# MPI Implementation of an Energy Landscape Visualisation Tool -Stochastic Hyperspace Embedding And Projection (SHEAP)

Mo Ji

28 June 2024

## Introduction

Energy landscape, also known as a potential energy surface (PES), is defined as a framework to describe the potential energy of a system as a function of the positions of its atoms [1–3]. PES is crucial to represent the structure and thermodynamics of systems of atoms configurations, where each point on the surface represents a different configuration corresponding to a potential energy level. By exploring through the PES, it is clear that the stable states (global minima), meta-stable states (local minima) and the transition states (saddle points) could be established. The system evolving towards equilibrium between minima could be described [4]. Based on this concept, a wide range of tools have been developed to simulate the PES, i.e. identify the minima and saddle points, such as basin hopping, ab initio random structure searching (AIRSS), and random tree construction (T-RRT) etc. [2, 5, 6]. Though the visualisation of the energy landscape is not necessary to calculate the thermodynamic properties or predict the structures, it could provide more insight on the PES as well as highlighting the key features, such as minima, saddle points and reaction pathways. However, visualising PES is a rather challenging task because the potential energy surface data generated by the simulations, such as first principle density function, often is high in dimension and large in quantity.

Dimensionality reduction is defined as the techniques to mapping of a high-dimensional data into a space of lower dimensionality [7]. To gain insights from the large PES datasets, dimensionality reduction techniques have been adapted to project high dimensional data into low dimensions while retaining the most important features [8]. Various dimensionality reduction techniques have been developed to map high dimensional datasets into 2D or 3D plots [9]. It is worth noting that there are also supervised algorithms for dimensionality reduction, such as linear discriminant analysis [10]. However it is out of the scope of this report, in terms of visualising PESs. The main unsupervised dimensionality techniques include linear approaches, for example the principal component analysis, and non linear approaches, such as the multidimensional scaling (MDS), Isomap, t- distributed stochastic neighbour embedding (t-SNE) and uniform manifold approximation and projection (UMAP) etc., i.e. manifold learning approaches [9]. Manifold learning approaches assume that the data lie on low dimensional manifold embedded within the original high- dimensional space, i.e. non- linear, also assumes that the dimensionality of the data sets is only artificially high [11]. Despite various dimensionality reduction techniques being available, each technique carries specific assumptions and thus limitations [9]. PCA carries out the embedding the data into a linear subspace that describes as much variance as possible. It is easy to apply and computationally cheap, however it assumes dataset sit in a linear subspace, which might not be an appropriate assumption [12]. Multidimensional scaling and Isomap are both manifold- based approaches, in which the aim is to retain pairwise distances (Euclidean distance for MDS and geodesic distance for Isomap). However MDS does not take neighbouring data points into account, and the performance of the Isomap is heavily dependent on the choice of parameters, such as number of neighbours (k - nearest neighbours) or the radius ($\epsilon$ - neighbourhoods) [7]. The t-SNE is also a manifold based approach, which converts the high- dimensional Euclidean distance into conditional probabilities, i.e. similarity of each pair of data. And a student t- distribution is used to illustrate the similarity of the data points in the low dimension. KL divergence is used as the cost function to minimise the conditional distribution between the high-dimensional and low- dimensional space [13]. However, it has been reported that t-SNE algorithm tends to preserve the local structure rather than the representative global representation [1]. As for UMAP, it adapts a topological approach, i.e. fuzzy simplicial complex, where a graph is generated by finding a fixed number of nearest neighbours in both high- dimensional and low- dimensional space. Then, cross entropy is used as the cost function to minimise the difference between the fuzzy simplicial complex in high- dimensional and low- dimensional space [14]. It has achieved improved performance in visualising high dimensional datasets, i.e. preserving both local and global structure of the data set, which is attributed to its cost function [2].

There are also existing visualisation tools for PES applications, such as disconnectivity graphs, and sketch maps. Disconnectivity graphs have been used to demonstrate the minimum energy pathways between local minima, but the graph itself does not reflect the spatial relationships among these basins, or the volume of each basin [4]. In terms of sketch maps, it applies a similar concept as t-SNE algorithm, i.e. minimising the pairwise distance difference between the high- dimensional and low- dimensional space. It has been initially developed to resolve issues such as poor sampling in the high dimensional space. However it has been reported that the approach failed to accurately represent the pairwise distance among the data points from different basins [15]. Therefore, SHEAP algorithm has been developed to fulfill the gap of representing the both local and global PES structure accurately.

To visualise the potential free energy surface, a structure descriptor is required to describe the atoms configuration, i.e. a vector to describe the atoms relative arrangements based on their atomic coordinates. The pairwise distance between atoms has been used in this reproduciblility study. This simple structure descriptor works in this case because Lennard-Jones potential is only dependent on the pairwise distance between two non- bonding atoms, which is given as [2] [16]:

$$\nu_{LJ}(r) = 4\varepsilon \left( \left[\frac{\sigma}{r}\right]^{12} - \left[\frac{\sigma}{r}\right]^{6} \right) \tag{1}$$

Where $\varepsilon$ is the attractive well depth, $\sigma$ is the distance where the potential between two atoms is zero, and $r$ is the pairwise distance. $\epsilon = 1$ and $\sigma = 2$ have been used in the original paper to generate the LJ data sets. Hence for LJ13 and LJ38 data sets, they represent a cluster of 13 or 38 atoms, i.e. a vector with 169 or 1444 dimensions, respectively.

In this project, the main aim is to reproduce the SHEAP algorithm in FORTRAN, as well as to optimise the algorithm for running larger data sets via implementing the Message Passing Interface (MPI). This allows the SHEAP algorithm to take advantage of the computation resources on the HPC facilities. There has been variations of t-SNE algorithms to speed up the computation, such as Barnes-Hut-SNE (approximate implementation) and (FIt-SNE) fast Fourier transformation accelerated interpolation-based t-SNE [2]. However they have not been implemented as it is out of the scope of this project. The objectives can be described as follows:

- Reproduce the SHEAP algorithm core functions. Demonstrate the layout of the local minima and the relative volume of the basins for LJ13 and LJ38 data sets.
- Improve efficiency, maintainability and readability of the code, optimise the code to handle larger data sets.
- Implement MPI for the SHEAP algorithm to accommodate large data sets.

**SHEAP- algorithm explanation**

SHEAP algorithm is a hybrid between using the conditional probabilities of t-SNE and the cost function of UMAP [2]. By using the UMAP cost function, it is expected to improve the global representation of the data in the low dimensional space. The probability of $p_{ij}$ is then given as:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \tag{2}$$

Where $p_{i|j}$ and $p_{j|i}$ are the conditional probability of data point i giving the condition of data point j, and data point j giving the condition of data point i. The conditional probability in the high dimensional space is given as:

$$p_{j|i} = \frac{exp[-d(x_i, x_j)^2/2\sigma_i^2]}{\sum_{k \neq i} exp[-d(x_i, x_j)^2/2\sigma_i^2]} \tag{3}$$

Where the $d(x_i, x_j)^2$ is the Euclidean distance between $x_i$ and $x_j$ in the high dimensional space, and the $\sigma$ is the variance of the conditional distribution, which is determined by the perplexity. The perplexity is given as an input parameter, which measures the effective number of neighbours for a given data points. It defines the local neighbourhood range for each data point. A small perplexity promotes a more accurate representation of the local connectivity, whereas a large perplexity tends to capture the overall global structure better. Exploring a range of perplexity values might be beneficial to understand the underlying data structure. The relationship between the perplexity and the conditional probability is given as:

$$Prep(P_i) = 2^{-\sum_j p_{j|i} log_2 p_{j|i}} \tag{4}$$

The conditional distribution probability in the low- dimensional space is described as a Student t distribution, which is given as:

$$q_{ij} = \frac{1}{(1 + d(y_i - y_j)^2)N(N-1)} \tag{5}$$

Where the $d(y_i, y_j)^2$ is the distance between $y_i$ and $y_j$ in the low-dimensional space. The reason to use the student t distribution over Gaussian distribution is to minimise the crowding problem. The heavier tailed distribution allows larger distance between clusters in the low- dimensional space, as the projection of the high dimensional distance into low- dimensional space might lead to a very large distance. SHEAP algorithm utilises the cost function from UMAP, rather than t-SNE to achieve better global representation of the data. The cost function is given as:

$$C = \sum_{i,j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) + (1 - p_{ij}) \log\left(\frac{1 - p_{ij}}{1 - q_{ij}}\right) \tag{6}$$

