Agglomerative Clustering via Maximum Incremental Path Integral: Implementation and Analysis

Unsupervised and Reinforcement Learning

Bernat Comas i Machuca

April 17, 2025

Contents

1 Introduction to the problem

The paper Agglomerative clustering via maximum incremental path integral (2013) [1] presents a new agglomerative clustering algorithm that computes cluster similarity based on path graphs that capture the data structure. To make this computation each cluster is treated as a dynamic system and the algorithm measures stability using the Path Integral, a concept extracted from statistical mechanics and quantum mechanics. In doing so, it introduces a novel hierarchical clustering approach by rethinking how to measure the similarity between clusters during the agglomerative process.

To define the structural descriptors that allow to compute this concept of similarity, we rely on a sample neighborhood graph. This allows the algorithm to avoid relying as much on pairwise distances (only used for graph initialization), and instead define the affinity between clusters by the amount of change of their stability when merged, which is measured through the path integral.

This has several advantages:

- Works well for data lying on low-dimensional manifolds.
- It does not rely on approximation or eigen-decomposition, making it more robust to noise.
- It does not make assumptions on data distribution, offering better generalization.
- Performs well on multi-scale data
- It is efficiently computed by the presented algorithm in linear time complexity

Over the course of this work, the Path Integral Clustering algorithm is going to be implemented in an efficient way, and then it will be compared to other well-known clustering algorithms. In this section we are going to introduce the problem and what is its novelty. Section 2 contains a brief description of the algorithm, without pseudocode. Section 3 contains details of the implementation that made it efficient. Section 4 describes the experiments made on the algorithm and its comparison to other state-of-the-art clustering algorithms. Finally the last section is devoted to unraveling the conclusions of the project.

1.1 Novelty of the Path Integral Clustering

The novelty of Path Integral Agglomerative clustering lies in the way it defines inter-cluster similarity and the algorithmic approach to compute it. Having the advantages stated in the previous section, its novelty can be summed up in the following points.

1. Defining a new linkage criterion based on a path integral formulation: Instead of typical linkage methods like single-link, complete-link, or average-link, which rely on direct distance metrics between points or clusters, this method computes the maximum incremental path integral (MIPI) between clusters.
2. The ”path integral” represents the cumulative similarity (or affinity) along the path connecting two clusters through a series of intermediate points. In this way, the method captures both local and global structure information in the dataset, which is an evolution from pairwise distances.

3. The MIPI criterion considers not just the shortest path between clusters, but the maximum incremental contribution a new link would make to the total path integral when merging clusters. This ensures clusters are merged in a way that preserves high-density connections and prevents weakly connected or noisy points from dominating cluster formation.

This is particularly useful as it balances local density and global connectivity, making it more robust in dealing with datasets where traditional linkage-based hierarchical clustering struggles, such as datasets with complex shapes or varying densities, noisy data, non-convex clusters, or when the distance metric alone isn't enough to capture cluster structure.

2 Algorithm description

This section contains a brief description of how the Path Integral Clustering algorithm works. The algorithm is based on the agglomerative clustering algorithm, which is a bottom-up approach to clustering. It starts with each sample as its own cluster and iteratively merges the most similar clusters until a stopping criterion is met. The stopping criterion in this case is the number of clusters we want to obtain.

2.1 Cluster initialization

Since we are working with an Agglomerative Clustering algorithm, we have to define how will the initial clusters be formed. A straightforward approach would be to assign each sample to its own cluster and iteratively merge them using Path Integral Clustering (PIC). However, the original paper proposes first using a nearest neighbour merging strategy. This algorithm begins by connecting each point to its nearest neighbour, progressively forming larger clusters. While this method can be efficient, it is highly sensitive to noise and outliers, as isolated points may act as unintended bridges between otherwise separate clusters, distorting the true structure of the data.

The algorithm of nearest neighbor merging consists of having each sample in its own cluster together with its nearest neighbor and then, the clusters that share samples are merged. This results in a number of initial clusters that is at least half the number of samples.

The full cluster initialization is defined in the file *"nearest_neighbour_init.py"*, in the function *"cluster_init"*.

2.2 Building the structural graph

We define the structural graph G as the directed graph where each initial sample X is a node and E the set of edges connecting them. Two samples are connected by an edge if and only if they are in its K closest neighbors. To compute distance the paper proposes using the Euclidean distance.

Using this graph G, we compute the weighted adjacency matrix W, which uses the pairwise similarities between the samples. Note that as we are only computing similarities between each point and its K closest neighbors, the rest of similarities will be 0. This means that our matrix will be sparse for big datasets, so we will be able to take advantage of that by using the appropiate functions in the implementation. A formal definition of the elements in W, can be found in Equation 1.

$$w_{ij} = \begin{cases} \exp\left(\frac{-\text{dist}(i,j)^2}{\sigma^2}\right), & \text{if } \mathbf{x}_j \in \mathcal{N}_i^K, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Where K is a free parameter to be set. Sigma squared (σ^2) on the other hand, is estimated by Equation 2 and represents the variance parameter used to compute the weighted adjacency matrix, controlling the sensitivity of point similarity based on their distance. This means that smaller σ^2 values result in stronger connections for closer points, while larger σ^2 values lead to more diffuse connections between distant points.

$$\sigma^2 = [\sum_{j=1}^{n} \sum_{x_j \in \mathcal{N}_i^3} dist(i,j)^2]/[3n(-ln(a))] \tag{2}$$

Where a is a parameter to be set.

