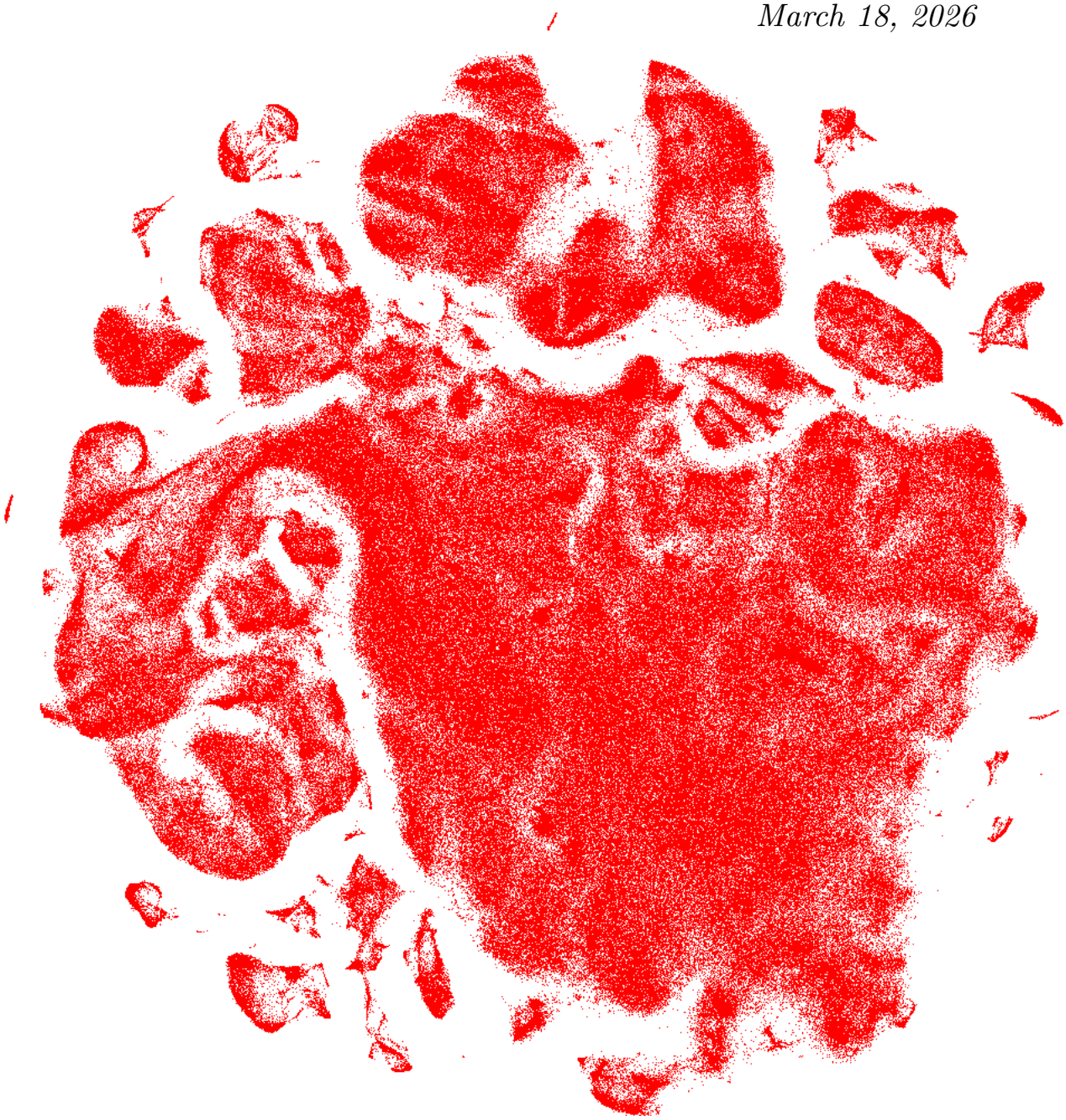
# FMM-t-SNE: Accelerating t-SNE with the fast multipole method

*March 18, 2026*



Lucas Frans Willem Furer

# FMM-t-SNE: Accelerating t-SNE with the fast multipole method

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Lucas Frans Willem Furer


Computer Graphics and Visualization Group
Department of Intelligent Systems
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

# FMM-t-SNE: Accelerating t-SNE with the fast multipole method

Author: Lucas Frans Willem Furer
Student id: 5071356
Email: L.F.W.Furer@student.tudelft.nl

**Abstract**

The t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm is a tool for analyzing high-dimensional data, such as RNA sequencing data from brain cells [1]. However, its applicability to large datasets is limited by the quadratic complexity of its gradient computation, which arises from all-to-all interactions between points. This computation can be interpreted as an N-body problem, enabling the use of approximation methods to accelerate evaluation.

This work investigates the fast multipole method (FMM), a hierarchical N-body solver based on multipole expansions, as an acceleration strategy for t-SNE. To further optimize performance, a time-varying $\theta$ parameter is introduced to adapt the approximation accuracy during optimization. FMM is integrated into the t-SNE framework and evaluated against existing approaches, including the Barnes–Hut (BH) method and the particle-mesh method (PM).

Results show that FMM outperforms other BH-like$^{\dagger}$ methods in terms of the error–time trade-off, but does not match the performance of PM. The proposed time-varying $\theta$ parameter enables FMM to achieve a lower embedding cost within the same runtime as PM, improving the cost–time trade-off despite not surpassing its error–time efficiency.

Thesis Committee:

| | |
|---|---|
| Chair: | Prof.Dr. K. Hildebrandt, Faculty EEMCS, TU Delft |
| Committee Members: | Prof.Dr.Ir. H.X. Lin, Faculty EEMCS, TU Delft |
| | J. Campolattaro |

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

Dimensionality reduction is a technique useful for analyzing and visualizing high-dimensional datasets. By transforming data into a lower-dimensional representation, it becomes possible to inspect structural patterns that would otherwise be difficult to observe. One widely used nonlinear dimensionality reduction method is the t-Distributed Stochastic Neighbor Embedding (t-SNE). T-SNE is designed to reveal structure in complex datasets by preserving local neighborhood relationships when projecting data into a low-dimensional space. As a result, it is capable of uncovering clusters and manifold structure in data such as image features, document embeddings or biological measurements. For example, the identification of cell types from RNA-sequences [1].

Despite its effectiveness for visual exploration, t-SNE suffers from computational limitations. Generating a single embedding for a large dataset can require substantial runtime, which limits its usefulness in interactive analysis and exploratory workflows. The computational bottleneck arises from the iterative optimization procedure used to construct the low-dimensional embedding. In each iteration, the algorithm updates the positions of embedded points by minimizing a cost function. This cost function measures the divergence between probability distributions defined in the high-dimensional and low-dimensional spaces. Evaluating the gradient of this cost function requires computing pairwise interactions between all points in the embedding. Because every point interacts with every other point, the computation scales quadratically with the number of data points.

This all-to-all interaction pattern can be interpreted as an instance of the N-body problem. In general, an N-body problem describes a system of entities that interact with each other, such as gravitational or electrostatic systems. Computing these interactions requires evaluating all pairwise forces, resulting in a computational cost proportional to $O(n^2)$. Such scaling quickly becomes prohibitively expensive for large systems. Methods have been developed to accelerate N-body computations by approximating distant interactions. Methods such as the Barnes–Hut method use hierarchical spatial decompositions to replace multiple Particle–Particle interactions with fewer Particle–Node interactions, thereby reducing computational complexity while maintaining acceptable accuracy.

This work explores the use of the Fast Multipole Method (FMM) to accelerate the computation of the derivative of t-SNE. The FMM is a hierarchical N-

body solver that approximates interactions between groups of particles through multipole expansions, enabling efficient Node–Node interactions. In this paper, the FMM is integrated into the t-SNE framework and evaluated against existing acceleration strategies, including the BH method and particle-mesh method (PM). Experimental results show that the FMM outperforms other BH-like$^{\dagger}$ methods in terms of the balance between computational speed and approximation accuracy, although it does not surpass the overall efficiency of the PM. To further improve the performance of BH-like$^{\dagger}$ methods, the concept of a time-varying $\theta$ parameter is introduced. This strategy allows the FMM to achieving a lower cost value within the same runtime as the PM method.

# Chapter 2

# Related Work

## 2.1 Dimensionality Reduction Methods

Stochastic Neighbor Embedding (SNE) [2] steers away from traditional approaches that preserve rigid geometric distances. Instead, SNE uses probabilistic similarities to represent the relationships between data points. In SNE, the high-dimensional distances between points are converted into conditional probabilities that represent the likelihood of one point choosing another as its neighbor. By focusing on these probabilities, SNE prioritizes the preservation of local neighborhood structures over global geometry. The goal of the algorithm is to find a low-dimensional configuration of points where the probability distribution matches the high-dimensional distribution as closely as possible, typically by minimizing the Kullback-Leibler divergence.

SNE suffers from two drawbacks: it has a difficult-to-optimize cost function [3] and it experiences the "crowding problem". The t-Distributed Stochastic Neighbor Embedding (t-SNE) [3] addresses these limitations through two modifications to the SNE framework. First, t-SNE replaces the asymmetric conditional probabilities with a symmetric version, which simplifies the gradient calculation. Second, t-SNE replaces the Gaussian distribution used in the low-dimensional space with a heavy-tailed Student's t-distribution.

## 2.2 N-Body Solvers

Particle-Particle Particle-Mesh ($P^3M$) [4] reduces computation time by collecting particles on a mesh and solving for the potential in Fourier space. The calculated potentials are then interpolated from the mesh to the particles. To maintain sufficient accuracy, the method falls back to the naive approach and performs Particle-Particle interactions for cells that are too close to each other. In the worst case, this requires $O(N^2)$ operations per iteration.

The Barnes–Hut method (BH) [5] works by subdividing the particles in an octree structure. This structure is recreated each iteration and keeps track of aggregated properties, such as total mass or electric charge. By allowing each particle to interact with a node in the tree, multiple Particle-Particle interactions are replaced by a single Particle-Node interaction. The accuracy parameter $\theta$, which is set by hand, controls how strict the conditions are for a particle to

accept a particular node instead of performing the interaction with all of its smaller child nodes. This method achieves an asymptotic improvement over the naive method, resulting in an $O(n \log(n))$ time complexity for each iteration.

The Multipole method (MM) [6] extends the BH method by adding multipole moments to increase the accuracy of the Particle-Node interactions.

The fast multipole method (FMM) [7] extends the BH method by replacing the single tree traversal with a dual-tree traversal. While the BH method computes the forces acting on each individual particle by traversing the tree structure, the FMM instead computes interactions between nodes. This approach facilitates Node-Node interactions, where the forces from a "source" node are collected and applied to a "sink" node. While these Node-Node interactions produce a higher degree of approximation error than the Particle-Node interactions used in the BH method, this is compensated for by the inclusion of multipole moments (at the source nodes) and multipole fields (at the sink nodes).

The FMM can also be formulated using Cartesian coordinates instead of spherical coordinates [8]. This formulation allows for symmetrized Node-Node interactions, which results in Newton's third law being satisfied by construction. The achieved time complexity of the FMM is $O(n \log(\theta))$, which is linear for a fixed $\theta$. However, maintaining constant error requires the accuracy parameter $\theta$ to scale with the size of the input, resulting in super-linear time complexity.

TreePM [9] and Gadget-4 [10] demonstrate that it is possible to combine the PM with tree methods to take advantage of their respective strengths. Such methods use the PM for computing long-range forces and the BH method or FMM for short-range forces.

## 2.3 Integration of N-Body Solvers into T-SNE

Barnes–Hut-SNE [11] and FIt-SNE [12] combine the BH method and PM respectively with the t-SNE method in order to accelerate the components that require solving an N-body problem. A GPU implementation of the PM for t-SNE is also presented in [13].

# Chapter 3

# Background

## 3.1 SNE

This section provides a more detailed explanation of how SNE reduces the dimensionality of data. The high-dimensional input data is defined in eq. (3.1).

$$\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, \quad \mathbf{x}_i \in \mathbb{R}^w \tag{3.1}$$

Here $\mathcal{X}$ is the high dimensional dataset, $n$ the number of entries and $w$ the dimensionality of the entries. The aim of the algorithm is to create a new dataset $\mathcal{Y}$ with matching points where the dimensionality is reduced to $v$, usually where $2 \leq v \leq 3$. This new dataset is called the embedded dataset and is defined in eq. (3.2).

$$\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}, \quad \mathbf{y}_i \in \mathbb{R}^v \tag{3.2}$$

At its core, SNE relies on assigning probabilities to the chance that point $i$ is a neighbor of point $j$. These probabilities are calculated with a separate Gaussian centered on each point. When the probabilities of every possible pair of points is calculated, the results are put in a matrix form. The probabilities for the original high-dimensional data are called $p_{i|j}$ and are calculated using eq. (3.3).

$$p_{i|j} = \frac{\exp(-d_{ij}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)} \tag{3.3}$$

The $d_{ij}$ variables are known as the dissimilarity between two points. It represents how different two points are and may be computed with the scaled squared Euclidean distance.

$$d_{ij}^2 = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2} \tag{3.4}$$

The value for $\sigma_i$ can be set by hand or by using the perplexity parameter. A higher perplexity will result in more neighbors having probabilities that are not close to zero. Through a binary search, can the values of $\sigma_i$ be found such that $\text{Perp}(P_i)$ from eq. (3.5) becomes as close as possible to the desired perplexity value.

$$\text{Perp}(P_i) = 2^{H(P_i)} \tag{3.5}$$

Here the $P_i$ represents the conditional probability distribution of point $i$ over all other points.

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i} \tag{3.6}$$

The neighbor probability matrix for the embedded dataset can be calculated using eq. (3.7).

$$q_{i|j} = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2)}{\sum_{k \neq i} \exp(-\|\mathbf{y}_i - \mathbf{y}_k\|^2)} \tag{3.7}$$

The goal of the SNE algorithm is to place the points in the embedded dataset such that we minimize the difference between $p_{i|j}$ and $q_{i|j}$. This is achieved by minimizing a cost function. The chosen cost function for SNE is the sum of Kullback-Leibler divergences between $p_{i|j}$ and $q_{i|j}$, which is illustrated in eq. (3.8).

$$C = \sum_i \sum_j p_{i|j} \log \frac{p_{i|j}}{q_{i|j}} = \sum_i KL(P_i||Q_i) \tag{3.8}$$

By differentiating $C$, it is possible to determine how the embedded dataset should be changed so that it matches the high-dimensional dataset more closely.

$$\frac{\partial C}{\partial \mathbf{y}_i} = 2 \sum_j (\mathbf{y}_i - \mathbf{y}_j)(p_{i|j} - q_{i|j} + p_{j|i} - q_{j|i}) \tag{3.9}$$

The resulting gradient can be interpreted as a sum of forces that are either pulling or pushing point $\mathbf{y}_i$ towards $\mathbf{y}_j$ depending on how much they are observed to be each others neighbor.

