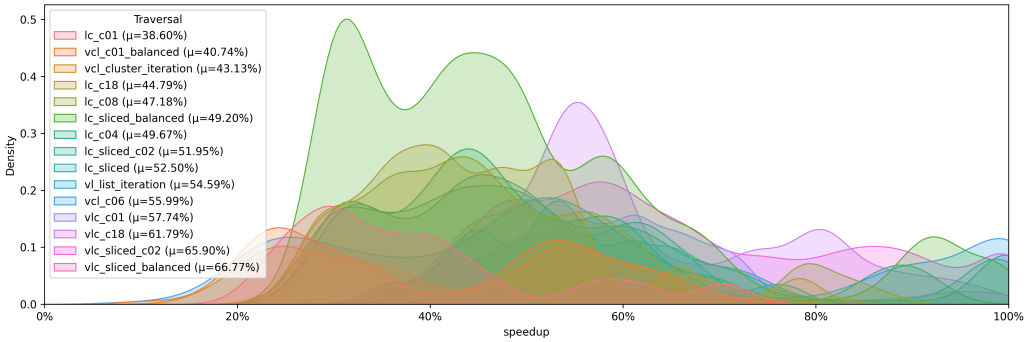


Figure 4.15.: The density plot shows the distribution of the Traversal option with respect to the relative speedup compared to the best configuration during each tuning phase. There are no clear peaks in the data, but we can see that the vlc_sliced_balanced option generally performed better than the other options with an expeced relative speedup of 66% compared to the best configuration.

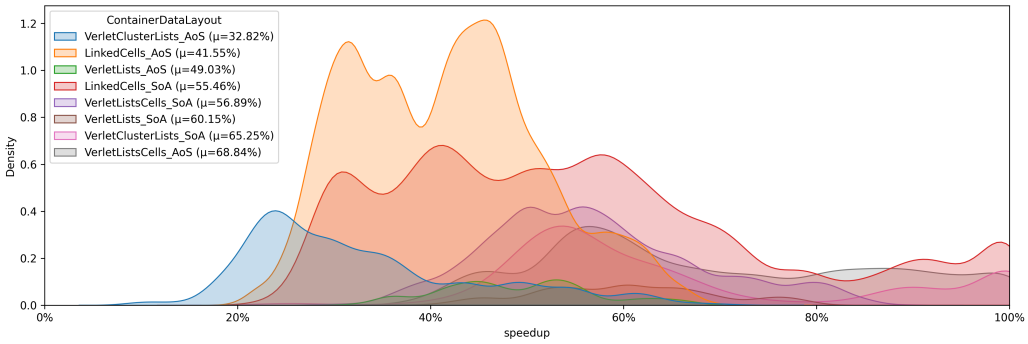


Figure 4.16.: Density plot showing the distribution of the container-datalayout option with respect to the relative speedup compared to the best configuration during each tuning phase. The VerletListCells_AoS container-data-layout is the best configuration in most cases with 68% of the runtime as the best configuration on average.

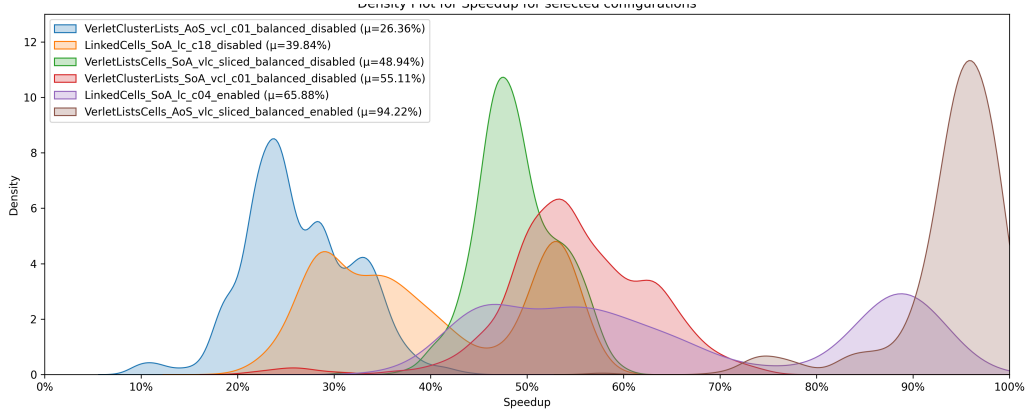


Figure 4.17.: The density plot shows the distribution of the collected configurations with respect to the relative speedup compared to the best configuration during each tuning phase. The VerletListCells_AoS_vlc_spliced_balanced_enabled configuration is the best configuration in most cases with 94% of the runtime as the best configuration on average.