We define the transition probability matrix P as the one-step transition probability from vertex i to vertex j based on the weighted adjacency matrix. It is computed using Equation 3, where D is the diagonal matrix of the weighted adjacency matrix W. The first equation, $P = D^{-1}W$, shows that the transition probabilities between points are derived by normalizing the weighted adjacency matrix W using the degree matrix D. The degree matrix D contains the sum of the weights for each point, ensuring that each row in the transition matrix sums to 1, as indicated by the second equation $\sum_{j=1}^{n} p_{ij} = 1$. This normalization ensures that the transition matrix P is row-stochastic, where each entry p_{ij} represents the probability of transitioning from point i to point j, based on their pairwise similarities.

$$\begin{gathered} P = D^{-1}W; \\ d_{ii} = \sum_{j=1}^{n} w_{ij} \text{ such that } \sum_{j=1}^{n} p_{ij} = 1 \end{gathered} \tag{3}$$

These computations are done in the file *"pic.py"*. The function *"_pairwise_similarity_-matrix"* computes W, the function *"_sigma_squared"* computes the variable with the same name and the function *"_transition_probability_matrix"* computes P.

2.3 Affinity measure: Path integral

The path integral of a cluster is computed by summing the paths in the cluster on the directed graph G, weighted by transition probabilities in P.

The path integral used is a discretization of the one seen in quantum mechanics, and is defined as a sum of the weighted contributions of all the paths in the cluster C, divided by the number of samples belonging to the cluster. This is called the path integral descriptor of a cluster C_a, which we call S_{C_a}, and is defined in Equation 4.

$$S_c = \frac{1}{|C|^2} \sum_{\gamma \in \Gamma_c} \Theta(\gamma) \tag{4}$$

Where, Γ_c is the set of all paths in C and $\Theta(\gamma)$ the weight of a path.

Equation 5 represents an efficient PIC implementation of the previous equation. It implements the same concept but avoids explicitly summing over all paths. Instead, it utilizes the transition matrix P_C for the cluster C. Here, $\boldsymbol{P}_C$ is the transition probability matrix for the points in C, and z is a scalar that typically influences the transition matrix. The term $(\boldsymbol{I} - z\boldsymbol{P}_C)^{-1}$ represents the inverse of a matrix that encodes the transitions between points in the cluster. Multiplying by the vector $\mathbf{1}$ (a column vector of ones) effectively sums over the resulting path weights.

Therefore, it efficiently computes the sum of path weights $\Theta(\gamma)$ over all paths in C using matrix operations, as opposed to directly summing over individual paths. The result is then normalized by $|C|^2$, yielding S_c, as in the first equation.

$$S_C = \frac{1}{|C|^2}\mathbf{1}^T(\boldsymbol{I} - z\boldsymbol{P}_C)^{-1}\mathbf{1} \tag{5}$$

We also define the conditional path integral descriptor $S_{C_a|C_a\cup C_b}$ between two clusters S_{C_a} and S_{C_b} as the path descriptor of the clustering resulting from the merge but only taking into account the paths that have starting and ending vertices in C_a. We compute the conditional path integral descriptor using Equation 6.

$$S_{C_a|C_a\cup C_b} = \frac{1}{|C_a|^2}\mathbf{1}_{C_a}^T(\boldsymbol{I} - z\boldsymbol{P}_{C_a\cup C_b})^{-1}\mathbf{1}_{C_a} \tag{6}$$

Now, we can compute the affinity A_{C_a,C_b} of the merging of two clusters C_a and C_b by the increment in the path integral descriptor shown in the previous equations 4 and 6, as can be seen in Equation 7. In addition to the path integral of each cluster separately S_{C_a} and S_{C_b}, the computation of the affinity uses the conditional path integral descriptor $S_{C_a|C_a\cup C_b}$, which computes the resulting path descriptor of the merge but only the ones that have starting and ending vertices in C_a.

$$A_{C_a,C_b} = (S_{C_a|C_a\cup C_b} - S_{C_a}) + (S_{C_b|C_a\cup C_b} - S_{C_b}) \tag{7}$$

The computations of the path integral are found in the file *"path_integral.py"*. The function that computes the affinity is found on the *pic.py* file, and is called *"compute_affinity"*. In addition to this we also implemented function *_compute_affinity_single_link*, used merely for testing purposes of the correct functioning of the global algorithm. This function computes the affinity between two clusters based on the S-link algorithm, and is not used in the final implementation.

2.4 Agglomerative algorithm

The input to our problem is a set of sample vectors together with the target number of clusters (that has to be predefined, as stated in the original paper).

The algorithm starts by building the graph G, weighted adjacency matrix W, and using it to obtain the transition probability matrix P. Then, cluster initialization is run to obtain the initial clusters.

The iterative part consists of merging the two most affine clusters we have until the number of clusters is the target one. To do so, we use the precomputed affinities between all pairs of clusters. The pseudocode of the algorithm is not provided because the coursework description clearly states not to include it.

The implementation of the full algorithm is found in the file *pic.py*, in the function *run*. We ultimately decided to leave some of the prints we used to check the correct functionining of the algorithm for testing purposes, although they are commented.

3 Description of the implementation

In this section we are going to describe the particularities of the implementation of the Path Integral Clustering Algorithm. The first section goes over the implementation techniques recommended by the paper, and the second one contains the extra decisions made during the implementation.