The direction in which each point in $\mathcal{Y}$ should be moved in order to improve its similarity to $\mathcal{X}$ is now known. What remains is to choose a method for updating $\mathcal{Y}$ using the derivatives. A natural first choice is steepest descent. However, this approach often leads to solutions that become trapped in suboptimal local minima. An alternative approach is to combine random jitter with steepest descent, where the random jitter gradually decreases to zero over time. This stochastic component enables the algorithm to escape certain local optima. The final update rule for the embedded dataset is given eq. (3.10).

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\partial C}{\partial \mathcal{Y}} + \mathcal{R} \tag{3.10}$$

Here $R$ is the random jitter and $t$ is the update-step counter.

The initial sampling of points in $\mathcal{Y}$ is taken from the following Gaussian distribution: $\mathcal{N}(0,\, 10^{-4}I)$. $\mathcal{Y}$ is then updated for 1000 iterations using eq. (3.10). The complete algorithm can be found in algorithm 5.

### 3.1.1 Optimizations

Annealing on the perplexity can reduce the impact of suboptimal local minima.

Sparsifying the $p_{i|j}$ matrix reduces the computational costs by eliminating the need to go over every entry. This can be done because of two reasons. First, the matrix is calculated once and then reused every iteration. Second, any value that is lower than a certain threshold can be considered as zero. The actual Sparsification of the matrix can be achieved by first calculating the full matrix and then setting each value lower than a certain threshold to zero. The resulting matrix can then be converted to a sparse representation, such as CSR or COO, and subsequently normalized to ensure that the sum of the probabilities remains equal to 1.

## 3.2 N-Body Problem

An N-body problem describes a system of entities that interact with each other in order to determine the behavior of the system. Classical examples include gravitational interactions between celestial bodies and electromagnetic interactions between charged particles. Although these interactions are typically associated with physical systems, any system with all-to-all interactions can be viewed as an N-body problem, as illustrated in fig. 3.1.

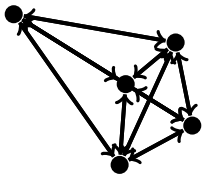


Figure 3.1: Solving the N-body problem for 5 points requires 10 pair interactions.

### 3.2.1 Naive

For a system of particles, several approaches exist to predict their future motion. Analytical solutions provide exact predictions for any time horizon, but such solutions are known only for systems where $N \leq 2$, or for certain special configurations when $N > 2$. In the general case, iterative numerical methods must be used instead. In the context of gravitational interactions, these methods are based on the gravitational potential in eq. (3.11) and the corresponding acceleration obtained from its gradient in eq. (3.12).

$$\Phi(\mathbf{x}_i) = -\mathbf{G} \sum_{j \neq i} \mu_j \frac{1}{\|\mathbf{x}_i - \mathbf{x}_j\|} \tag{3.11}$$

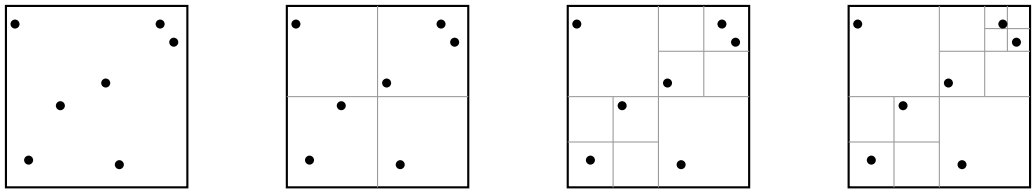
$$-\nabla\Phi(\mathbf{x}_i) = -\mathbf{G}\sum_{j\neq i}\mu_j\frac{(\mathbf{x}_i-\mathbf{x}_j)}{\|\mathbf{x}_i-\mathbf{x}_j\|^3} \tag{3.12}$$

Here $\mu_j$ denotes the mass of particle $j$.

A limitation of iterative methods is the accumulation of numerical error, where inaccuracies introduced in one iteration propagate and compound in subsequent steps. For chaotic systems such as gravitational dynamics, this sensitivity to initial errors makes reliable long-term predictions extremely challenging. A naive implementation for computing particle interactions is shown in algorithm 6. The time complexity of a single iteration of this algorithm is $O(n^2)$, which serves as a baseline for evaluating more efficient N-body algorithms.

### 3.2.2 Barnes–Hut

The first step of the Barnes–Hut (BH) method is the construction of an octree, which represents the hierarchical spatial decomposition of the particles and must be rebuilt at each iteration. The octree is constructed by initializing a root node that contains all particles in the system together with a tight square bounding box. This root node is then subdivided into eight child nodes, four nodes in the case of a two-dimensional space, each with half the side length of the parent node. By recursively applying this subdivision process to newly created nodes, the particles are progressively partitioned until each leaf node contains a single particle. A two-dimensional illustration of this procedure is shown in fig. 3.2.



(a) Root Node (b) Level 1 (c) Level 2 (d) Level 3

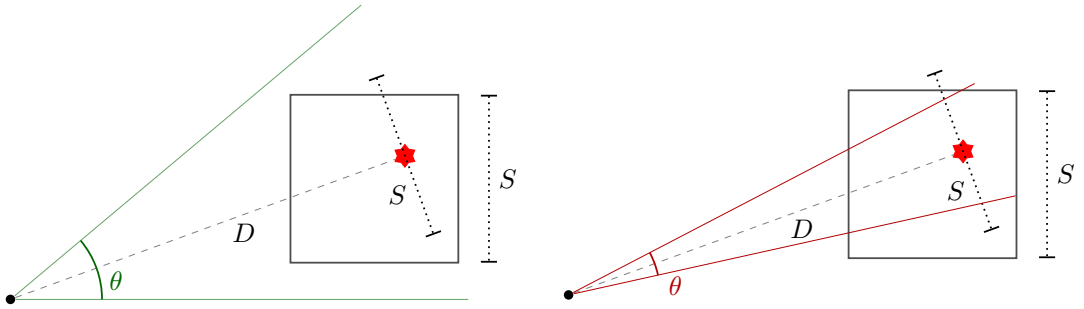
Figure 3.2: Construction of a Quadtree. Each cell is subdivided into four quadrants if it contains more than a single point.

The second step of the algorithm consists of assigning aggregate information to each node based on the particles it contains. In the BH method, each node stores the total mass of its particles and the corresponding center of mass. These quantities can be computed efficiently by first evaluating them at the leaf nodes. Once the mass and center of mass have been determined for each leaf node, the properties of the parent nodes can be obtained recursively by aggregating the values of their child nodes.

Before the constructed tree can be used for force evaluation, a criterion is required to determine when a Particle–Node interaction provides a sufficiently accurate approximation. For this purpose, a descent criterion is used that depends on an accuracy parameter $\theta$. The condition is defined in eq. (3.13).

$$\frac{S}{D} < \theta \tag{3.13}$$

Here, $S$ denotes the side length of the node and $D$ the distance between the particle and the node's center of mass. The parameter $\theta$ can be interpreted as the minimum viewing angle required for the node to be treated as a single aggregated source. An illustration of this viewing angle is shown in fig. 3.3.




(a) Accepted interaction (low accuracy). (b) Rejected interaction (high accuracy).

Figure 3.3: Comparison of descent criterion for two nodes, where one interaction is rejected and another accepted.

After the tree construction has been completed, the force computation can begin. For each particle, the total force exerted by the system is accumulated by performing a Particle–Node interaction starting at the root node. If the descent criterion is satisfied, the interaction between the particle and the node is approximated using the node's aggregated properties. Otherwise, the tree traversal continues with the node's children. In cases where even the smallest node fails to satisfy the descent criterion, the interaction is evaluated directly as a Particle–Particle interaction. Once the forces on all particles have been computed, the particle velocities and positions are updated, after which the entire procedure is repeated. The complete algorithm is shown in algorithm 7.

### 3.2.3 Reverse Barnes–Hut

Reverse Barnes–Hut (rBH) differs from the standard BH formulation in that, rather than computing the force on a single particle by traversing the tree as a source structure, it computes the force exerted by an individual source particle

on the tree treated as a sink. The underlying principles remain largely the same, with the exception that each node must store its accumulated acceleration. This acceleration is subsequently propagated down the tree until it reaches the individual particles.

Reverse Barnes–Hut is rarely used in practice because it generally yields lower accuracy than the standard BH approach. The reduced accuracy arises from the assumption of a uniform force field within each node, which results in accumulated forces being passed to the children without modification. This approximation can introduce significant errors, even for interactions occurring at relatively large distances.

# Chapter 4

# FMM-Accelerated T-SNE

This chapter explains the details surrounding the adaptation of the FMM into t-SNE. Section 4.1 explains the details of the t-SNE algorithm. Section 4.2 compares the t-SNE terms from Section 4.1 to their N-body gravitational counterpart. This is done in order to create a connection between the two different concepts. Section 4.3 reformulates t-SNE in order to make it more compatible with the N-body context. Section 4.4 explains the FMM in detail. Section 4.5 discusses the specific changes that are required to adapt the FMM to work with the t-SNE method. Section 4.6 introduces the concept of varying the $\theta$ parameter to speed up the BH-like$^{\dagger}$ methods. Finally, Section 4.7 summarizes the main design choices that were made for implementing the FMM for t-SNE.

## 4.1 T-SNE

T-SNE is a method that builds upon SNE. It alleviates the crowding problem and the optimization problem of SNE. It does this by symmetrizing the neighborhood probabilities along with the cost function and by replacing the Gaussian used on the embedded dataset with the Student's t-distribution.

### 4.1.1 Crowding/Optimization Problem

The crowding problem occurs when points near the center of the embedding are compressed by attractive forces arising from points that are moderately distant in the high-dimensional space. This leads to a visualization where points are crushed together in the center of the map, obscuring the distinct cluster structures present in the high-dimensional data. This phenomena is not unique to SNE since it can also be found in the Sammon method. The optimization problem refers to the difficult-to-optimize cost function used by SNE which increases the runtime of the method.

### 4.1.2 Student's T-Distribution

SNE calculates its neighborhood probabilities in the embedded dataset with equation eq. (3.7). A Gaussian is used which tends to collect a large part of its probability near the center. The t-SNE method attempts to solve this by

replacing the Gaussian from eq. (3.7) with Student's t-distribution which is shown in eq. (4.1).

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq l (1+\|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}}, \quad \text{where } q_{ii} = 0 \tag{4.1}$$

The purpose of using the t-distribution is twofold. First, it helps to alleviate the crowding problem. Second, it makes the entries in the $q_{ij}$ matrix symmetric. An additional desirable property is that this distribution is less computationally expensive than the previously used Gaussian.

### 4.1.3 Symmetrizing Probabilities and the Cost Function

T-SNE also symmetrizes the $p_{ij}$ matrix, shown in eq. (4.2), while still using a Gaussian to calculate the probabilities.

$$p_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{k \neq l} \exp(-d_{kl}^2)} \tag{4.2}$$

Equation (4.2) uses the same dissimilarity equation, eq. (3.4), as the unsymmetrical version in eq. (3.3). By changing the normalization to include all the values in the probability matrix, instead of only normalizing each neighbor of a point, can a symmetric version for the probabilities be created. However, there is a downside to this approach.

Consider a point $\mathbf{x}_i$ for which all pairwise distances are large. Such a point consequently has low values of $p_{ij}$ for all $j$'s. Because all $p_{ij}$ values for this point are small, its low-dimensional location has little influence on the cost function. To avoid this issue, the equation for $p_{ij}$ can be symmetrized in a different way, as shown in eq. (4.3).

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}, \quad \text{where } p_{ii} = 0 \tag{4.3}$$

The new equation takes the normalized sum of the probabilities from SNE, where $i$ belongs to $j$ and vise versa. This guarantees that $p_{ij}$ is equal to $p_{ji}$. The advantage is that $\sum_j p_{ij} > \frac{1}{2n}$ holds for every $\mathbf{x}_i$, ensuring that each point contributes more uniformly to the cost function. The cost function is changed to eq. (4.4) which was first introduced in [14]. Taking the derivative of this equation yields eq. (4.5).

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} = KL(P||Q) \tag{4.4}$$

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_j (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j)(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1} \tag{4.5}$$

Similarly to SNE, the derivative contains both an attractive and a repulsive component. This decomposition provides insight into the behavior of the gradient. All points exert a certain repulsive force on each other in the embedded dataset, which can only be overcome by high similarity in the high-dimensional dataset.

### 4.1.4 Update Scheme

The update Scheme has the following changes compared to SNE:

- The random jitter has been replaced with a momentum term. This momentum term takes the difference between the current and previous position of the points in the embedded dataset and adds this to the derivative. The $\alpha$ parameter determines the strength of the momentum and is set to 0.5 for $t < 250$ and 0.8 for $t \geq 250$.
- An "early exaggeration" method is added which multiplies all values of the $p_{ij}$ matrix by 4 when $t < 50$. In [11] this "early exaggeration" is used for $t < 250$.

The total number of iterations is set to 1000 and the learning rate is set adaptively [15] with an initial value of 100. The complete algorithm for t-SNE can be found in algorithm 1.

**Algorithm 1:** Simple version of t-Distributed Stochastic Neighbor Embedding.

**Data:** dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$,
perplexity $\mathcal{P}$,
iteration amount $\mathcal{T}$,
learning rate $\eta$,
momentum $\alpha$

**Result:** low dimensional data $\mathcal{Y}^{(t)}$

**Algorithm:**

compute $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ using eq. (3.5) with $\mathcal{P}$

compute $p_{i|j}$ using eq. (3.3) with $\Sigma$

set $p_{ij} = \dfrac{p_{j|i} + p_{i|j}}{2n}$

initialize $\mathcal{Y}^{(0)}$ with $\mathcal{N}(0,\, 10^{-4}I)$

**for** $t = 1$ **to** $\mathcal{T}$ **do**

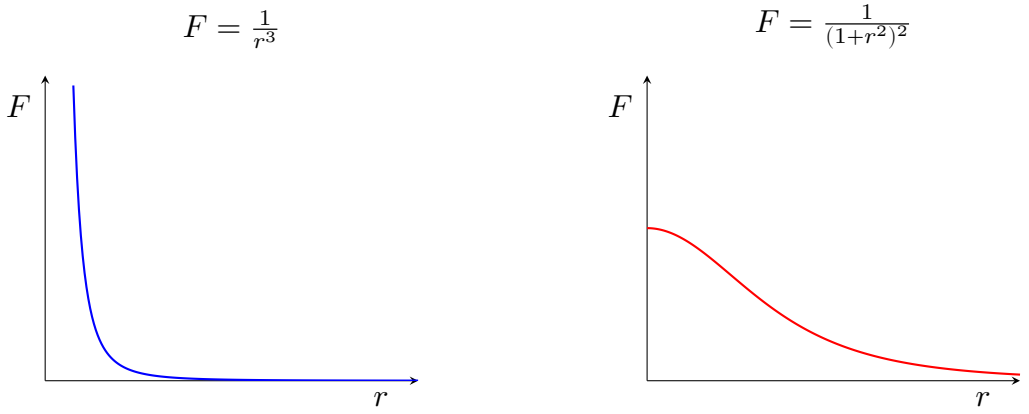
- compute $q_{ij}$ using eq. (4.1)
- compute gradients using eq. (4.5)
- set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \dfrac{\partial C}{\partial \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{t-2})$

## 4.2 T-SNE Compared to Gravitational

Many concepts in t-SNE have analogies in gravitational dynamics. This section discusses these similarities, as well as their differences, in order to build intuition about how they relate to each other. The comparison begins with the formula for gravitational acceleration in eq. (3.12) and the derived formula for $F_{rep}$ in eq. (4.9). The following differences can be observed.