4.5.4. Creating the Fuzzy Rules

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

1. Preprocess the data according to to approach used (see next section).
2. Train the decision trees and prune the tree to avoid overfitting.
3. Select the best perfoming 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 are going to create two different kind 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 paramters 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 good configurations, we first remove all configurations perfoming worse than a certain threshold. As depicted in Figure 4.18 we chose to only include configurations that have a relative speedup of at least 70% as this still leaves us with a large enough dataset to train the decision trees.

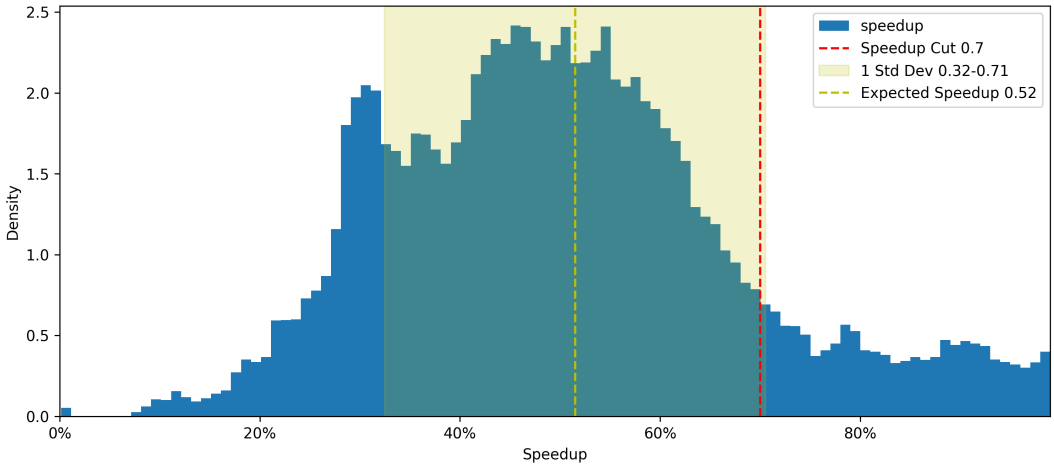


Figure 4.18.: The density plot shows the distribution of all the collected configurations with respect to the relative speedup compared to the best configuration during each tuning phase. This distribution shows that the average tested configuration performs just 50% as good as the best. With some configurations beeing 10 times slower than the best configuration. We see that it is of utmost importance to efficiently find good configurations as testing all possible configurations causes a huge performance loss.

Afterward we group together all configurations that have been evaluated in the same tuning phase and aggregate all the present values of tunable parameters into a single term. This term representing all *good* values for the paramters 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 liveinfo 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 4.19.

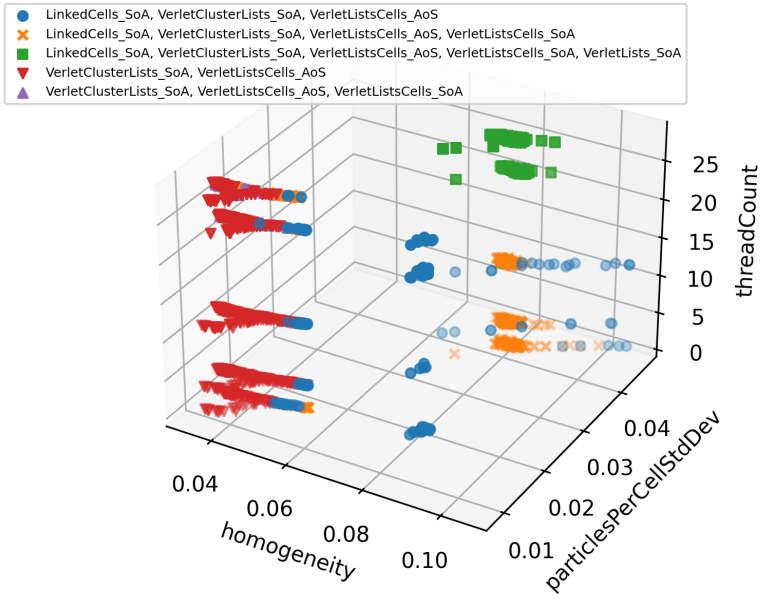


Figure 4.19.: The scatterplot shows the influence of the variables `homogeneity`, `particlesPerCellStdDev` and `threadCount` on the optimal `ContainerDataLayout` parameter. We can clearly see regions which are going to be learned by the decision tree.