During the implementation, efficiency was our priority. The full implementation of the algorithm is contained in the files *"pic.py"*, *"nearest_neighbour_init.py"* and *"path_integral.py"*. The rest of files are helpers that have contributed to the results obtained but are not part of the implementation itself.

3.1 Efficient computation of the path integral

The main ideas laid out by the paper about the implementation are on the computation of the path integral between two clusters. They use Theorem 1 to avoid computing a matrix inverse but instead solve a linear system.

Theorem 1. s_{ij} always converges, and $s_{ij} = \left[(I - zP_{\mathcal{C}})^{-1}\right]_{ij}$, i.e., the (i,j)-element of $(I - zP_{\mathcal{C}})^{-1}$, where $P_{\mathcal{C}}$ is the submatrix of P by selecting the samples in $\mathcal{C}$. If we define $S_{\mathcal{C}} = [s_{ij}]_{i,j\in\mathcal{C}}$, we have $S_{\mathcal{C}} = (I - zP_{\mathcal{C}})^{-1}$. Then, we can compute the path integral as the structural descriptor of cluster $\mathcal{C}$ as follows:

$$S_{\mathcal{C}} = \frac{1}{|\mathcal{C}|^2}\mathbf{1}^T S_{\mathcal{C}}\mathbf{1} = \frac{1}{|\mathcal{C}|^2}\mathbf{1}^T (I - zP_{\mathcal{C}})^{-1}\mathbf{1} \tag{8}$$

where $\mathbf{1}$ is all-one column vector.

Proposition 1 $(I - zP_{\mathcal{C}})$ is a strictly diagonally dominant matrix with the ∞-norm condition number no more than $(1 + z)/(1 - z)$.

Both Theorem 1 and Proposition 1 are proven in the original paper [1], so we will not go into more detail here. Taking them into account, we see that the computation of $S_{\mathcal{C}}$ only involves solving a linear system

$$\begin{aligned} (I - zP_{\mathcal{C}})\mathbf{y} &= \mathbf{1}, \\ S_{\mathcal{C}} &= \frac{1}{|\mathcal{C}|^2}\mathbf{1}^T\mathbf{y}. \end{aligned} \tag{9}$$

For a large cluster, $(I - zP_{\mathcal{C}})$ is sparse, which means it follows Proposition 1 and can be solved using iterative methods. This will help us compute both the incremental path integral and the path integral descriptor.

$$S_{c_a|c_a \cup c_b} = \frac{1}{|c_a|^2} \mathbf{1}_{c_a}^T (I - zP_{c_a \cup c_b})^{-1} \mathbf{1}_{c_a} \tag{10}$$

To compute the path integral descriptor as shown in Equation 8, we first compute $(I - zP_{c_a \cup c_b})$, which is straightforward. Then, following equation 9, we solve the linear system (we can use *spsolve*). Finally, multiplying by the transposed matrix of 1s is the same as summing all the elements in the matrix resulting from the linear equation's solving, and finally we can compute the multiplication with the fraction. Computing the incremental path descriptor takes advantage of the same trick.

3.2 Implementation decisions

To ensure that our algorithm runs in linear time, we must precompute as much as possible, including the graph, its transition matrices, and the affinities between all pairs of clusters. Since the algorithm merges clusters greedily, we aim to avoid recalculating affinities at each iteration. Instead, we will compute the affinities for all cluster pairs upfront, and only update those affected by a merge. When two clusters are merged, we only need to compute the affinities between the new cluster and the remaining clusters, and we can avoid recalculating all pairwise affinities.

In addition to this, we are using a heap to store all the cluster pairs and their affinities for fast extraction in runtime. This brought us to use a set to store the clusters that are active in a moment, to avoid having to traverse the heap to remove all pairs of clusters that contains one of the merged ones at each iteration. This means that at every iteration we are flagging the merged clusters as inactive, computing the affinities between the new cluster and all the previous ones, and adding them to the heap.

Due to the nature of our algorithm, we only implement the fit-predict method. This is because the algorithm does not learn from data but instead applies the same process to the samples to cluster them independently from the data it has seen.

Another decision we made to improve efficiency that was not stated in the paper is to take advantage that the algorithm always computes K-nearest-neighbours and also 3-nearest neighbours by using the same graph for both. This results in only having to compute the K-nearest-neighbours graph and then use it to compute the 3-nearest-neighbours graph (for K¿3, otherwise we compute K=3 and extract the other graph). This is because for K¿3 the 3-nearest-neighbours graph is a subset of the K-nearest-neighbours graph, and this can save us a lot of time considering we compute the nearest neighbours of all the samples in the dataset. Furthermore, in the computation of the pairwise similarity matrix, we use the distances calculated by the KNN algorithm to prevent calculating them again.

The file *"path_integral.py"* contains the implementation of the path integral descriptor and the incremental path integral descriptor, but there is also an implementation for the inefficient computation of the incremental path integral which was used to verify that the linear system solving method proposed by the paper was correct. This implementation is not used in the final version of the algorithm, but it is there for testing purposes.

All the operations that involve the clustering algorithm use numpy, using matrices as numpy arrays, and taking advantage of the different tools we have to work with sparse matrices (*spsolve*, *csc_matrix*, *eighsh*). This is because the algorithm is designed to work with large datasets, and we want to avoid using too much memory. We also use the *scipy* library to compute the nearest neighbours graph, as it is a well-known library for scientific computing in Python, and the *heapq* library for efficient heap implementation.