Where the first half of the cost function is still the same as the t-SNE algorithm, i.e. $C = \sum_{i,j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$. The t-SNE cost function itself is the Kullback- Leibler (KL) divergence. And the second half of the cost function is the repulsive contribution from $q_{ij}$. The cost function gradient in SHEAP is given by:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4N(N-1) \sum_j \left(p_{ij} - q_{ij}\left(\frac{1 - p_{ij}}{1 - q_{ij}}\right)\right) q_{ij}(\mathbf{y}_i - \mathbf{y}_j) \tag{7}$$

By knowing the gradient for each data point, the gradient descent algorithm could then be applied to minimise the cost function.

Another important feature for SHEAP algorithm is to calculate the hard sphere. Hard sphere has been introduce to demonstrate the size of each basin in the PES. SHEAP algorithm uses a ' hard sphere' rather than a point to represent a local minimum. The size of the hard sphere is determined by the number of similar structures, which is determined by a similarity threshold of high- dimensional distance. The concept is to introduce a cost function, i.e. hard-sphere potential, so that the layout of the hard spheres cannot have overlaps. The hard-sphere potential is given as:

$$U_{HS} = \frac{1}{2}\gamma_{HS} \sum_i \sum_{j \neq i} \alpha_{ij}^2, \tag{8}$$

where $\gamma_{HS}$ is the hard core strength and

$$\alpha_{ij} = \begin{cases} \frac{R_i + R_j - \|\mathbf{y}_i - \mathbf{y}_j\|_2}{2} & \text{if } \|\mathbf{y}_i - \mathbf{y}_j\|_2 < R_i + R_j, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

In this way, the low-dimensional positions could be adjusted according to the hard sphere radius. The cost function of the hard sphere radius has only been added into the optimisation algorithm when it is close to convergence. That is, the optimisation will be divided into two parts. The first part of optimisation only aiming to adjust the low dimensional positions to minimise the cost function due to the difference in conditional probability of $p_{ij}$ and $q_{ij}$.

The convergence criteria being used in this case is the norm of the gradient vector, i.e. monitoring the gradient magnitude during gradient descent process. The exponential Moving Average (EMA) has been applied to smooth out the fluctuations during the gradient descent, i.e. a more stable output of gradient magnitude. The logarithm of the gradient norm has also been used to represent the gradient changes during the optimisation process to handle the large changes of the gradient norm values, such as large gradient norm values at the early stage of gradient descent and very small values when it is closer to convergence.

## Methodology

The reproducing SHEAP algorithm has been carried out in FORTRAN, and CMake has been applied to automate generating Makefile, installing libraries, as well as managing the system dependencies. The profiling and the tests have been carried out on Apple MacBook Pro with M1 pro chip and 16 GB memory, and on the Icelake HPC at University of Cambridge. Local machine used a number of rank of 1- 9

where number of ranks were applicable, and Icelake used number of ranks from 9 to 144. Valgrind has been applied to check for memory leaks. lldb has been used to debug. Xcode and instrument has been applied for profiling the performance of the algorithm. Intrinsic FORTRAN timing functions have been used to time the functions. Each functions run for 100 iterations and the average time was calculated to minimise the fluctuation during testing speed of the algorithm. The optimisation process has followed the strategies listed as below:

- The initialisation method and the optimisation method within the SHEAP algorithm will be investigated. A better way of initialising low dimensional positions might reduce the computational cost significantly. Additionally, the gradient descent is expected to be the most computational expensive part of the algorithm, therefore it is worth exploring different optimisation options.
- Other optimisation strategies include (1) Using optimisation flags: The o3 flag for aggressive optimisation, such as loop unrolling, vectorization etc. ; (2) Hoisting: It is to move operations out of loop, if they are independent from the loop iteration; (3) Data locality: The goal is to minimise the cache misses via using the data in the cache as much as possible. (4) Stride access: The preference is to write continuously while dealing with matrices, so that the cache only needs to be updated once the cache line is filled. (5) Minimising branch: The aim is to reconstruct the conditional statements (if statements). This is because these statements might reduce the predictability of the code while being executed, i.e. increasing computational cost.
- OpenMP: It has already been applied extensively in the original code.
- MPI: It allows for parallel computing, i.e. using multiple ranks, to speed up the computation. In this report, a master and worker rank distribution system has been established. That is, the master rank oversees the work load of each rank, while each individual worker rank is working on calculating the gradient vector.

## Prototyping

The prototyping process has been demonstrated in Figure 1. Four key areas have been identified for the SHEAP algorithm, i.e. file reading and data pre-processing, high-dimensional data calculation, low-dimensional calculation and the optimisation. Data will firstly be parsed into the programme and normalised. Then based on the pre- set criteria, the duplicated points will be removed from the dataset to reduce the workload. High dimensional calculation will then be carried out. Firstly, the high dimension distance, i.e. Euclidean distance, will be calculated based on the data vectors. Then the sigma will be calculated for each point based on pre- determined perplexity value. Thus, the conditional distribution probability, i.e. $p_{ij}$, of the high dimension could then be established. The next step is to initialise the low dimensional positions, as well as initialising the hard sphere for each data point. Optimisation algorithms, such as two-point steepest descent (TPSD) and Adaptive Moment Estimation (ADAM), will be applied to minimise the cost between the high- dimensional distribution probability and the low-dimensional distribution probability.

The original SHEAP code has utilised TPSD gradient descent. TPSD gradient descent improves the accuracy and the convergence rate via considering the gradients from two points, i.e. the current point and a nearby point. The algorithm can be described as below.

**Algorithm 1** Two-Point Steepest Descent (TPSD)

1: Initial low-dimensional position, perturbation size $\delta$
2: **while** cost $\geq$ threshold **do**
3: Compute gradient at current position.
4: Perturb the current position, computer the gradient at the perturbed position
5: Calculate refined gradient also add noise to the refined gradient
6: Calculate step size
7: Update the low dimension positions
8: Calculate the cost function
9: **end while**

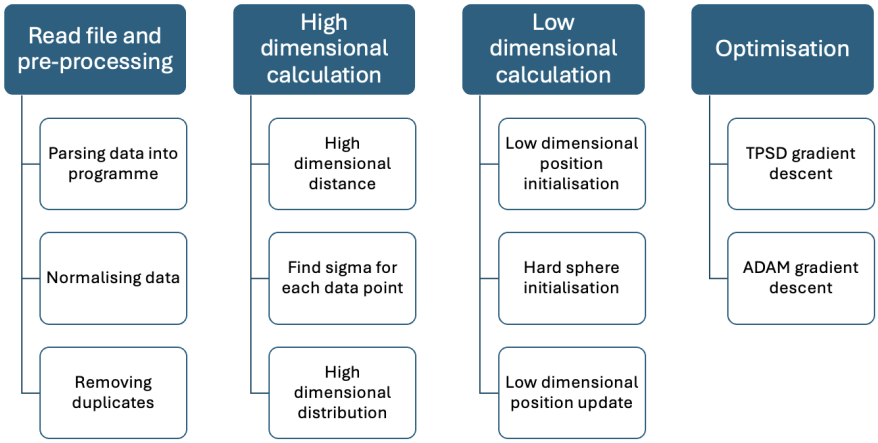



Figure 1: Prototyping of the SHEAP algorithm. The overall view gives the insights of the required modules and functions