As descibed previously we use `gaussian` membership functions for each linguistic term of the consequent linguistic variables. The placement of the values is irrelevant and we just place them in a way that they don't overlap. Figure 4.20 and Figure 4.21 show the resulting linguistic variables for a antecedent linguistic variable (homogeneity) and a consequent linguistic variable (Newton3) respectively. The visualization of the other variables follows a similar pattern, but are more complex due to the higher number of terms and are therefore not shown here.

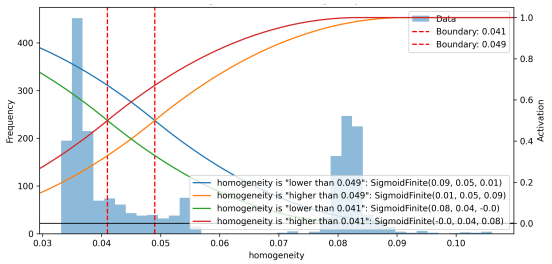


Figure 4.20.: This figure shows the linguistic variable for the homogeneity attribute. We see the different fuzzy sets created from the decision trees. The background shows the histogram of all homogeneity values present in the dataset.

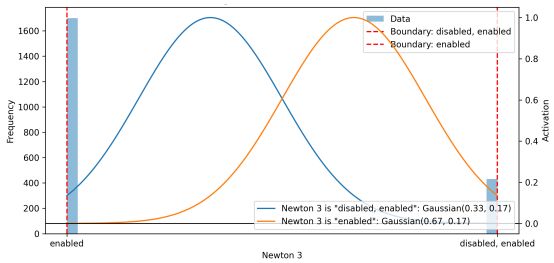


Figure 4.21.: This figure shows the linguistic variable for the Newton3 attribute. We see the two `gaussian` membership functions representing the two possible values for the Newton3 attribute.

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 the predicted values of the parameter. As we are using the MOM method, the output will always correspond to the fuzzy set with the highest membership value and will 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 the numerical *suitability*-value of a configuration under the current conditions. Therefore each possible configruation 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, but 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 classication based approach with `terrible`, `bad`, `average`, `good` and `excellent` as the possible linguistic terms (see Figure 4.22). The calculated suitability values of each configuration during a specific tuning phase are then just 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 4.22. The output mapping for the suitability approach is shown in **??**.

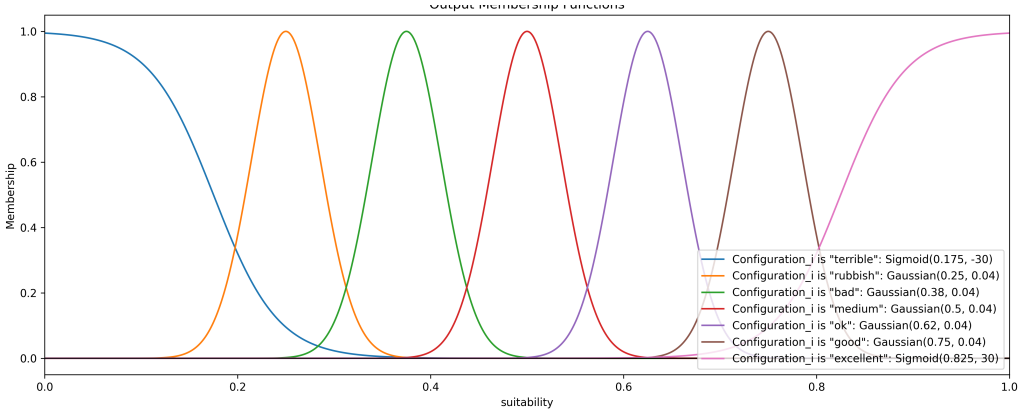


Figure 4.22.: Linguistic terms for the suitability variables. The fuzzy sets consist of `sigmoid` membership functions at the borders and `gaussian` membership functions in the middle. The values are placed in coherent way. This time we will use the COG defuzzification method and will make use of the 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 liveinfo 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 an evaluate its performance.

To measure the performance of the tuning strategy, we also use the scenarios present in md_flexible. The benchmarks are run on the CoolMuc-2 cluster at the Leibniz Supercomputing Centre in Garching .