4 Experiments

In this section we will conduct experiments comparing different clustering algorithms on several datasets. The algorithms under evaluation include Affinity Propagation, Complete Link, Average Link, Simple Link, Zeta clustering (Zell), Difussion clustering, and of course, our implementation of PIC. Each algorithm will be tested on six datasets: MNIST, USPS, Breast Cancer (Wisconsin), and three differently-structured synthetic random datasets with 1000 samples each. We are going to divide the experiments section into several subsections. In the first subsection, we are going to explain the decisions regarding the methodology of the experiments. In the second one, we are going to talk about the metrics we are using to compare the models. Next, we are going to analyse the performance of the algorithms over the synthetic datasets. In section four, we are going to run the algorithms on the complete datasets to observe what are the differences in their performances, using a diverse set of metrics. Finally, we are going to analyse how does the introduction of gaussian noise and structural noise affect the performance of the different algorithms, similar to what is done in the original paper.

4.1 Methodology

The choice of the algorithms we are using is due to the fact that we only have found implementations for the Affinity Propagation, Complete, Average and Simple Links. Although in the project we only had to implement PIC, we decided to also implement some of the algorithms that get the most similar results in the paper, and the selected algorithms to implement are the diffusion Kernel (D-kernel) and Zeta function clustering (Zell), because they are shown to be better than PIC in some domains by the original paper. To do the PIC experiments, we are using exactly the same parameters recommended by the original paper, which are: $K = 20$, $a = 0.95$ and $z = 0.01$. The rest of the algorithms are using their default parameters, as we are not trying to tune them but rather compare them.

The datasets chosen are also extracted from the paper: MNIST and USPS. We opted not to use the FRGC ver2.0, PubFig, and Caltech datasets for this study due to their large size, which would require significant computational resources and time for processing, apart from the approach the paper takes on loading them, which uses spatial pyramid features. Instead, we have decided to incorporate the Breast-cancer wisconsin dataset because it is a well-known benchmark in the medical domain, containing a high-dimensional, real-world classification and being widely used. In addition, being a binary classification dataset, it stands out from the rest and means that the algorithms will have to iterate until only two clusters are remaining, apart from being something not tested in the original paper.

It is worth noting that in our work, we used the original USPS dataset containing 9,298 samples, rather than the 11,000-sample version referenced in the paper. We chose to work with the authentic USPS dataset, as the source and details of the augmented version with additional samples used by the paper were unclear.

We don't have access to the synthetic datasets that are used in the paper, and for this reason we have decided to create our own. It will be through the synthetic datasets that we will study how does structural and gaussian noise affect our implementation of the clustering algorithm. To have datasets as diverse as possible, we are going to use three different creation methods, the first one using *moons*, second one using concentric *circles* and last one using *blobs*. Our synthetic datasets will have 1000 samples, 2 features for ease of representation (same as the ones in the paper) and 10 target clusters.

Table 1: Statistics of used datasets. Synthetic represents the three synthetic datasets: *blobs*, *moons* and *circles*. For the synthetic datasets, noise will be added for the noise analyisis, but not for the clustering analysis.

Dataset	**USPS**	**MNIST**	**BC-Wisconsin**	**Synthetic**
No. of samples	9 298	5 139	569	1 000
No. of clusters	10	5	2	10
Min. cluster size	708	980	212	-
Max. cluster size	1 553	1 135	357	-
Noise added	No	No	No	Yes
Dimensionality	256	784	30	2

4.2 Metrics to be used

The metrics we will be using to compare the methods include the same ones that are used in the paper: Normalized Mutual Information (NMI) score and Clustering Error (CE). This will enable us to have comparable results to the ones presented. As both of these are external metrics we decided to incorporate some internal ones in addition to NMI and CE: Silhouette score, Davies-Bouldin index, and Calinski-Harabasz index. We have selected these three internal metrics because they are the ones recommended as internal criteria for a wide range of situations by the "Clustering evaluation and validation" chapter of the material of the Unsupervised Learning course [2]. Their implementations can be seen in Equations 11 to 15.

Normalized Mutual Information (NMI):

$$\text{NMI}(U, V) = \frac{2 \times I(U; V)}{H(U) + H(V)} \tag{11}$$

where:

- $I(U; V)$ is the mutual information between cluster assignment U and ground truth labels V.
- $H(U)$ and $H(V)$ are the entropies of U and V.

Clustering Error (CE):

$$\text{CE} = 1 - \frac{1}{N} \sum_{i=1}^{N} \not{\vdash}\{\hat{y}_i = y_i\} \tag{12}$$

where:

- N is the total number of samples.
- $\hat{y}_i$ is the predicted cluster label after optimal matching.
- y_i is the ground truth label.
- $\nVdash\{\cdot\}$ is the indicator function (1 if true, 0 otherwise).

Silhouette Score:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{13}$$

where:

- $a(i)$ is the average distance from point i to all other points in the same cluster.
- $b(i)$ is the minimum average distance from point i to points in a different cluster.

The overall Silhouette Score is the mean of $s(i)$ over all points.

Davies-Bouldin Index (DBI):

$$\mathrm{DB} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{S_i + S_j}{M_{ij}} \right) \tag{14}$$

where:

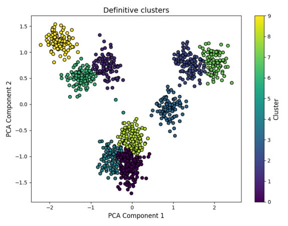
- k is the number of clusters.
- S_i is the average distance between each point in cluster i and its centroid.
- M_{ij} is the distance between the centroids of clusters i and j.