- Instead of multiplying by a constant factor of $-\mathbf{G}$, a normalization term $Z$ is used. This normalization requires an accumulation of information about every interaction, which is not present in regular N-body dynamics. The normalization can be interpreted as a value that accumulates contributions from every interaction. After all interactions have been computed, a post-processing step divides the results by this value. Because this normalization is always positive, the direction of $F_{rep}$ is inverted compared to gravity, producing a pushing effect between points rather than a pulling effect.
- Another difference is the absence of $\mu$ in $F_{rep}$. This value represents the mass of a particle and increases the strength of the force it emits. In t-SNE, each point is treated as equally important and there is therefore no notion of mass or weight. This can be interpreted as every point having a mass of 1, which explains why the term does not appear in the equation.
- The decay rate of the force, or in the case of t-SNE, the derivative show different behavior for gravity and t-SNE. For gravity, the decay scales with $1/r^3$, while for t-SNE it is $1/(1+r^2)^2$, where $r$ is the distance between two points. A desirable side effect of this alternative decay function is that it is well behaved near zero. Unlike gravitational forces, where particles that are close to each other can produce extremely large forces, the decay used in t-SNE avoids this issue and therefore does not require a softening parameter. The shapes of these decay functions are illustrated in fig. 4.1.
- Whereas gravitational simulations are typically modeled with only a single type of force, t-SNE uses two: an attractive and a repulsive component. These forces are computed independently and then combined. This means that the attractive component can be ignored for our N-body view of the repulsive component.
- Particles under the effect of gravity typically have their velocity updated using the calculated acceleration at each iteration, after which the new positions are determined from the updated velocities. t-SNE does not explicitly model velocity and instead updates the positions directly from the acceleration, which is equivalent to the gradient. A pseudo-velocity term nevertheless exists, defined as the difference between the current point positions and those from the previous iteration.
- The final difference is that t-SNE does not require high numerical accu-

racy. Gravitational simulations, which predict the behavior of physical systems, require high accuracy to remain physically realistic. T-SNE has no direct physical interpretation. The objective is only to reduce the cost function and obtain a meaningful qualitative structure in the embedding. A method that is capable of reducing the cost function while displaying large approximation error is considered acceptable.




(a) Force decay of gravity.

(b) Force decay of t-SNE.

Figure 4.1: Visualizing the force decay of gravity and t-SNE.

## 4.3 Reformulate T-SNE

Before explaining the FMM, the t-SNE update scheme shall first be reformulated to an N-body context. Doing this first will assist in integrating the FMM into t-SNE. The first step is to separate the $p$ and $q$ matrices in eq. (4.5). Expanding the multiplication of $(p_{ij} - q_{ij})$ with the remaining terms in eq. (4.5) yields eq. (4.6).

$$\frac{\delta C}{\delta Y_i} = 4\left(\sum_j p_{ij}\frac{(y_i - y_j)}{1 + \|y_i - y_j\|^2} - \sum_j q_{ij}\frac{(y_i - y_j)}{1 + \|y_i - y_j\|^2}\right) \tag{4.6}$$

This separation allows the two components to be computed independently and subsequently combined to form the final derivative. While the $p$ matrix is sparsified to reduce computational cost, the $q$ matrix cannot be sparsified. Instead, the computation of the $q$ matrix is treated as an N-body problem. The repulsive component can be isolated as shown in eq. (4.7).

$$\frac{\delta C}{\delta Y_i} = 4(F_{attr,i} - F_{rep,i}) \quad \text{where } F_{rep,i} = \sum_j q_{ij}\frac{(y_i - y_j)}{1 + \|y_i - y_j\|^2} \tag{4.7}$$

Substituting the definition of $q_{ij}$ from eq. (4.1) into $F_{rep,i}$ yields eq. (4.8).

$$F_{rep,i} = \sum_j \sum_{k \neq l} \frac{(1 + \|y_i - y_j\|^2)^{-1}(y_i - y_j)}{(1 + \|y_k - y_l\|^2)^{-1}(1 + \|y_i - y_j\|^2)} \tag{4.8}$$

Further simplification can be obtained by combining the factors $(1 + \|y_i - y_j\|^2)$ in the numerator and denominator into a single term. The sum over $k$ and $l$ are substituted with $Z$ which results in eq. (4.9).

$$F_{rep,i} = \frac{1}{Z} \sum_j \frac{(y_i - y_j)}{(1 + \|y_i - y_j\|^2)^2} \quad \text{where} \quad Z = \sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1} \tag{4.9}$$

$$F_{attr,i} = \sum_j p_{ij} \frac{(y_i - y_j)}{1 + \|y_i - y_j\|^2} \tag{4.10}$$

The reason for formulating the equation as such is to make the comparison with gravitational formulas as clear as possible. This formulation is not unique. Related expressions appear in prior work, such as [11] and [12], which employ a form similar to eq. (4.11). Despite their apparent differences, these formulations are algebraically equivalent.

$$\frac{\delta C}{\delta Y_i} = 4 \left( \sum_{j \neq i} p_{ij} q_{ij} Z(\mathbf{y}_i - \mathbf{y}_j) - \sum_{j \neq i} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j) \right) \tag{4.11}$$

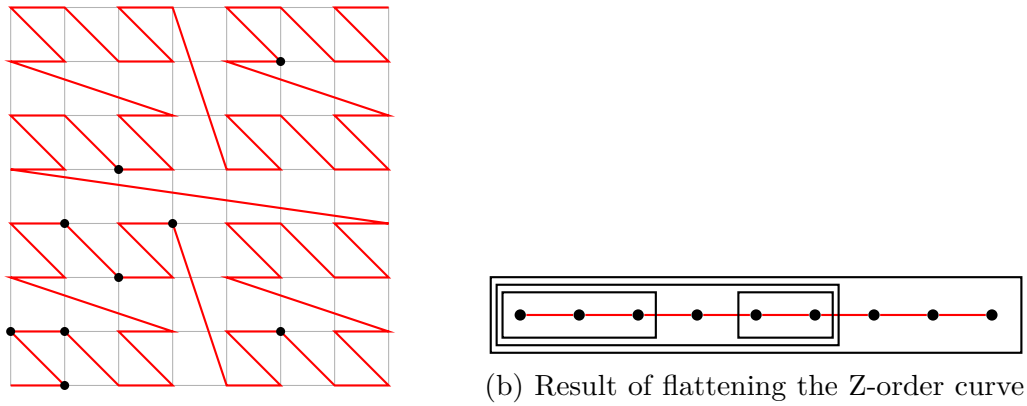
## 4.4 Fast Multipole Method

The FMM differs from the BH method by considering Node–Node interactions rather than Particle–Node interactions. Instead of treating a single particle as the sink and a node as the source, both the sink and the source are represented by nodes. In addition, the FMM typically employs a different tree construction strategy, often based on space-filling curves, to achieve an $O(n)$ time complexity.

To perform a Node–Node interaction, an approximation of the field over the sink node, induced by the mass distribution summary of the source node, must be constructed. A straightforward approach would be to approximate the source node by a center-of-mass distribution and evaluate the resulting field at the center of the sink node. However, this approach assumes a uniform field across the sink node, which introduces significant errors, even at large distances. The FMM overcomes this limitation through the use of multipole expansions. By replacing the center-of-mass representation with multipole moments, and the uniform field approximation with higher-order multipole fields, a more accurate representation of the interaction is obtained. These multipole expansions are based on Taylor series approximations of the underlying potential field, and their accuracy can be improved by increasing the expansion order.

### 4.4.1 Z-Order Curve Tree Construction

The tree construction used in the BH method proceeds by recursively subdividing nodes until each node contains at most a single point. This process has a time complexity of $O(n \log(n))$. Certain space-filling curves enable the construction of a similar hierarchical structure. Although not all space-filling curves are suitable for generating quadtree structures, the Hilbert curve and the Z-order curve are appropriate choices. These curves are effective because their projections, when sorted, produce natural square subdivisions of the domain. In this process, N-dimensional point coordinates are mapped to a single scalar value that determines the ordering of the points. When using the Z-order or Hilbert curve, points that are close in N-dimensional space tend to remain close in the resulting ordering. For the Z-order curve, this ordering is illustrated in fig. 4.2.



(a) Order determined by the Z-order curve.

(b) Result of flattening the Z-order curve for the given points.

Figure 4.2: Visualization of the Z-order curve. Points on the grid have an order dictated by the space filling curve which are then flattened to a one-dimensional representation. The hierarchical boxes in this one-dimensional representation show which points lie in the same node.

The tree construction can be summarized in 5 steps. First, the position of the points are interpolated onto a grid which can be achieved by converting the real numbered coordinates to non-negative integer numbered coordinates. Second the order for each point is calculated using the space filling curve. In the case of the Z-order curve, this can be calculated using the Morton code [16]. Third, the points are sorted by their order using a radix sort algorithm. Fourth, each point is iterated over in order and placed in their appropriate leaf node. The depth of the leaf nodes is dictated by the ”tree depth” hyperparameter. Fifth, all parent nodes are created through a bottom-up construction.

### 4.4.2 Multipole Moments

A multipole moment is a way to to describe the distribution of mass in a node. The difference between it and the center-of-mass summary is in how much information it captures. The amount of information can be controlled by setting the expansion order $n$, where higher order multipole moments contain more information. In order to calculate the multipole moments for a node, the center of mass must be know. Once that is calculated for each node, can eq. (4.12) be used to determine the multipole moments.

$$\mathbf{M}_B^n = \sum_{p \in \text{particles}(B)} \mu_p (\mathbf{x}_p - z_B)^{(n)} \tag{4.12}$$

The notation of $\mathbf{x}^{(n)}$ indicates the $n$-fold outer product of the vector $\mathbf{x}$, the $\mu_p$ is the mass of particle $p$ and $z_B$ represents the center of mass of the node $B$. When calculating a multipole moment up to order $p$, then each moment where $1 \leq n \leq p$ must be calculated as well. This results in a collection of tensors of increasing rank. There is also the inclusion of the scalar total-mass term $m$, which can be interpreted as the zeroth-order multipole. The shape of a quadrupole moment, where $p = 2$, can be seen in eq. (4.13).

$$m, \quad \langle m_x, m_y, m_z \rangle, \quad \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} \\ . & m_{yy} & m_{yz} \\ . & . & m_{zz} \end{bmatrix} \tag{4.13}$$

Blank entries in the rank 2 tensor of the quadrupole moment represent redundant data. These values do not need to be stored since all tensors in a multipole moment are symmetric. As the rank of the tensor increases, so does the amount of memory saved by not including these unnecessary entries. While a rank 2 tensor only saves 3 entries, a tensor of rank 3 already saves 17 entries by using this symmetric system. This advantage can be seen more clearly in eq. (4.14).

$$\left[ \begin{bmatrix} m_{xxx} & m_{xxy} & m_{xxz} \\ . & m_{xyy} & m_{xyz} \\ . & . & m_{xzz} \end{bmatrix} \begin{bmatrix} . & . & . \\ . & m_{yyy} & m_{yyz} \\ . & . & m_{yzz} \end{bmatrix} \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & m_{zzz} \end{bmatrix} \right] \tag{4.14}$$

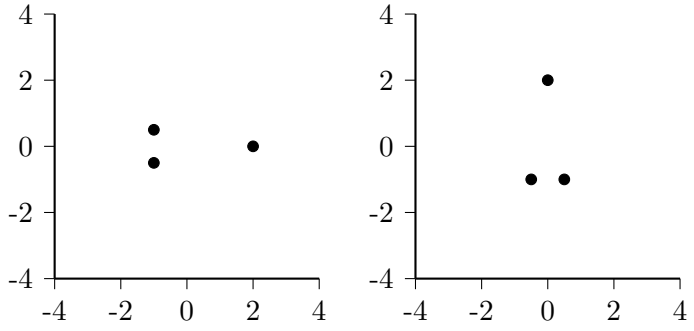
whether such a symmetric setup is beneficial depends on how much memory is saved by not storing redundant data and how much the computation time increases for a symmetric lookup.

Creating a multipole moment for a node, can be done with eq. (4.12). When a node is not a leaf, its multipole moment can be computed more efficiently by querying its children. Rather than iterating over all particles contained in the node, each child node can be treated as a particle. Additionally, the multipole

moments of the children must also be added to the parent. This procedure is formalized in eq. (4.15).

$$\mathbf{M}_B^n = \sum_{C \in \text{children}(B)} \mu_C (\mathbf{x}_C - z_B)^{(n)} + \mathbf{M}_C^n \tag{4.15}$$

In order to demonstrate what these multipole moments actually represent, a two dimensional scenario is shown in fig. 4.3 and their respective multipoles in eq. (4.16).




(a) Horizontal scenario. (b) Vertical scenario.

Figure 4.3: Two different configurations of points where one is horizontally distributed and one vertically.

$$\{3, \quad \langle 0,0 \rangle, \quad \begin{bmatrix} 6 & 0 \\ . & 0.5 \end{bmatrix}\} \qquad \{3, \quad \langle 0,0 \rangle, \quad \begin{bmatrix} 0.5 & 0 \\ . & 6 \end{bmatrix}\} \tag{4.16}$$

The quadrupole moments in the two scenarios capture information about the shape of the node. The $m_{xx}$ entry in the rank-2 tensor of scenario A is larger than that of scenario B, indicating that scenario A exhibits a stronger horizontal spread. A similar interpretation applies to the values of $m_{yy}$, which reflect the vertical spread of the distribution. The $m_{xy}$ term describes how closely the distribution aligns with a diagonal orientation. Another notable feature is that the dipole moment, represented by the rank-1 tensor, contains only zeros in both scenarios. This is a consequence of placing the expansion point at the center of mass.