To investigate the effect of the gradient descent algorithm on the performance of the SHEAP algorithm, ADAM gradient descent has also been implemented for the SHEAP algorithm. It adapts the learning rate via calculating the first and second moments of the gradient, i.e. the average of the gradients and the average of the squared gradient. The ADAM algorithm can be described as follow. Four parameters are required to be defined, i.e. $\beta_1$, $\beta_2$, learning rate $\alpha$ and a small constant, $\epsilon$ to avoid division by zero. For a given gradients $g_t$ at a time step t, the first moment can be calculated as :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{10}$$

And the second moment (velocity) is given as :

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{11}$$

Therefore the moment term is then:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{12}$$

And the velocity term is written as:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{13}$$

Finally, the parameters $\theta_t$ can be updated as:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{14}$$

The structure of data file for LJ13 and LJ38 has been explored. There are three rows for each data points, i.e. number of dimensions, vectors describing structure in the configuration space, and other information for the data point, i.e. ion label, ion number, ion formula, ion symmetry, ion volume, ion energy and point count. In the original code, the read file function allocates the maximum number of the data points, i.e.120000, for all the arrays and matrices. It could be a waste of memory if the actual number of data points is significantly smaller than the maximum number. Additionally, the maximum number might need to change manually for each dataset, especially if the size of data sets are large. Therefore, the read file function has been re-written to allow dynamic allocation of the arrays and matrices. It counts the total number of rows within the data file first, before parsing the data into the programme. Therefore, the memory will be allocated based on the actual size of the data, which is crucial for handling larger datasets.

Another important consideration is parallelisation computing. OpenMP (Open Multi- Processing) has been widely applied in the original code, which allows multiple threads to be executed simultaneously. It distributes the execution of the loops or tasks among the available threads to balance the workload.

OpenMP utilises multi threads optimisation based on the shared memory. The idea is to combine both OpenMP and MPI to optimise the speed of calculation.

The other side of the parallel computation is to enable the SHEAP algorithm running large datasets. It has been mentioned in the introduction that the dataset size could be around 100000 for modern structure predictions, which might be very time- consuming to carry out dimensionality reduction on a single rank. By implementing MPI, the SHEAP algorithm could run large dataset on HPC, which utilises the computational power of multiple ranks. This could be crucial for scientific research, especially in structure prediction. Due to the nature of the gradient descent, the sequential computation does not allow parallelisation, therefore the effort has been made on the loss gradient calculation. The strategy for MPI has been summarised as below, Figure 2. Loss gradient calculation includes calculating the low dimensional probability, calculating the gradient and then update the gradient vector. The gradient descent loop will run on the master rank, because the calculation is sequential. The work load, i.e. calculating the conditional probability and the gradient vector for each data point, will be distributed from the master rank to the worker ranks. To ensure the balance of work load, the work load for each rank has been calculated as below.

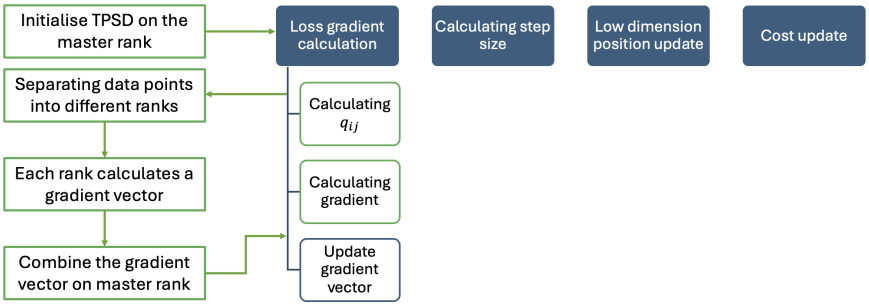



Figure 2: MPI Implementation strategy for the SHEAP algorithm.

**Algorithm 2** Work Distribution

$N$ is the number of data points, $n$ is the total number of ranks, and $r$ is the rank number.
The total load is $N(N-1)/2$
The load for each rank is $N(N-1)/2n$
**for** i = 1, N **do**
  $load = load + (N - i)$
  **if** $load \geq rN(N-1)/2n$ **then**
    $start = i$
  **end if**
  **if** $load \geq (r+1)N(N-1)/2n$ **then**
    $end = i - 1$
  **end if**
**end for**

## Results

### (1) PCA initialisation vs random initialisation

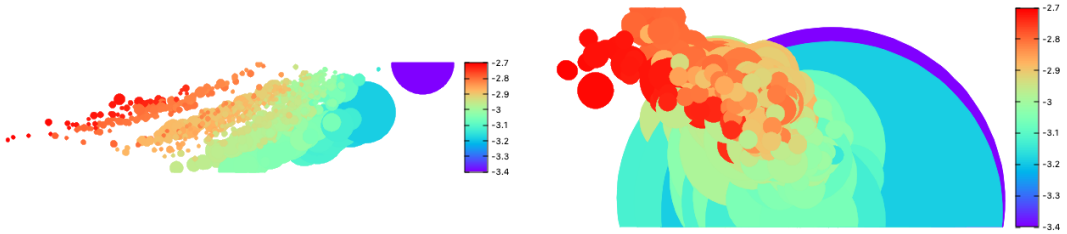
The random initialisation approach first initialises a random matrix, which is normalised and orthogonalised using the Gram-Schmidt process. The low- dimensional positions are calculated by performing a matrix multiplication between the random matrix and the high dimensional data. The low- dimensional positions are then normalised by the average sigma values and a projection compression factor.

The PCA initialisation approach initialises the low- dimensional position using principle component

analysis (PCA). Eigenvalues and eigenvectors are calculated using the Linear Algebra package (LAPACK). The number of eigenvectors and eigenvalues selected is determined by the dimension of the low-dimensional positions. These eigenvectors are then used to initialize the low-dimensional positions. Normalisation of the low- dimensional positions using the average sigma value is also performed.

The results of the random initialisation and the PCA initialisation are shown in Figure 3 and 5 for the datasets of LJ13 and LJ38, respectively. It can be seen that PCA initialisation clearly demonstrates a more structured initialisation on the low-dimensional positions compared to the random initialisation. This was expected to achieve a faster convergence rate due to the more optimised starting positions of the low- dimensional data points. However, the performance of the SHEAP algorithm by two initialisation approaches are comparable for the LJ13 and LJ38 data set. The running times averaged 2.8 seconds for LJ13 and 15 seconds for LJ38 datasets for both initialisation approaches. Figure 4 and 6 showed the low-dimensional positions after 100 iterations for both LJ13 and LJ38 datasets by both initialisation approaches.

It can be seen that PCA initialisation might have achieved better convergence than random initialisation at the early stage of gradient descent for LJ13 data set, but there was little difference for LJ38 data set in terms of visualisation. That is, the low-dimensional data positions of the LJ13 data set are closer to the final results for the PCA initialisation after 100 iterations, whereas no difference was observed for LJ38 data set. Additionally, the differences in convergence rate became negligible during the later stage, i.e. hard sphere growth stage, of gradient descent for LJ13 and LJ38 data set. Therefore, it suggests that PCA initialisation might provide better starting positions for the low-dimensional data positions, but the speed difference between PCA initialisation and random initialisation were minimal in this case. PCA initialisation might provide better performance for more complex dataset, i.e. where the initial starting positions plays a more important role.