Calinski-Harabasz Index (CH):

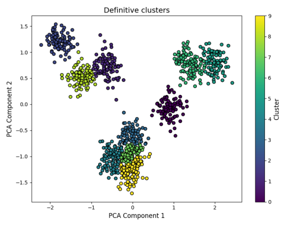
$$\mathrm{CH} = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{N - k}{k - 1} \tag{15}$$

where:

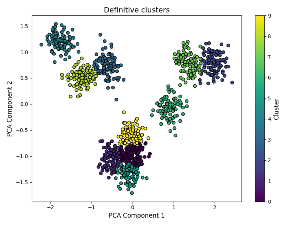
- N is the total number of points.
- k is the number of clusters.
- $\mathrm{Tr}(B_k)$ is the trace of the between-cluster dispersion matrix.
- $\mathrm{Tr}(W_k)$ is the trace of the within-cluster dispersion matrix.



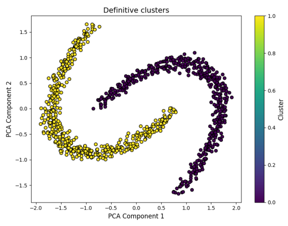
(a) A-Link (0.8482)



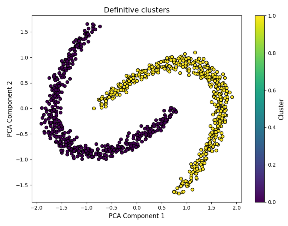
(b) PIC (0.8429)



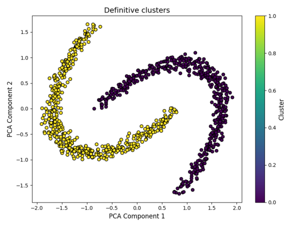
(c) Zell (0.8403)



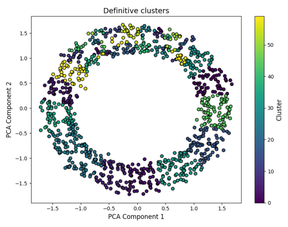
(d) PIC (1.0)



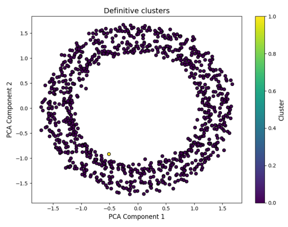
(e) A-Link (1.0)



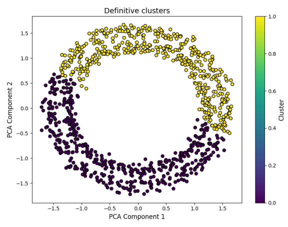
(f) S-Link (1.0)



(g) AP (0.083)



(h) S-Link (0.002)



(i) D-kernel (0.00023)

Figure 1: Clustering results on the three synthetic datasets. Between parentesis, the NMI score for the algorithm presented on the respective dataset.

4.3 Synthetic datasets

In this section, we present the results on the synthetic datasets. In Figure 1 the three best algorithms for each one of the synthetic datasets are presented. The complete data for the synthetic datasets can be observed in Table 2.

The results obtained from the synthetic datasets reveal several interesting patterns. In the case of the first dataset, *blobs*, shown in images (a), (b), and (c) of Figure 1, all algorithms performed well, with the top three achieving nearly identical results, each surpassing 0.84 in Normalized Mutual Information (NMI). This shows the ease of separating clusters in this dataset.

For the second dataset, *moons*, results indicate that while three algorithms managed to perfectly separate the data, suggesting the dataset is not particularly challenging, the remaining algorithms showed significantly weaker performance. This disparity can be attributed to the specific characteristics and structure of the dataset, which may not align well with the assumptions of those methods.

The third dataset, *circles*, proved to be the most difficult for the evaluated algorithms. As similarity-based clustering methods, the algorithms struggled to capture the circular structure inherent in the data, resulting in consistently poor performance. In this case, the best-performing algorithm was Affinity Propagation (AP), though its

NMI score of 0.08 can not be seen as a good result, as is represented in Figure 1. While it is problematic that AP picks up on noise, leading to suboptimal clustering, its performance still seems relatively better when compared to the other algorithms, which fail with scores below 0.01. In this context, AP's score of 0.08 may appear to be a significant improvement. However, despite this apparent advantage, a score of 0.08 still indicates poor clustering performance, and its ability to incorporate noise and not being able to identify the correct number of clusters actually undermines its effectiveness, highlighting the overall challenge of dealing with this dataset.

Compared to the results reported in the original paper, ours are not as distinctly favorable toward the PIC algorithm. Nevertheless, it remains evident that PIC consistently performs well on both the first and second datasets, being a very close second-best on *blobs* and the joint best one on *moons*. However, similar to the other algorithms evaluated, PIC struggles with the third dataset, showing its common limitation when dealing with data of this structure.

4.4 Algorithm comparison

This section presents a comparative analysis of our clustering algorithms across the three non-synthetic datasets: USPS, MNIST and BC-Wisconsin, similarly as done in the original paper. The results are summarized in Table 2 and Table 3. The first table presents the results in terms of Normalized Mutual Information (NMI), while the second one shows the clustering error. We decided to only add the results for the synthetic datasets to the NMI table because we had computed them for noise analysis (with structural noise at 0), and we thought it would be interesting to see how they compared to the rest of the datasets, although the original paper does not add them.