Equation (4.17) provides a Particle-Node interaction expressed in terms of multipole moments. The acceleration experienced by $A$ due to $B$ is denoted by $\mathbf{C}_{B \to A}^{1,p}$, where $p$ indicates the expansion order.

$$\mathbf{C}_{B \to A}^{1,p} = \sum_{n=0}^{p-1} \frac{(-1)^n}{n!} \nabla^{(n+1)} g(\mathbf{R}) \odot \mathbf{M}_B^n \tag{4.17}$$

Here $\odot$ denotes a tensor inner product, $p$ the expansion order and $g(\mathbf{R})$ the Green's function, which for gravity is $1/\|\mathbf{R}\|$). Working out eq. (4.17) for a third order expansion ($p = 3$), yields eq. (4.18) which can be read using Einstein's sum convention.

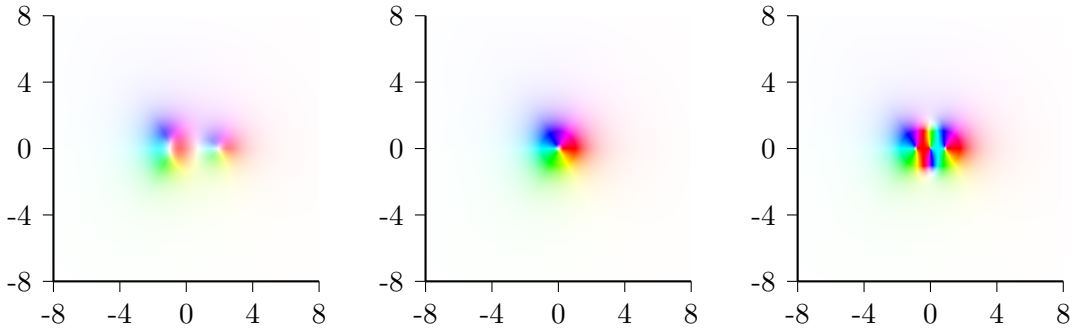
$$\mathbf{C}^1_{B\to A,i} = \mathbf{M}^0_B(\mathbf{R}_i(D^0 + \tfrac{1}{2}\tilde{\mathbf{M}}^2_{Bii}D^1 + \tfrac{1}{2}\mathbf{R}_i\mathbf{R}_j\tilde{\mathbf{M}}^2_{Bij}D^2)) \tag{4.18}$$

Here $\tilde{\mathbf{M}}^2_B \equiv \mathbf{M}^2_B/\mathbf{M}^0_B$ and $D^m$, which are the derivatives from the Taylor expansion, can be calculated using eq. (4.19).

$$D^m \equiv \left(\frac{1}{r}\frac{\partial}{\partial r}\right)^m g(r)\bigg|_{r=\|\mathbf{R}\|} \tag{4.19}$$

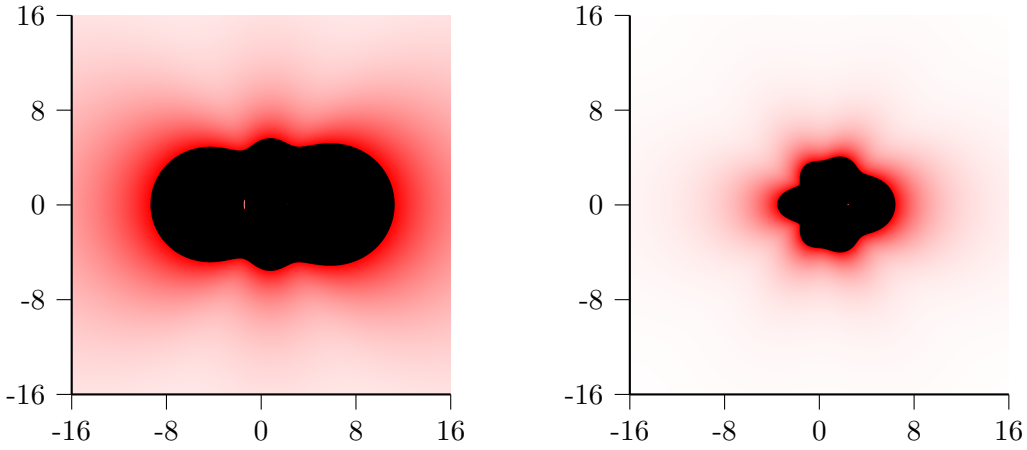
### Multipole Method

Replacing the Particle–Node interactions of the BH method with the interaction defined in eq. (4.18) results in a similar but more accurate approach known as the multipole method (MM). The effect of this modified interaction on the generated field for the horizontal scenario is illustrated in fig. 4.4. The relative errors produced by the BH and the MM are shown in fig. 4.5.




(a) Naive. (b) BH. (c) MM.

Figure 4.4: Visualization of the unnormalized repulsive component of the t-SNE derivative generated by the naive, BH and MM. saturation is determined by the radial distance of the field vector while the hue is determined by the polar angle.




(a) BH. (b) MM.

Figure 4.5: Visualization of the relative error of the field generated by the BH and MM. Saturation determines the strength of the error up to 0.1, above which the error is displayed as black.

### 4.4.3 Multipole Fields

A multipole field, consists of a collection of tensors of increasing rank. Its function is to give information about how the accumulated acceleration changes in certain directions within a node. In order to illustrate this, the shape of a multipole field that is required for an expansion order of 1 is shown in eq. (4.20).

$$\langle a_x, a_y, a_z \rangle, \quad \begin{bmatrix} \frac{da_x}{dx} & \frac{da_x}{dy} & \frac{da_x}{dz} \\ . & \frac{da_y}{dy} & \frac{da_y}{dz} \\ . & . & \frac{da_z}{dz} \end{bmatrix} \tag{4.20}$$

Here the first element represents the acceleration at the node's center and the second element represents how this acceleration changes with respect to the position in the node. As the expansion order increases, so do the number of tensors and their rank in our multipole fields thus allowing them to store more information. The multipole fields are symmetric. which comes from the fact that $\frac{da_i}{dj}$ will always be equal to its inverse.

Every Node-Node interaction generates multipole fields which have to be accumulated at the sink node. The lowest rank tensor of the multipole field is equal to the acceleration and can be calculated using eq. (4.18). The other tensors require a different equation. Generalizing for the field order yields eq. (4.21). Setting $m = 1$ results in the same equation as eq. (4.18) and is the reason why $\mathbf{C}^{1,p}_{B\to A}$ was labeled as the acceleration. There also exists a $\mathbf{C}^{0,p}_{B\to A}$

term and it is equal to the potential.

$$\mathbf{C}^{m,p}_{B\to A} = \sum_{n=0}^{p-m} \frac{(-1)^n}{n!} \nabla^{(n+m)} g(\mathbf{R}) \odot \mathbf{M}^n_B \tag{4.21}$$

Node-Node interactions using quadrupole moments, need to store $\mathbf{C}^1_{B\to A}$, $\mathbf{C}^2_{B\to A}$ and $\mathbf{C}^3_{B\to A}$. $\mathbf{C}^1_{B\to A}$ can be found in eq. (4.18). Working out $\mathbf{C}^2_{B\to A}$ and $\mathbf{C}^3_{B\to A}$ results in eq. (4.22) and eq. (4.23). Higher order field equations can be worked out using eq. (4.21) but will not be listed for $p > 3$.

$$\mathbf{C}^2_{B\to A,ij} = \mathbf{M}^0_B(\delta_{ij} D^1 + \mathbf{R}_i \mathbf{R}_j D^2) \tag{4.22}$$

$$\mathbf{C}^3_{B\to A,ijk} = \mathbf{M}^0_B((\delta_{ij}\mathbf{R}_k + \delta_{jk}\mathbf{R}_i + \delta_{ki}\mathbf{R}_j) D^2 + \mathbf{R}_i \mathbf{R}_j \mathbf{R}_k D^3) \tag{4.23}$$

Once all the multipole fields have collected their coefficients from all of their interactions, where $\mathbf{C}^m_A = \sum_B \mathbf{C}^m_{B\to A}$, can the process of passing down the fields of each node to their children begin. This process is started form the root node where the multipole fields are recentered to their child nodes or particles, and ends when all the particles have received their multipole fields from their parents. A multipole field can be recentered using eq. (4.24).

$$\mathbf{C}^{m,p}_{\text{new}} = \sum_{n=0}^{p-m} \frac{1}{n!} (z_{\text{old}} - z_{\text{new}})^{(n)} \odot \mathbf{C}^{m+n,p}_{\text{old}} \tag{4.24}$$

Once all the particles have collected their multipole fields, the $\mathbf{C}^1_{B\to A}$ can be added to the value that was computed from Particle-Particle interactions to obtain the final acceleration.

#### Reverse Multipole Method

A reverse multipole method (rMM) can be created by replacing the Node-Particle interactions of the rBH method with the multipole interactions. Each node will track their multipole fields and pass down their values. The complete algorithm for the rMM can be found in algorithm 9.

### 4.4.4 Multipole Descent Criterion

There are multiple different descent criterion that can be used for the FMM. Equation (3.13) from the BH method makes for a suboptimal choice since it does not take the size of both nodes into account. Equation (4.25) is a more suitable choice since it does consider the sizes of both nodes. This new criterion

will result in nodes seeing each other under a smaller effective angle for the same $\theta$ compared to the descent criterion of the BH method [10].

$$\frac{S_a + S_B}{D} < \theta \tag{4.25}$$

In order to create a coherent overview of all the equations, a complete algorithm of the FMM can be found in algorithm 11.

### 4.4.5 Symmetric Interactions

The FMM as explained in the previous sections, has treated interactions asymmetrically, i.e., with a single sink and source component. by treating the interactions of the FMM in a symmetric way can the computations of the field moment interactions partly be reused, which saves computational effort. There are three aspects of the method that must be changed to accommodate symmetric interactions.

When computing a mutual interaction between two nodes, node $A$ and node $B$, it is possible to reuse the multipole fields of $\mathbf{C}^2$ and $\mathbf{C}^3$ when working with an expansion order of $p = 3$. This can be done because $\mathbf{M}_A^0\mathbf{C}_A^2 = \mathbf{M}_B^0\mathbf{C}_B^2$ and $\mathbf{M}_A^0\mathbf{C}_A^3 = -\mathbf{M}_B^0\mathbf{C}_B^3$. This fact follows directly from eq. (4.22) and eq. (4.22). To exploit this symmetry, $\mathbf{C}_A^n$ is not stored directly. Instead, $\tilde{\mathbf{C}}_A^n = \mathbf{M}_A^0\mathbf{C}_A^n$ is used when $2 \leq n \leq 3$. This approach reduces computational effort and ensures conservation of moments, with the sum of all forces vanishing to within machine precision [17]. After all interactions are calculated, the fields have to be divided by their mass in order to undo the mass multiplication necessary for the symmetrical interactions.

Since the symmetrical interactions now calculate the forces for both particles or nodes, the tree traversal has to be altered. The symmetric traversal algorithm can be found in algorithm 2. Algorithm 2 contains the normalization variable found in t-SNE which is not present in conventional N-body problems.

**Algorithm 2:** A recursive symmetric fast multipole method Node-Node tree traversal for t-SNE.

```
function symFMM_NN_TRAVERSE(norm, N_A, N_B)
    if DESCENT_CRITERION(N_A, N_B, θ) then
        N_A.C, N_B.C += SYM_REP_TSNE(N_A, N_B) using eq. (4.9)
        norm += 2 · (N_A.M_0 + N_B.M_0) · NORM_TSNE(N_A, N_B)
    else if IS_LEAF(N_A) then
        if N_A == N_B then
            foreach unique: {p_A, p_B} ∈ N_A.points × N_B.points do
                p_A.acc, p_B.acc += SYM_REP_TSNE(p_A, p_B)
                norm += 2 · NORM_TSNE(N_A, N_B)
        else
            foreach p_A ∈ N_A.particles do
                symFMM_PN_TRAVERSE(norm, p_A, N_B)
    else if IS_LEAF(N_B) then
        if N_A == N_B then
            foreach unique: {p_A, p_B} ∈ N_A.points × N_B.points do
                p_A.acc, p_B.acc += SYM_REP_TSNE(p_A, p_B)
                norm += 2 · NORM_TSNE(N_A, N_B)
        else
            foreach p_B ∈ N_B.particles do
                symFMM_PN_TRAVERSE(norm, p_B, N_A)
    else
        if N_A == N_B then
            foreach unique: {N_a, N_b} ∈ N_A.children × N_B.children do
                symFMM_NN_TRAVERSE(norm, N_a, N_b)
        else
            foreach N_a ∈ N_A.children do
                foreach N_b ∈ N_B.children do
                    symFMM_NN_TRAVERSE(norm, N_a, N_b)
```

**Algorithm 3:** A recursive symmetric fast multipole method Point-Node tree traversal for t-SNE.

```
function symFMM_PN_TRAVERSE(norm, p, N)
    if DESCENT_CRITERION(p, N, θ) then
        p.C, N.C += SYM_REP_TSNE(p, N) using eq. (4.9)
        norm += 2 · N.M_0 · NORM_TSNE(, N)
    else if IS_LEAF(N) then
        foreach p_child ∈ N.points do
            p.acc, p_child.acc += SYM_REP_TSNE(p_A, p_B)
            norm += 2 · NORM_TSNE(p, p_child)
    else
        foreach N_child ∈ N.children do
            symFMM_PN_TRAVERSE(norm, p, N_child)
```

## 4.5 FMM Specific Changes

Integrating the FMM with t-SNE, presents a few changes that have to be addressed. This section will cover these FMM specific changes.

### 4.5.1 T-SNE Derivatives

The FMM requires derivatives in order to perform the required Taylor expansions. Since t-SNE uses a different equation for the "forces" than gravity, the derivatives have to be recalculated. Equation (4.19) can be used to work out the needed derivatives. A problem is that the potential generated by the repulsive component for t-SNE is not known. What is known is the potential for gravity and its 3 derivatives, which are shown in eq. (4.26).

$$D^0 = \frac{1}{r}, \quad D^1 = -\frac{1}{r^3}, \quad D^2 = \frac{3}{r^5}, \quad D^3 = -\frac{15}{r^7} \tag{4.26}$$

From eq. (4.9), it follows that $D^1 = \frac{1}{(1+r^2)^2}$ in the case of t-SNE. When interpreted through the analogy with gravitational dynamics, this corresponds to the acceleration term.

Unlike gravitational systems, however, this expression is not derived from a physical potential. Instead, it originates from the Student's t-distribution used in t-SNE to model similarities in the embedded space. In particular, t-SNE employs a heavy-tailed kernel of the form $(1 + r^2)^{-1}$, which corresponds to a Student's t-distribution with one degree of freedom (i.e., a Cauchy distribution).