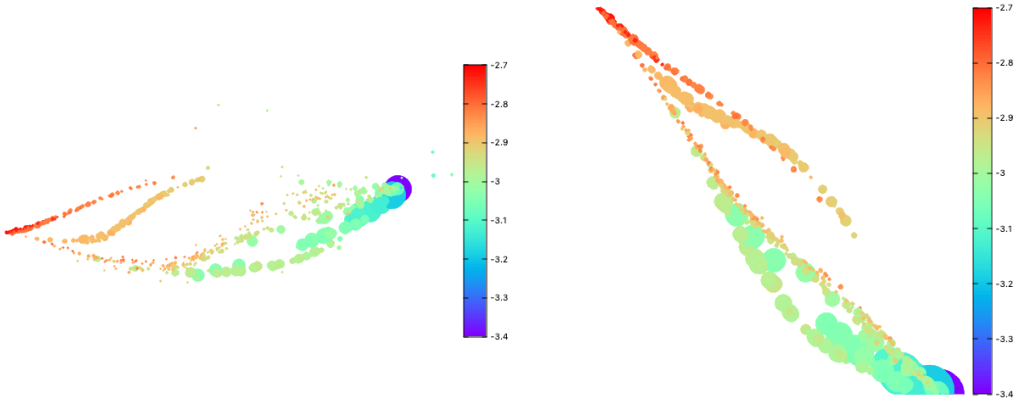


**(a)** **(b)**

Figure 3: (a) PCA initialisation result of LJ13 dataset, and (b) Random initialisation result of LJ13 dataset.

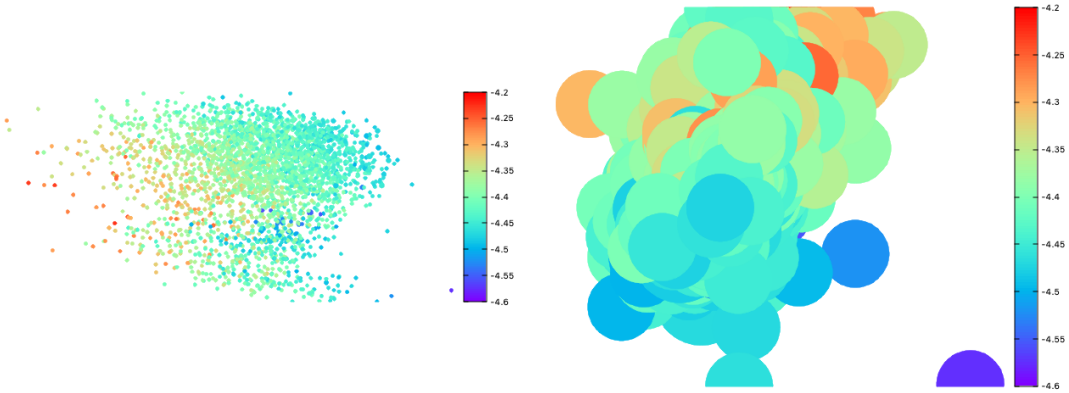
### (2) TPSD vs ADAM

As mentioned, TPSD The convergence rate of TPSD and ADAM have been investigated using both LJ13 and LJ38 data sets. Each gradient descent was running 100 times to obtain a representative view of the performance. Figure has summarised the elapsed time for ADAM approach. It can be seen that the LJ13 data set took an average of 2.8 seconds to converge, and LJ38 data set took an average of 34.5 seconds to converge. The difference in time for both data sets were attributed to the number of data points and the dimension of each data point. In comparison, Figure 7 and 8 summarised the performance of TPSD approach. LJ13 data set took an average of 3.8 second, whereas LJ38 data set only took an average of 26 seconds. It can be seen from the figures that the ADAM performed slightly better than TPSD on a small data set, i.e. LJ13. Whereas TPSD convergence rate was significantly quicker than ADAM on the large data set, i.e. LJ38. This could be due to TPSD takes a more blunt approach, i.e. using steepest descent without relying on the momentum or velocity. For a large data



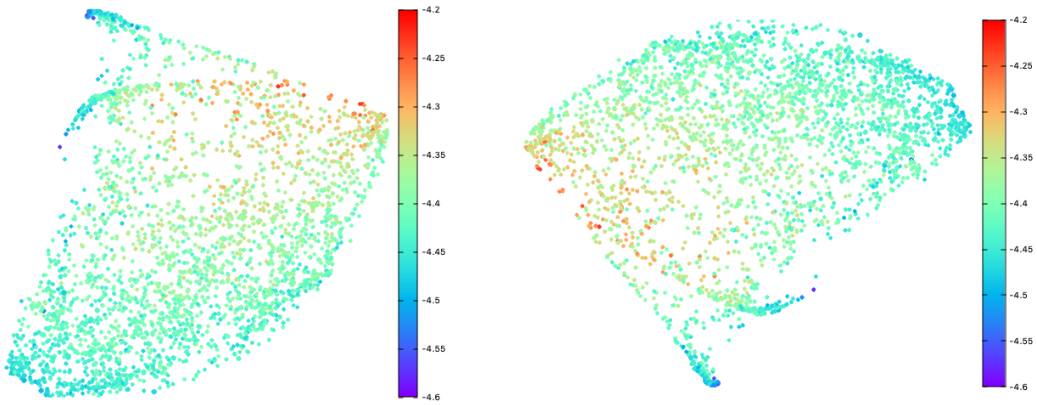

**(a)** **(b)**

Figure 4: (a) LJ13 dataset result after 100 iterations using PCA initialisation, and (b) LJ13 dataset result after 100 iterations using random initialisation.




**(a)** **(b)**

Figure 5: (a) PCA initialisation result of LJ38 dataset, and (b) Random initialisation result of LJ38 dataset.

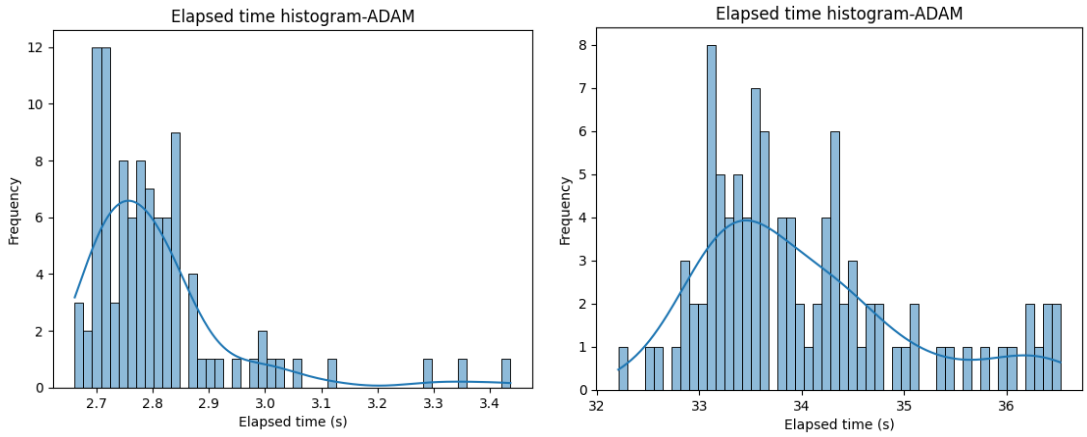



**(a)** **(b)**

Figure 6: (a) LJ38 dataset result after 100 iterations using PCA initialisation, and (b) LJ38 dataset result after 100 iterations using random initialisation.