Table 2: Quantitative clustering results in NMI. The best values are in bold.

Dataset	USPS	MNIST	BC-W	Blobs	Moons	Circles
AP	0.5176	0.4170	0.2596	0.8028	0.2772	**0.0831**
A-link	0.01128	0.0016	0.0151	**0.8482**	**1.0**	0.0
S-link	0.0019	0.0016	0.0102	0.7067	**1.0**	0.002
C-link	0.1592	0.0016	0.0102	0.8225	0.5640	0.0
Zell	0.5743	**0.6293**	**0.5679**	0.8404	0.0677	0.0001
D-kernel	0.2455	0.0038	0.0102	0.8262	0.5616	0.0002
PIC	**0.8708**	0.0063	0.0102	0.8430	**1.0**	0.00005

Table 3: Quantitative clustering results in Clustering error.The best values are in bold.

Dataset	USPS	MNIST	BC-Wisconsin
AP	0.9272	0.9527	0.8963
A-link	0.8285	0.7785	0.3673
S-link	0.8328	0.7785	0.3691
C-link	0.7634	0.7785	0.3691
Zell	0.4450	**0.3592**	**0.0844**
D-kernel	0.6698	0.7772	0.3691
PIC	**0.1150**	0.7772	0.3691

The Normalized Mutual Information (NMI) metric quantifies the similarity between clustering assignments and ground truth labels, where higher values indicate better alignment. According to the original paper, the Path Integral Clustering algorithm outperformed the rest on the USPS and MNIST datasets. However, the current implementation and evaluation reveal some notable differences.

On the USPS dataset, PIC indeed achieves the highest NMI value of 0.8708, very far from the rest of the algorithms, confirming the original paper's claim and proving its strong performance in capturing the underlying cluster structure in a somewhat challenging dataset, considering we are clustering written numbers solely using the pixels. The NMI result presented by the original paper was 0.825 which is close enough to the one we obtained, and this is a good indicator that our implementation is working correctly.

For MNIST, on the other hand, we are presented with a significant divergence from the paper's findings. While PIC is reported as the top performer in the original study, it achieves only 0.0063 NMI here, with Zell clearly outperforming it at 0.6293. This inversion suggests that in this replication, either PIC struggled to converge to meaningful clusters, or its assumptions were not well-suited to the MNIST data in this particular implementation. It could also imply that Zell, perhaps under different parameter settings or with the specific MNIST preprocessing used here, was better able to exploit the dataset's structure.

On BC-Wisconsin, which was not reported in the original paper, Zell again clearly leads with an NMI of 0.5679, followed by AP at 0.2596. All other methods, including PIC, performed very poorly here, indicating that Zell may be particularly effective in handling this dataset's cluster characteristics, likely benefitting from its capacity for detecting more globally separated clusters, as seen in the CH scores of table 6.

Table 4: Quantitative clustering results in Silhouette score

Dataset	**USPS**	**MNIST**	**BC-Wisconsin**
AP	0.077	0.0558	0.0870
A-link	0.2806	**0.7832**	0.6339
S-link	0.3089	**0.7832**	**0.6606**
C-link	0.1945	**0.7832**	**0.6606**
Zell	0.0681	-0.0389	0.3263
D-kernel	**0.6698**	0.6349	**0.6606**
PIC	0.1390	0.5941	**0.6606**

Table 5: Quantitative clustering results in DBI

Dataset	**USPS**	**MNIST**	**BC-Wisconsin**
AP	1.8701	1.5212	1.3622
A-link	1.1553	**0.1325**	0.6798
S-link	**0.4623**	**0.1325**	**0.4497**
C-link	2.4111	**0.1325**	**0.4497**
Zell	2.0193	3.3001	1.3537
D-kernel	2.1858	3.0414	**0.4497**
PIC	2.3121	1.5937	**0.4497**

Table 6: Quantitative clustering results in Calinski-Harabasz

Dataset	USPS	MNIST	BC-Wisconsin
AP	68.3675	25.6472	39.0207
A-link	22.2013	52.8586	32.4822
S-link	4.5830	52.8586	27.8726
C-link	242.2214	52.8586	27.8726
Zell	**697.2199**	**185.1767**	**258.9817**
D-kernel	337.163	46.1865	27.8726
PIC	517.6182	27.6675	27.8726

The Silhouette Score, which measures how similar an object is to its own cluster compared to other clusters, revealed that on the MNIST dataset, the A-link, S-link, and C-link methods all achieved identical and notably high scores of 0.7832. This indicates that these methods produced clusters with a high degree of compactness and separation. On the USPS dataset, the D-kernel method outperformed others with a Silhouette Score of 0.6698, suggesting it formed well-defined clusters in this setting. For the BC-Wisconsin dataset, the S-link, D-kernel, and PIC methods all tied with a strong score of 0.6606.

When considering the Davies-Bouldin Index (DBI), where lower values signify better clustering, consistent patterns are found. On the USPS dataset the S-link method achieves the lowest DBI at 0.4623, outperforming other algorithms in terms of minimizing intra-cluster variance while maximizing inter-cluster separation. For the BC-Wisconsin dataset, the S-link, C-link, D-kernel, and PIC methods shared a DBI value of 0.4497, proving their effectiveness. Zell, D-kernel, and PIC generally performed worse in this metric on MNIST and USPS, despite competitive performances elsewhere.