The repulsive term in the gradient is obtained by differentiating this kernel and combining it with the normalization term $Z$.

Although no physical potential is defined in the original formulation of t-SNE, it is possible to introduce a scalar function whose gradient reproduces the required output. This "pseudo-potential" should therefore be interpreted purely as a mathematical construct that enables the use of multipole expansions, rather than as a quantity with physical meaning. By applying eq. (4.19) in reverse, such a potential can be constructed. Evaluating the first three derivatives of this potential yields the expressions shown in eq. (4.27).

$$D^0 = \frac{-0.5}{1+r^2}, \quad D^1 = \frac{1}{(1+r^2)^2}, \quad D^2 = \frac{-4}{(1+r^2)^3}, \quad D^3 = \frac{24}{(1+r^2)^4} \tag{4.27}$$

### 4.5.2 Z Accumulation

As described in [11], accumulation of the normalization value $Z$ is performed by delaying the computation of the final derivative and instead evaluating an unnormalized version. For each interaction, the value $(1+\|\mathbf{y}_k-\mathbf{y}_l\|^2)^{-1}$ is added to $Z$. In the case of a Point–Node interaction, where the point acts as the sink and the node as the source, this value is multiplied by the mass of the node.

When transitioning to the FMM formulation, two modifications arise. First, in addition to Point–Point and Point–Node interactions, Node–Point and Node–Node interactions must also be considered. Second, multipole expansions are introduced in order to improve the accuracy of the interaction approximations. To account for the first modification, the $Z$ contribution is multiplied by the product of the masses of the two interacting entities. For Node–Point and Point–Node interactions, this reduces to multiplication by the mass of the node. The second modification is handled by applying the derivatives from eq. (4.27) within the formulation given in eq. (4.21).

## 4.6 Varying $\theta$ Parameter

Previous sections established that strict numerical accuracy is not required for t-SNE. The primary objective is to obtain an embedding that reveals natural cluster structure in the data, which is achieved by minimizing the associated cost function. An important observation is that t-SNE does not require a uniform level of accuracy throughout the optimization process. An approach that begins with lower accuracy and gradually increases it can therefore be acceptable, provided that the final embedding achieves a sufficiently low cost. Based on this observation, a modification to BH–like$^\dagger$ methods is proposed in which the parameter $\theta$ is adjusted at each iteration.

Ideally, the value of $\theta$ should start relatively large and gradually decrease toward a lower bound. The specific function used to control this behavior is given in eq. (4.28).

$$\theta = l + (0.5 + 0.5 \cdot \cos(\pi \cdot \min(t/999, 1))) \cdot (h - l) \tag{4.28}$$

Here $t$ is the iteration counter and $l$ and $u$ are the lower and upper value of $\theta$ respectively. A visualization of this function can be found in fig. 4.6.

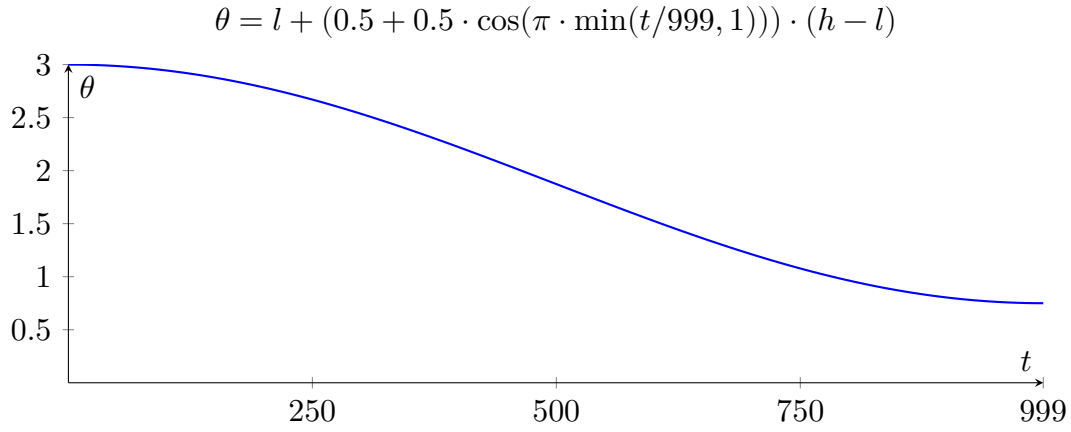



Figure 4.6: Evolution of $\theta$ as a function of $t$ with $l = 0.75$ and $u = 3$.

## 4.7 Details

In this section, the main design choices underlying the construction of the FMM-accelerated t-SNE algorithm are summarized, and the components introduced throughout the chapter are consolidated into a coherent framework.

First, the multipole expansion is restricted to quadrupole order. Although higher-order multipoles could, in principle, improve accuracy, their additional computational and memory costs are unlikely to provide meaningful benefits for t-SNE. Second, the "maximum number of points per node" ($n_{max}^{node}$) is limited to 16. Empirical evaluation indicates that this value performs well across BH–like$^{\dagger}$ methods. Smaller node sizes increase tree depth and traversal overhead, whereas larger capacities lead to an excessive number of Point–Point interactions. The related "tree depth" hyperparameter is determined using eq. (5.1), which allows the Z-order curve tree construction to perform effectively. Third, the best-performing variant of the algorithm employs symmetric interactions. Fourth, although multipole moments and fields are inherently symmetric tensors, this symmetry is not exploited in storage or computation. In the two-dimensional setting considered here, the memory savings obtained by omitting

redundant tensor entries are relatively small. Consequently, the implementation stores the full tensors. Finally, the complete algorithm integrates all previously discussed components: the reformulated t-SNE gradient, FMM-based approximation of the repulsive forces, accumulation of the normalization constant $Z$, symmetric interactions, adaptive descent based on the FMM criterion, and a time-varying $\theta$ parameter. The complete algorithm is presented in algorithm 4.

**Algorithm 4:** FMM accelerated t-Distributed Stochastic Neighbor Embedding.

**Data:** dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$,
perplexity $\mathcal{P}$,
iteration amount $\mathcal{T}$,
learning rate $\eta$,
momentum $\alpha$,
accuracy parameter $\theta$

**Result:** low dimensional data $\mathcal{Y}^{(t)}$

**Algorithm:**

compute $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ using eq. (3.5) with $\mathcal{P}$

compute $p_{i|j}$ using eq. (3.3) with $\Sigma$

set $p_{ij} = \dfrac{p_{j|i} + p_{i|j}}{2n}$

initialize $\mathcal{Y}^{(0)}$ with $\mathcal{N}(0,\, 10^{-4}I)$

**for** $t = 1$ **to** $\mathcal{T}$ **do**

- $\forall \mathbf{x} \in \mathcal{X}, \quad \mathbf{x}.deriv = \langle 0, 0 \rangle$
- $norm = 0$
- create tree structure with nodes $\mathcal{N} = \{\mathbf{N}_1, \mathbf{N}_2, \dots, \mathbf{N}_N\}$
- calculate $(z, \mathbf{M})$ **ForEach** node using eq. (4.12)
- symFMM_NN_TRAVERSE($norm, \mathbf{N}_{root}, \mathbf{N}_{root}$) using algorithm 2
- $\forall \mathbf{N} \in \mathcal{N}$ translate and sum multipole fields to children using eq. (4.24)
- $\forall \mathbf{N} \in LEAF(\mathcal{N})$ translate and sum $\mathbf{N}.\mathbf{C^0}$ to $\mathbf{N}.points$
- $\forall \mathbf{x} \in \mathcal{X}, \quad \mathbf{x}.deriv = \mathbf{x}.deriv/norm$
- $\dfrac{\partial C}{\partial \mathcal{Y}_i} = 4(F_{attr,i} - \mathbf{x}_i.deriv)$ using eq. (4.10)
- set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \dfrac{\partial C}{\partial \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{t-2})$

# Chapter 5

# Experiments

This chapter evaluates the performance of several N-body approximation methods in the context of t-SNE. The datasets, parameter settings, and evaluation metrics used throughout the experiments are first introduced. Subsequently, a series of experiments investigates suitable hyperparameter settings and compares the computational efficiency and approximation accuracy of the considered methods.

## 5.1 Experimental Setup

### 5.1.1 Datasets

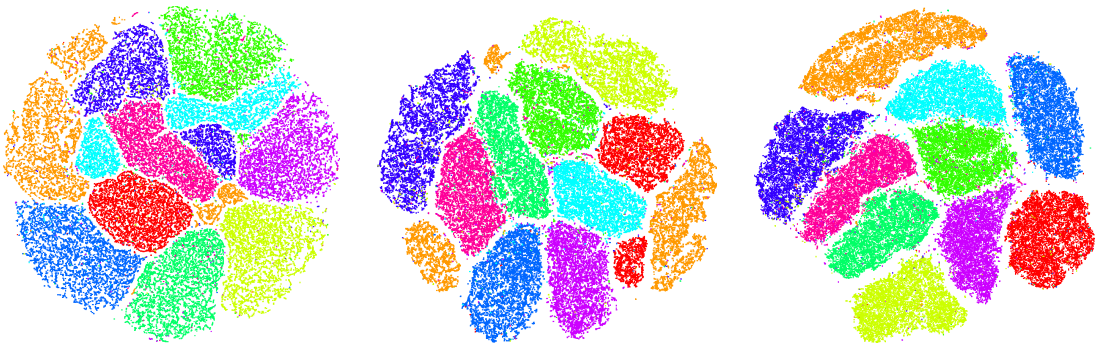
Three datasets were used in the experiments. The first two datasets are the "digits MNIST" dataset [18] and the "fashion MNIST" dataset [19]. Both contain 70 000 samples and are used for the majority of the experiments.

The third dataset is the "1.3 Million Brain Cells from E18 Mice" dataset [20]. Due to its significantly larger size, explicit error calculations become prohibitively expensive. Consequently, this dataset is only used in experiments that do not require direct error evaluation.

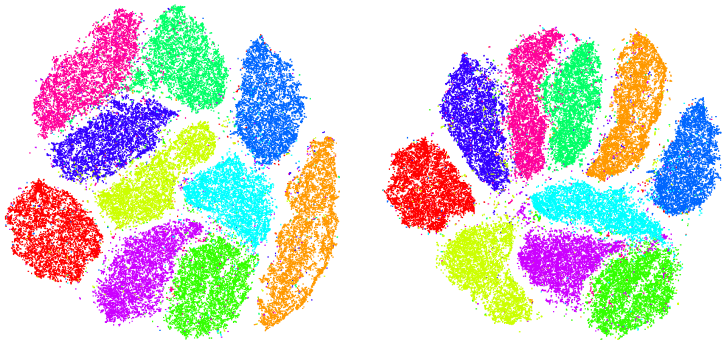
### 5.1.2 Parameter Settings

The perplexity parameter influences the amount of non-zero entries of the sparse $P$ matrix. As this amount increases, the computational cost increases accordingly. However, the perplexity value is not chosen solely based on computational efficiency, as different values reveal structural patterns in the data at different scales. The influence of the perplexity parameter on cluster formation is illustrated using the "digits MNIST" dataset in fig. 5.1. As the perplexity value increases, each data point maintains a larger effective neighborhood. This strengthens longer-range attractive bonds and reduces the risk of fragmenting clusters into sub-components. When this value is set too high however, it can lead to incorrect cluster merging. Figures 5.1a and 5.1b exhibit fragmented clustering structures, which is consistent across both the "digits MNIST" and "fashion MNIST" datasets. Perplexity values of 30, 50 and 100 produce the expected number of identifiable clusters for the "digits MNIST" dataset. Consequently, a

perplexity of 30 is used in subsequent experiments unless stated otherwise. The results are expected to generalize to other perplexity values, as the computation of the repulsive component of the derivative is independent of this parameter.



(a) Perplexity: 5 (b) Perplexity: 15 (c) Perplexity: 30



(d) Perplexity: 50 (e) Perplexity: 100

Figure 5.1: t-SNE embeddings of the "digits MNIST" dataset obtained using the BH method with $\theta = 0.5$ for different perplexity values.

All BH–like$^{\dagger}$ methods use a maximum of 16 points per tree node, which was found empirically, in section 5.2.1, to work well for all the BH-like$^{\dagger}$ methods. Experiments that vary the accuracy parameter evaluate values in the range $0.5 \leq \theta \leq 3.0$ as values below or above will resulted in either too much error or prohibitively long runtimes.

### 5.1.3 Evaluation Metrics

Approximation accuracy is measured using the mean relative error (MRE) of the repulsive component of the t-SNE gradient. The attractive component is identical across all evaluated methods and therefore does not contribute to differences in approximation accuracy. During error evaluation, iteration steps are computed using the full derivative of the naive method.

Many experiments report results using plots with the mean relative error on the y-axis and the average time per iteration on the x-axis. Both quantities are averaged over 1000 t-SNE iterations. Each data point corresponds to a complete t-SNE run with a specific value of the parameter that is to be tested. Multiple runs with varying values for the parameter illustrate how runtime and approximation error change as the parameter increases. Throughout the rest of the chapter we shall refer to this type of plot as the "average error/time" plot.

The term "acceptable error" refers to the observation in [11] that the BH method produces sufficiently accurate results when $\theta = 0.5$.

## 5.2 Method Experiments

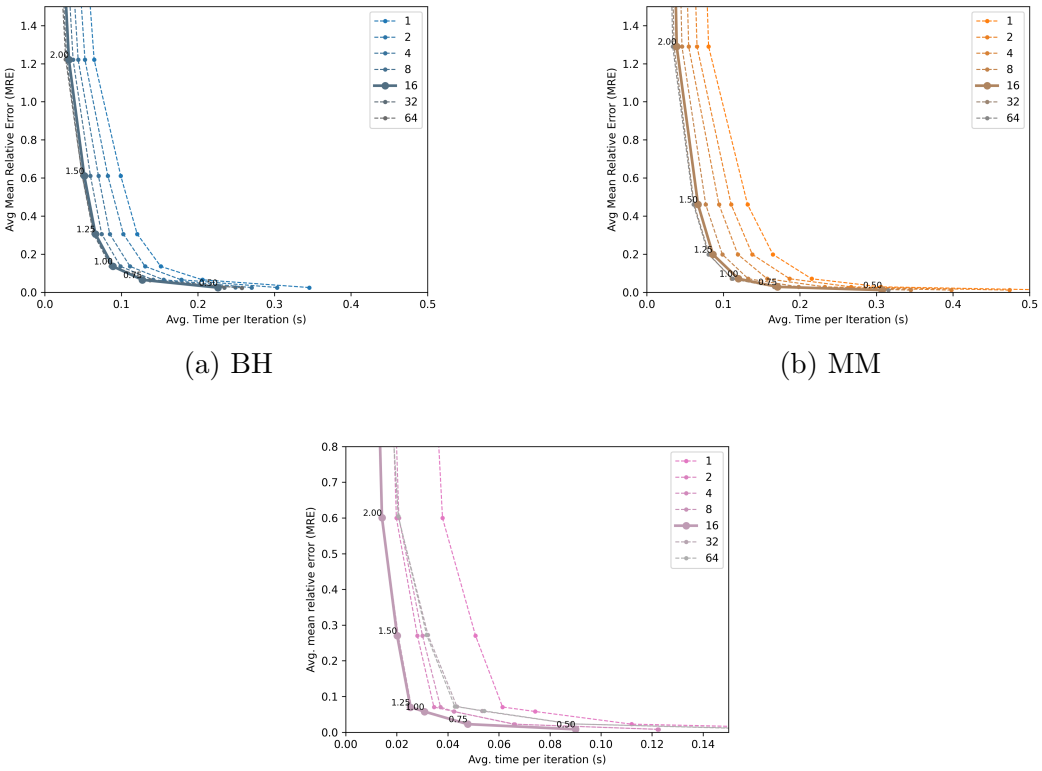
### 5.2.1 Maximum Points Per Node / Tree Depth

The BH method subdivides nodes until each leaf node contains a single point. Although this strategy produces accurate approximations, additional performance improvements can be achieved by terminating subdivision earlier. Node subdivision can therefore stop once the number of contained points falls below a specified threshold.