add specs

5.0.1. Exploding Liquid Benchmark

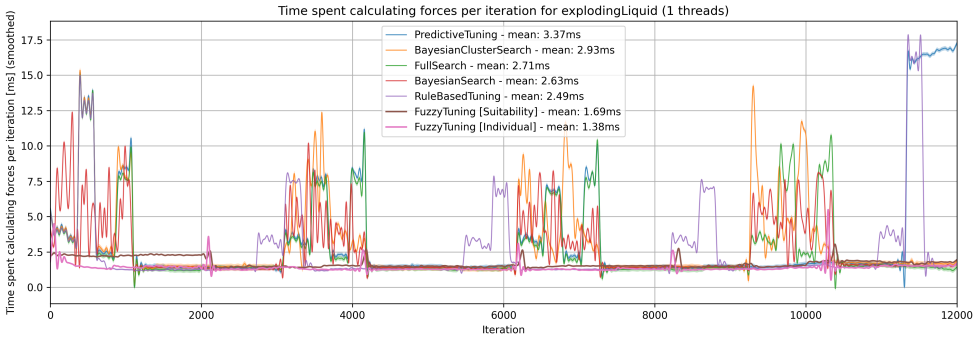
The exploding liquid benchmark simulates a high-density liquid that quickly expands outwards as the simulation progresses. As the data of this scenario was included in the training data, we expect the fuzzy tuning technique to perform well. The benchmark is repeated with 1, 12, 24 and 28 threads, but for brevity, we only include the results for 1 and 28 threads as the other results are very similar. Every scenario is run with all available tuning strategies in AutoPas.

We use the `timeSpentCalculatingForces` metric to evaluate the performance of the tuning strategies as it gives a good indication of the performance of the configuration. The resulting timings are shown in 5.1a and 5.2a for 1 and 28 threads, respectively.

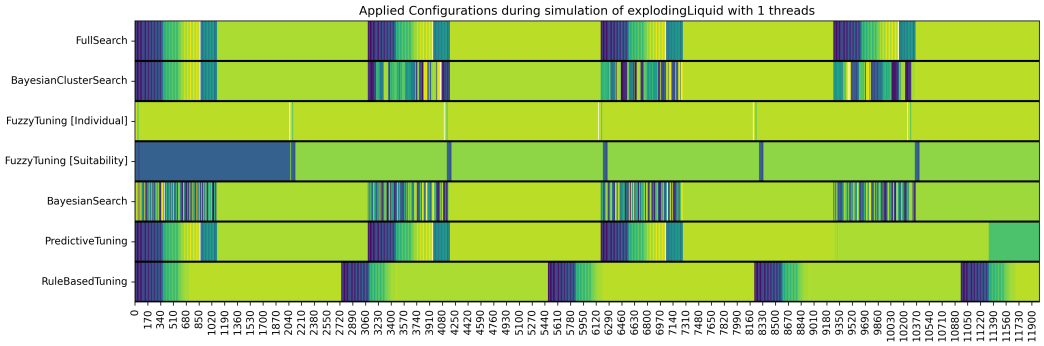
As timing data is prone to noise, we also include a heatmap of the selected configurations in 5.1b and 5.2b respectively. We can see that every tuning strategy selects a similar configuration most of the time. There are also some bigger deviations in both fuzzy tuning strategies, but we can see from the associated timings that these configurations are also very performant and do not significantly harm the total performance.

The last plot in 5.1c and 5.2c shows the total time spent calculating the forces for each tuning strategy. In every benchmark the fuzzy tuning strategies performed the best, with the individual tuning approach performed slightly better than the suitability approach. By looking at the timing data, we can see that the slight performance increase is due to the suitability approach taking longer to select a configuration as it has to evaluate way more fuzzy rules than the individual tuning approach.

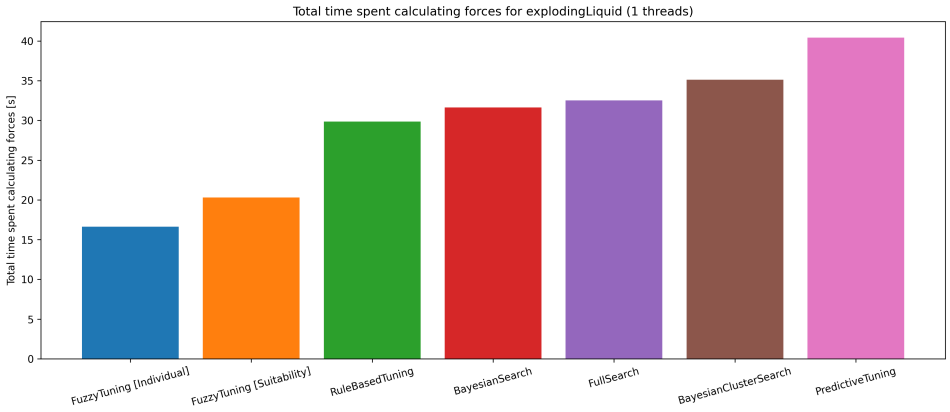
A huge factor in the performance of the fuzzy tuning strategies is their relatively low tuning overhead. As the fuzzy-tuning strategy only needs a single iteration to perform the *slow* fuzzy rule evaluation, it is very fast compared to the other tuning strategies. Especially as all of the predicted configurations mostly tend to be very performant. We can see from the heatmap that most other tuning strategies use much longer tuning phases, where there is a higher possibility of testing bad configurations. This is especially true for the *full* tuning strategy, which tests every possible configuration in the search space.