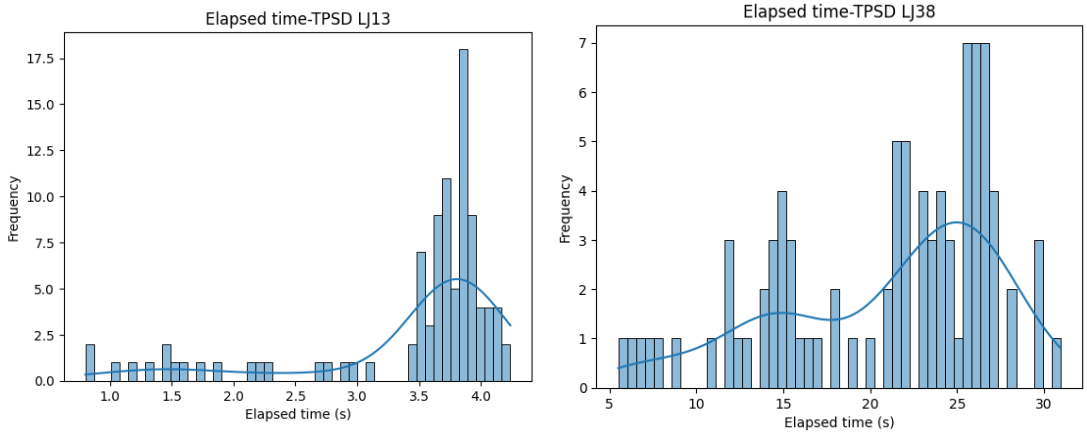
set, such as LJ38, there might be more local minima and saddle points compared to LJ13. Therefore, TPSD might achieve a faster convergence rate by only using the steepest descent approach, i.e. moving towards the global minimum more quickly. It is worth noting that 3% of the trials using TPSD approach did not converge before reaching the maximum steps. It is unclear what has caused the instability of the TPSD algorithm during the reproduction trials, as the original code did not seem to have such an

issue. Further investigation will be required to optimise the TPSD code. Therefore, TPSD has been kept as the first choice of gradient descent algorithm, due to its faster convergence rate at a large dataset.




**(a)** **(b)**

Figure 7: Elapsed time for running LJ13 (a) and LJ38 (b) data sets via ADAM optimisation algorithm.




**(a)** **(b)**

Figure 8: Elapsed time for running LJ13 (a) and LJ38 (b) data sets via TPSD optimisation algorithm.

### (3)MPI implementation

Message Passing Interface, MPI, has been developed to communicate among different ranks in a HPC setup. The main advantage for adding MPI function in this case is to enable the SHEAP algorithm to run large data sets. It has been achieved by splitting loss gradient calculation tasks into sub sections. That is, each rank calculate the low- dimensional conditional probability, i.e. $q_{ij}$, for a subset number of data points. Then the gradient vector will be gathered on the master rank, the low dimensional positions will then be updated also on the master rank before they are being sent out to the workers rank for the next iteration calculation.

It is, however, worth noting that the communication, i.e. message passing, comes with the overhead. The cost increases with increasing number of processes, as well as the size of the message being passed around. That is, each worker rank needs to send and receive messages, i.e. starting and ending number, to and from the master rank. The local gradient descent is also required to be sent to master rank from each worker rank, then finally, the updated low dimensional positions will be sent back to the worker ranks for the next iteration. The performance of the MPI implemented SHEAP algorithm has been summarised as below in Figure 9. Comparable trends have been observed for the elapsed time of both

the LJ13 and LJ38 data set. That is, the running time has increased between the rank number 1 and 4, then plateaued between rank number 4 and 6. The running time kept increasing from rank number 6 to 10. The communication cost became significant with a rank number of 10, i.e. the running time has almost doubled, i.e. 50% of running speed, for LJ13 data set using 10 ranks comparing to only using 1 rank. It is worth noting that the running speed has only slowed down by 35 % for LJ38 data set by using 10 ranks. This might be due to a larger low-dimensional position matrix of LJ38 data set. That is, though the communication overhead is also larger for a larger matrix, a higher percentage of the running time has been occupied by calculating the loss gradient vectors. This can be validated by the running time of LJ13 and LJ38 data sets without using MPI. It took 3.8 seconds in average for LJ13 data set, whereas an average of 35 seconds was required for the LJ38 data set, i.e. the loss gradient vectors calculation took significantly longer than LJ13 data set.

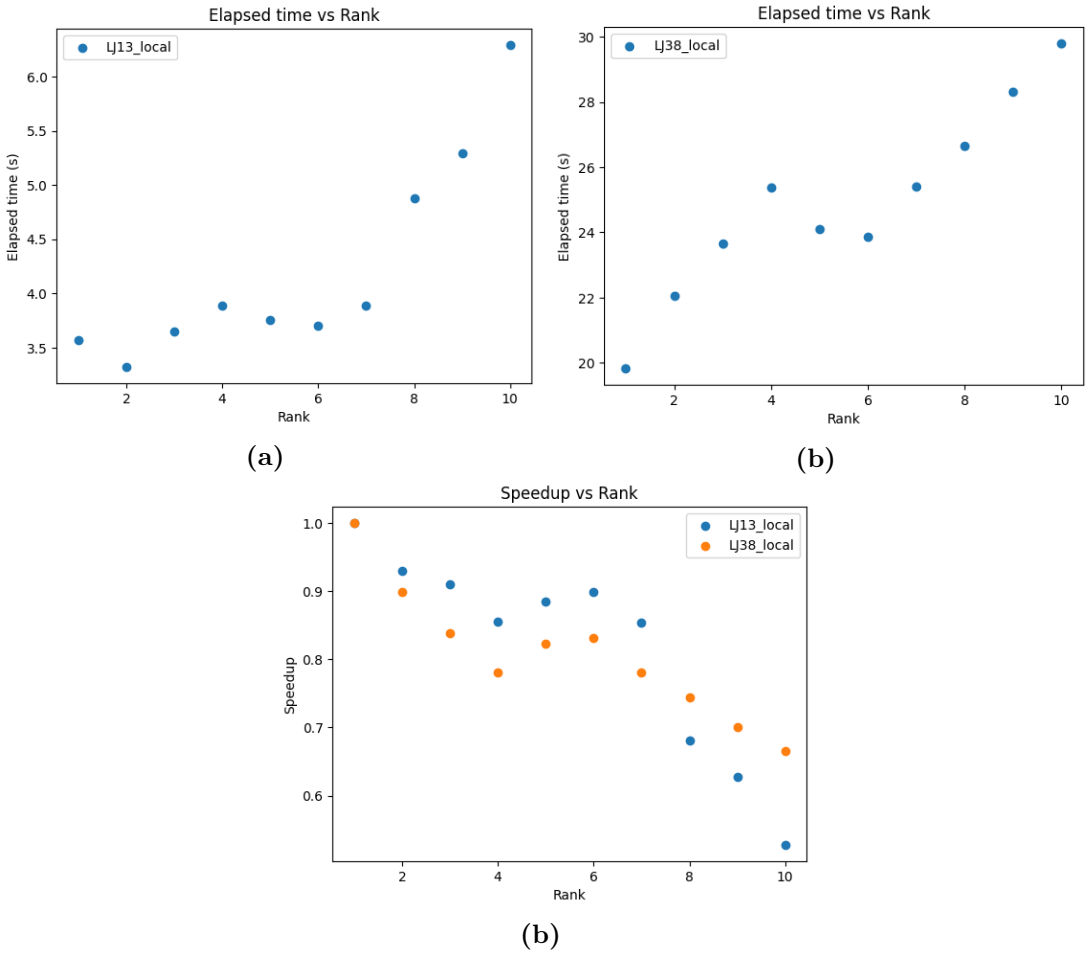
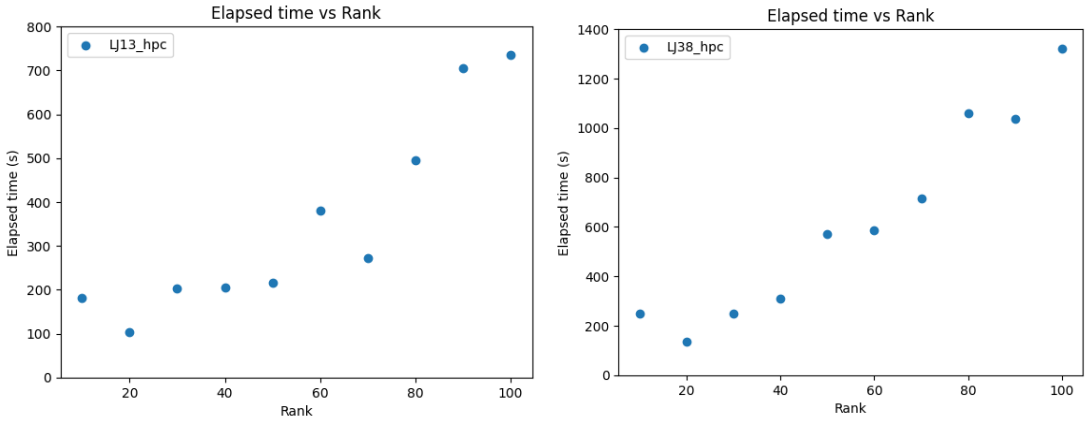



Figure 9: Elapsed time for running LJ13 (a) and LJ38 (b) data sets and speed up percentage against different number of ranks via MPI and TPSD optimisation algorithm.