The Calinski-Harabasz Index (CHI), which rewards high between-cluster dispersion and low within-cluster dispersion, offered a contrasting perspective. The Zell method achieved exceptionally high CHI scores on all three real datasets, particularly on BC-Wisconsin, where it recorded a value of 258.98 clearly surpassing other algorithms. On the USPS dataset, Zell again led with a CHI of 697.22, followed by PIC and D-kernel with notable but lower scores. Although Zell consistently achieved high CHI values, it did not perform as well in the Silhouette and DBI metrics, suggesting that while it can create globally dispersed clusters, they might lack local compactness.

In summary, the results highlight that S-link and A-link methods exhibit balanced, reliable performance across all datasets and metrics, making them strong general-purpose and fast clustering techniques. The D-kernel method excels particularly on the USPS dataset if we only look at internal metrics, while Zell demonstrates outstanding performance in terms of global cluster dispersion (as measured by CHI) but falls behind in terms of compactness and separation as captured by the Silhouette and DBI scores.

We see that for our particularly implemented PIC algorithm, internal metrics show mixed results across the different datasets. In terms of Silhouette score, PIC performs relatively well on BC-Wisconsin, achieving the highest score of 0.6606, but fares worse on USPS and MNIST, with scores of 0.1390 and 0.5941, respectively. The DBI values reflect a similar trend, where PIC has the highest value of 2.3121 on USPS, suggesting less distinct clusters, while it performs better on BC-Wisconsin, with a DBI of 0.4497, indicating more compact and well-separated clusters. As for Calinski-Harabasz, PIC achieves high values on USPS (517.6182) and BC-Wisconsin (27.8726), but struggles on MNIST, with a low value of 27.6675. These results indicate that while the PIC algorithm may generate good clustering quality in some cases, its performance is dataset-dependent, with significant variation in the internal metric scores across different domains.

That being said, when using internal metrics to evaluate clustering results on labeled data, it's essential to keep in mind that these metrics evaluate the inherent structure of the clustering, without considering the true labels. They do not directly indicate how well the clusters align with the predefined classes, which means an algorithm can achieve strong internal scores while still poorly matching the actual labels, especially if the data's natural structure doesn't correspond to the given categories. The results observed here illustrate this situation clearly.

Nonetheless, internal metrics remain valuable even in labeled scenarios as ours, as they provide complementary information about the structure of the data and quality of the formed clusters. Using internal metrics can help detect situations where the labeled classes themselves might not reflect the underlying structure of the data, or where an algorithm achieves high label agreement through overfitting or by forming poorly separated clusters. In this sense, combining internal and external metrics offers a more complete and reliable assessment than the one presented in the original paper.

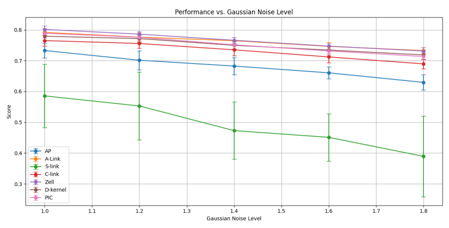
4.5 Noise analysis

In this section we will run experiments using the previously described synthetic dataset incorporating different levels of gaussian noise and structural noise to see how do the different methods behave. Because the addition of gaussian and structural noise is done randomly, we have followed the same practices from the original paper and repeated all the experiments 20 times, showing the mean and the standard deviation at each noise level.

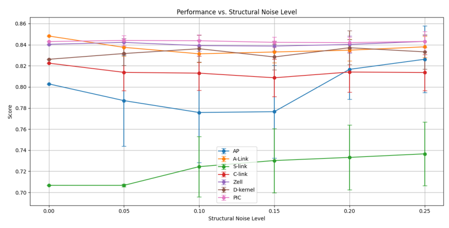
For the first dataset, *blobs*, there is a clear decline on the performance of all algorithms when applying gaussian noise. Despite this overall decline, the relative ranking of the algorithms remains mostly consistent, with each method maintaining its position in comparison to the others throughout the noise levels. Zell is the one that gets the top results at all noise points but the rest of algorithms are very close, with the exception of S-link, AP and C-link, which are clearly performing worse. It is notable to see that the standard deviation of the results is proportional to the NMI scores the algorithms are getting, meaning that the algorithms that are performing better are also more stable.

The structural noise results on the first dataset are very relevant because we can see the resistance of PIC to noise, as stated by the original paper. While most of the algortihms are affected by the structural noise, PIC even increases its NMI score keeping it at over 0.84 with a quarter of the dataset removed. Its standard deviation is also small.

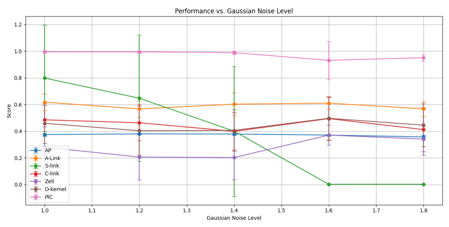
On the second dataset, *moons*, the results are fully supporting what the paper stated, not only does PIC perform the best, but it also is the most resistant to noise. The gaussian noise almost has no effect on the performance of the algorithm, and the structural noise does not affect it. It is relevant to see how A-link and S-link have a steep decline in score with the addition of noise, showing immense variability between the different runs, while PIC is the only one that manages to keep its score.