(a)

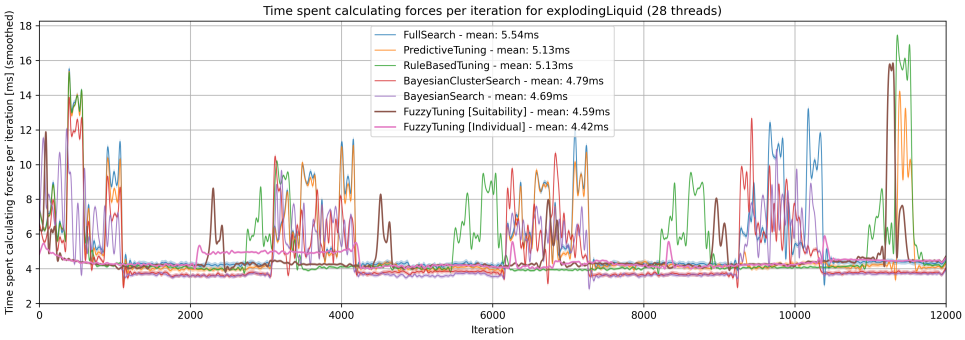


(b)

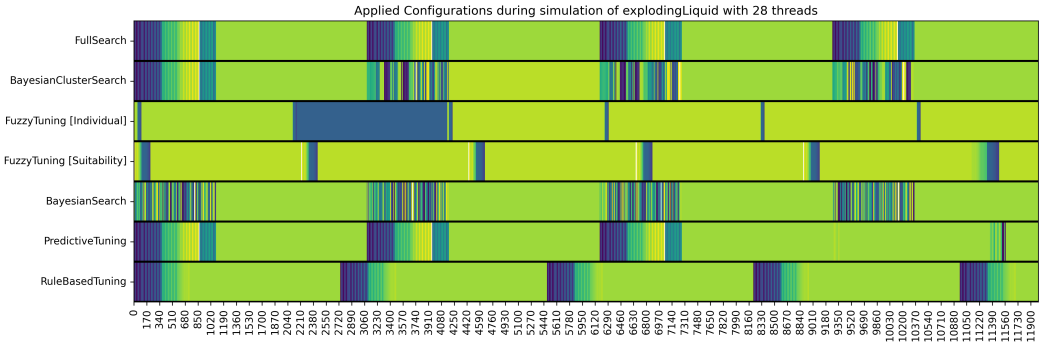


(c)

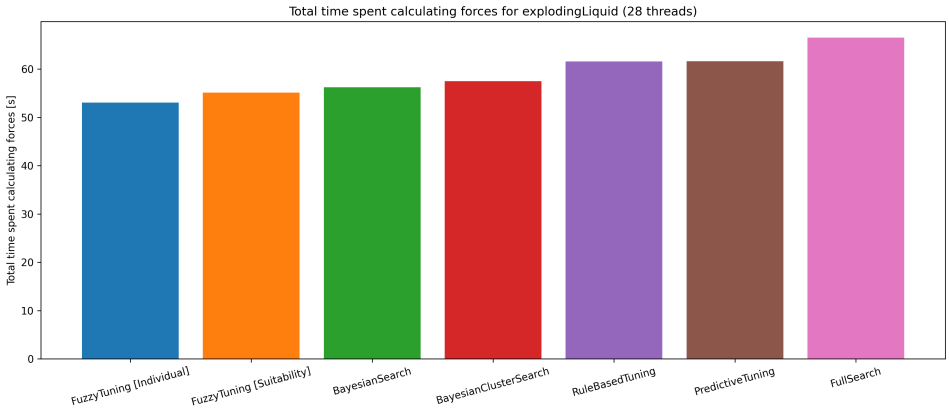
Figure 5.1.: (a) Total time spent calculating the forces for the exploding liquid benchmark with 1 thread. (b) Heatmap of the tested configurations during the benchmark. The meaning of the colors are displayed in A.1. (c) Total time spent calculating the forces for each tuning strategy.



(a)



(b)



(c)

Figure 5.2.: (a) Total time spent calculating the forces for the exploding liquid benchmark with 28 threads. (b) Heatmap of the tested configurations during the benchmark. The meaning of the colors are displayed in A.1. (c) Total time spent calculating the forces for each tuning strategy.
Note: Due to the high performance overhead of multithreading, using 28 threads results in a worse performance than using only 1 thread for this benchmark.

