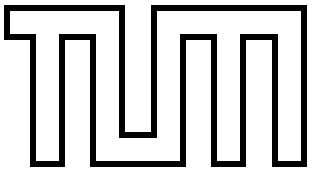


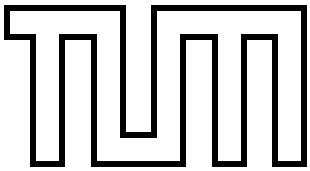
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploring Fuzzy Tuning Technique for Molecular Dynamics Simulations in AutoPas

Manuel Lerchner



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploring Fuzzy Tuning Technique for Molecular Dynamics Simulations in AutoPas

Remove all TODOS

Untersuchung von Fuzzy Tuning Verfahren für Molekulardynamik-Simulationen in AutoPas

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

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

Munich, 10.08.2024 Manuel Lerchner

Contents

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 given a specific container. 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:

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

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 $\tilde{A}$.

Those definitions arise naturally by extending the classical set operations into the realm of fuzzy sets. The similarity between the classical and fuzzy set operations are depicted in Table 2.2.

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.

Classical Set Operation	Fuzzy Set Operation
$x \in A \cap B \iff x \in A \wedge x \in B$	$\mu_{\tilde{A} \cap \tilde{B}}(x) = \mu_{\tilde{A}}(x) \wedge \mu_{\tilde{B}}(x)$ $= \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$
$x \in A \cup B \iff x \in A \vee x \in B$	$\mu_{\tilde{A} \cup \tilde{B}}(x) = \mu_{\tilde{A}}(x) \vee \mu_{\tilde{B}}(x)$ $= \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$
$x \in \bar{A} \iff x \notin A \iff \neg(x \in A)$	$\mu_{\neg \tilde{A}}(x) = \neg \mu_{\tilde{A}}(x)$ $= 1 - \mu_{\tilde{A}}(x)$

Table 2.2.: Similarities between classical and fuzzy set operations

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

- **Weighted Average**
  The Weighted Average method calculates the average of all the input values weighted by their membership values. Contrary to the Centroid method which integrates over the whole domain, the Weighted Average method only considers a 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

A. Appendix

A.1. Glossary

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

md_flexible A flexible molecular dynamics simulation framework built on top of AutoPas. xiii, 5

A.2. LiveInfoLogger Data Fields

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

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

Define the metric

Which unit?

A.3. TuninData Fields

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

Date The date and time when the data was collected.

Iteration The current iteration number of the simulation.

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

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

check https://med

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

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

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

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

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

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

Smoothed A smoothed version of the reduced performance data.

List of Figures

List of Tables

Bibliography

[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.

[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.