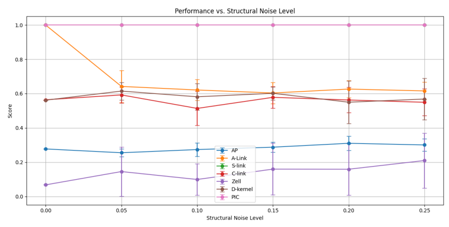
(a) Gaussian noise level (Blobs)



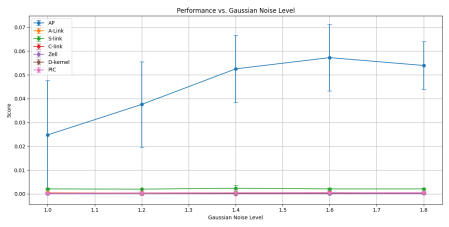
(b) Structural noise level (Blobs)



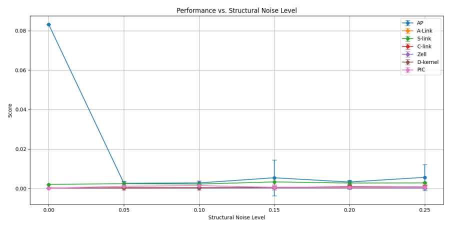
(c) Gaussian noise level (Moons)



(d) Structural noise level (Moons)



(e) Gaussian noise level (Circles)



(f) Structural noise level (Circles)

Figure 2: Mean and standard deviation of NMI results for each one of the datasets when applying gaussian noise (left column) and structural noise (right column). Every experiment was repeated 20 times. Structural noise level is defined as the proportion of removed points from the original dataset.

On the *circles* dataset, which we had shown to be more challenging to our algorithms, we see that all of the algorithms perform poorly, with the best one being AP with a score of 0.08. Here we can see a curious effect, with how the addition of gaussian noise has a favorable effect on the AP algorithm, fitting the noise and increasing its NMI score. With the structural noise the opposite happens, and all of the algorithms end up performing equally bad.

In conclusion, we have seen that PIC is the most structural-noise resistant algorithm in our experiments, while having very promising results under the effect of gaussian noise.

5 Conclusions

This paper has presented some challenges in the implementation of the Path Integral Clustering algorithm. The unavailability of the original synthetic datasets used in the paper made it difficult to replicate the results. This however permited us to create our own datasts, which were diverse enough to have very varied results that challenged the claims of the paper. In addition, the original paper clearly states the hyperparameter combinations that they use to do the testing, so this was straightforward and we did not need to experiment with them.

Thanks to the paper's step-by-step development, implementing the Path Integral Agglomerative Clustering was straightforward. Every computation step, constructing the kernel matrix, summing contributions along candidate paths, and normalizing the resulting affinities, was laid out in algorithmic form, so one could follow the derivation without diving into the full formalism of quantum-mechanical path integrals. In practice, this meant we could treat the "path integral" simply as a weighted sum over link sequences, implement the recurrence relations directly, and easily test against our datasets. The only aspect left largely to our own judgment was the overall workflow: how to organize data preprocessing, similarity initialization, and merge order, so we experimented with a couple of pipeline layouts before choosing the heap one presented.

The main issue we found during the implementation was regarding the heap. When trying to dinamically update the clusters that were removed at each merging point, we were modifying the affinities of the wrong clusters, which was hard to detect and took substantially longer than expected. We ended up using the approach explained, where a set is used to store the active clusters and a heap is used to store the clustering affinities for easy access to the pair with the highest one. This meant that we could keep the heap inmutable and it allowed the algorithm run in linear time, which was one of the main goals of the project.

Regarding the claims of the paper, we have seen that the PIC algorithm indeed outperforms state-of-the-art clustering algorithms in some datasets, but not in all of them. We have seen that the algorithm is very resistant to structural noise, and that it performs well on the USPS dataset, but it does not perform as well on MNIST. For the BC-Wisconsin dataset, which is a biclass classification dataset, Zell also takes the edge in external metrics, but it is very interesting to observe that it is the worst performer both in Silhouette score and DBI. In addition, PIC takes considerably longer than the algorithms it is being compared to in this report.

The PIC algorithm on the other hand was proven to be very resistant to gaussian and specially structural noise clearly beating the rest in most of the cases. The one dataset where it performed poorly is in the *circles* dataset, which turned out to be the most challenging for all of the algorithms.

It is important to note that in "easier" datasets (compared to USPS, MNIST, and BC-Wisconsin), such as *blobs* and *moons*, the PIC algorithm consistently outperforms most other algorithms, with A-link and S-link being its closest competitors in terms of NMI. These latter two are relatively simple algorithms. In the case of the *moons* dataset, we observe that all other algorithms perform poorly, which underscores the effectiveness of PIC in handling such datasets.

In conclusion, this work has proven that the PIC algorithm is a highly promising and reliable clustering method, showing notable resistance to noise and strong performance across a variety of datasets. While it does not consistently outperform all other algorithms in every dataset, differently from what was laid out in the original paper, it has proven to be a reliable algorithm, often surpassing alternative methods and delivering consistent results. The implementation developed in this work successfully runs in linear time, and has been evaluated on both datasets proposed in the original paper and additional, diverse ones. Additionally, we carried out a detailed analysis of how the algorithm behaves with different levels of noise, which confirmed its practical usefulness and reliability in difficult situations.

6 Bibliography

[1] Wei Zhang et al., *Agglomerative clustering via maximum incremental path integral*, 2013 (Link)

[2] Javier Béjar, URL - 2025 Spring Term *Clustering evaluation and validation*