6. Future Work

6.1. A

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`. 30, 45–47

COG Center of Gravity Defuzzification Method. 29, 38

FIS Fuzzy Inference System. 28–30

md_flexible A flexible molecular dynamics simulation framework built on top of AutoPas. 5, 24, 30, 40

MOM Mean of Maximum Defuzzification Method. 29, 38

SOM Start of Maximum Defuzzification Method. 23

A.2. LiveInfoLogger Data Fields

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

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

Define the metric

Which unit?

A.3. TuninData Fields

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

Date — The date and time when the data was collected.

Iteration — The current iteration number of the simulation.

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

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

check https://med

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

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

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

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

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

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

Smoothed — A smoothed version of the reduced performance data.

A.4. ANTLR4 Rule Parser Grammar

The following is the grammar for the domain-specific language used by the Rule Parser to parse the rule base supplied by the user. The grammar is defined using the ANTLR4 parser generator.

Listing A.1: ANTLR4 Rule Parser Grammar

```
1 grammar FuzzyLanguage;
2
3 // Rule File
4 rule_file           : settings linguistic_variable*
5                       output_mapping fuzzy_rule* EOF
```

```
6                       ;
7
8   // Settings as key-value pairs
9   settings            : 'FuzzySystemSettings' ':'
10                      (IDENTIFIER ':' STRING)*
11                      ;
12
13  // Fuzzy Variable
14  linguistic_variable
15                      : 'FuzzyVariable' ':' 'domain' ':' STRING
16                        'range' ':' '(' NUMBER ',' NUMBER ')'
17                       fuzzy_term+
18                      ;
19
20  fuzzy_term
21                      : STRING ':' function
22                      ;
23
24  function
25                      : IDENTIFIER '(' NUMBER (',' NUMBER)* ')'
26                      ;
27
28  // Fuzzy Rule
29  fuzzy_rule
30                      : 'if' fuzzy_set 'then' fuzzy_set
31                      ;
32
33  fuzzy_set
34                      : '(' fuzzy_set ')'          # Brackets
35                      | fuzzy_set '&&' fuzzy_set   # And
36                      | fuzzy_set '||' fuzzy_set   # Or
37                      | '!' fuzzy_set              # Negate
38                      | STRING '==' STRING         # Select
39                      ;
40
41  // Output Mapping
42  output_mapping
43                      : 'OutputMapping' ':'
44                       output_entry+
45                      ;
46
47  output_entry
48                      : STRING ':' pattern_mapping+
49                      ;
50
51  pattern_mapping
52                      : NUMBER '=>'
53                        config_pattern (',' config_pattern)*
54                      ;
55
56  config_pattern
57                      : '[' (IDENTIFIER '=' STRING)
58                          (',' IDENTIFIER '=' STRING)* ']'
59                      ;
60
61  // Lexer Rules
62  WS
63                      : [ \t\n\r\f]+ -> skip
64                      ;
65
66  COMMENT             : '#' .*? '\r'? '\n' -> skip
67                      ;
68
69  STRING
70                      : '"' (~["\r\n] | '""')* '"'
71                      {setText(getText().substr(1, getText().size()-2));}
72                      ;
73
74  NUMBER
75                      : '-'? INT ('.' [0-9]+)? EXP?
76                      ;
77
78  fragment INT
79                      : '0'
80                      | [1-9] [0-9]*
81                      ;
82
83  fragment EXP
84                      : [Ee] [+-]? [0-9]+
85                      ;
86
87  IDENTIFIER
88                      : [a-zA-Z0-9_]+
89                      ;
```

A.5. Scenarios used for Data Generation

The following scenarios were used to generate the data to train the decision tree models used to generate the fuzzy rules. The scenarios are defined in the `.yaml` configuration file used by the `AutoPas` library.