All BH–like† methods except the FMM use this parameter directly. The FMM instead constructs the tree using an $O(n)$ algorithm based on Morton codes. This approach replaces the "maximum number of points per node" ($n_{max}^{node}$) parameter with a fixed tree depth. Assuming a roughly uniform distribution of points within a circular domain, the tree depth can be estimated from the number of points $N$ and $n_{max}^{node}$, using eq. (5.1). In this experiment, the BH method, MM and sFMM were evaluated using the "average error/time" plot with $n_{max}^{node}$ as the tested value. Since the values for these hyperparameters can influence the speed and accuracy of the method, it constitutes as the first experiment.

$$\text{tree depth} = \left\lceil \log_4 \left( \frac{1}{\pi/4} \frac{N}{n_{max}^{node}} \right) \right\rceil \tag{5.1}$$

From fig. 5.2, the configuration with a single point per node consistently produces the worst performance. This behavior is expected since it will result in the maximum possible tree depth and number of nodes. This increases traversal and tree creation time without improving approximation accuracy in a meaningful way. As $n_{max}^{node}$ increases, subdivision terminates earlier, reducing tree size and therefore computational cost. The results show a consistent improvement in runtime until the optimal value is reached.




(a) BH

(b) MM

(c) sFMM

Figure 5.2: Evaluation of the $n_{max}^{node}$ hyperparameter. The bold curve corresponds to a value of 16, which was found to be optimal.

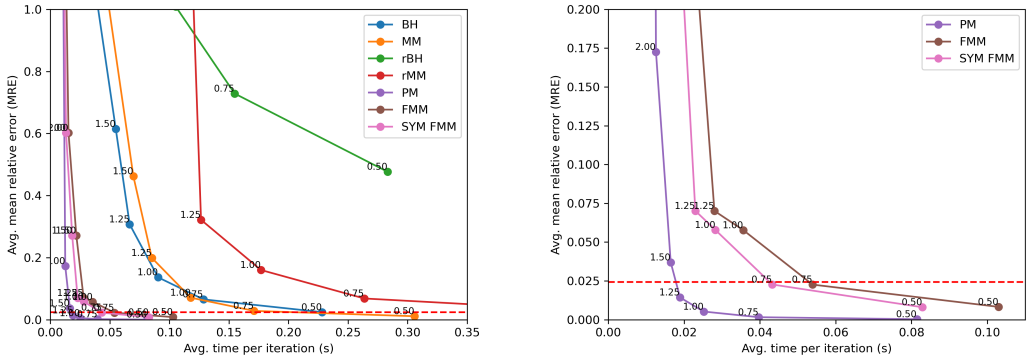
Figure 5.2 shows that the BH method and MM perform best when $n_{max}^{node} = 16$ or $n_{max}^{node} = 32$, while the sFMM performs best when $n_{max}^{node} = 8$ or $n_{max}^{node} = 16$. What is notable in fig. 5.2c is that $n_{max}^{node} = 8$ and $n_{max}^{node} = 16$ result in similar performance. Other values also perform similar to each other. This behavior can be explained by the discontinuous nature of the tree depth parameter. Since the tree depth must be a positive integer, different values for $n_{max}^{node}$ may map to the same effective depth. In particular, both $n_{max}^{node} = 8$ and $n_{max}^{node} = 16$ result in a tree depth of 7, and therefore produce identical hierarchical decompositions. Consequently, their performance is indistinguishable.

Across all tested BH-like† methods, a value of 16 for $n_{max}^{node}$ provides the best trade-off between computational efficiency and approximation accuracy. Importantly, the close agreement between BH, MM, and sFMM indicates that the inclusion of multipole expansions does not significantly alter the optimal choice of this hyperparameter. Thus, the structural properties of the tree

dominate this behavior rather than the specific interaction scheme used.

### 5.2.2 Computation Time Versus Error

The second experiment compares the computational efficiency of the methods by analyzing the trade off between runtime and approximation error across different accuracy parameters. This is done by using the "average error/time" plot with the accuracy parameter as the tested value. As this experiment directly evaluates the methods without alterations to their algorithm, it is performed as one of the first experiments.




(a) all method

(b) top 3 methods

Figure 5.3: N-body solver effectiveness under varying $\theta$ parameter. The horizontal red line represents the "acceptable error".

The Pareto analysis in fig. 5.3 reveals several structural insights.

First, the reverse methods (rBH and rMM) perform substantially worse than their non-reverse counterparts. This can likely be attributed to the inherent inaccuracies of Node–Point interactions. While rMM partially mitigates this through multipole interactions, the underlying approximation remains less accurate than Point–Node interactions. As a result, these methods accumulate larger errors.

Second, Both the FMM and the sFMM Method outperform all other BH–like$^{\dagger}$ methods across the tested accuracy range. The sFMM consistently achieves lower errors for smaller runtimes than the standard FMM. The reason that the FMM outperforms the other BH-like$^{\dagger}$ methods is most likely due to the Node-Node interactions reducing the computational costs compared to Point-Node interactions, while maintaining accuracy with multipoles. This reduction in computation cost is more pronounced in the sFMM, which can be explained by omitted calculations in symmetric interactions.

Third, although prior literature suggests that the FMM should outperform the PM in high-accuracy scenarios [10], our results do not confirm this. The PM consistently dominates the Pareto frontier. Several explanations are plausible:

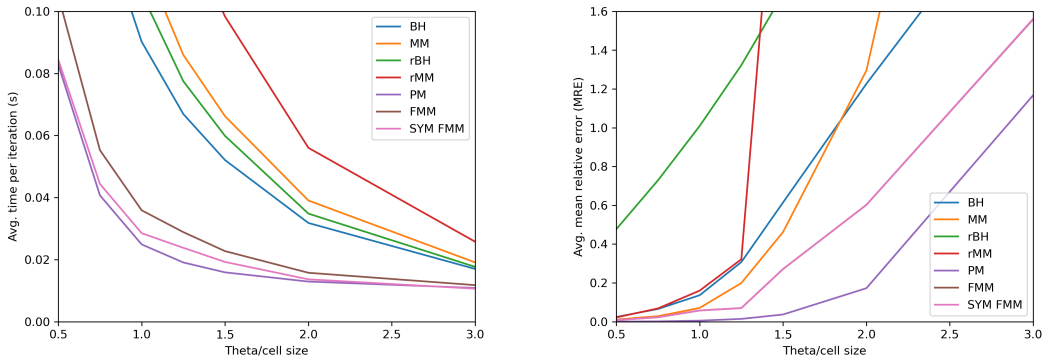
- **Error structure:** The PM primarily exhibits error in close-range interactions. Since t-SNE actively pushes nearby points apart during optimization, such close interactions occur less frequently as the embedding evolves. Consequently, the effective PM error in practice is lower than expected.
- **Dimensionality advantage:** In the two dimensional setting of t-SNE, the PM constructs a grid over a square rather than a cube, reducing cell count and improving efficiency.
- **Sub-optimal Z accumulation:** The Z accumulation scheme presented in this paper is an approximation of the true normalization. This approximation could be responsible for creating derivatives that are normalized incorrectly.
- **Implementation maturity:** The PM implementation has undergone long-term optimization and refinement, potentially giving it an engineering advantage.

These factors likely explain why the PM maintains superior empirical performance despite theoretical expectations favoring FMM in certain scenarios.

### 5.2.3 $\theta$ Effect

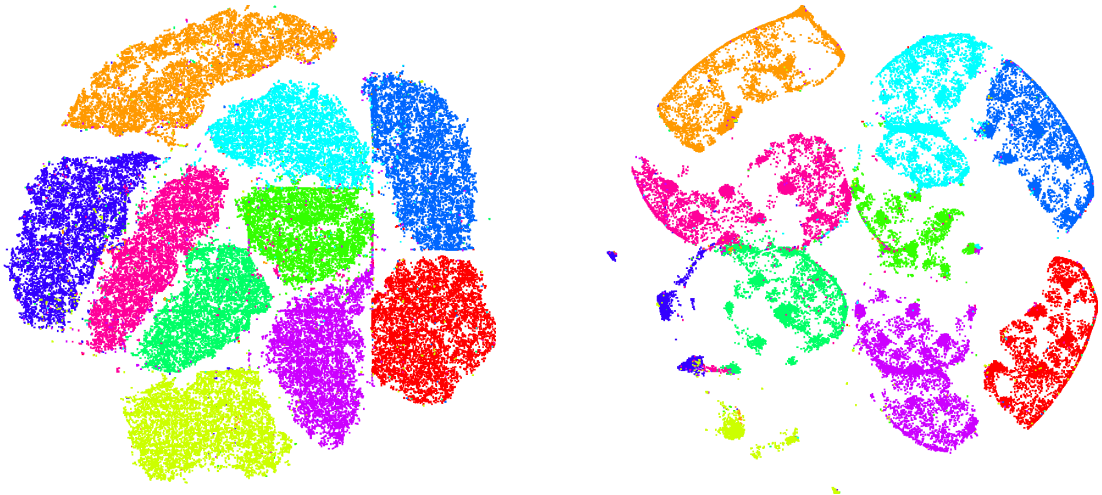
This experiment evaluates how the accuracy parameter $\theta$ influences runtime, approximation accuracy and embedding quality. The MRE and the "time per iteration" are measured independently for each method and averaged over 1000 iterations. The final embeddings for the sFMM and BH using a high $\theta$ are shown in order to see the artifacts they produce. Since the choice of an appropriate value for the accuracy parameter can affect the subsequent experiment, this experiment must be conducted beforehand.

Increasing the accuracy parameter in fig. 5.4a results in a monotonic decrease in "time per iteration" accompanied by increasing approximation error in fig. 5.4b. For the FMM, sFMM, and PM, the decrease in runtime become negligible for $\theta$ values larger than approximately 1.5. Beyond this point the runtime curves flatten while the approximation error increases rapidly. Values larger than $\theta = 3$ are expected to yield time savings small enough compared to the expected error to not be a viable option. The BH, rBH, MM and rMM do still show significant runtime reductions even when $\theta = 3$.




(a) Avg. time per iteration (S). (b) Avg. MRE.

Figure 5.4: Affect of accuracy parameter on time and accuracy on N-body solvers.



(a) sFMM (b) BH

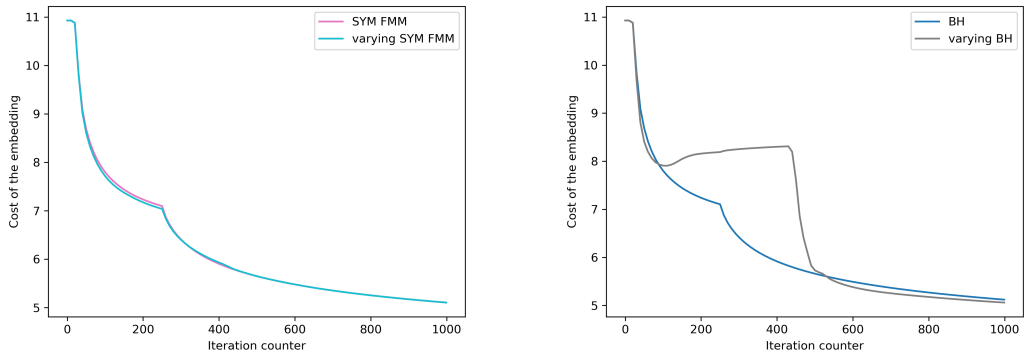
Figure 5.5: Final Embedding using the sFMM and BH method with $\theta = 2$.

Visual inspection of the embeddings in fig. 5.5 reveals that the BH method produces noticeable artifacts at $\theta = 2$, whereas the sFMM produces substantially cleaner embeddings for the same value. The artifacts in fig. 5.5a exhibit square-like patterns. This is most likely explained by the bounding boxes of the underlying tree structure, which are axis-aligned. When approximation becomes too aggressive, these structural boundaries accumulate or otherwise distort points on their edges. In contrast, the BH artifacts in fig. 5.5b form rounded arches. This may arise from the descent criterion allowing multiple points to accept interactions from a single large node thus resulting in circular artifacts. The reason for why the sFMM remains more visually stable under large $\theta$ values is most likely due to the multipole Node-Node interactions producing smoother derivatives. Even if not fully accurate, these derivatives preserve sufficient structural coherence to prevent severe geometric distortions.