MPI implemented SHEAP algorithm has also been trialed on the Cambridge HPC facilities. The number of ranks has been chosen from 10 to 100 to examine the performance of the algorithm. Each node contains 76 ranks on a node of the Cambridge HPC- Icelake, therefore a maximum of 2 nodes have been allocated for the experiments. The testing results have been summarised in Figure 10. Comparable trends have been observed for both data sets, i.e. the running time increases with increasing number of node. For both data sets, the running time only fluctuated slightly between the rank number up to 40, then it increased significantly between rank number 60- 100. The communication overhead became very significant if a large number of ranks have been assigned to carry out the tasks. For example, MPI implemented SHEAP algorithm took over 700 seconds to complete the computation with a rank number of 100, compared to only 3.8 seconds with the single rank SHEAP algorithm for LJ13 data set. It is worth noting that the LJ13 dataset is an extreme case to demonstrate the effect of number of ranks on the computational time, as the size of the low dimensional positions matrix is only $794 \times 2$. MPI

implementation would become useful to leverage the power of the parallel computing when the data points are significant higher, such as 100,000 points.

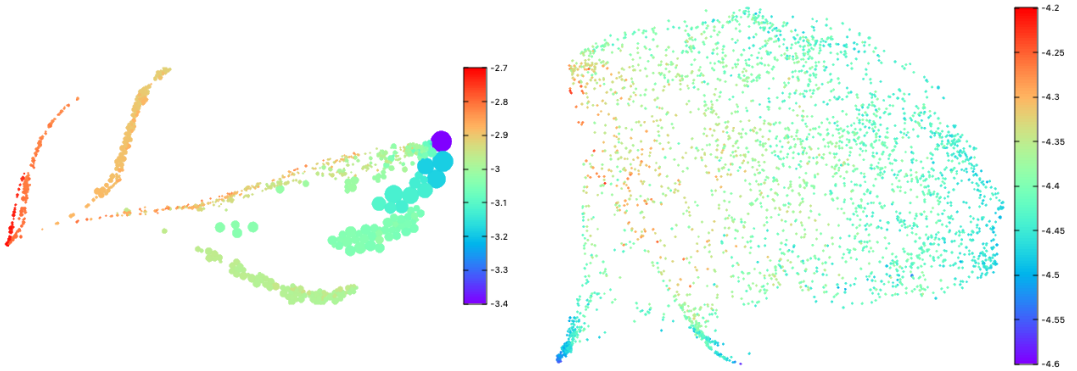



**(a)** **(b)**

Figure 10: Elapsed time for running LJ13 (a) and LJ38 (b) data sets on Cambridge HPC-Icelake via MPI and TPSD optimisation algorithm.

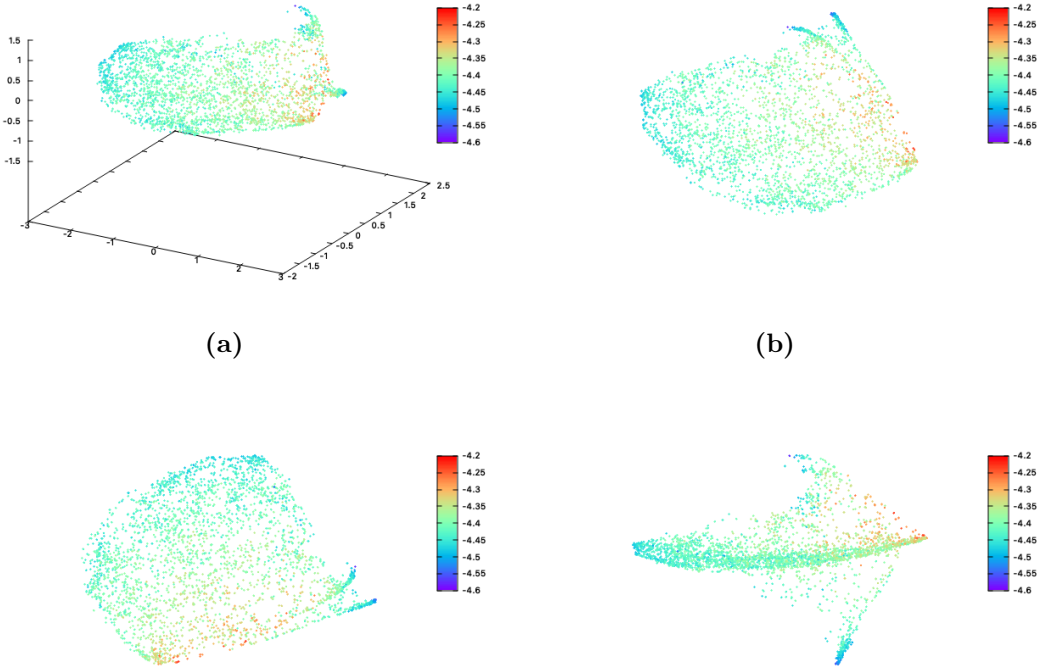
### Reproduciblity

One of the objectives of this project is to reproduce the results from the original paper. The LJ13 and LJ38 data sets have been provided in the example folder of the original SHEAP package. The 2D SHEAP maps have been successfully reproduced for LJ13 and Lj38 data sets. The results have been shown in Figure 11. The colour of each data sphere represents the ion energy of the data point, and the size of the sphere demonstrates the size of the basin volume, i.e. the number of points with comparable minima. It can be seen from Figure 11 (a) that the "single funnel" topology has been successfully reproduced. A large basin has been presented on the right side of the projection, indicating a stable structure. It also suggested that the assumption of manifold learning, i.e. the data sitting in an artificially high dimension, might be validated for the LJ13 data set. It is however worth noting that the SHEAP algorithm visualisation is dependent on the parameters, such as the choice of perplexity and the minimum hard sphere radius. More details will be discussed in the following section. As for LJ38 data set, Figure 11 (b) showed the "brain-like" topology with a double funnel energy landscape. The two separate funnels at the bottom of the landscape could be difficult to describe without the visualisation technique. Three-dimensional SHEAP map has also been reproduced for LJ38 data set, as shown in Figure 12. It can be clearly seen from the figure that the two separate funnels were protruding out of the plane. The "brain-like" topology is actually a plane, which was not showing clearly in the 2 dimensional plot. The other descriptor, SOAP, has not been used in the reproducibility trials, as the aim of the project is to focus on implementing MPI and optimising the SHEAP algorithm.



**(a)** **(b)**

Figure 11: (a)A 2D SHEAP map reproduced from using LJ13 data set (794 distinct minima), the projection used a perplexity of 30, with a minimum sphere radius $R_0 = 0.005$. (b) A 2D SHEAP map reproduced from using LJ38 data set (2966 distinct minima), the projection used a perplexity of 30, with a minimum sphere radius $R_0 = 0.01$The colour represents the ion energy from the data set, and the radius of each point corresponds to the size of the basin volume.