Listing A.2: explodingLiquid.yaml

```
container :                          [LinkedCells, VarVerletListsAsBuild, VerletClusterLists,
    VerletLists, VerletListsCells, PairwiseVerletLists]
verlet-rebuild-frequency :           10
verlet-skin-radius-per-timestep :    0.02
fastParticlesThrow :                 false
verlet-cluster-size :                4
selector-strategy :                  Fastest-Mean-Value
data-layout :                        [AoS, SoA]
traversal :                          [lc_c01, lc_c01_combined_SoA, lc_c04, lc_c04_HCP,
    lc_c04_combined_SoA, lc_c08, lc_c18, lc_sliced, lc_sliced_balanced, lc_sliced_c02,
    vcl_c01_balanced, vcl_c06, vcl_sliced, vcl_sliced_balanced, vcl_sliced_c02, vl_list_iteration
    , vlc_c01, vlc_c18, vlc_sliced, vlc_sliced_balanced, vlc_sliced_c02, vlp_c01, vlp_c18,
    vlp_sliced, vlp_sliced_balanced, vlp_sliced_c02, vvl_as_built]
tuning-strategies :                  []
tuning-interval :                    1000
tuning-samples :                     10
functor :                            Lennard-Jones (12-6) AVX intrinsics
newton3 :                            [disabled, enabled]
cutoff :                             2
box-min :                            [0, 0, 0]
box-max :                            [15, 60, 15]
cell-size :                          [1]
deltaT :                             0.00182367
sorting-threshold :                  8
iterations :                         12000
boundary-type :                      [periodic, periodic, periodic]
Sites:
  0:
    epsilon :                        1
    sigma :                          1
    mass :                           1
Objects:
  CubeClosestPacked:
    0:
      particle-spacing :             1
      box-length :                   [14, 6, 14]
      bottomLeftCorner :             [0.5, 27, 0.5]
      velocity :                     [0, 0, 0]
      particle-type-id :             0
vtk-filename :                       explodingLiquid
vtk-write-frequency :                10000000
use-tuning-logger :                  false
output-suffix :
log-level :                          info
no-flops :                           false
no-end-config :                      true
no-progress-bar :                    false
load-balancer :                      InvertedPressure
load-balancing-interval :            100
subdivide-dimension :                [true, true, true]
```

Listing A.3: spinodalDecompositionEquilibration.yaml

```
container :                          [LinkedCells, VerletClusterLists, VerletLists, VerletListsCells
    , PairwiseVerletLists]
verlet-rebuild-frequency :           10
verlet-skin-radius-per-timestep :    0.05
fastParticlesThrow :                 false
verlet-cluster-size :                4
selector-strategy :                  Fastest-Absolute-Value
data-layout :                        [AoS, SoA]
traversal :                          [lc_c01, lc_c01_combined_SoA, lc_c04, lc_c04_HCP,
    lc_c04_combined_SoA, lc_c08, lc_c18, lc_sliced, lc_sliced_balanced, lc_sliced_c02, ot_c01,
    ot_c18, vcl_c01_balanced, vcl_c06, vcl_sliced, vcl_sliced_balanced, vcl_sliced_c02,
    vl_list_iteration, vlc_c01, vlc_c18, vlc_sliced, vlc_sliced_balanced, vlc_sliced_c02, vlp_c01
    , vlp_c18, vlp_sliced, vlp_sliced_balanced, vlp_sliced_c02, vlp_c08, vvl_as_built]
tuning-strategies :                  []
tuning-interval :                    1000
tuning-samples :                     3
functor :                            Lennard-Jones (12-6) AVX intrinsics
newton3 :                            [disabled, enabled]
cutoff :                             2.5
box-min :                            [-0.25, -0.25, -0.25]
box-max :                            [46, 46, 46]
```

```
cell-size :                          [1]
deltaT :                             0.00182367
sorting-threshold :                  8
iterations :                         10000
boundary-type :                      [periodic, periodic, periodic]
Sites:
    0:
    epsilon :                        1
    sigma :                          1
    mass :                           1
Objects:
    CubeGrid:
    0:
        particles-per-dimension :    [30, 30, 30]
        particle-spacing :           1.5
        bottomLeftCorner :           [0.5, 0.5, 0.5]
        velocity :                   [0, 0, 0]
        particle-type-id :           0
thermostat:
    initialTemperature :             1.4
    targetTemperature :              1.4
    deltaTemperature :               2
    thermostatInterval :             10
    addBrownianMotion :              true
vtk-filename :                       SpinodalDecomposition_equilibration
vtk-write-frequency :                10000
use-tuning-logger :                  false
output-suffix :
log-level :                          info
no-flops :                           false
no-end-config :                      false
no-progress-bar :                    false
load-balancer :                      InvertedPressure
load-balancing-interval :            100
subdivide-dimension :                [true, true, true]
```