### 5.2.4 Varying $\theta$

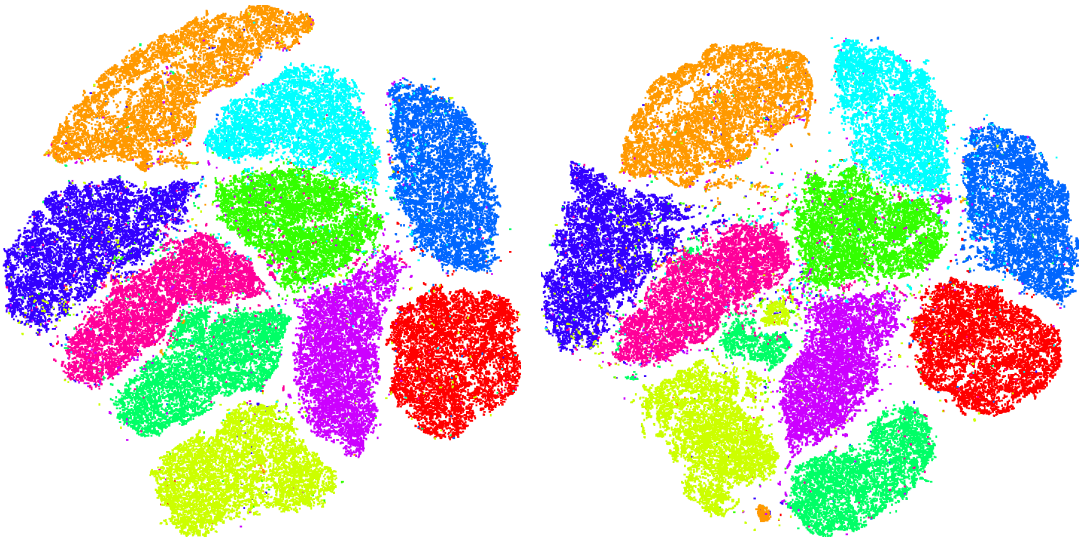
This experiment examines the effect of varying $\theta$ on computational cost, embedding quality, and the "average cost/time". The first plot illustrates the evolution of the embedding cost throughout the t-SNE optimization process for both the sFMM and BH methods. This is followed by a visualization of the final embeddings obtained using t-SNE with varying sFMM (v-sFMM) and varying BH (v-BH). The final plot uses the "average error/time" plot in which the error metric is replaced by the embedding cost, with the varying accuracy parameter as the tested value. Although both methods are initialized with $\theta = 3$, only the target (reduced) values of $\theta$ are displayed. As varying $\theta$ constitutes a non-standard modification to BH-like$^{\dagger}$ methods, it is examined last to assess its potential impact on computational efficiency.

The cost trajectory of the v-sFMM in fig. 5.6 closely follows the sFMM. The v-BH method shows a noticeable deviation from the BH method, where The embedding cost stagnates between approximately iterations 150 and 500 before decreasing again during later iterations. Despite the strong approximations used in the early iterations of the v-sFMM and the v-BH, they converge to slightly lower final embedding costs than their non-varying counterparts. This behavior is unexpected since we are allowing more error. The reason why the embedding cost of the sFMM follows a more stabile decreasing line for a high $\theta$ value is likely due to its use of multipole Node–Node interactions. Even at high $\theta$ values, interactions between large nodes yield smooth repulsive derivatives that maintain coherent point spacing. While these derivatives may lack precision for a high $\theta$ value, they preserve enough structure to allow convergence once $\theta$ decreases. BH, however, relies on simpler interactions without multipoles. When $\theta$ is initially high, the resultant distortion of the repulsive derivative prevents



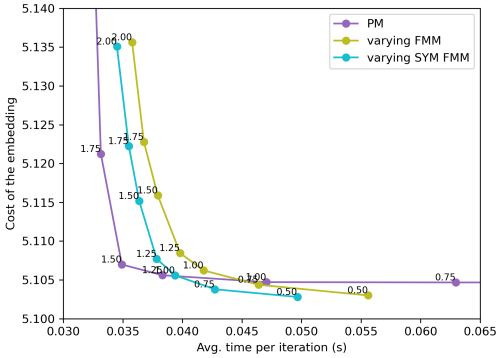

(a) V-sFMM (b) V-BH

Figure 5.6: The cost of the embedding at each iteration of the t-SNE method.

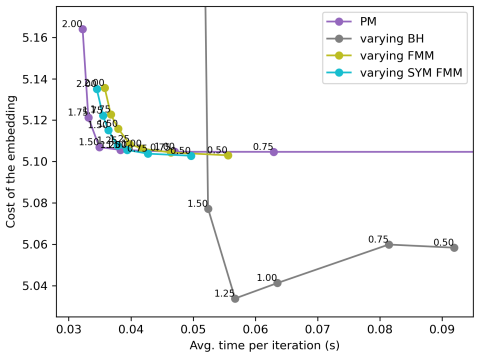


(a) V-sFMM (b) V-BH

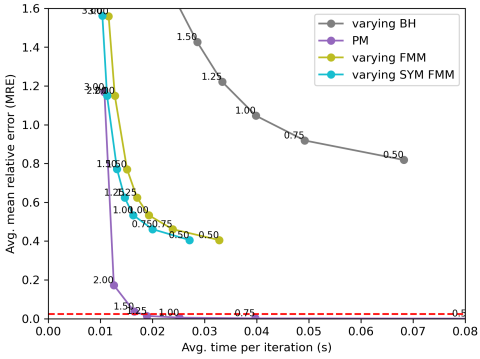
Figure 5.7: Resulting embedding of the t-SNE method.




(a) V-FMM, v-sFMM and PM.




(b) V-BH, v-FMM, v-sFMM and PM.




(c) V-BH, v-FMM, v-sFMM and PM.

Figure 5.8: Effect of varying $\theta$ values on the "average error/time" and "average cost/time" ratio.

the embedding from converging, which is eventually corrected by the $\theta$ value lowering throughout the iteration process.

The visual comparison in fig. 5.7 shows a clear degradation in the clustering produced by the v-BH method. This reduction in quality, which is not present in the v-sFMM, can be explained by the same distortion that prevented the cost of the embedding from decreasing between iteration 150 and 500. This result also implies that a lower final cost does not necessarily mean a better clustering quality.

Figure 5.8c shows that the BH-like† methods exhibit higher average MRE when $\theta$ varies. The reason for this larger error is the initially large $\theta$ preventing the average from reaching a lower value. In fig. 5.8b, the v-BH achieves a lower final cost than both v-sFMM and PM. However, as demonstrated visually in fig. 5.7b, this does not correspond to superior clustering. The v-sFMM, by

contrast, achieves lower "time per iteration" than the PM with any of its tested accuracy parameters, for minimum $\theta$ values of 0.5 and 0.75, while maintaining visually stable embeddings. This makes it a viable acceleration strategy over the PM.

# Chapter 6

# Discussion

This chapter synthesizes the experimental results presented in Chapter 5 and evaluates their implications for the use of the FMM and sFMM in accelerating t-SNE. The discussion focuses on the overall performance of the evaluated N-body solvers, the relationship between approximation error and embedding quality, and the impact of a varying $\theta$. Finally, practical parameter recommendations are provided based on the observed experimental behavior.

## 6.1 Overall Performance of FMM for T-SNE

The experiments demonstrate that the FMM and sFMM consistently outperform other BH–like$^{\dagger}$ approaches across the tested accuracy range. In particular, the sFMM achieves lower approximation errors for similar runtimes compared to the FMM.

The improved performance of the FMM can largely be attributed to the use of Node–Node interactions and multipole interactions. While the BH method relies on Point–Node interactions, the FMM aggregates groups of particles and evaluates their influence collectively through multipole expansions. This significantly reduces the number of required interactions while maintaining acceptable approximation accuracy. Furthermore, the symmetric interaction scheme used in the sFMM avoids redundant computations, which further improves computational efficiency.

Despite these advantages, the experiments show that the PM consistently dominates the ”average error/time” Pareto frontier. Several factors may explain this result.

First, the PM method benefits from a mature and highly optimized implementation, whereas the FMM implementation used in this work is relatively new and may not yet fully exploit available performance optimizations. Second, the structure of the approximation error differs between the methods. The PM primarily introduces errors in short-range interactions, which is less likely to occur in t-SNE since points push each other apart during optimization. As a result, the impact of these short-range errors becomes less significant during later optimization stages. Third, the two-dimensional nature of the t-SNE embedding space allows the PM to operate on a relatively small grid, which reduces the

computational overhead associated with mesh-based calculations. Finally, the used Z accumulation may create sub-optimal normalization.

Overall, these results indicate that while the FMM does not surpass the PM in terms of "average error/time" efficiency, it provides an improvement over other BH–like† methods.

## 6.2 Impact of Error on Embedding Quality

The experiments show that the sFMM remains more stable than the BH method when larger approximation parameters are used. For higher values of $\theta$, the BH method begins to introduce visible artifacts in the embeddings, such as distorted or fragmented clusters. In contrast, the sFMM produces embeddings that remain visually coherent under similar approximation settings.

This difference is likely related to the interaction schemes used by the two methods. The BH method relies on point–node interactions, which can lead to stronger inaccuracies when aggressive approximations are applied. The sFMM, on the other hand, uses Node–Node interactions and multipole expansions, which provide a smoother approximation of distant particle interactions.

As a result, the sFMM is able to tolerate larger approximation errors while still maintaining reasonable embedding quality, whereas the BH method becomes visually unstable at comparable $\theta$ values.

## 6.3 Effectiveness of Varying $\theta$

The final experiment investigated whether gradually varying $\theta$ during the optimization process could improve computational efficiency. The underlying idea is that early iterations of t-SNE primarily perform coarse global organization of the embedding, while later iterations refine local cluster structures. As a result, stronger approximations may be acceptable during early stages of the optimization.

The results indicate that this strategy can be beneficial for FMM-based methods. The v-sFMM closely follows the optimization trajectory of the regular sFMM while achieving slightly lower final embedding costs and reduced average iteration times. This suggests that aggressive approximations in early iterations do not significantly disrupt the optimization process and may even accelerate convergence.

In contrast, the varying $\theta$ strategy appears less stable for the BH method. The experiments show periods where the embedding cost stagnates before decreasing again later in the optimization process. This indicates that the stronger early

approximations may introduce distortions that the algorithm must subsequently correct.

Overall, these results suggest that dynamically varying the approximation strength can be a useful strategy for FMM-based solvers but do work as effectively for the BH method.

## 6.4 Recommendations

Based on the experimental results, several practical recommendations can be made for applying accelerated t-SNE methods in practice.

First, for tree-based methods such as the BH method, a maximum node size of approximately 16 points provides a good trade-off between tree depth and interaction cost. Smaller values increase the depth of the tree and therefore the traversal overhead, while larger values reduce the effectiveness of the spatial hierarchy.

Second, when choosing between different N-body solvers, the PM currently provides the best empirical error-time performance. However, the v-sFMM remains an attractive alternative. When trying to optimize for the cost of the embedding, the v-SFMM is able to outperform the PM by reaching a lower cost within the same runtime.

# Chapter 7

# Conclusion

This work compared BH-like† methods, such as the FMM, against the PM method in the context of t-SNE repulsive derivative computation. Additionally, a varying $\theta$ value was proposed to increase the efficiency of each BH-like† method.

First, regarding the error–time trade-off, we found that the methods rank from least to most efficient as: rBH<rMM<BH<MM<FMM<sFMM<PM. The sFMM was found to achieve a speedup of 2.5-3x over BH using $\theta = 0.5$ while maintaining equivalent error. Contrary to theoretical expectations [10], neither the FMM nor the sFMM yielded results that outperformed the PM method for any of the tested accuracy parameters.

Second, under large $\theta$ values, the FMM and sFMM produce more stable and visually coherent embeddings than the BH method using the same value. The smoother multipole Node–Node interactions are better at preventing severe structural artifacts.

Third, introducing a varying $\theta$ parameter yields different outcomes for BH and FMM. The varying BH shows substantial deviation in cost trajectories and produces degraded clustering quality. In contrast, the v-FMM and v-sFMM maintain stable convergence behavior and show less visual degradation than v-BH. We therefore argue that a varying $\theta$ parameter can safely be employed for these methods.

Although the v-sFMM does not outperform PM in terms of error, it achieves a lower final cost within the same runtime as PM using any of the tested accuracy parameter values using minimum $\theta$ values of 0.5 and 0.75. These values remain sufficiently small to avoid visible artifacts. While the v-BH can reach an even lower final cost, fig. 5.7 has shown us that this does not translate into improved clustering quality.

Overall, we have demonstrated that combining the sFMM with a varying $\theta$ strategy yields a faster descent to low-cost embeddings compared to PM, while preserving visual quality. Although PM remains superior in the strict error–time sense, the v-sFMM offers a competitive alternative when convergence speed toward low embedding cost is the primary objective.

# Chapter 8

# Future Work

A concept that has not been explored in this paper is parallelization. All of the various solvers presented so far could benefit from CPU or even GPU parallelization in order to speed up their computation. Many papers have already been published exploring this such as [10], [21] and [22]. Not only do the N-body solver benefit from using multiple cores, the sparse matrix creation and traversal are also highly parallelizable. A GPU implementation of the PM for t-SNE has already been created [23]. A similar approach can be taken to further improve the FMM for t-SNE which might outperform existing methods. For a single compute node, it is unlikely that a parallel FMM based approach would outperform the PM method. This balance does change when considering multiple compute nodes [21]. Since the PM method requires all-to-all communication, it has worse scaling with respect to the number of compute nodes compared to the FMM. This creates potential for a faster t-SNE solver when dealing with larger datasets that require such node clusters.

The methods explored in this paper did not explore the use of hybrid approaches such as the PM-FMM seen in [10]. This hybrid approach tries to combine the speed of the PM method for long range force with the accuracy of the FMM for short range forces. This method has the potential to increase both the speed and accuracy of the existing method.

The FMM has to consider all permutations of the interaction between a point and node. Not all interactions result in an equal accuracy when working with the same $\theta$. By adjusting the $\theta$ value for each type of interaction, can one type of interaction be prevented from dominating the produced error.

The multipole order of the implementation of the FMM in this paper did not exceed quadrupoles. Higher order multipoles can be used to further increase the accuracy of the methods at the cost of calculation time. This trade-off has the potential to make the FMM more efficient in calculating the derivatives of t-SNE.

The inherent symmetry of the multipole moments/fields was not exploited. The additional memory that is used to store the redundant symmetrical entries could feasibly make the FMM slower, especially when working with higher order multipoles.

While t-SNE usually restricts itself to a two-dimensional visualization, it can also produce three-dimensional visualizations. An increase in dimensions might show a different trend in efficiency for the tested methods.

# Bibliography


[1] G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger, "Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data," *Nature Methods*, vol. 16, no. 3, pp. 243–245, Mar. 2019, ISSN: 1548-7105. DOI: 10.1038/s41592-018-0308-4. [Online]. Available: https://doi.org/10.1038/s41592-018-0308-4.

[2] G. E. Hinton and S. Roweis, "Stochastic Neighbor Embedding," in *Advances in Neural Information Processing Systems*, vol. 15, MIT Press, 2002. Accessed: Sep. 26, 2025. [Online]. Available: https://papers.nips.cc/paper_files/paper/2002/hash/6150ccc6069bea6b5716254057a194ef-Abstract.html.

[3] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html.

[4] J. W. Eastwood, R. W. Hockney, and D. N. Lawrence, "P3M3DP—The three-dimensional periodic particle-particle/ particle-mesh program," *Computer Physics Communications*, vol. 19, no. 2, pp. 215–261, Apr. 1980, ISSN: 0010-4655. DOI: 10.1016/0010-4655(80)90052-1. Accessed: Sep. 30, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0010465580900521.

[5] J. Barnes and P. Hut, "A hierarchical o(n log n) force-calculation algorithm," *Nature*, vol. 324, no. 4, pp. 446–449, 1986. DOI: 10.1038/324446a0.