**(a)** **(b)**

**(c)** **(d)**

Figure 12: (a)A 3D SHEAP map reproduced from using LJ38 data set (2966 distinct minima), the projection used a perplexity of 30, with a minimum sphere radius $R_0 = 0.01$. (b) view angle (0,0) (c) view angle (0, 90) (d) view angle (90,0).

**Effect of parameters on the visualisation results**

The choice of the perplexity and the minimum sphere radius parameter might affect the final visualisation results significantly. Because perplexity determines the number of the local neighbours, and the sphere radius parameters determines the minimum radius of the data points, i.e. affecting the cost function. Therefore the effect of the perplexity and sphere radius parameter choices on the visualisation results have been investigated. Figure showed the visualisation results of using a range of perplexity value from 8 to 50, with a sphere radius parameter of 0.005. The results have been summarised in Figure 13. It can be seen that perplexity affects the visualisation results of LJ13 data set significantly with different perplexity values, i.e. 8, 15 or 50. Figure 14(a) showed isolated regions of local minima, i.e. perplexity value of 8. With increasing value of perplexity from 8 to 15, a more comparable visualisation result, i.e. "single funnel" topology, have been observed. With further increasing the perplexity value to 50, the visualisation result was comparable to the original result, i.e. using perplexity value of 30. Additionally, the minimum sphere radius parameter changed the radius of each data point, but the relative position of the data points remained unaffected by increasing the minimum radius from 0.005 to 0.05. The "single funnel" topology could still be observed, Figure 13 (d). As for LJ38 data set, little or no effect of the perplexity values has been observed on the visualisation results, Figure 13 (e) and (f). The "brain-like" cluster of minima with two separate "funnels" have been visualised successfully with perplexity values between 8 and 50. Perplexity value determines the effective number of neighbours. That is, a smaller value might emphasis more on the local connectivity of the the data, whereas a large value could promote to represent global structures. Different perplexity values were chosen for different data sets in the original paper, based on the fact that the choice of perplexity value is dependent on the data set itself. Future work might be able to carry out on how to choose an optimum perplexity value, as certain data set visualisation results can be sensitive towards the parameter.

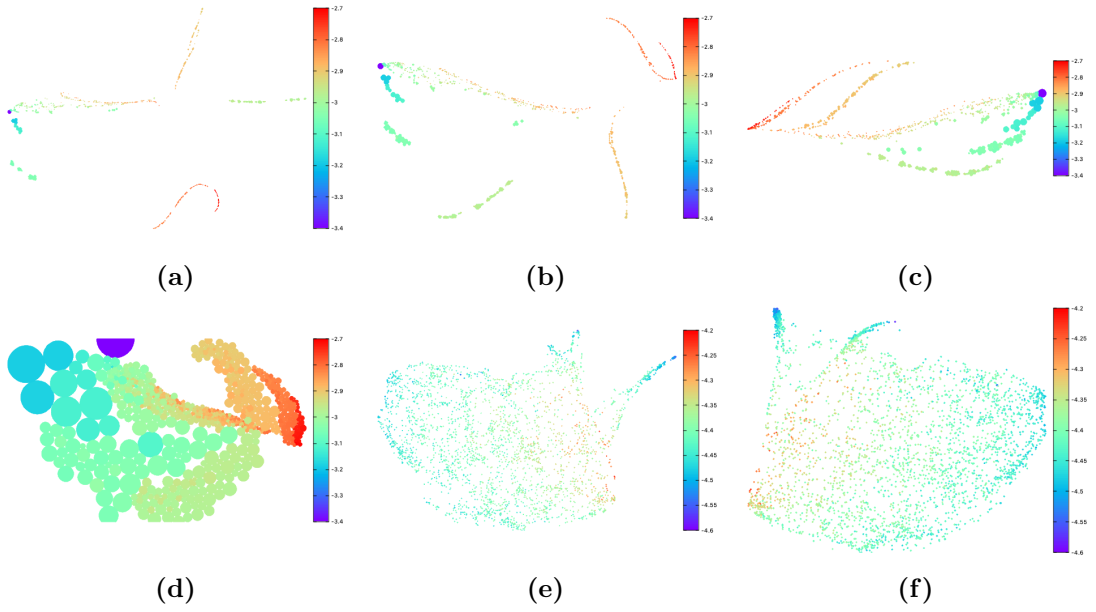



Figure 13: (a)- (d)A 2D SHEAP map reproduced from using LJ13 data set (794 distinct minima), (e)-(f)A 2D SHEAP map reproduced from using LJ38 data set (2966 distinct minima). (a) The projection used a perplexity of 8, with a minimum sphere radius $R_0 = 0.005$, LJ13. (b) The projection used a perplexity of 15, with a minimum sphere radius $R_0 = 0.005$, LJ13. (c) The projection used a perplexity of 50, with a minimum sphere radius $R_0 = 0.005$, LJ13. (d) The projection used a perplexity of 50, with a minimum sphere radius $R_0 = 0.05$, LJ13. (e) The projection used a perplexity of 8, with a minimum sphere radius $R_0 = 0.01$, LJ38. (f) The projection used a perplexity of 50, with a minimum sphere radius $R_0 = 0.01$, LJ38.

**Profiling and Optimisation**

Profiling has been carried out on the SHEAP algorithm to investigate the time consumed on each part of the code. Table 1 has summarised the time percentage for each part of the code. The OpenMP has been left on to reflect real perfomance of the alogrithm, and the number of thread chosen was 8. It can be seen from the table that the most of the running time was consumed by the tpsd gradient descent function, i.e. 73.8% and 87.5% for LJ13 and LJ38 data set respectively.

| | LJ13 | | LJ38 | |
|---|---|---|---|---|
| code | percentage (%) | time (s) | percentage (%) | time (s) |
| read file | 11 | 0.284 | 6.7 | 0.689 |
| normalisation | 0.3 | 0.007 | 0.2 | 0.018 |
| high dimension distribution | 14.7 | 0.381 | 5.5 | 0.56 |
| low dimension distribution | 0.1 | 0.002 | 0.1 | 0.01 |
| tpsd | 73.8 | 1.91 | 87.5 | 8.95 |

Table 1: Actual running time and the percentage of each module using LJ13 and LJ38 datasets.

Regarding the tpsd gradient descent function, two optimisation loops have been applied, i.e. low- dimensional position optimisation without and with the hard sphere growth stage. The first optimisation of position took 28.7 % of the whole running time, and 61 % was taken by the second optimisation loop, i.e. hard sphere growth, as shown in Table 2. It can be seen that the loss gradient calculation took approximately 90% of the tpsd function running time. Therefore, optimising the loss gradient position function and loss gradient core function have been carried out.

| | LJ13 | | LJ38 | |
|---|---|---|---|---|
| code | percentage (%) | time (s) | percentage (%) | time (s) |
| loss gradient position | 33.5 | 0.501 | 28.7 | 4.39 |
| loss gradient core | 58.4 | 0.874 | 61 | 9.32 |
| point radius growth | 0 | 0.001 | 0 | 0.004 |
| gradient add. noise | 4.7 | 0.071 | 0.8 | 0.137 |
| calculate step size | 1.6 | 0.024 | 0.3 | 0.046 |

Table 2: Actual running time and the percentage of tpsd function using LJ13 and LJ38 datasets.

First optimisation has been carried out to reduce the number of loops during optimisation process. Rather than using the total number of data points, the reduced number of points, i.e. the point count $> 0$, has been applied. This was achieved by establishing a separate array to accommodate only the 'valid' data points. There are three potential benefits by doing it, (1) Reducing the number of loops. In LJ13 dataset, the original data point number was 10000. This has been reduced to 794 by removing the duplicated points. (2) Removing the branching, i.e. the original code uses if statements to skip the 'invalidated' data points. (3) More memory efficient. As only the valid data points being involved in the calculation, the cost of reading and writing matrices can be reduced significantly. This is crucial for MPI implementation, as minimising the communication cost is very favourable. As a result, this has reduced the running time of LJ13 data set from 12 seconds to 2.8 seconds in average. Whereas little improvements have been observed for LJ38 data set, as only 34 data points out of 3000 was removed as duplicates.