Listing A.4: spinodalDecomposition.yaml

```
container :                          [LinkedCells, VerletClusterLists, VerletLists, VerletListsCells
    , PairwiseVerletLists]
verlet-rebuild-frequency :           15
verlet-skin-radius-per-timestep :    0.2
fastParticlesThrow :                 false
verlet-cluster-size :                4
selector-strategy :                  Fastest-Absolute-Value
data-layout :                        [AoS, SoA]
traversal :                          [lc_c01, lc_c01_combined_SoA, lc_c04, lc_c04_HCP,
    lc_c04_combined_SoA, lc_c08, lc_c18, lc_sliced, lc_sliced_balanced, lc_sliced_c02, ot_c01,
    ot_c18, vcl_c01_balanced, vcl_c06, vcl_sliced, vcl_sliced_balanced, vcl_sliced_c02,
    vl_list_iteration, vlc_c01, vlc_c18, vlc_sliced, vlc_sliced_balanced, vlc_sliced_c02, vlp_c01
    , vlp_c18, vlp_sliced, vlp_sliced_balanced, vlp_sliced_c02, vlp_c08, vvl_as_built]
tuning-strategies :                  []
tuning-interval :                    5000
tuning-samples :                     3
functor :                            Lennard-Jones (12-6) AVX intrinsics
newton3 :                            [disabled, enabled]
cutoff :                             2.5
box-min :                            [-0.75, -0.75, -0.75]
box-max :                            [239.25, 239.25, 239.25]
cell-size :                          [1]
deltaT :                             0.00182367
sorting-threshold :                  8
iterations :                         30000
boundary-type :                      [periodic, periodic, periodic]
Sites:
  0:
    epsilon :                        1
    sigma :                          1
    mass :                           1
Objects:
thermostat:
  initialTemperature :               0.7
  targetTemperature :                0.7
  deltaTemperature :                 2
  thermostatInterval :               10
  addBrownianMotion :                false
vtk-filename :                       SpinodalDecomposition
vtk-write-frequency :                30000
checkpoint :                         output/SpinodalDecomposition_equilibration/
    SpinodalDecomposition_equilibration_10000.pvtu
use-tuning-logger :                  false
output-suffix :
log-level :                          info
no-flops :                           false
no-end-config :                      false
no-progress-bar :                    false
```

```
load-balancer :                 InvertedPressure
load-balancing-interval :       100
subdivide-dimension :           [true, true, true]
```

Listing A.5: fallingDrop.yaml

```
container :                     [LinkedCells, VerletClusterLists, VerletLists, VerletListsCells]
verlet-rebuild-frequency :      10
verlet-skin-radius-per-timestep : 0.1
fastParticlesThrow :            false
verlet-cluster-size :           4
selector-strategy :             Fastest-Absolute-Value
data-layout :                   [AoS, SoA]
traversal :                     [lc_c01, lc_c08, lc_c18, lc_sliced_c02, vcl_c01_balanced, vcl_c06, vcl_cluster_iteration, vl_list_iteration, vlc_c01, vlc_c18, vlc_sliced_c02]
tuning-strategies :             []
tuning-interval :               2500
tuning-samples :                3
functor :                       Lennard-Jones (12-6) AVX intrinsics
newton3 :                       [disabled, enabled]
cutoff :                        3
box-min :                       [0, 0, 0]
box-max :                       [49.5612, 29.5612, 37.296]
cell-size :                     [1]
deltaT :                        0.0005
sorting-threshold :             8
iterations :                    15000
boundary-type :                 [reflective, reflective, reflective]
Sites:
  0:
    epsilon :                   1
    sigma :                     1
    mass :                      1
Objects:
  CubeClosestPacked:
    0:
      particle-spacing :        1.12246
      box-length :              [48, 28, 10]
      bottomLeftCorner :        [1, 1, 1]
      velocity :                [0, 0, 0]
      particle-type-id :        0
  Sphere:
    0:
      center :                  [18, 15, 30]
      radius :                  6
      particle-spacing :        1.12246
      velocity :                [0, 0, 0]
      particle-type-id :        0
globalForce :                   [0, 0, -12]
vtk-filename :                  fallingDrop
vtk-write-frequency :           1000
use-tuning-logger :             false
output-suffix :
log-level :                     info
no-flops :                      false
no-end-config :                 true
no-progress-bar :               false
load-balancer :                 InvertedPressure
load-balancing-interval :       100
subdivide-dimension :           [true, true, true]
```

A.6. Benchmark

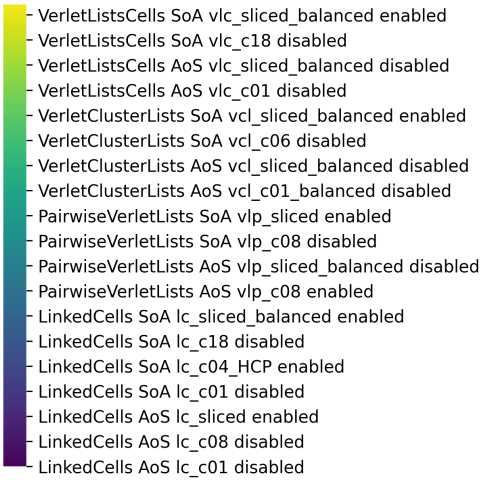


Figure A.1.: Colorbar for the heatmaps used in the 5 section.

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.

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

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

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

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

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

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

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

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