[6] H. Rein and S.-F. Liu, "REBOUND: An open-source multi-purpose N-body code for collisional dynamics," en, *Astronomy & Astrophysics*, vol. 537, A128, Jan. 2012, Publisher: EDP Sciences, ISSN: 0004-6361, 1432-0746. DOI: 10.1051/0004-6361/201118085. Accessed: Sep. 30, 2025. [Online]. Available: https://www.aanda.org/articles/aa/abs/2012/01/aa18085-11/aa18085-11.html.

[7] J. Ambrosiano, L. Greengard, and V. Rokhlin, "The fast multipole method for gridless particle simulation," *Computer Physics Communications*, vol. 48, no. 1, pp. 117–125, Jan. 1988, ISSN: 0010-4655. DOI: 10.1016/0010-4655(88)90029-X. Accessed: Sep. 30, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/001046558890029X.

[8] W. Dehnen, "A Very Fast and Momentum-conserving Tree Code," *The Astrophysical Journal*, vol. 536, pp. L39–L42, Jun. 2000, ISSN: 0004-637X. DOI: 10.1086/312724. Accessed: Oct. 31, 2025.

[9] J. S. Bagla, "TreePM: A code for Cosmological N-Body Simulations," *Journal of Astrophysics and Astronomy*, vol. 23, no. 3-4, pp. 185–196, Dec. 2002, arXiv:astro-ph/9911025, ISSN: 0250-6335, 0973-7758. DOI: 10.1007/BF02702282. Accessed: Sep. 30, 2025. [Online]. Available: http://arxiv.org/abs/astro-ph/9911025.

[10] V. Springel, R. Pakmor, O. Zier, and M. Reinecke, "Simulating cosmic structure formation with the GADGET-4 code," *Monthly Notices of the Royal Astronomical Society*, vol. 506, no. 2, pp. 2871–2949, Jul. 2021, arXiv:2010.03567 [astro-ph], ISSN: 0035-8711, 1365-2966. DOI: 10.1093/mnras/stab1855. Accessed: Sep. 30, 2025. [Online]. Available: http://arxiv.org/abs/2010.03567.

[11] L. van der Maaten, *Barnes-hut-sne*, 2013. arXiv: 1301.3342 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1301.3342.

[12] G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger, "Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data," *Nature Methods*, vol. 16, no. 3, pp. 243–245, Mar. 2019, ISSN: 1548-7105. DOI: 10.1038/s41592-018-0308-4. Accessed: Dec. 15, 2025.

[13] N. Pezzotti et al., *GPGPU Linear Complexity t-SNE Optimization*, Aug. 2019. DOI: 10.48550/arXiv.1805.10817. arXiv: 1805.10817 [cs]. Accessed: Jan. 3, 2026.

[14] J. Cook, I. Sutskever, A. Mnih, and G. Hinton, "Visualizing Similarity Data with a Mixture of Maps," en, in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, ISSN: 1938-7228, PMLR, Mar. 2007, pp. 67–74. Accessed: Oct. 26, 2025. [Online]. Available: https://proceedings.mlr.press/v2/cook07a.html.

[15] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–307, Jan. 1988, ISSN: 0893-6080. DOI: 10.1016/0893-6080(88)90003-2. Accessed: Oct. 15, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608088900032.

[16] G. M. Morton, "A computer oriented geodetic data base; and a new technique in file sequencing," International Business Machines Ltd., Ottawa, Canada, Technical Report, 1966.

[17] W. Dehnen, "A Hierarchical O(N) Force Calculation Algorithm," *Journal of Computational Physics*, vol. 179, no. 1, pp. 27–42, Jun. 2002, ISSN: 00219991. DOI: 10.1006/jcph.2002.7026. arXiv: astro-ph/0202512. Accessed: Oct. 7, 2025.

[18] H. K., *Mnist dataset*, https://www.kaggle.com/datasets/hojjatk/mnist-dataset, Accessed: 2025-01-06, Kaggle, 2025.

[19] Zalando Research, *Fashion-mnist dataset*, https://www.kaggle.com/datasets/zalando-research/fashionmnist, Accessed: 2025-01-06, Kaggle, 2025.

[20] 10x Genomics, *1.3 million brain cells from e18 mice, single cell gene expression dataset by cell ranger v1.3.0*, https://www.10xgenomics.com/datasets/1-3-million-brain-cells-from-e-18-mice-2-standard-1-3-0, Published: 9 February 2017; accessed 6 January 2025, 10x Genomics, Feb. 2017.

[21] B. Kohnke, C. Kutzner, and H. Grubmüller, "A GPU-Accelerated Fast Multipole Method for GROMACS: Performance and Accuracy," *Journal of Chemical Theory and Computation*, vol. 16, no. 11, pp. 6938–6949, Nov. 2020, ISSN: 1549-9618. DOI: 10.1021/acs.jctc.0c00744. Accessed: Jan. 3, 2026.

[22] D. Potter, J. Stadel, and R. Teyssier, *PKDGRAV3: Beyond Trillion Particle Cosmological Simulations for the Next Era of Galaxy Surveys*, Sep. 2016. DOI: 10.48550/arXiv.1609.08621. arXiv: 1609.08621 [astro-ph]. Accessed: Nov. 7, 2025.

[23] N. Pezzotti et al., *GPGPU Linear Complexity t-SNE Optimization*, Aug. 2019. DOI: 10.48550/arXiv.1805.10817. arXiv: 1805.10817 [cs]. Accessed: Jan. 3, 2026.

[24] L. F. W. Furer, *First symmetric cartesian fmm for t-sne with adaptive theta*, https://github.com/LucasFurer/MasterThesisLucas, 2026.

# Appendix A

# Algorithms

**Algorithm 5:** Simple version of Stochastic Neighbor Embedding.

**Data:** dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$,
perplexity $\mathcal{P}$,
iteration amount $\mathcal{T}$,
learning rate $\eta$

**Result:** low dimensional data $\mathcal{Y}^{(t)}$

**Algorithm:**

compute $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ using eq. (3.5) with $\mathcal{P}$
compute $p_{i|j}$ using eq. (3.3) with $\Sigma$
initialize $\mathcal{Y}^{(0)}$ with $\mathcal{N}(0,\, 10^{-4}I)$
**for** $t = 1$ **to** $\mathcal{T}$ **do**
    compute $q_{i|j}$ using eq. (3.7)
    compute gradients using eq. (3.9)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta\dfrac{\partial C}{\partial \mathcal{Y}} + \mathcal{R}$

## A.1 Naive N-Body Algorithm

**Algorithm 6:** A naive algorithm for gravitational N-body.

**Data:** particles $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$,
delta time $t$

$\forall \mathbf{p} \in \mathcal{P}, \quad \mathbf{p}.acc = \langle 0, 0, 0 \rangle$
**foreach** $\mathbf{p}_{sink} \in \mathcal{P}$ **do**
    **foreach** $\mathbf{p}_{source} \in \mathcal{P}$ **do**
        $\mathbf{p}_{sink}.acc$ += GRAVITY($\mathbf{p}_{sink}, \mathbf{p}_{source}$)
update velocities and position

## A.2 Barnes–Hut N-Body Algorithm

**Algorithm 7:** A Barnes–Hut implementation for gravitational N-body.

**Data:** particles $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$,
delta time $t$,
accuracy parameter $\theta$

$\forall \mathbf{p} \in \mathcal{P}, \quad \mathbf{p}.acc = \langle 0, 0, 0 \rangle$
create tree structure using nodes $\mathcal{N} = \{\mathbf{N}_1, \mathbf{N}_2, \dots, \mathbf{N}_N\}$
calculate $(z, m)$ **ForEach** node
**foreach** $\mathbf{p} \in \mathcal{P}$ **do**
  collect acceleration using BH_TRAVERSE($\mathbf{p}, \mathbf{N}_{root}$)
update velocities and position

**Algorithm 8:** A recursive Barnes–Hut tree traversal.

**function** BH_TRAVERSE($\mathbf{p}_{sink}, \mathbf{N}_{source}$)
  **if** DESCENT_CRITERION($\mathbf{p}_{sink}, \mathbf{N}_{source}, \theta$) **then**
    $\mathbf{p}_{sink}.acc$ += GRAVITY($\mathbf{p}_{sink}, \mathbf{N}_{source}$)
  **else if** IS_LEAF($\mathbf{N}_{source}$) **then**
    **foreach** $\mathbf{p}_{child} \in \mathbf{N}_{source}.particles$ **do**
      $\mathbf{p}_{sink}.acc$ += GRAVITY($\mathbf{p}_{sink}, \mathbf{p}_{child}$)
  **else**
    **foreach** $\mathbf{N}_{child} \in \mathbf{N}_{source}.children$ **do**
      BH_TRAVERSE($\mathbf{p}_{sink}, \mathbf{N}_{child}$)

## A.3 Reverse Multipole Method N-Body Algorithm

**Algorithm 9:** A reverse multipole method implementation for gravitational N-body.

**Data:** particles $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$,
delta time $t$,
accuracy parameter $\theta$

$\forall \mathbf{p} \in \mathcal{P}, \quad \mathbf{p}.acc = \langle 0, 0, 0 \rangle$
create tree structure with nodes $\mathcal{N} = \{\mathbf{N}_1, \mathbf{N}_2, \dots, \mathbf{N}_N\}$
calculate $(z, \mathbf{M})$ **ForEach** node using eq. (4.12)
**foreach** $\mathbf{p} \in \mathcal{P}$ **do**
  collect multipole fields using rMM_TRAVERSE($\mathbf{N}_{root}, \mathbf{p}$)
$\forall \mathbf{N} \in \mathcal{N}$ translate and sum multipole fields to children with eq. (4.24)
for every leaf node, add $\mathbf{C^0}$ to the acceleration
update velocities and position

**Algorithm 10:** A recursive reverse multipole method tree traversal.

**function** rMM_TRAVERSE($\mathbf{N}_{sink}, \mathbf{p}_{source}$)
  **if** DESCENT_CRITERION($\mathbf{N}_{sink}, \mathbf{p}_{source}, \theta$) **then**
    $\mathbf{N}_{sink}.\mathbf{C}$ += GRAVITY($\mathbf{N}_{sink}, \mathbf{p}_{source}$) using eq. (4.21)
  **else if** IS_LEAF($\mathbf{N}_{sink}$) **then**
    **foreach** $\mathbf{p}_{child} \in \mathbf{N}_{sink}.particles$ **do**
      $\mathbf{p}_{child}.acc$ += GRAVITY($\mathbf{p}_{child}, \mathbf{p}_{source}$)
  **else**
    **foreach** $\mathbf{N}_{child} \in \mathbf{N}_{sink}.children$ **do**
      rMM_TRAVERSE($\mathbf{N}_{child}, \mathbf{p}_{source}$)

## A.4 Fast Multipole Method N-Body Algorithm

**Algorithm 11:** A fast multipole method implementation for gravitational N-body.

**Data:** particles $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$,
delta time $t$,
accuracy parameter $\theta$

$\forall \mathbf{p} \in \mathcal{P}, \quad \mathbf{p}.acc = \langle 0, 0, 0 \rangle$
create tree structure with nodes $\mathcal{N} = \{\mathbf{N}_1, \mathbf{N}_2, \dots, \mathbf{N}_N\}$
calculate $(z, \mathbf{M})$ **ForEach** node using eq. (4.12)
collect multipole fields using FMM_TRAVERSE($\mathbf{N}_{root}, \mathbf{N}_{root}$)
$\forall \mathbf{N} \in \mathcal{N}$ translate and sum multipole fields to children using eq. (4.24)
for every leaf node, translate and add $\mathbf{C^0}$ to the to the acceleration of the contained points
update velocities and position

**Algorithm 12:** A recursive fast multipole method tree traversal.

```
function FMM_TRAVERSE(N_sink, N_source)
    if DESCENT_CRITERION(N_sink, N_source, θ) then
        N_sink.C += GRAVITY(N_sink, N_source)
    else if IS_LEAF(N_sink) then
        foreach p_child ∈ N_sink.particles do
            MM_TRAVERSE(p_child, N_source)
    else if IS_LEAF(N_source) then
        foreach p_child ∈ N_source.particles do
            rMM_TRAVERSE(N_sink, p_child)
    else
        foreach N_sinkChild ∈ N_sink.children do
            foreach N_sourceChild ∈ N_source.children do
                FMM_TRAVERSE(N_sinkChild, N_sourceChild)
```

# Appendix B

# Implementation/Hardware Details

The code for running the algorithms and tests was written in c++20 and was compiled using the MSVC compiler. The source code can be found in [24]. All tests were performed on a system with 32 gigabytes of RAM and a Intel core i5-12400f CPU. All algorithms were run using only a single thread.

Several libraries were used throughout the project, all of which shall be listed here. The OpenGL API was used for displaying the embedded space as it is being optimized. Vector operations were handled by the OpenGL mathematics library. In order to make the computations of the multipole interactions more wieldy, we use tensors from the Fastor library. The responsibility of loading sparse matrices was left to the Eigen library. Although Eigen has tensor types too, it was considered less suitable for our needs then Fastor. The radix sorting algorithm from Boost is used in order to maintain the $O(n)$ time complexity that the FMM promises. The implementation of the PM was obtained from [12] which will be used to test our methods against.

While the tests in the c++ code can generate csv files, they cannot turn them into a plot. For this we have used python, specifically the matplotlib library. We also used libraries like pandas and numpy to handle the initial input data. Libraries like sklearn, scanpy and openTSNE were then used to process them and obtain the P matrices.

# Appendix C

# Method Time Complexities

Table C.1: Time complexity of all tested methods where $n$ is the number of points and $m$ is the number of cells.

| Method | Time complexity |
|---|---|
| Barnes&Hut | $O(n \log(n))$ |
| Barnes&Hut multipole | $O(n \log(n))$ |
| Barnes&Hut reverse | $O(n \log(n))$ |
| Barnes&Hut reverse multipole | $O(n \log(n))$ |
| Fast multipole method | $O(n \log(\theta))$ |
| Symmetric fast multipole method | $O(n \log(\theta))$ |
| Particle mesh | $O(n + m \log(m))$ |

## Appendix D

# Abbreviations and Definitions

**BH** Barnes–Hut Method.

**rBH** Reverse Barnes–Hut Method.

**MM** Multipole Method.

**rMM** Reverse Multipole Method.

**FMM** Fast Multipole Method.

**sFMM** Symmetric Fast Multipole Method.

**PM** Particle-Mesh method.

**v-** Method prefix indicating that the time-varying $\theta$ parameter is used.

**BH-like** Umbrella term for BH, rBH, MM, rMM, FMM and sFMM.

$\theta$ **(Theta)** Accuracy parameter of the tree-based N-body solvers.