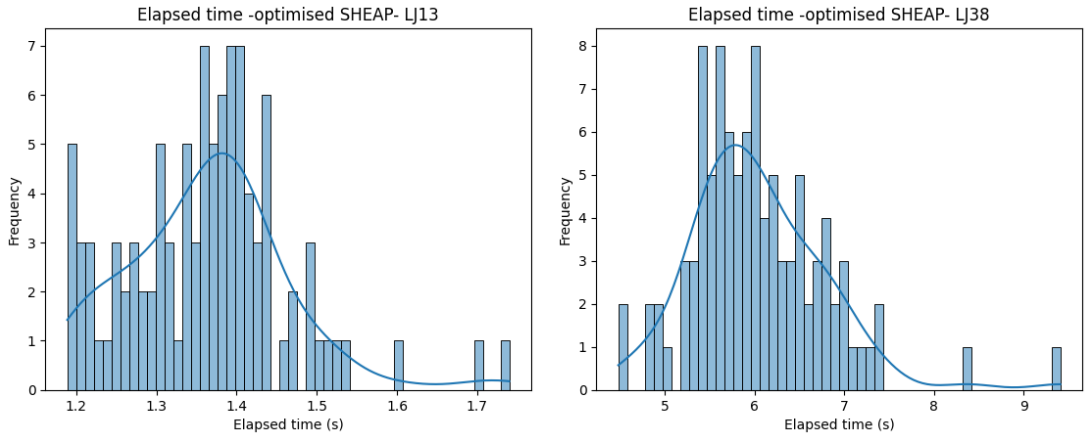
The second optimisation was completed on changing all the matrices into 1 dimensional arrays during optimisation. That is, the low dimensional position matrix and gradient matrix have been flattened into 1D array during optimisation. To access the data point with a certain index i, the index can be written as follow.

```
do i = 1, index
     index_i = (i - 1)*low_dim_params%low_dimension + 1
```

```
index_ii = i*low_dim_params%low_dimension
do j = i + 1, index
   index_j = (j - 1)*low_dim_params%low_dimension + 1
   index_jj = j*low_dim_params%low_dimension
   ...
```

Additionally, branching has been minimised by removing the if statement in the functions. One example is to avoid using `if(point_count(ni).eq.0) cycle` within the loop. Anther strategy is using `merge` functions rather than `if` statements. An example is updating the exaggeration factor. The function has been written as: `exaggeration = merge(1.0_sp, exaggeration_init, condition)`. Therefore no branching was required in this case. Also two separate loss gradient functions, i.e. with and without the hard sphere growth, have been applied onto two separate loops to avoid branching conditions, i.e. checking the growth status at each iteration via `if` statement.

The cost calculation has also been removed to further increase the efficiency of the code. As a result, the speed of SHEAP algorithm has been significantly improved, i.e. 1.4 second for LJ13 data set, and 6.49 second for LJ38 data set. This shows a 9 times and 2 times speed up compared to the original SHEAP code respectively, as shown in Figure 14.




**(a)** **(b)**

Figure 14: Elaspsed time for running LJ13 (a) LJ38 (b) with the optimised SHEAP code. The figures show the time distribution of 100 runs of the programme.

Several other improvements have been made to improve the SHEAP algorithm maintainability. Parameters module has been extended to manage all variables and parameters of the whole SHEAP algorithm. The parameters and variables parsing have been using *Type* construct in FORTRAN. This is to increase the maintainability and the readability of the code. Six types of parameters and variables, i.e. high dimensional parameters, low dimensional parameters, optimisation parameters, file data, high dimensional results, low dimensional results have been established. In this case, due to the nature of the SHEAP algorithm, separating the high- dimensional and low- dimensional parameters and results help the data structure being managed more easily.

## Summary

SHEAP algorithm has been successfully reproduced, and MPI implementation and optimisation have been carried out to improve the maintainability, readability of the code, as well as gaining the capability of running large data sets. It has also been found that the perplexity values could affect the topology of the visualisation significantly in LJ13 data set, therefore selecting appropriate perplexity values can be added into the future work. It has also been found low dimensional position matrix initialisation done by random projection or PCA projection yielded comparable results, and ADAM optimisation algorithm performed better on LJ38 dataset, whereas the tpsd outperformed ADAM on large dataset, i.e .LJ38 in

this case. The final optimised SHEAP code achieved 9 times faster running speed on LJ13 data set, and 2 times faster on LJ38 data set.

# Bibliography


[1] A. R. Oganov and M. Valle, "How to quantify energy landscapes of solids," *The Journal of Chemical Physics*, vol. 130, p. 104504, 03 2009.

[2] B. W. B. Shires and C. J. Pickard, "Visualizing energy landscapes through manifold learning," *Phys. Rev. X*, vol. 11, p. 041026, Nov 2021.

[3] D. J. Wales and T. V. Bogdan, "Potential energy and free energy landscapes," *The Journal of Physical Chemistry B*, vol. 110, no. 42, pp. 20765–20776, 2006. PMID: 17048885.

[4] J. P. K. Doye, M. A. Miller, and D. J. Wales, "Evolution of the potential energy surface with size for lennard-jones clusters," *The Journal of Chemical Physics*, vol. 111, p. 8417–8428, Nov. 1999.

[5] D. J. Wales, "Exploring energy landscapes," *Annual review of physical chemistry*, vol. 69, no. 1, pp. 401–425, 2018.

[6] L. Jaillet, F. J. Corcho, J.-J. Pérez, and J. Cortés, "Randomized tree construction algorithm to explore energy landscapes," *Journal of computational chemistry*, vol. 32, no. 16, pp. 3464–3474, 2011.

[7] F. Anowar, S. Sadaoui, and B. Selim, "Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne)," *Computer Science Review*, vol. 40, p. 100378, 2021.

[8] C. O. S. Sorzano, J. Vargas, and A. P. Montano, "A survey of dimensionality reduction techniques," *arXiv preprint arXiv:1403.2877*, 2014.

[9] L. Van Der Maaten, E. O. Postma, H. J. Van Den Herik, *et al.*, "Dimensionality reduction: A comparative review," *Journal of Machine Learning Research*, vol. 10, no. 66-71, p. 13, 2009.

[10] J. Ye and S. Ji, "Discriminant analysis for dimensionality reduction: An overview of recent developments," *Biometrics: Theory, Methods, and Applications. Wiley-IEEE Press, New York*, 2010.

[11] A. J. Izenman, "Introduction to manifold learning," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 5, pp. 439–446, 2012.

[12] L. Cao, K. S. Chua, W. K. Chong, H. P. Lee, and Q. Gu, "A comparison of pca, kpca and ica for dimensionality reduction in support vector machine," *Neurocomputing*, vol. 55, no. 1-2, pp. 321–336, 2003.

[13] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[14] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction. arxiv 2018," *arXiv preprint arXiv:1802.03426*, vol. 10, 1802.

[15] M. Ceriotti, G. A. Tribello, and M. Parrinello, "Simplifying the representation of complex free-energy landscapes using sketch-map," *Proceedings of the National Academy of Sciences*, vol. 108, no. 32, pp. 13023–13028, 2011.

[16] X. Wang, S. Ramírez-Hinestrosa, J. Dobnikar, and D. Frenkel, "The lennard-jones potential: when (not) to use it," *Physical Chemistry Chemical Physics*, vol. 22, no. 19, pp. 10624–10633, 2020.